ADSP-CM41x Mixed-Signal Control Processor

with ARM Cortex-M4/ARM Cortex-M0 and 16-Bit ADCs Hardware Reference

Revision 1.0, December 2018

Part Number
82-100127-01

Analog Devices, Inc.
One Technology Way
Norwood, MA 02062-9106

ANALOG DEVICES

Notices

Copyright Information

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, Blackfin, Blackfin+, CrossCore, EngineerZone, EZ-Board, EZ-KIT Lite, EZ-KIT Mini, EZ-Extender, SHARC, SHARC+, A^2B, and VisualDSP++ are registered trademarks of Analog Devices, Inc.

SigmaStudio is a trademark of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

Contents

ARM Cortex-M4 Core 0 Memory Sub-System

Security

System Crossbars (SCB)

Clock Generation Unit (CGU)

Oscillator Comparator Unit (OCU)

System Protection Unit (SPU)

Dynamic Power Management (DPM)

System Event Controller (SEC)

Trigger Routing Unit (TRU)

Trigger Timing Unit (TTU)

Static Memory Controller (SMC)

Flash Controller (FLC)

System Memory Protection Unit (SMPU)

Cyclic Redundancy Check (CRC)

Direct Memory Access (DMA)

General-Purpose Ports (PORT)

Watchdog Timer (WDOG)

General-Purpose Counter (CNT)

Pulse-Width Modulator (PWM)

Universal Asynchronous Receiver/Transmitter (UART)

Two-Wire Interface (TWI)

Controller Area Network (CAN)

Serial Peripheral Interface (SPI)

Serial Port (SPORT)

Analog-to-Digital Converter Controller (ADCC)

Digital-to-Analog Converter Controller (DACC)

Fast Over Current Protection (FOCP)

Harmonic Analysis Engine (HAE)

Sinus Cardinalis (SINC) Filter

MATH Accelerator Unit

Floating-Point Saturation Unit (FSAT)

Logic Block Array (LBA)

Mailbox (MBOX)

Voltage Monitoring Unit (VMU)

FFT Accelerator Block (FFTB)

Reset Control Unit (RCU)

System Design and Safety

System Block (SYSBLK) and PADS

Boot ROM and Booting the Processor

System Watchpoint Unit (SWU)

JTAG debug and Serial Wire Debug Port (SWJ-DP)

Preface

Thank you for purchasing and developing systems using an ADSP-CM41x processor from Analog Devices, Inc.

Purpose of This Manual

The ***ADSP-CM41x Mixed-Signal Control Processor with ARM® Cortex®-M4/M0 and 16-Bit ADCs Hardware Reference*** provides architectural information about the ADSP-CM41x processors. This hardware reference provides the main architectural information about these processors. The architectural descriptions cover functional blocks, buses, and ports, including all features and processes that they support. For information about programming the ARM core in the ADSP-CM41x processor, visit the ARM Information Center at:

http://infocenter.arm.com/help/

For timing, electrical, and package specifications, see the ***ADSP-CM41x Processor Data Sheet***.

Intended Audience

The ADSP-CM41x processor uses the ARM Cortex-M4/M0 processors. The manual assumes the audience has a working knowledge of the ARM processor architecture and instruction set.

What's New in This Manual

This is the first revision (1.0) of the *ADSP-CM41x Mixed Signal Control Processor with ARM Cortex-M4/M0 and 16-Bit ADCs Hardware Reference*.

Technical or Customer Support

You can reach customer and technical support for processors from Analog Devices in the following ways:

- Post your questions in the processors and DSP support community at ***EngineerZone®***:

  http://ez.analog.com/community/dsp
- Submit your questions to technical support at ***Connect with ADI Specialists***:

  http://www.analog.com/support
- E-mail your questions about software/hardware development tools to:

  processor.tools.support@analog.com
- E-mail your questions about processors and DSPs to:

processor.support@analog.com (world wide support)

processor.china@analog.com (China support)

- Contact your Analog Devices sales office or authorized distributor. Locate one at:

  http://www.analog.com/adi-sales

- Send questions by mail to:

  Analog Devices, Inc.

  Three Technology Way

  P.O. Box 9106

  Norwood, MA 02062-9106 USA

Product Information

Product information can be obtained from the Analog Devices Web site.

Analog Devices Web Site

The Analog Devices Web site, http://www.analog.com, provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

To access a complete technical library for each processor family, go to: http://www.analog.com/processors/technical_library The manuals selection opens a list of current manuals related to the product as well as a link to the previous revisions of the manuals. When locating your manual title, note a possible errata check mark next to the title that leads to the current correction report against the manual.

Also note, MyAnalog.com is a free feature of the Analog Devices Web site that allows customization of a Web page to display only the latest information about products you are interested in. You can choose to receive weekly e-mail notifications containing updates to the Web pages that meet your interests, including documentation errata against all manuals. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Visit MyAnalog.com to sign up. If you are a registered user, just log on. Your user name is your e-mail address.

EngineerZone

EngineerZone is a technical support forum from Analog Devices. It allows you direct access to ADI technical support engineers. You can search FAQs and technical information to get quick answers to your embedded processing and DSP design questions.

Use EngineerZone to connect with other DSP developers who face similar design challenges. You can also use this open forum to share knowledge and collaborate with the ADI support team and your peers. Visit http://ez.analog.com to sign up.

Supported Processors

The following is the list of Analog Devices, Inc. processors.

Blackfin+® (ADSP-BF7xx) Processors

The name *Blackfin+* refers to the enhanced fixed-point Blackfin core architecture featured by the ADSP-BF70x processor product line, which is a family of 16-bit embedded processors.

Blackfin® (ADSP-BF6xx/BF5xx) Processors

The name *Blackfin* refers to the fixed-point core architecture featured on the following processors: ADSP-BF5xx and ADSP-BF6xx.

SHARC® (ADSP-21xxx) Processors

The name *SHARC* refers to the high-performance, 32-bit, floating-point core architecture featured on the following processors: ADSP-2116x, ADSP-2126x, ADSP-213xx, and ADSP-214xx. These processors can be used in speech, sound, graphics, and imaging applications.

SHARC+® (ADSP-SC5xx, ADSP-215xx) Processors

The name *SHARC+* refers to the enhanced high-performance, 32-bit, floating-point core architecture featured on the following processors: ADSP-215xx/ADSP-SC5xx. The connected SHARC+ ADSP-SC5xx processors also contain an ARM® Cortex®-A5 core. These products can be used in speech, sound, graphics, and imaging applications.

The following is the list of Analog Devices, Inc. processors supported by the IAR Embedded WorkBench® development tools. For information about the IAR Embedded WorkBench product and software download, go to http://www.iar.com/en/Products/IAR-Embedded-Workbench .

Mixed-Signal Control Processors

The ADSP-CM40x processors are based on the ARM Cortex-M4 core and are designed for motor control and industrial applications.

The ADSP-CM41x processors are based on the ARM Cortex-M4 and ARM Cortex-M0 cores and are designed for motor control and industrial applications.

Notation Conventions

Text conventions used in this manual are identified and described as follows. Additional conventions, which apply only to specific chapters, may appear throughout this document.

Example	Description
File > ***Close***	Titles in reference sections indicate the location of an item within the CrossCore Embedded Studio IDE's menu system (for example, the ***Close*** command appears on the ***File*** menu).
`{this | that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`. One or the other is required.
`[this | that]`	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional `this` or `that`.
`[this, …]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `Letter Gothic` font.
`filename`	Non-keyword placeholders appear in text with italic style format.
NOTE:	***NOTE:*** For correct operation, ... A note provides supplementary information on a related topic. In the online version of this book, the word ***NOTE:*** appears instead of this symbol.
CAUTION:	***CAUTION:*** Incorrect device operation may result if ... ***CAUTION:*** Device damage may result if ... A caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word ***CAUTION:*** appears instead of this symbol.
ATTENTION:	***ATTENTION:*** Injury to device users may result if ... A warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. In the online version of this book, the word ***ATTENTION:*** appears instead of this symbol.
Registers/Bits	All registers and bits in this manual are linked (clickable) to their respective descriptions in the "Register Descriptions" of each chapter.
Miscellaneous Conventions	Interrupt and internal signals are shown in all caps with no other formatting. For example the SPDIFn_RX or SCLK signal or the PKTE0_IRQ interrupt. An overbar denotes an active-low signal as in $\overline{\text{SYS\_FAULT}}$.

Register Documentation Conventions

Register diagrams use the following conventions:

- The descriptive name of the register appears at the top with the short form of the name.
- If a bit has a short name, the short name appears first in the bit description, followed by the long name.
- The reset value appears in binary in the individual bits and in hexadecimal to the left of the register.

- Bits marked *X* have an unknown reset value. Consequently, the reset value of registers that contain such bits is undefined or dependent on pin values at reset.
- Shaded bits are reserved

NOTE: To ensure upward compatibility with future implementations, write back the value that is read for reserved bits in a register, unless otherwise specified.

Register description tables use the following conventions:

- Each bit's or bit field's access type appears beneath the bit number in the table in the form (read-access/write-access). The access types include:
  - R = read, RC = read clear, RS = read set, R0 = read zero, R1 = read one, Rx = read undefined
  - W = write, NW = no write, W1C = write one to clear, W1S = write one to set, W0C = write zero to clear, W0S = write zero to set, WS = write to set, WC = write to clear, W1A = write one action
- Many bit and bit field descriptions include enumerations, identifying bit values and related functionality. Unless otherwise indicated (with a prefix), these enumerations are decimal values.

1 Introduction

The ADSP-CM41xF family of mixed-signal control processors is based on the ARM Cortex-M4 processor core with floating-point unit operating at frequencies up to 240 MHz, and the ARM Cortex-M0 processor core operating at frequencies up to 100 MHz. The processors integrate up to 160 KB of SRAM memory with ECC, up to 1 MB of flash memory with ECC, accelerators and peripherals optimized for motor control and photo-voltaic (PV) inverter control, and an analog module consisting of up to two 16-bit SAR-type ADCs, one 14-bit on M0 ADC and one 12-bit DAC. The ADSP-CM41xF family operates from a single voltage supply, generating its own internal voltage supplies using internal voltage regulators and an external pass transistor.

By integrating a rich set of industry-leading system peripherals and memory, the ADSP-CM41xF mixed-signal control processors are the platform of choice for next-generation applications that require RISC programmability and leading-edge signal processing in one integrated package. These applications span a wide array of markets in power conversion and include Solar PV inverters, motor/power control, and battery charging/control.

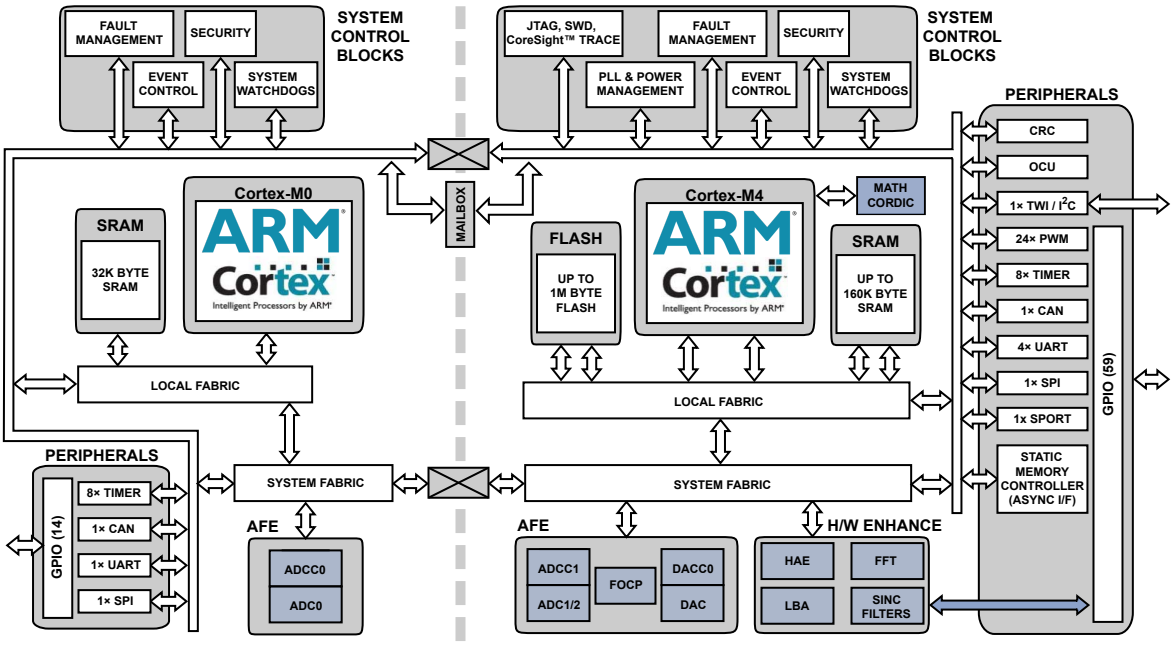


Figure 1-1: ADSP-CM41x Functional Block Diagram

ARM Cortex-M4 and ARM Cortex-M0 Processor Cores

The ARM Cortex-M4, core is a 32-bit reduced instruction set computer (RISC). It uses 32-bit buses for instruction and data. The length of the data can be 8/16/32 bits. Most of the registers only have 32-bit access. The length of the instruction word is 16 or 32 bits. The M4F core-memory subsystem consists of the ARM Cortex-M4 core, the main memory group, the MATH/CORDIC coprocessor, and the ARM Cortex-M4P subsystem control/ status registers. The ARM Cortex-M4F subsystem operates in its own CCLK0 clock domain at speeds up to 240 MHz.

The ARM Cortex-M0 is a 32-bit ultra low gate count RISC. It uses 32-bit buses for instruction and data. The length of the data can be 8/16/32 bits. Most of the registers only have 32-bit access. The length of the instruction word is 16 or 32 bits. The M0 subsystem consists of the ARM Cortex-M0 core, its local ARM Cortex-M0P platform SRAM, and its own communications peripherals (SPI, UART, and CAN), instrumentation (ADCC), and infrastructure (SEC, TRU, WDT). The ARM Cortex-M0 subsystem operates in its own SCLK0 clock domain at speeds up to 100 MHz.

The ADSP-CM41xF family consistes of several product variants that may contain one or both ARM cores (see the product data sheet). To discover which cores the processor contains, use the `EMUID_CHIPID` register as follows.

- The processor has the ARM Cortex-M4 and ARM Cortex-M0 dual core, the `EMUID_CHIPID` register reads 0x00000402.
- The processor has the ARM Cortex-M4 single core, the `EMUID_CHIPID` register reads 0x00000412.

Power and Clock Management

The processor contains the following modules that control power and clocking.

Dynamic Power Management (DPM)

The Dynamic Power Management (DPM) unit of the processor controls transitions between different power saving modes.

Reset Control Unit (RCU)

The Reset Control Unit (RCU) supports separate reset control for various chip sub-systems. For deterministic operation programmers should ensure there is no activity between separate chip sub-systems during reset activity. Several global reset options are also supported.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

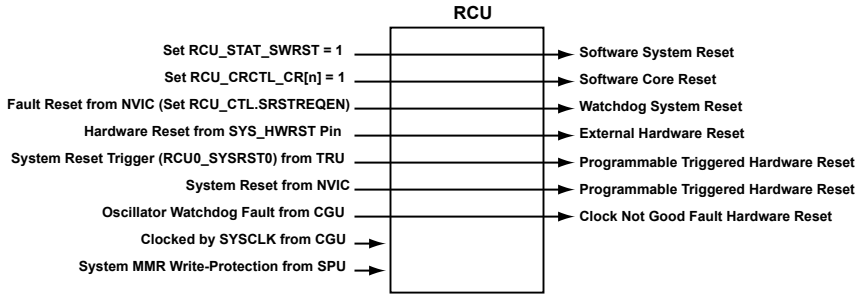


Figure 1-2: RCU System Diagram

Clock Generation Unit (CGU)

The Clock Generation Unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a single high-speed clock running at a programmable multiple of the SYS_CLKIN0 input clock frequency. Clocks for chip sub-systems are individually divided down to user program requirements. The PCU allows the application software to control the PLL module operation. (see Figure 6-2 ADSP-CM41x Clock Domains (Detail)).

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

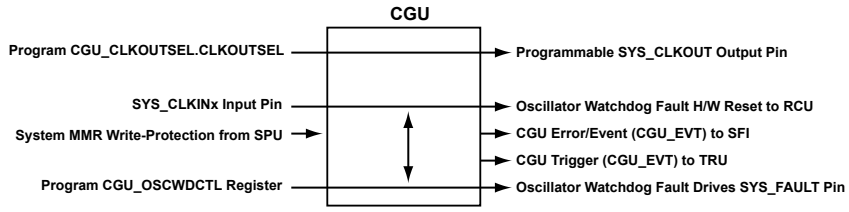


Figure 1-3: CGU System Diagram

Oscillator Comparator Unit (OCU)

The OCU monitors the frequency of an input clock using an additional low-frequency oscillator input. If the frequency drift of the input clock exceeds the specification, a fault is issued. The OCU can also detect gross frequency errors on either clock.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

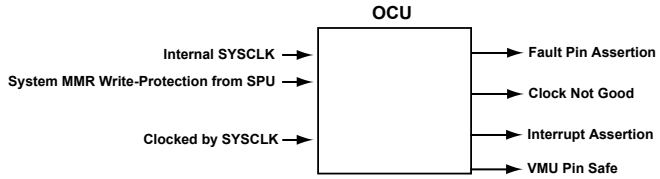


Figure 1-4: OCU System Diagram

System Interrupts and Triggers (SEC/TRU/TTU/CPTMR/GP Timer)

The following modules provide interrupt and trigger management functions for the processor.

System Event Controller (SEC)

The ARM Cortex-M4 and Cortex-M0 cores have intrisic handling of interrupt events. The two SEC units manage the interface between ISR events and system fault creation for the ARM Cortex-M4 and Cortex-M0 systems, respectively.

SEC is basically an encapsulation of the ARM NVIC module and System Fault Interface (SFI).

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

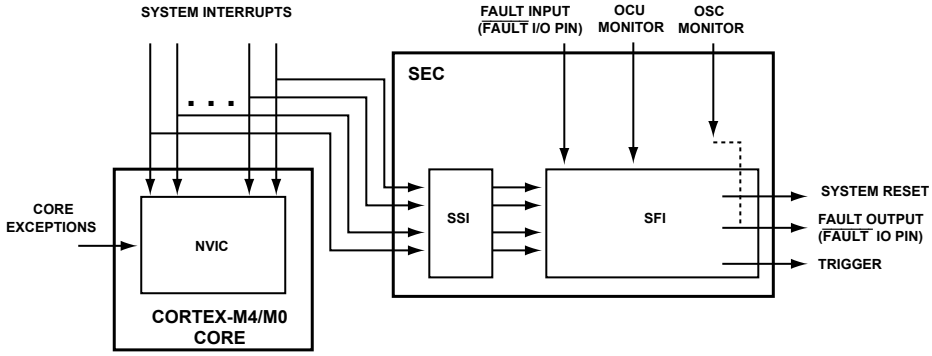


Figure 1-5: SEC Block Diagram

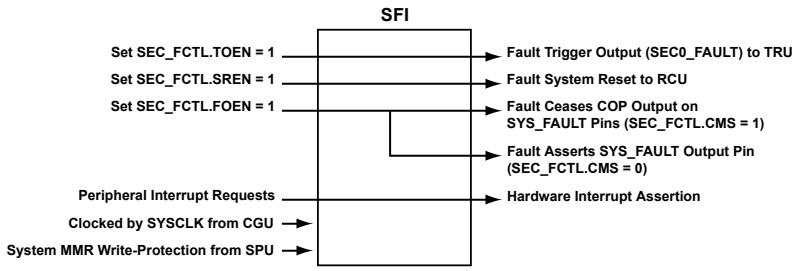


Figure 1-6: SFI Block Diagram

Trigger Routing Unit (TRU)

The Trigger Routing Unit (TRU) provides system-level sequence control without core intervention. The TRU maps trigger masters (generators of triggers) to trigger slaves (receivers of triggers). Slave endpoints can be configured to respond to triggers in various ways. Multiple TRUs may be provided in a multiprocessor system to create a trigger network.

Common applications enabled by the TRU include the following:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes
- Software triggering

- Synchronization of concurrent activities

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

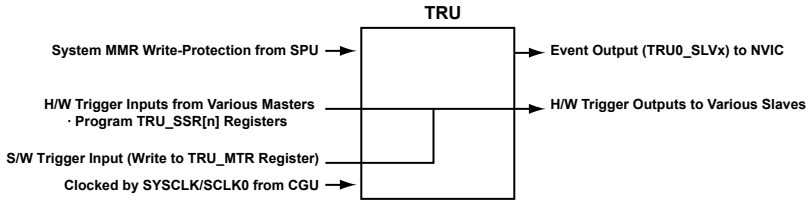


Figure 1-7: TRU System Diagram

Trigger Timing Unit (TTU)

The Trigger Timing Unit (TTU) provides a simple way to generate event trigger signals with precise timing relationships. Triggers can then be routed to or from the TTU using the Trigger Routing Unit (TRU). The TTU is designed in a scalable, modular fashion. The available output trigger generators can be organized into independent trigger delay groups, each sensitive to its own input trigger.

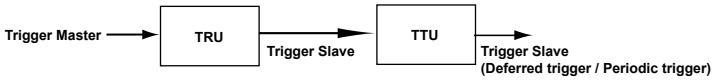


Figure 1-8: TTU Block Diagram

Capture Timer (CPTMR)

The Capture Timer (CPTMR) module measures the total on-time of the input signal between two triggers. The module performs this measurement by counting clock pulses during the on-time of the input signal. The total on-time is captured when the stop trigger is received.

GP Timer

The ADSP-CM41xF processors provide two sets of eight general- purpose (GP) timers, one set primarily associated with each processor core. Each timer has an external pin that can be configured either as a PWM or timer output, as an input to clock the timer, or as a mechanism for measuring pulse widths and periods of external events. These timers can be synchronized to an external clock input on the `TM0_ACLKx` pins, an external `TM0_CLK` input pin, or to the internal SCLK0.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

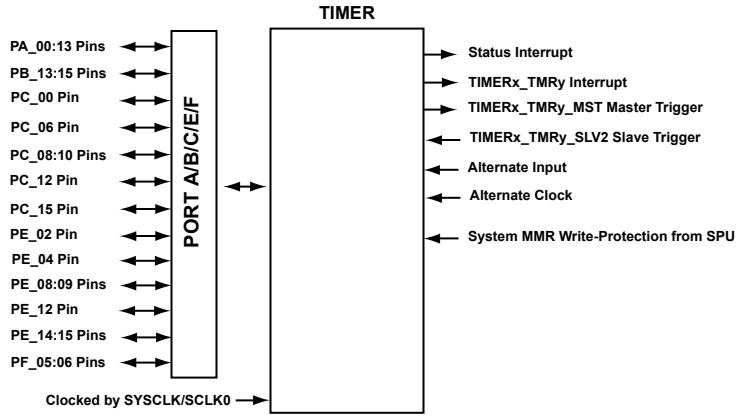


Figure 1-9: Timer Block Diagram

System Memory (SMC/SMPU/FLC)

The following section describes the memory architecture of the processor.

Static Memory Controller (SMC)

The Static Memory Controller (SMC) is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. It provides a glueless interface to various external memories and peripheral devices, including SRAM, ROM, EPROM, NOR flash memory, and FPGA/ASIC devices.

The SMC acts as an SCB slave. The processor SCB interconnect fabric arbitrates accesses to the SMC. The SMC connects to signal pins for memory control (such as read, write, output enable, and memory select lines).

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

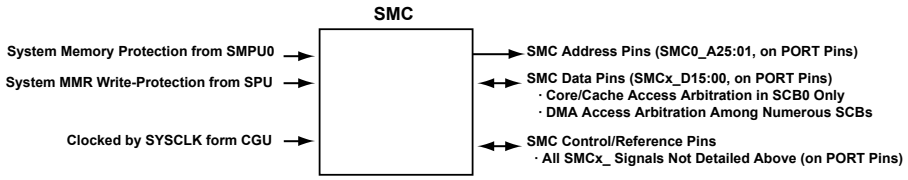


Figure 1-10: SMC System Diagram

System Memory Protection Unit (SMPU)

The System Memory Protection Unit (SMPU) provides a flexible way of protecting memory regions against read or write access from any or all masters in the system. In addition, it can guard against memory access depending on security privileges of the system master.

On the ADSP-CM41x, an SPMU is provided for each ARM Cortex core's main memory and for external memory.

Flash Controller (FLC)

The processor features ECC-protected embedded parallel flash memory. It consists of up to 1024 KB of main memory and up to 8 KB of information memory.

Direct Memory Access (DMA/MDMA/CRC)

DMA Controller (DMA)

The processors use Direct Memory Access (DMA) to transfer data within memory spaces or between a memory space and a peripheral. The processors can specify data transfer operations and return to normal processing while the fully integrated DMA controller carries out the data transfers independent of processor activity.

DMA transfers can occur between memory and a peripheral or between one memory and another memory. Each Memory-to-memory DMA stream uses two channels, where one channel is the source channel, and the second is the destination channel. Most peripherals have at least one dedicated DMA channel associated with them.

All DMAs can transport data to and from all on-chip and off-chip memories. Programs can use two types of DMA transfers, descriptor-based or register-based. Register-based DMA allows the processors to directly program DMA control registers to initiate a DMA transfer. On completion, the control registers may be automatically updated with their original setup values for continuous transfer. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. Descriptor-based DMA transfers allow multiple DMA sequences to be chained together and a DMA channel can be programmed to automatically set up and start another DMA transfer after the current sequence completes.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

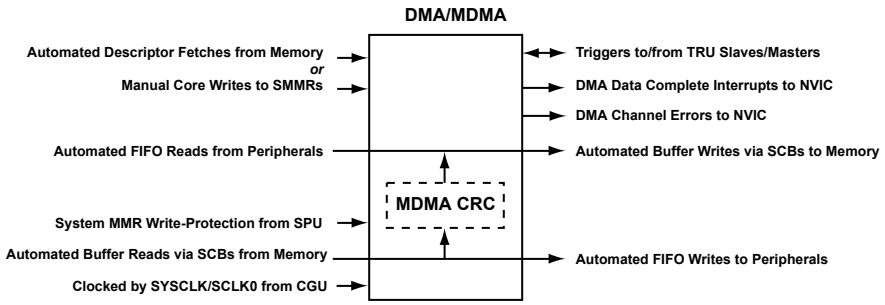


Figure 1-11: DMA System Diagram

Memory DMA Controllers (MDMA)

The processor supports memory-to-memory DMA operations which include two standard memory DMA channels with CRC protection (32-bit bus width, run on SCLK0).

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

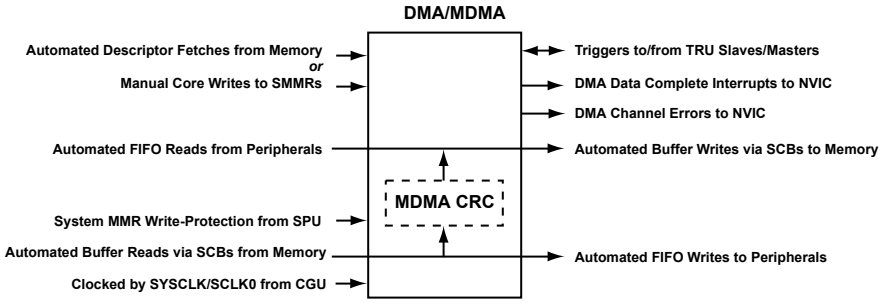


Figure 1-12: MDMA System Diagram

Cyclic Redundancy Check (CRC)

The Cyclic Redundancy Check (CRC) peripheral performs the cyclic redundancy check (CRC) of the block of data that is presented to the peripheral. The peripheral provides a means to verify periodically the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects. It is based on a CRC32 engine that computes the signature of 32-bit data presented to the peripheral.

The dedicated hardware compares the calculated signature of the operation to a pre-loaded expected signature. If the two signatures fail to match, the peripheral generates an error. The source channel of the memory-to-memory DMA channels can provide data. The CRC optionally forwards data to memory through the destination DMA channel. Alternatively, the peripheral supports data presented by core write transactions.

The CRC peripheral implements a reduced table-look-up algorithm to compute the signature of the data. The CRC uses a programmable 32-bit CRC polynomial to generate the lookup table (LUT) contents automatically.

More CRC peripheral modes allow for initializing large memory sections with a constant value, or for verifying that sections of memory are equal to a constant value.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

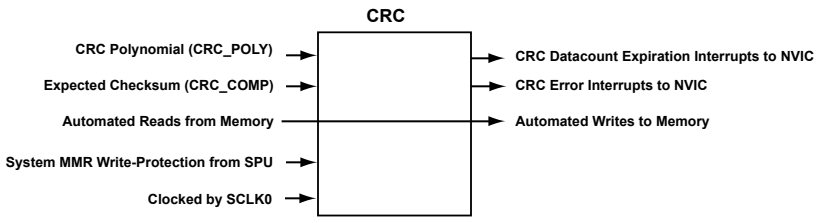


Figure 1-13: CRC System Diagram

Peripherals

By integrating a rich set of industry-leading system peripherals and memory, the ADSP-CM41x mixed-signal control processors are the platform of choice for next-generation applications that require RISC programmability and leading-edge signal processing in one integrated package. These applications span a wide array of markets in power conversion and include Solar PV inverters, motor/power control, and battery charging/control.

- General-Purpose I/O (GPIO) peripherals
- Dedicated pin peripherals

General-Purpose I/O (GPIO) Peripherals

The processor has up to 102 general-purpose I/O pins mapped across up to seven ports (PORT A through PORT G). Each pin can be configured individually to serve as a GPIO pin or as a peripheral-specific pin.

GPIO Ports (PORT)

When configured in the default GPIO mode, the PORT pins allow for the processor to interface to system components to provide handshaking functionality as either inputs or outputs. When in output mode, open-drain output is supported. A single MMR access can be used to sense or set individual pins or a complete port of 16 pins.

Additionally, each GPIO pin can optionally be configured to raise a system interrupt on the processor via a dedicated pin interrupt (PINT) block, and all peripheral functions are controlled via a set of port multiplexing registers, with specific settings defined in the data sheet.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

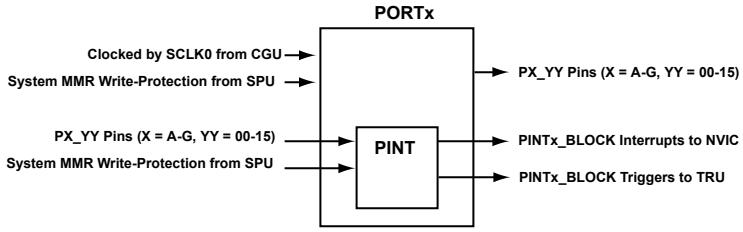


Figure 1-14: PORT System Diagram

Serial Peripheral Interface Ports (SPI)

The Serial Peripheral Interface (SPI) is an industry-standard synchronous serial link that supports communication with multiple SPI-compatible devices. The baseline SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. The two data pins allow full-duplex operation to other SPI-compatible devices. Two extra (optional) data pins are provided on specific SPIs to support quad SPI operation. Enhanced modes of operation such as flow control, fast mode, and dual-I/O mode (DIOM) are also supported. In addition, a direct memory access (DMA) mode allows for transferring several words with minimal CPU interaction.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

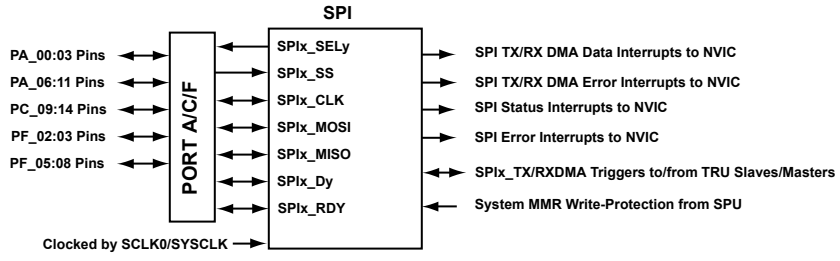


Figure 1-15: SPI System Diagram

Serial Port (SPORT)

The programmable serial ports (SPORTs) support various protocols for serial data communication and provide a glueless hardware interface to many industry-standard data converters and codecs. They have high data rates and dual half-duplex datapaths and are ideal for establishing a direct serial connection among two or more processors in a multiprocessor system, as many processors provide compatible serial interfaces.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

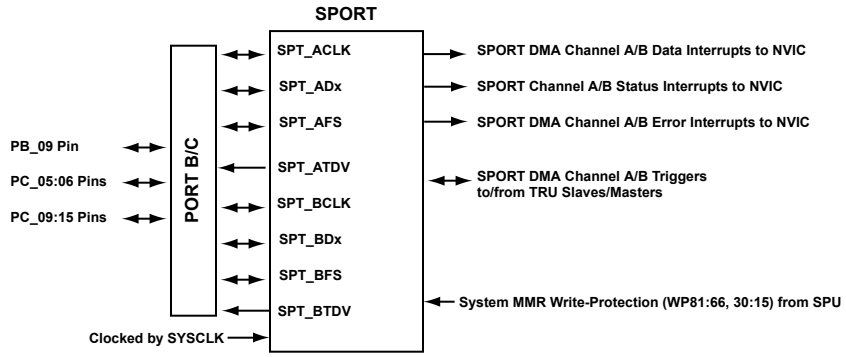


Figure 1-16: SPORT System Diagram

Universal Asynchronous Receiver/Transmitter (UART)

The Universal Asynchronous Receiver/Transmitter (UART) module is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word lengths, stop bits, bit rates, and parity-generation options. The UART includes interrupt-handling hardware. Multiple events can generate interrupts.

The UART is logically compliant to EIA-232E, EIA-422, EIA-485 and LIN standards, but usually requires external transceiver devices to meet electrical requirements. In IrDA (Infrared Data Association) mode, the UART meets the half-duplex IrDA SIR (9.6/115.2 Kbps rate) protocol. In multi-drop bus mode, the UART meets the full-duplex MDB/ICP v2.0 protocol.

UART3 is used for booting and flash firmware upgrade.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

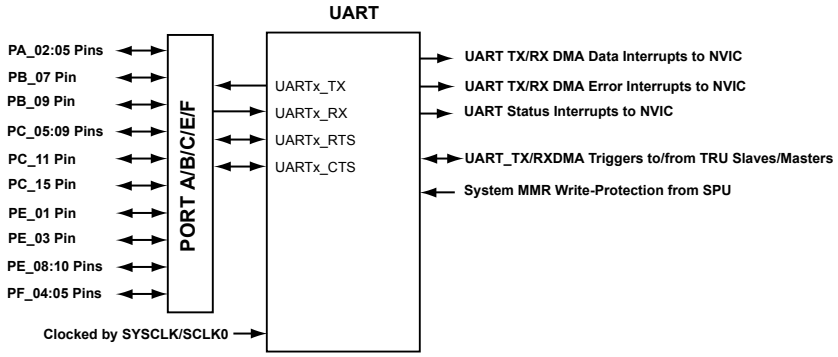


Figure 1-17: UART System Diagram

Pulse-Width Modulator (PWM)

The Pulse-Width Modulator (PWM) module is a flexible and programmable waveform generator. It supports full HPPWM. It has three HPPWM units.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

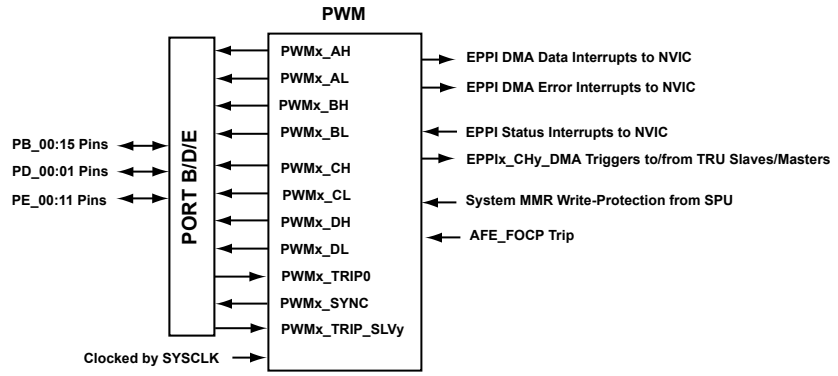


Figure 1-18: PWM System Diagram

General-Purpose Counter (CNT)

The General-Purpose Counter (CNT) converts pulses from incremental position encoders into data that is representative of the actual position of the pulse. This conversion is done by integrating (counting) pulses on one or two inputs. Since integration provides relative position, some devices also feature a zero-position input (zero marker). The GP counter can use the zero position input feature to establish a reference point for verifying that the acquired position does not drift over time. In addition, the GP counter can use the incremental position information to determine speed, if the time intervals are measured.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

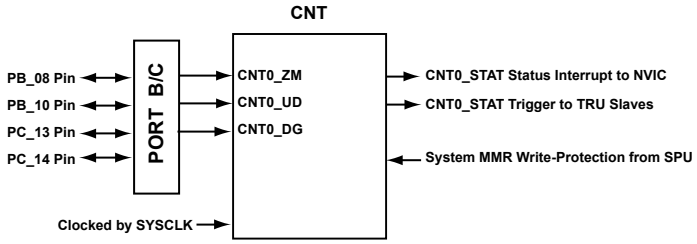


Figure 1-19: CNT System Diagram

Controller Area Network (CAN)

The processor contains two Controller Area Network (CAN) module based on the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. The CAN protocol is compatible with the control applications. It can communicate reliably over a network and incorporates CRC checking, message error tracking, and fault node confinement.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

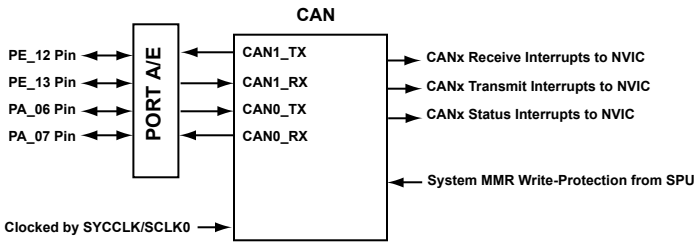


Figure 1-20: CAN System Diagram

ADC Controller (ADCC)

The processor includes an ADC Controller (ADCC) module that provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). The processor initiates analog-to-digital conversions, based on either external or internal events

Sinus Cardinalis (SINC) Filter

The sinus cardinalis (SINC) filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. See System Accelerators (FFTB/HAE/SINC/MATH).

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

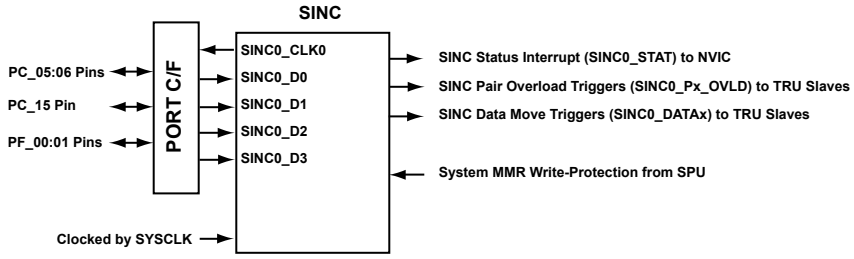


Figure 1-21: SINC System Diagram

Dedicated Pin Peripherals

The following peripherals have dedicated pins on the processor.

Two-Wire Interface (TWI)

The processor has a Two-Wire Interface (TWI), that provides a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I^2C bus standard. Additionally, the TWI module is fully compatible with serial camera control bus (SCCB) functionality for easier control of various CMOS camera sensor devices.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

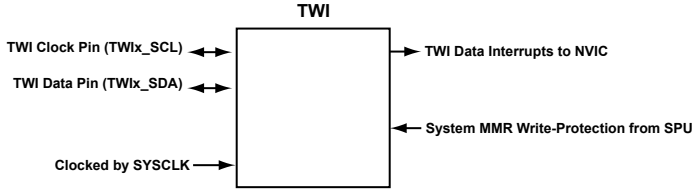


Figure 1-22: TWI System Diagram

Boot Mode Pin

`SYS_BMODE0` is used for selecting the boot mode of processor.

System Accelerators (FFTB/HAE/SINC/MATH)

The following sections provide information about the high-performance acceleration engines on the processor.

FFT Accelerator Block (FFTB)

The FFT Accelerator Block (FFTB) provides input signal spectrum analysis. The input can be provided by the core or the FFTB can perform memory to memory FFT/IFFT operations (up to 512 points) without core software intervention. Additionally, the FFTB architecture allows in-built data conversion for various sensor input formats, windowing, and comb filtering the input signal before FFT. The FFTB performs operations on complex FFT output spectrum such as spectrum averaging, square magnitude computation, and band power limit detection.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

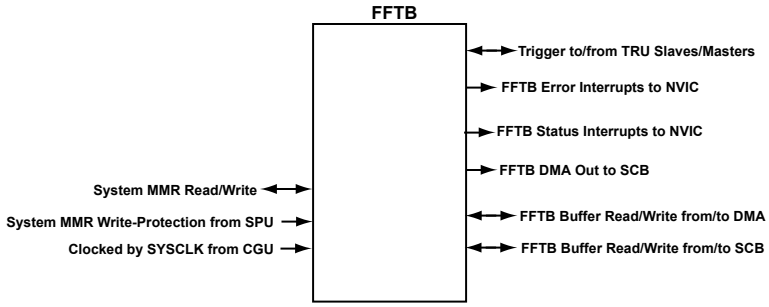


Figure 1-23: FFTB System Diagram

Harmonic Analysis Engine (HAE)

The Harmonic Analysis Engine (HAE) analyzes harmonic frequencies present on the voltage and current input samples. The HAE receives input samples from two source channels whose frequencies are 45–65 Hz. The HAE then processes the input samples and produces output results. The output results consist of power quality measurements of the fundamental and up to 12 more harmonics.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

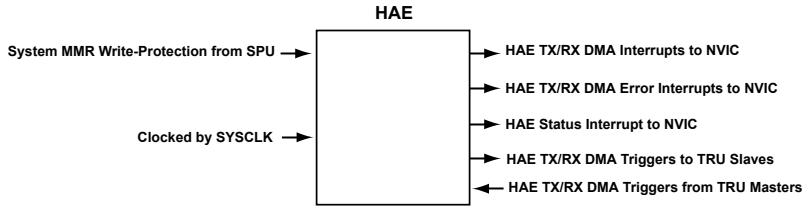


Figure 1-24: HAE System Diagram

Sinus Cardinalis (SINC) Filter

The Sinus Cardinalis (SINC) Filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. A SINC filter converts the bit stream from a sigma-delta front-end modulator into a digital word representing the signal level presented to the modulator.

The filter consists of a set of integration and decimation stages implemented directly in logic for efficient execution. The SINC filter supports capture of current or voltage feedback signals from an isolating analog-to-digital converter (ADC). Each modulator bit stream connects to two SINC filters: a primary filter for controlling feedback; a secondary filter for overcurrent detection. The SINC module includes four filter channels and two modulator clock generators.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

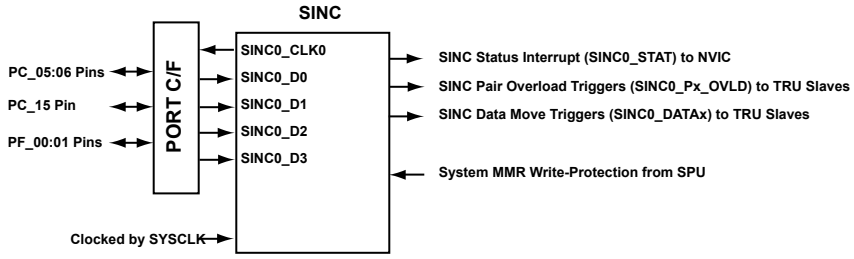


Figure 1-25: Sinus Cardinalis (SINC) Filter

MATH Accelerator Unit

The MATH Accelerator Unit accelerates highly accurate single-precision floating-point computations of common transcendentals such as trigonometric and exponential functions and their inverses. It provides faster reciprocals and square roots than provided by the Cortex-M4. It also provides accelerated functions to convert between rectangular and polar coordinates. Most operations by this tightly-coupled accelerator complete in a deterministic number of core clock cycles.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

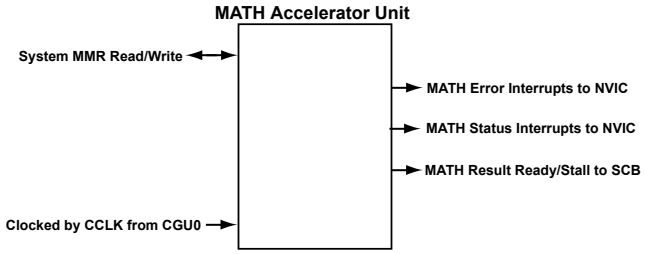


Figure 1-26: MATH Accelerator Unit

Security and Protection (SPU/MBOX)

The following modules provide system safety and security.

System Protection Unit (SPU)

In a system with multiple system MMR masters, configurations of peripherals can be changed unintentionally leading to bad data or even system malfunctions. The peripherals are shared resources in the system. The System Protection Unit (SPU) restricts access to certain MMRs, similar to the functionality of a semaphore.

The SPU also protects peripherals based on security settings. It is part of the overall security infrastructure of the processor.

MailBox (MBOX)

Communication between the cores is provided by a 4 KB MBOX (mailbox) organized as 1024 32-bit words. The MBOX is a simulated dual port RAM. The MBOX is located in the ARM Cortex-M0 clock domain. The SRAM is

protected from soft and hard errors using ECC with minimum 1-bit correction and 2-bit detection. The MBOX may be used as a sole communication device between cores, or as a communication device for synchronizing data transfers and protection between the die.

Safety (WDOG/VMU)

Signal Watchdog

The eight general-purpose Watchdog Timer (WDOG) timers feature modes to monitor off-chip signals. The Watchdog Period mode monitors whether external signals toggle with a period within an expected range. The Watchdog Width mode monitors whether the pulse widths of external signals are within an expected range. Both modes help to detect undesired toggling (or lack thereof) of system-level signals.

Voltage Monitor Unit (VMU)

The Voltage Monitoring Unit (VMU) provides over-voltage and under-voltage detection by monitoring the internal and external power pins and generates asynchronous signals if the voltages exceed or drop below the programmable limits. The VMU also generates a control signal for the power sequencing requirements of the embedded FLASH, even if the VMU is disabled.

For more information about System MMR Write-Protection from SPU, refer to the ADSP-CM41x Write-Protect and Secure Peripheral Registers.

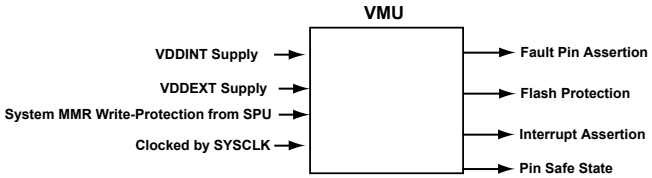


Figure 1-27: VMU System Diagram

JTAG Debug and Serial Wire Debug Port

The ADSP-CM41x processor support standard debugging options, JTAG and SWD for the ARM Cortex cores. The processor supports both Embedded Trace Macrocell (ETM) and Instrumentation Trace Macrocell (ITM).

Additional debug resources are provided by the following modules.

- The System Crossbars (SCB) are the fundamental building blocks of a switch-fabric style for (on-chip) system bus interconnection. The SCBs connect system bus masters to system bus slaves, providing concurrent data transfer between multiple bus masters and multiple bus slaves. The SCB provides sustainable throughput for simultaneous transactions in the system with configurable Quality of Service for each type of transaction (traffic) as required. A hierarchical model, built from multiple SCBs, provides a power and area efficient system interconnect, which satisfies the performance and flexibility requirements of a specific system.
- The System Watchpoint Unit (SWU) is a single module used for transaction monitoring. The SWU is attached to each system slave through the system crossbar interface and provides ports for all address channel signals for

the system crossbar. The SWU does not have ports for the read/write data channel signals or the low-power interface signals.

ADCC and DACC

The processor contains an Analog-to-Digital Converter Controller (ADCC) and a Digital-to-Analog Converter Controller (DACC).

Analog Front End (AFE)

The processors contain one ADC attached to the ARM Cortex-M0 core and two ADCs plus one DAC attached to the ARM Cortex-M4 core. Control of these data converters is simplified by two powerful on-chip Analog-to-Digital Converter Controller (ADCC) and a Digital-to-Analog Converter Controller (DACC). The ADCC and DACC are integrated seamlessly into the software programming model, and they efficiently manage the configuration and real-time operation of the ADCs and DACs.

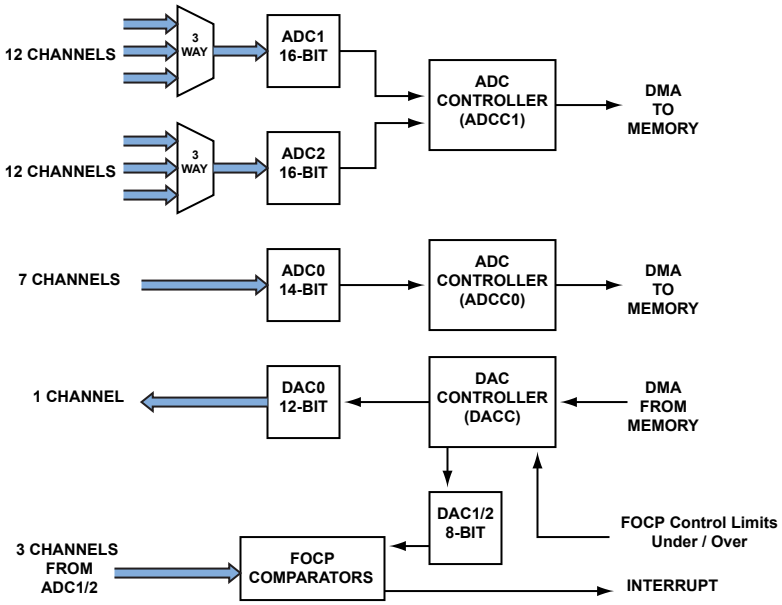


Figure 1-28: AFE Diagram

For AFE related pins and interconnections, refer to the product datasheet.

Fast Over Current Protection (FOCP)

The processor supports protection from over-current on its analog channels, via dedicated comparators in the analog front end subsystem (AFE).

System Block (SYSBLK) and Pads

System Block (SYSBLK) and PADS are collection of additional registers that are part of several peripherals and core systems. For example, SYSBLK covers few registers for SCB/DMA_MUX/LROM_STAT and PADS covers few registers for Debounce/FOCP/PORTS/VMU.

2 ARM Cortex-M0 Core 1 Memory Sub-System

The ADSP-CM41x family of processors that have an optional ARM Cortex-M0 processor core with integrated SRAM memory and peripherals.

The ARM Cortex-M0 Subsystem is a self-contained processor subsystem provided for safety supervisory purposes. It has tightly coupled 32 KB SRAM, peripherals, and GPIOs, and on the ADSP-CM41x, is configured to control its own 7-input auxiliary ADC core on the processor die. The interfaces and buses in the system are arranged so that the application can control the degree of interaction and observation that the ARM Cortex-M4 and ARM Cortex-M0 processors have with each other, from total access to almost complete independence.

The following terms that are used in this chapter:

- *Cortex core* refers to the ARM Cortex-M0 core and core peripherals.
- *Cortex memory* refers to the portions of the Cortex memory map that are part of the memory model for the Cortex core (for example, SRAM), but are not part of the system memory (memory-mapped registers) or external memory.
- *ADSP-CM41x processor* or *processor* refers to the combination of the Cortex core, Cortex memory, system peripherals (for example, UART, SPI, and SPORT), and system memories.

This document describes the ARM Cortex-M0 core and memory architecture used on the ADSP-CM41x processor, but does not provide detailed programming information for the ARM processor. For more information about programming the ARM processor, visit the ARM Information Center:

- http://infocenter.arm.com/help/

The applicable documentation for programming the ARM Cortex-M0 processor include:

- ARM Cortex-M0 Devices Generic User Guide
- ARM Cortex-M0 Technical Reference Manual

Peripherals and Infrastructure in M0 Subsystem Domain

The following tables identify the peripherals and infrastructure in the ARM Cortex-M0 Subsystem domain.

Table 2-1: Peripheral Block

Peripheral block	Description
SPI0	Serial Peripheral Interface
UART0	Universal Asynchronous Receiver Transmitter
ADCC0	Analog to Digital Converter Controller
CAN0	Controller Area Network
TIMER0 x 8	General Purpose Timer

Table 2-2: Infrastructure Block

Infrastructure block	Description
DDE0–3	DMA Engine
TRU0	Trigger Routing Unit
SEC0	System Event Controller
WDT0	Watchdog Timer
PORTA	General Purpose Port
PINT0	Peripheral Interrupt Module
PADS0	Pads Interface
SYSBLK0	System Block Interface
SPU0	System Protection Unit
SMPU0	System Memory Protection Unit
SWU0–1	System Watch-point Unit

Cortex-M0 Memory Features

The ARM Cortex-M0 core memory architecture includes the following features:

- An local tightly coupled memory sub-system that supports 32 KB of ECC-protected, zero wait state main SRAM memory for data and code storage.
- ARM Cortex-M0 bus architecture with a single AHB interface that provides simple integration to all system peripherals and memory.
- External memory interface that contains up to 4 banks, each bank with 128 KB of space.

Common System Memories:

- 1024 KB of ECC-protected flash memory (512 KB x 2 banks) common for both the ARM Cortex-M4P and the ARM Cortex-M0S. Flash access is via the Cortex-M4P domain and clock, and accessing flash from the ARM Cortex-M0S is slower.

- 4 KB MBOX mailbox memory (with ECC) for inter-processor communication.
  - MBOX supports exclusive reads and writes from either core, to implement process synchronization semaphores.
  - Shared with the ARM Cortex-M0S

Cortex-M0 Memory Functional Description

The following sections provide the functional description for the Cortex-M0 core memory sub-system.

CM41X_M0 M0P Interrupt List

Table 2-3: CM41X_M0 M0P Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
2	M0P_SRAM_EERR	M0P supervisor sram ecc error	Level	

CM41X_M0 M0P Trigger List

Table 2-4: CM41X_M0 M0P Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 2-5: CM41X_M0 M0P Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
18	M0P_NMI_SLV0	M0P NMI Trigger Slave 0	Pulse

ARM Cortex-M0 Bus Interconnect

The following sections breifly describe the Cortex-M0 bus blocks.

ARM Cortex-M0 Supervisor System Peripherals

These are connected to a bus which is mastered by the SPU0 unit. This bus is clocked at the ARM Cortex-M0 clock rate by SCLK0 and is reset by $\overline{\text{SYSRST}}$. The SPU0 unit accepts qualified transaction requests from three sources:

- The ARM Cortex-M4 controller,
- The ARM Cortex-M0 supervisor, and
- Certain DMA-capable system peripherals via the main system fabric.

Supervisor (ARM Cortex-M0) Tightly-Coupled Memory

The 32 KB is connected to the ARM Cortex-M0 Supervisor bus through a zero-latency matrix. Access by other masters outside the M0 Platform is facilitated by the Supervisor Peripheral Fabric, implemented in the bus protocol. This memory and interconnect are within the SCLK0 clock domain and are reset by SYSRST.

- Access by DMA to this memory is controlled by the System Memory Protection Unit SMPU0, which can be programmed to grant or deny write or all access by any memory master in the ADSP-CM41x system outside the ARM Cortex-M0P to up to 4 regions within the ARM Cortex-M0 Tightly-Coupled Memory address space.
- DMA access to these memories is monitored by the System Watchpoint Unit SWU1, which can be programmed to watch for accesses to selected addresses and to signal interrupts or trigger events.

System Memories

These are slave memory spaces which are connected to the Main System SCB Fabric. This is a high-concurrency, high-throughput fabric, residing within the SYSCLK domain and reset by SYSRST. Concurrent access is possible by different masters to any of the six slave memory spaces at the same time.

- The devices which can act as masters on the Main System bus fabric include
  - The ARM Cortex-M4P Controller
  - The ARM Cortex-M0 Supervisor and its DMA-capable peripherals
  - DMA-capable system peripherals, in three groups called System Crossbars (SCBs).
- The slave memory spaces in the System Memory Fabric include
  - The SMC System Memory Controller, which supports connection to off-chip memory through the parallel memory port
  - The FFTB's embedded memory
- Additional slaves include bus bridges to the following:
  - The ARM Cortex-M4 DMA Port to the ARM Cortex-M4 Tightly-Coupled Memories
  - The ARM Cortex-M0 DMA Port to the ARM Cortex-M4 Tightly-Coupled Memories
  - The Controller System Peripherals
  - The Supervisor System Peripherals

Posted System Writes (NormSys Write versus PostSys Write)

The ARM Cortex-M0 supports only PostSysWrite mode. Non-exclusive ARM Cortex- M0 core writes to system space are posted, meaning that the Cortex memory interface accepts the MMR write transaction from the Cortex core and immediately returns an OKAY response. The Cortex core proceeds with subsequent instructions, while independently, the Cortex memory interface forwards the posted write to the system fabric (bus interconnect). Only

one write may be posted at any time, so if a subsequent write arrives while the first is pending, the Cortex core is stalled until the Cortex memory interface can accept the new transaction.

If a system fabric (bus interconnect) ERROR response is returned from a posted write, the Cortex core cannot signal an exact HardFault exception because it has already gone on to execute further instructions. In this case, the Cortex memory captures the address of the erroneous transaction in the field.

An erroneous transaction can be configured to generate an interrupt service request to both cores. Set the `SYSBLK_SCB_RESP_CFG.PWENA` bit to enable the interrupt. The `SYSBLK_SCB_RESP_CFG.PWERR` bit holds a sticky indication of erroneaous error irregardless of interrupt enable. The `SYSBLK_SCB_RESP_CFG.PWERR` status bit is write-1-clear.

Bus Timeout

The ADSP-CM41x bus interconnect fabric provides a Fault-tolerant Access Timeout protection, provided by SPUs or TESYS Timeout units. Accesses can be programmed to Timeout in SYSCLK units.

Programming the Timeout includes:

- Programming the Timeout Value in SYSCLKs
- Enabling the Timeout to the Interrupt

Table 2-6: Bus Timeout

To (Slave)	From M4	From M0	From MDMA	From M4 Peripheral DMAs	From M0 Peripheral DMAs
M4 ROMs	none	n/a	n/a	n/a	n/a
M4 SRAM	none	SYSBLK0	none	none	none
M4 Flash	none	SYSBLK0	none	none	none
M4 local peripherals (MATH, Flash Controller, M4P)	none	n/a	n/a	n/a	n/a
SPU1 controls	SPU1	SYSBLK0 & SPU1	SPU1	n/a	n/a
M4 Peripheral / System MMRs	SPU1	SPU1	SPU1	n/a	n/a
FFTB memory	SYSBLK1	SYSBLK0	none	none	none
SMC external memory	SYSBLK1	SYSBLK0	none	none	none
MBOX Port1 and MMRs	SPU1	SYSBLK0	SPU1	n/a	n/a
M0 SRAM	SYSBLK1	none	none	none	none
SPU0 controls	SPU0	SPU0	SPU0	n/a	n/a
M4 Peripheral and System MMRs	SYSBLK1	SYSBLK0 & SPU0	SPU0	n/a	n/a
MBOX Port 0 and MMRs	n/a	SYSBLK0	n/a	n/a	n/a

As shown in the table, the time-out feature is implemented via three blocks: SYSBLK1, SPU0 and SPU1. The programming structure of the SYSBLK1 and SPU0/SPU1 for timeout is similar, but uses different sets of registers. The following is a code example for SYSBLK timeout programming.

1. Program the timeout value using the SYSBLK_SCB_TIMEOUT_VALUE register. Set the SYSBLK_SCB_RESP_CFG.TOENA bit (=1).
2. Wait for the timeout interrupt and once the interrupt is asserted and the program is in the handler the following code can be executed.

```
void C1_BUS_TIMEOUT_Handler()
 {
 unsigned int stat = *pREG_SYSBLK0_SCB_RESP_CFG;
 if(stat & BITM_SYSBLK_SCB_RESP_CFG_TOERR)
 {
 *pREG_C1_AHB_RESP_CFG |= BITM_SYSBLK_SCB_RESP_CFG_TOERR;
 __SEV();   __ISB();
 }
/*process interrupt*/
 }
```

The following programming example is used to configure and service the SPU timeout event. The example can also be used for SPU1 if the corresponding SPU1 registers are used in place of the indicated SPU0 registers.

1. Install the following handler code for the SPU0 Timeout event:

```
void SPU0_TIMEOUT_Handler ()
{
unsigned int stat = *pREG_SPU0_STAT;
if(stat & BITM_SPU_STAT_TIRQ)
{
time_outerr++;
*pREG_SPU0_STAT |= BITM_SPU_STAT_TIRQ;
__SEV();   __ISB();
}
/*process interrupt*/
}
```

2. Program the timeout value in the SPU_TIMEOUT register.
3. Set the SPU_CTL.TOMON bit (=1) to enable the monitor.

To detect a pending timeout transaction the program should check the SPU_STAT.TOERR bit. This bit is set once the timeout condition is hit, whether or not the interrupt request is enabled.

Cortex-M0 Code and Data SRAM

The unified internal SRAM space provides both code and data memory for the ARM Cortex-M0 processor core, allowing a configurable partition between code and data space. In addition, the SRAM features a DMA access port

for read/write access by other master devices on the system fabric. The SRAM and all its interfaces operate in the ARM Cortex-M0 core clock domain (SCLK0). The SRAM supports exclusive accesses. The SRAM can be accessed at the maximum SCLK0 speed in zero-wait-state. For access timing information, see the product data sheet.

SRAM Features

The SRAM has the following features:

- Up to 32 KB SRAM capacity
- Zero wait-state performance
- ECC protection (SECDED)

SRAM Bank Organization on the ARM Cortex-M0 Subsystem

The ARM Cortex-M0S platform has a contiguous 32K Bytes SRAM created from four 8K Bytes single-ported SRAM macros, where both code as well as data can reside.

ECC Error Handling

The following sections provide information on the Error Correcting Code (ECC) feature of the M0 core.

ECC Error Signaling

The `SYSBLK_SRAM0_ECC.FADDR` M0's bus interface register is associated with reporting errors on the various SRAM read ports.

ECC Error Interrupt Handlers

The M0 SRAM array is ECC protected against single bit failures and will detect multi-bit failures. Multi-bit failures signal a BUSERR to the requesting port (Core SCB or DMA). In addition, regardless of port, the Fault Status bit (`SYSBLK_SRAM0_ECC.FLTST`) is set and the array address of the first failure is captured in the ECC Fault Address bits (`SYSBLK_SRAM0_ECC.FADDR`). The `SYSBLK_SRAM0_ECC.FADDR` bit does not change after `SYSBLK_SRAM0_ECC.FLTST` is asserted, even if there are new ECC errors. Writing a one to the `SYSBLK_SRAM0_ECC.FLTST` bit clears it and allows a new ECC error address to be captured in the `SYSBLK_SRAM0_ECC.FADDR` bit.

Setting the `SYSBLK_SRAM0_ECC.IEN` bit, asserts the M0_SRAM_ECC_IRQ interrupt when the `SYSBLK_SRAM0_ECC.FLTST` bit is set. This interrupt is routed internally to the M0P ARM CORTEX-M0 processor and externally to the ARM Cortex-M4 processor. Clearing the `SYSBLK_SRAM0_ECC.FLTST` bit or disabling the `SYSBLK_SRAM0_ECC.IEN` bit disables this interrupt.

The ARM Cortex-M0S SRAM does not accept new transactions from either port for two cycles after an ECC error is detected.

Memory and MMR Access Latencies

This section provides information on the ARM Cortex-M0 core read and write memory-mapped register access latencies.

The number of M0 core clock wait states incurred for reads and writes to various system address ranges is specified by the ***M0 Wait States By Target Address Space*** table. Note that the baseline time in the ARM Cortex-M0 architecture for a load-store instruction to zero-wait-state memory is two core clock cycles. In the following examples the results of division are rounded up to the next higher integer.

Ratio_CS = (CCLK0 frequency ÷ SYSCLK frequency)

Ratio_M0 = (CCLK0 frequency ÷ SCLK frequency)

ARM Cortex-M0 Read from UART1 in M0 space (RWS = 0), at any Ratio_M0: Read wait states = 1 + RWS = 1

ARM Cortex-M0 Read from UART1 in M4 space (RWS = 0), at Ratio_M0 = 2:1: Read wait states = 2 + (5 + RWS) ÷ Ratio_M0 = 2 + (5 + 0) ÷ 2 = 2 + 3 = 5.

Table 2-7: ARM Cortex-M0 Wait States By Target Address Space

Target Space	Read Wait States	Write Wait States
ARM Cortex-M4 Boot ROM + Logic ROM	n/a (not accessible)	n/a
ARM Cortex-M4 Main SRAM	0 + (14 ÷ Ratio_M0)	1
ARM Cortex-M4 Core peripherals (MATH, Cortex PPB)	n/a	n/a
Flash Memory (min)	0 + (6 + (8 ÷ RatioCS)) ÷ Ratio_M0	n/a (Read only)
Flash Memory (max)	0 + (6 + (11 ÷ RatioCS)) ÷ Ratio_M0	n/a
M0 Core peripherals (PPB)	0	0
Mailbox memory ports	2 + (5 + RWS) ÷ Ratio_M0	2 + (5 + WWS) ÷ Ratio_M0
ARM Cortex-M4 System Peripheral MMRs	2 + (5 + RWS) ÷ Ratio_M0	2 + (5 + WWS) ÷ Ratio_M0
ARM Cortex-M0 System Peripheral MMRs	1 + RWS	1 + WWS
ARM Cortex-M0 SRAM	0	0
FFT memories	0 + 6 ÷ Ratio_M0	1
HAE memories	2 + 8 ÷ Ratio_M0	2 + 14 ÷ Ratio_M0
SMC	TBD	TBD

Information about the wait states in individual peripherals can be found in the Table 3-10 MMR Wait States by Peripheral table.

Boot Sequence

The ARM Cortex-M0 supervisor is booted entirely under control of the ARM Cortex-M4 main controller, in the following sequence:

- At system reset, both the ARM Cortex-M4 and ARM Cortex-M0 and the enclosing systems are reset.
- On deassertion of system reset, the ARM Cortex-M4 boots while the ARM Cortex-M0 core is retained in reset. The ARM Cortex-M4 Boot ROM takes no action regarding the ARM Cortex-M0. At this time all system resources on the ARM Cortex-M4 side and the ARM Cortex-M0 side are accessible to the ARM Cortex-M4, while the ARM Cortex-M0 core alone is held in reset.

After deassertion of the system reset, the user program performs the following tasks.

- Starts on the ARM Cortex-M4 and initializes itself. The ROM code does various self check operations and asserts the system fault pin if unable to complete the initialization procedure. On proper init completion, control is passed to user.
- Initializes and configures the system as desired (clocking, memory initialization, system protection modes, interrupt handlers and so on).
- Copies the ARM Cortex-M0 supervisor executable to the ARM Cortex-M0 SRAM from Flash, either by DMA or by direct processor memory operations, and validates the copy (for example using MDMA plus CRC or direct comparison with flash).
- Optionally may initialize the rest of the ARM Cortex-M0 SRAM (to init ECC states).
- Configures the system as desired to enable or restrict access to the ARM Cortex-M0 subsystem by the ARM Cortex-M4 system and vice versa, using address-range-based protection (SPU, SPMU) and register-granularity protection (LOCK, SPU) as appropriate.
- Initializes the ARM Cortex-M0's vector table relocation register (`SYSBLK_M0_VTOR`) in the C0 unit to point to the initial stack pointer and vector table in the downloaded ARM Cortex-M0 executable.

The ARM Cortex-M0 is then allowed to come out of reset when the ARM Cortex-M4 application writes the `RCU_CRCTL` registers in the RCU (See the Reset Control Unit (RCU) chapter). When this occurs the ARM Cortex-M0 performs the following tasks.

- The ARM Cortex-M0 fetches its initial stack pointer and reset vector address from the location pointed to by `SYSBLK_M0_VTOR`.
- Begins executing code at the reset vector address, usually within the ARM Cortex-M0's SRAM.
- Initializes itself, and optionally initializes the rest of its ARM Cortex-M0 SRAM (ECC states).
- Configures the ARM Cortex-M0 subsystem as desired to enable or restrict access by the surrounding system (ARM Cortex-M4 core or DMA peripherals).
- Configures the ARM Cortex-M0S subsystem monitor units (SPU, SMPU, SWU, timeout) to monitor M0 operations.

- Begins its supervisory tasks.

Cortex-M0 Registers

The Cortex-M0 core related registers are part of the SYSBLK unit. See System Block (SYSBLK) and PADS.

The following are the set of registers that are specific to the core:

- SYSBLK_IRQ_LATENCY: Register to program the M0 IRQ latency.
- SYSBLK_M0_VTOR: Register to program Vector Table Offset Register.
- SYSBLK_SYS_STCALIB: Register to program the Systick Calibration.
- SYSBLK_SISTAT0 to SYSBLK_SISTAT28: Registers to read the status of which interrupt is asserted.

The Cortex-M0 specific registers are titled SCS1_ prefix. For example, the ACTLR register is mentioned as SCS1_ACTLR.

Refer to the ***CM41X_M0 SCS1 MMR Register Addresses*** table.

3 ARM Cortex-M4 Core 0 Memory Sub-System

The ADSP-CM41x family of processors are based on the ARM Cortex-M4 processor core with floating-point unit and integrated SRAM memory, flash memory, accelerators, and peripherals.

The processor provides sufficient memory to support micro-controller based applications. This memory includes 160K bytes of internal tightly coupled SRAM that can be partitioned into any one of six choices for code/data memory partitioned into 32K byte blocks and 1M byte of tightly coupled flash memory with ECC. The processor also includes a static memory controller for interface to external devices or memories.

Note the following terms, which are used in this chapter:

- *Cortex core* refers to the ARM Cortex-M4 core with floating-point support and core peripherals.
- *Cortex memory* refers to the portions of the Cortex memory map that are part of the memory model for the Cortex core (for example, SRAM Flash and boot ROM), but are not part of the system memory (memory mapped registers) or external memory.
- *ADSP-CM41x processor* or *processor* refers to the combination of the Cortex core, Cortex memory, system peripherals (for example, UART, SPI, and SPORT), and system memories.

This document describes the ARM Cortex-M4 core and memory architecture used on the ADSP-CM41x processor, but does not provide detailed programming information for the ARM processor. For more information about programming the ARM processor, visit the ARM Information Center:

- http://infocenter.arm.com/help/

The applicable documentation for programming the ARM Cortex-M4 processor include:

- ARM Cortex-M4 Devices Generic User Guide
- Cortex -M4 Technical Reference Manual

Peripherals and Infrastructure on the ARM Cortex-M4 Sub-system Domain

The following tables identify the peripherals and infrastructure on the ARM Cortex-M4 Sub-system domain.

Table 3-1: Peripheral Blocks

Peripheral block	Description
SPI1	Serial Peripheral Interface
UART1–4	Universal Asynchronous Receiver Transmitter
ADCC1	Analog to Digital Converter Controller
CAN1	Controller Area Network
TIMER1 x 8	General Purpose Timer
SPORT0	Serial Port
TWI0	Twin Wire Interface
DACC0	Digital to Analog Converter Controller
HPPWM0–2	High-performance Pulse Width Module
HAE0	Harmonic Analysis Engine
SINC0	Sinus Cardinals Filter
CRC0	Cyclic Redundancy Check Unit
FFT0	Fast Fourier Transform Unit
ROT0	Rotary Counter
CPT0	Capture Timer
LBA0	Logic Block Array

Table 3-2: Infrastructure Blocks

Infrastructure block	Description
DDE4–13	DMA Engine
TRU1	Trigger Routing Unit
SEC1	System Event Controller
WDT1	Watchdog Timer
PORTB–F	General Purpose Port
PINT1–5	Peripheral Interrupt Module
PADS1	Pads Interface
SYSBLK1	System Block Interface
SPU1	System Protection Unit
SMPU1–2	System Memory Protection Unit
SWU2–6	System Watch-point Unit
TTU0	Trigger Trimming Unit

Table 3-3: ARM Cortex-M4 Tightly Coupled Unit

M4 Tightly coupled unit	Description
MATH0	Math Accelerator Unit

Cortex-M4 Memory Features

The ARM Cortex-M4 core memory architecture includes the following features:

- An internal memory sub-system supporting:
  - Up to 160K bytes of zero waitstate and configurable SRAM
  - Up to 1M of internal embedded, low-latency, parallel flash memory

Cortex-M4 Memory Functional Description

The following sections provide the functional description for the Cortex-M4 core memory sub-system.

CM41X_M4 M4P Interrupt List

Table 3-4: CM41X_M4 M4P Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
5	M4P_BCODE_ERR	M4P Boot Code CRC Error	Level	
9	M4P_LOCKUP	M4P Lockup Error (Fault only; not an interrupt)	Level	
10	M4P_BUS_FAULT	M4P Bus Fault	Edge	
14	M4P_CORE_SRAM_EERR	M4P SRAM Core ECC Error	Level	
15	M4P_DMA_SRAM_EERR	M4P SRAM DMA ECC Error	Level	
158	M4P_SOFT_INT0	M4P Software Interrupt 0	Edge	
159	M4P_SOFT_INT1	M4P Software Interrupt 1	Edge	
160	M4P_SOFT_INT2	M4P Software Interrupt 2	Edge	
161	M4P_SOFT_INT3	M4P Software Interrupt 3	Edge	
162	M4P_SOFT_INT4	M4P Software Interrupt 4	Edge	
163	M4P_SOFT_INT5	M4P Software Interrupt 5	Edge	

CM41X_M4 M4P Register List

The ARM Cortex-M4 platform module (M4P) provides the interface to the L1 SRAM. A set of registers governs M4P operations. For more information on M4P functionality, see the M4P register descriptions.

Table 3-5: CM41X_M4 M4P Register List

Name	Description
M4P_BROMERR	Boot ROM Error Register
M4P_BUSFLT	Bus Fault Error Information Register
M4P_SRAM_CFG	SRAM Configuration Register
M4P_SRAM_EEADDR_CORE	SRAM ECC Error Address (Core) Register
M4P_SRAM_EEADDR_DMA	SRAM ECC Error Address (DMA) Register
M4P_SRAM_INITDONE	SRAM Initialization Done Status Register
M4P_SRAM_RFRADDR	SRAM Refresh Address
M4P_SRAM_RFRPER	SRAM Refresh Period Register
M4P_STCALIB	SysTick Calibration Register

Cortex-M4 Code and Data SRAM

The unified internal SRAM space provides both code and data memory for the ARM Cortex-M4 processor core, allowing a configurable partition between code and data space. In addition, the SRAM features a DMA access port for read/write access by other master devices on the system fabric. The SRAM and all its interfaces operate in the ARM Cortex-M4 core clock domain (CCLK). The SRAM supports exclusive accesses. The SRAM can be accessed at the maximum CCLK speed in zero wait states. For access timing information, see the product data sheet.

SRAM Features

The SRAM has the following features:

- Up to 160K byte SRAM Capacity
- Zero wait-state performance at maximum CCLK speed
- Dynamically configurable between code space and data SRAM space partitions
- ECC protection (SEC-DED)
- Exclusive access support
- Two 32-bit buses for ARM Cortex-M4 core access to code space (MEM_ICODE, MEM_DCODE)
- One 32-bit bus for ARM Cortex-M4 core access to SRAM space (MEM_SYS)
- One 32-bit bus for system DMA access to code and SRAM spaces (SRAM_DMA)
- Defined maximum DMA response latency due to collisions with core activity. Programmable up to 16 cycles (default =8 cycles)

SRAM Bank Organization

The SRAM resources are divided into banks for efficiency (as shown in the *SRAM Address Fields And Mapping* figure), reducing the conflicts between accesses from various sources. These sources include: core fetch versus core load/store versus DMA. Usually, programs do not need to worry about the internal SRAM organization, other than conflict management.

- MSB Striping (banking) divides memory by 32K byte regions into separate ConfigBanks. Accesses to different ConfigBanks never cause conflict.
- LSB striping divides memory into four 32-bit lanes, so that accesses in different lanes do not conflict.

  LSB striping efficiently spaces out consecutive address reads/writes across multiple banks. This reduces overall stalls and reduces ECC penalty for byte/word write accesses.

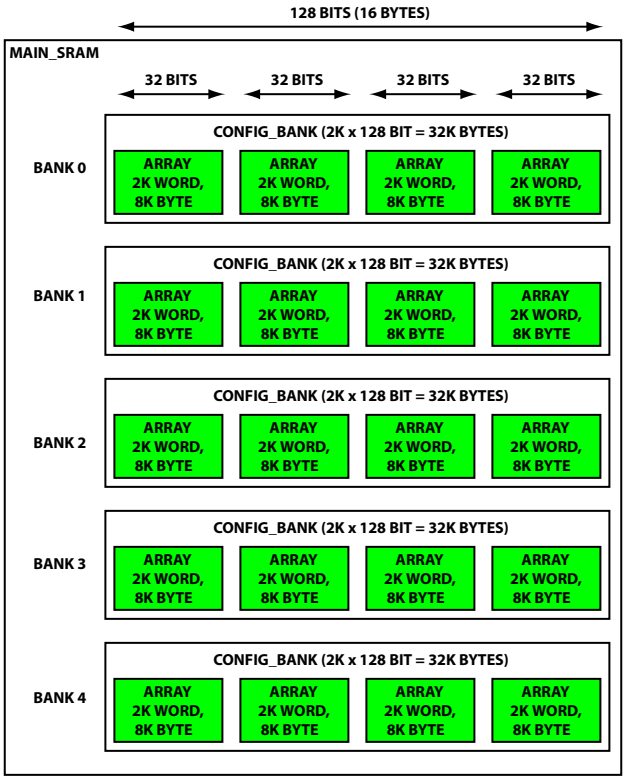


Figure 3-1: SRAM Address Fields And Mapping

Table 3-6: SRAM Address Fields And Mapping

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	MSB			Array Address (11 bits, 2K)											LSB (Array-Bank)		Byte	
0	0	(CFG Banks)																	

SRAM Partitioning Using Config Banks

The address range of main SRAM depends on the configuration settings in the M4P_SRAM_CFG.CDBANKS register field, as shown in the see ***Memory Map Configuration, Code and SRAM Regions*** figure. It shows an implementation using five config banks. Memory available is denoted by the shaded regions. It is an example of active blocks as per the SRAM_CODE configuration.

Table 3-7: SRAM Bank Configuration Base Addresses

Config	31	30	29	28	27	26	25	24	23	22	21	20	Description
CODE	0	0	0	1	0	0	0	0	0	0	0	0	CODE at 0x1000_0000
DATA	0	0	1	0	0	0	0	0	0	0	0	0	DATA at 0x2000_0000

The memory resources in the ARM Cortex-M4 memory main SRAM are divided into two or more ConfigBanks. Each ConfigBank may appear in either the code or the SRAM region, but may not appear in both. The settings in the supervisor-only M4P_SRAM_CFG.CDBANKS register field specify how many contiguous ConfigBanks appear in the CODE segment, starting at the lowest address (0x1000_0000) The remaining ConfigBanks appear in a continuous address range in the DATA segment, ending at the highest populated address (0x2000_0000 + SRAM_SIZE 1 ((0x2000_0000 + SRAM_SIZE 1), or 0x2002_FFFF)).

When the value of the M4P_SRAM_CFG.CDBANKS register is changed, resulting in changes to the populated regions in the memory map, the contents of newly-accessible memory ranges are UNSPECIFIED. The user should not assume that the contents of any specific memory range are transferable between the CODE region to the SRAM region when the M4P_SRAM_CFG.CDBANKS register is changed.

It is important to manage the M4P_SRAM_CFG.CDBANKS register when booting or initializing an application, so that the intended memory map is configured before the application is copied from the boot source into the active locations in the CODE or SRAM regions. For example, the program may choose a linker control file which specifies 96 KB of CODE (0x1000_0000 to 0x1001_7FFF) and 64 KB of DATA SRAM (0x2001_8000 to 0x2002_7FFF). The linker may automatically support copying read-only sections of the application from nonvolatile (Flash) memory regions to the SRAM regions at start-up time in an init function. It is important that the M4P_SRAM_CFG.CDBANKS register is initialized to the intended value (here, 3) before init is called. This task can be performed in the reset handler.

Programs should not attempt an active access to a memory bank when changing the configuration for that bank, including instruction fetches. A program that changes the M4P_SRAM_CFG.CDBANKS register must reside either outside SRAM (for example, in Flash), or in an SRAM bank that is unaffected by that change. Programs should implement the following items. The code that is used can vary depending on the assembler.

- Disable interrupts
- DMA inactive and the Cortex-M0 core avoiding the Cortex-M4 memory space
- Instruction-Sync-Boundary (ISB)
- cdbank change memory write

- Enable interrupts

NOTE: The initial SP (Stack Pointer) recorded in the vector table should be set to point to the top of populated data SRAM. This address is always populated as long as there is at least one ConfigBank allocated to DATA. If an application allocates all confgBanks to CODE, then the initial SP must be set to point into CODE space instead of DATA space.

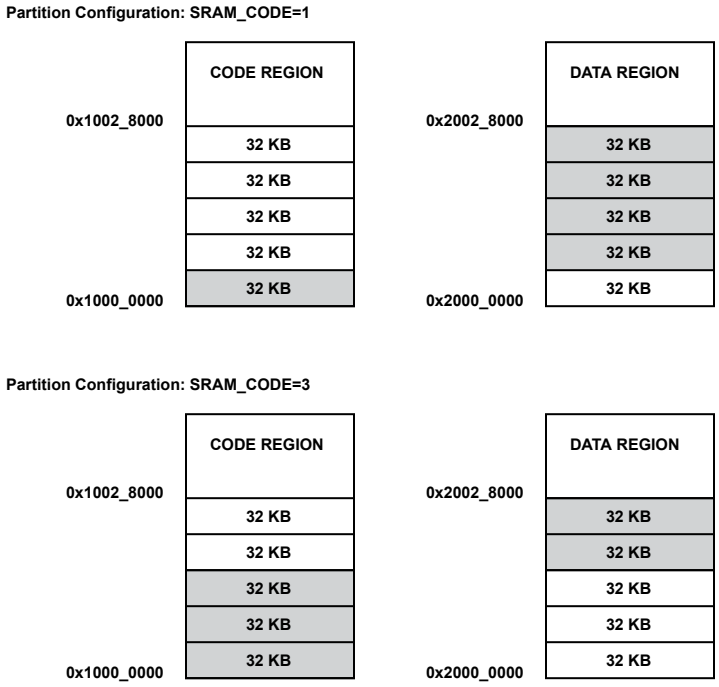


Figure 3-2: Memory Map Configuration, Code and SRAM Regions

SRAM Posted System Writes (NormSysWrite versus PostSysWrite)

The `M4P_SRAM_CFG.POSTWR` bit controls the behavior of nonexclusive writes by the ARM Cortex-M4 core to system space (outside the ARM Cortex-M4 memory). This allows control of the trade off between performance and precise error detection.

In NORMSYSWRITE mode (`M4P_SRAM_CFG.POSTWR` =0), non-exclusive ARM Cortex-M4 core writes to system space are performed with normal system fabric (bus interconnect) write transactions. The Cortex core must wait several core clock cycles for the target peripheral or device to return a bus response (OKAY or ERROR) before proceeding with further instructions. If an ERROR response is returned, the Cortex core generates a precise HardFault exception at the instruction that caused the error. Typically, an ERROR response results from an invalid address, an invalid data size, or a protection violation such as writing to a read-only location or accessing a privileged resource in non-privileged mode.

In POSTSYSWRITE mode (`M4P_SRAM_CFG.POSTWR` =1), non-exclusive ARM Cortex-M4 core writes to system MMR space are posted, meaning that the Cortex memory interface accepts the write transaction from the Cortex core and immediately returns an OKAY response. The Cortex core proceeds with subsequent instructions, while independently, the Cortex memory interface forwards the posted write to the system fabric (bus interconnect).

Several System MMR writes in a burst may be posted without stall up to the depth of an internal FIFO (4 deep). If further accesses are posted faster than the system FIFO can complete the transactions, the core stalls.

If a system fabric (bus interconnect) ERROR response is returned from a posted write, the Cortex core cannot signal an exact HardFault exception because it has already gone on to execute further instructions. In this case, the Cortex memory captures the address of the erroneous transaction in the M4P_BUSFLT.ADDR field, sets the M4P_BUSFLT.STAT bit to 1, and asserts the interrupt. The handler for this interrupt can determine the offending address using the M4P_BUSFLT register and take appropriate action. The interrupt is cleared by writing a 1 to the M4P_BUSFLT.STAT bit.

In the ADSP-CM41x, only writes by the M4 core to system MMRs may be posted. Writes by the M4 Core to system memories (for example, the M0 SRAM or the FFT buffers) do not post. While individual reads or writes are sometimes useful, the most performance-efficient way to manage the bulk contents of these memories is by using MDMA.

Memory Initialization

The main SRAM memory can be initialized in hardware. This is initiated by writing the M4P_SRAM_CFG.INIT control bit to a 1. The progress of the initialization can be monitored using the M4P_SRAM_CFG.INITDONE status bit. While initialization is in progress, all access attempts to the SRAM on the core or DMA buses are stalled until initialization completes. After initialization, all memory locations contain zeroed data, without ECC error.

Memory initialization happens to all main banks in parallel. Memory initialization on the ADSP-CM41x takes approximately 33k core clock cycles. Memory initialization is typically performed automatically by the CM41x Boot ROM program after each system reset, before the user's application is executed.

By default, the boot performs a memory initialization. If user memory initialization has to be performed, the program must execute from flash as all SRAM memory locations do not contain any data.

1. All DMA operations to the M4P Main SRAM must be stopped, including both DMA controller streams and any DMA-capable peripherals, accelerators, or other processors which may write to the M4P Main SRAM via its DMA port.
2. Execute a memory barrier instruction (_DSB or ISB) between the final write to memory and the write to the M4P_SRAM_CFG.INIT MMR bit.

It is permissible to synchronize the software to the end of the init operation by simply reading any memory location in the array, as follows:

1. Write the M4P_SRAM_CFG.INIT MMR bit to 1.
2. Execute a system memory barrier instruction (_DSB or _ISB).
3. Read a memory location in the main M4P SRAM.

NOTE: Memory initialization is required before first accessing the memory after power-on, as uninitialized memory contains uninitialized ECC states and any read or partial-word write accesses to uninitialized memory may cause ECC error detections. The boot code initializes the full Cortex-M4 SRAM region by default.

NOTE: Memory initialization must not be executed from a routine inside SRAM, nor using a stack pointer inside SRAM. It is recommended that memory initialization either take place from ROM (in the ADSP-CM41x processor BOOT ROM performs the initialization) or Flash memory, followed immediately by a re-initialization of the stack pointer. Alternatively, memory initialization may be performed from code and stack resident in off-chip L3 space.

```
register uint32_t *cfgp = &(pADI_M4P->SRAM_CFG);
register int *memp = (any address in M4P SRAM);
register uint32_t cfgd, memd;
// set up to write MMR to cause memory init
cfgd = *cfgp | BITM_M4P_SRAM_CFG_INIT;
__dsb();                // barrier instruction
*cfgp = cfgd;           // start memory init
__dsb();                // barrier instruction;
memd = *memp;           // sync to end of memory init
```

Refresh Function, Background Memory Scrubber

Refresh is a background operation that performs an atomic read/write of memory locations to allow periodic ECC correction to the physical memory arrays.

A basic refresh cycle requires three cycles of access to a memory sub-bank without any intervening processor or DMA activity. In these cycles the following operations occur.

The refresh operation works in the background, waiting for a refresh opportunity of three consecutive foreground idle cycles. If the refresh 3-cycle sequence is pre-empted by foreground activity, the sequence is retried the next time that idle cycles present a refresh opportunity. (This ensures the refresh activity never corrupts or un-does a foreground write.)

When a successful refresh operation completes, the refresh address increments and the refresh attempts to access the next spot in the physical array.

The refresh operates on the physical array, it does not care about the bank data/instruction programming.

If an ECC error is encountered during the refresh sequence it is reported and the error is captured in the SRAM Refresh Address register (M4P_SRAM_RFRADDR).

Because performing a refresh operation consumes power and the expected error rate of these SRAM is small, a SRAM refresh period register is provided (M4P_SRAM_RFRPER). This register holds the number of cycles to wait between a successful refresh operation and a refresh attempt to the next address. Due to the background operation of refresh the exact time between two successful refresh operations is variable. The default refresh period is the minimum.

ECC Error Signaling

The Cortx M4P provides error signaling to report uncorrectable ECC errors of two or more bits within an aligned 32-bit ECC unit. When the main SRAM detects an uncorrectable ECC error in an access on any interface, the details of the erroneous access are captured in one of the reporting registers (described below) as appropriate, and an interrupt (if enabled) is signaled on the corresponding interrupt port.

- The M4P_SRAM_EEADDR_CORE register reports on the M4's ICODE, DCODE, SYS read ports, and the Refresh port, in that priority order. This register reports individual errors using its M4P_SRAM_EEADDR_CORE.EERR bit and multiple errors using its M4P_SRAM_EEADDR_CORE.MEERR bit field.
- The M4P_SRAM_EEADDR_DMA register reports errors on the DMA read port. This register reports individual errors using its M4P_SRAM_EEADDR_DMA.EERR bit and multiple errors using its M4P_SRAM_EEADDR_DMA.MEERR bit field.

When an error is detected in the core, the register's M4P_SRAM_EEADDR_CORE.EERR bit is set. The details are captured in the read-only M4P_SRAM_EEADDR_CORE.ADDR and M4P_SRAM_EEADDR_CORE.BUS bit fields.

When an error is detected in the DMA, the M4P_SRAM_EEADDR_DMA.EERR bit is set.

If multiple simultaneous ECC errors are detected on more than one of the interfaces, the M4P_SRAM_EEADDR_CORE.MEERR (core), and M4P_SRAM_EEADDR_DMA.MEERR (DMA) bits are set. One of the failing accesses is selected according to the above priority scheme. Only the details of the selected access are captured in the register. In this instance, the multiple error bit behaves as a read-only attribute and is not cleared by W1C.

Sequential MERRs (Overrun): If a reporting register's M4P_SRAM_EEADDR_CORE.EERR bit is already set from a previous event when a new ECC error is detected, then the registers' M4P_SRAM_EEADDR_CORE.MEERR (core), and M4P_SRAM_EEADDR_DMA.MEERR bit is asserted, and the details of the previous event (offset address, bus interface ID) are overwritten. In this instance, the multiple error bit can and should be cleared by a W1C in the handler.

The M4P_SRAM_EEADDR_CORE.EERR or M4P_SRAM_EEADDR_DMA.EERR bits have W1C semantics and are cleared by the associated interrupt handler. It is recommended that the M4P_SRAM_EEADDR_DMA.MEERR bit field also be cleared by W1C in the handler.

ECC Error Interrupt Handlers

The purpose of the ECC error interrupts (for CORE and DMA) is to notify the application that information in the main SRAM has been lost, while presenting the full details of the loss, and to allow the application a means to prevent further actions which might otherwise be corrupted by the memory error. Such actions might include shutting down active processes, communicating details of the error to off-chip recipients, and/or logging the error details in on-chip storage.

Note that the interrupts are not the same as exact exceptions: they occur a few cycles after the error detection, so the handler does not return to the instruction which caused the error. The error handler cannot be used to attempt to replace the erroneous data from some other source and to replay the failing instruction.

The M4P_SRAM_EEADDR_CORE.MEERR (core), and M4P_SRAM_EEADDR_DMA.MEERR (DMA) bits indicate to the handler that information in the reporting registers is incomplete and that at least one other access has caused an error which could not be represented in the reporting registers. This may be relevant to statistical or logging purposes.

Memory Built-In Self Test (MEMST)

The ADSP-CM41x provides a Memory Built-In Self Test (MEMST) engine, that validates the functionality of every embedded SRAM array and provides a definite pass/fail indication to the application. MEMST is not a background task, it is a process which requires exclusive access to all SRAM resources in the processor system (see Restrictions below). The MEMST module is invoked from user application code and is not controlled or supported by the boot ROM.

Features

The MEMST module has the following features:

- MEMST provides a fully automated SRAM test.
- All SRAMs are tested. The BOOT ROM and Logic ROM are NOT tested. Internal flash is not tested.
- The SYSBLK_MEMST_CTL register in the C1 block is used to control MEMST.
  - The SYSBLK_MEMST_CTL.EN control bit enables User mode. It must be cleared between test runs to reset the test logic.
  - The SYSBLK_MEMST_CTL.RUN control bit starts the MEMST test sequence.
  - The SYSBLK_MEMST_CTL.BUSY status bit indicates when the test sequence has finished.
  - The SYSBLK_MEMST_CTL.FAIL status bit indicates the success of the test. It is not valid until SYSBLK_MEMST_CTL.BUSY is LOW.

MEMST Handler Operation

An MEMST handler operates generally as follows.

- The MEMST handler must follow the restrictions noted above: it must be located in Flash memory, and once MEMST starts, it must avoid using any SRAM resources, and must use ARM core registers only.
- Stop all DMA operation of peripherals in both the ARM Cortex-M4 and ARM Cortex-M0 subsystems: SPIs, SPORTs, UARTs, SINC, ADCCs, DACCs, and MDMA.
- Stop the FFTB and HAE accelerators.
- Disable all interrupts. Take care to ensure ARM exceptions do not occur.
- Other system configuration which does not employ SRAM or interrupts may remain operational. These include:
  - GPIO states and the Logic Block Array (LBA),
  - Timers
  - TRU triggers

  - peripherals which do not use or are not configured to use DMA or SRAM, for example the UART, SPI, SPORT, and/or CAN (if programmed entirely using ARM Cortex processor registers and peripheral MMRs)
  - watchdog monitors such as SEC, WDT, VMU, OCU and OSCWDOG (using hardware signaling only via TRU and/or PinSafeState)
  - Emulator/debug access (provided access to SRAMs is avoided, for example through memory display windows in the debugger GUI).
- Handlers must obey the restriction to not employ any SRAM or stack. Options include
  - asserting the $\overline{\texttt{SYS\_FAULT}}$ pin using the SEC
  - communicating with an external device using peripherals such as UART, CAN or SPI in MMR-based modes
- If the MEMST operation was successful (`SYSBLK_MEMST_CTL.FAIL` =0), the stack and C runtime environment must be reinitialized; a system reset is recommended but not required.

MEMST Programming Model

The following restrictions and recommendations should be followed to ensure proper operation of the MEMST module.

- Before running MEMST, all DMA operation by peripherals must be stopped.
- The MEMST monitor code must be run from Flash, as SRAM contents are destroyed during test.
- Code must run without stack (using core registers only) as SRAM is unavailable during test.
- Embedded memories within peripherals must not be used during test, including the MBOX memory, the ETF Debug/Trace memory, and the FFTB and HAE accelerators.
- All interrupts must be disabled, and care must be taken not to incur any ARM processor exceptions, as the interrupt/exception entry sequence attempts to push processor state to the stack in SRAM, which is not allowed during MEMST.
- It is recommended to run MEMST from the ARM Cortex-M4. In this case, the ARM Cortex-M0 should be stopped (or reset) to prevent it accessing its SRAM during MEMST.
- If MEMST is run from the ARM Cortex-M0, the ARM Cortex-M4 must first be placed in a state (or routine) in which it no longer accesses SRAM until after the test is complete.

Memory and MMR Access Latencies

This section provides information on the ARM Cortex-M4 core read and write memory-mapped register access latencies.

The time required to perform a read or write access by the ARM Cortex-M4 Core is specified in this section. This is the time required for the processor load or store instruction to execute, described in units of the local core clock (CCLK0 for the ARM Cortex-M4), as measured by the cores SysTick unit.

The time required to perform the load/store, in local core clocks, is:

TLDST = 1 + coreWaitStates + (Ratio_CtoS × sysWaitStates)

where:

coreWaitStates = CoreWS from the ***ARM Cortex-M4 Wait State Formulas by Target Space*** table, for a given master (ARM Cortex-M4) and target space.

syseWaitStates = SysWS from the ***ARM Cortex-M4 Wait State Formulas by Target Space*** table, for a given master (ARM Cortex-M4) and target space.

Ratio_CtoS = (CCLK0 frequency ÷ SYSCLK frequency).

In the ***ARM Cortex-M4 Wait State Formulas by Target Space*** table:

- POSTWR. Wait States for Writes may depend on the Posted Write mode set by the `M4P_SRAM_CFG.POSTWR` bit. Use the appropriate column in the ***ARM Cortex-M4 Wait State Formulas by Target Space*** tavle for the posted write mode. If `M4P_SRAM_CFG.POSTWR` =0, all system writes are direct (non-posted). If `M4P_SRAM_CFG.POSTWR` =1, all system writes are posted, up to the depth of the ABB-to-APB FIFO.
- ASYNC A. The variable A is either 0 or 1, and denotes the ARM Cortex-M4 System Bus Synchronization mode set by the `M4P_SRAM_CFG.SYNCCLK` bit, as given by the ***ARM Cortex-M4-System Bus Synchronization Mode*** table.
- PHASE Φ. Some accesses may cross the core to system clock boundary, and if so, a clock phase alignment delay Φ is added. This delay is from 0 to (Ratio_CtoS – 1) core clocks, and is dependent on the instantaneous alignment of the two clocks at the time of the access.

Table 3-8: ARM Cortex-M4 Wait State Formulas by Target Space

Target Space	Read		Write (Non-Posted)		Write (Posted)	
	CoreWS	SysWS	CoreWS	SysWS	CoreWS	SysWS
M4 Boot ROM + Logic ROM	1	0	N/A	N/A	N/A	N/A
ARM Cortex-M4 Main SRAM	0	0	0	0	0	0
ARM Cortex-M4P Core peripherals (MATH, Flash, Cortex Private Peripheral Bus (PPB))	0	0	WWS	0	same	same
Mailbox Port P0						
Mailbox Port P1	1 + 2A + φ	3+A	0 + 2A + φ	1 + A	0	0

Table 3-8: ARM Cortex-M4 Wait State Formulas by Target Space (Continued)

Target Space	Read		Write (Non-Posted)		Write (Posted)	
	CoreWS	SysWS	CoreWS	SysWS	CoreWS	SysWS
ARM Cortex-M4 System Peripheral MMRs	1 + 2A + φ	2+A+RWS	0 + 2A + φ	2 + A + WWS	0	0
ARM Cortex-M0 System Peripheral MMRs	1 + 2A + φ	8 + A + RWS	0 + 2A + φ	8 + A + WWS	0	0
ARM Cortex-M0 SRAM	3 +A + φ	10 + A	0 + A + φ	4 + A	same*1	same*1
FFT	3+A+ φ	6+A	0+A+ φ	4+A	same*1	same*1
HAE						
SMC						

*1 Posted Write mode only affects accesses to peripheral MMRs. It does not affect accesses to system memory spaces such as the SMC, M0 SRAM, or the FFT or HAE accelerator memory buffers

Table 3-9: ARM Cortex-M4-System Bus Synchronization Mode

SRAM_CFG.SYNCCLK Mode	Async delay factor A	Notes
0 = Asynchronous Mode	A=1	Bus Synchronizers Active
1 = Synchronous Mode	A=0	Bus Synchronizers Bypassed

NOTE: If the clock ratio Ratio_CtoS is not an integer (for example 240:96 MHz where MHz operation is a 5:2 ratio), then the bus synchronization mode must be Asynchronous. The expressions in the ***M4 Wait State Formulas by Target Space*** table describe the additional delays (A, 2A, etcetera) required for the system-to-core bus synchronizers.

If the clock ratio Ratio_CtoS is an integer (for example 240:80 MHz, where MHz operation is a 3:1 ratio), then the bus synchronization mode may be either synchronous or asynchronous. If the mode is synchronous, then no additional synchronizer delay is imposed.

Posted ARM Cortex-M4 Writes FIFO Delay

In PostWR mode, any single write or burst of writes by the ARM Cortex-M4 up to the depth of the AHB-APB FIFO is executed with no wait states. This FIFO has a depth of four. Additional writes after the FIFO fills may incur a wait state delay up to the delay of a non-posted write. The FIFO drains at a rate related to the number of system wait states for the given peripheral, as specified in the ***MMR Wait States by Peripheral*** table.

TWDRAIN = ((WWS + 1) × Ratio_CS) × Nwrites

Once the FIFO has drained to the extent that it is no longer full, additional posted writes execute with no wait states.

Table 3-10: MMR Wait States by Peripheral

Peripheral	Space	Wait States		Units
		Read (RWS)	Write (WWS)	
ADCC0	Cortex-M0	1	0	SYSCLK
CAN0 1, 2	Cortex-M0	1	0	SYSCLK
DMA0–3	Cortex-M0	0	0	SYSCLK
PINT0	Cortex-M0	0	0	SYSCLK
PORTA	Cortex-M0	0	0	SYSCLK
SEC0	Cortex-M0	2	1	SYSCLK
SMPU0	Cortex-M0	1	1	SYSCLK
SPI0	Cortex-M0	1	0	SYSCLK
SPU0	Cortex-M0	2	1	SYSCLK
SWU0–1	Cortex-M0	1	1	SYSCLK
TEPADS0	Cortex-M0	2	2	SYSCLK
TESYS0	Cortex-M0	0	0	SYSCLK
TIMER0	Cortex-M0	1	0	SYSCLK
TRU0	Cortex-M0	2	1	SYSCLK
UART0	Cortex-M0	0	0	SYSCLK
WDOG0	Cortex-M0	0	0	SYSCLK
ADCC1	Cortex-M4	1	0	SYSCLK
CAN1 1, 2	Cortex-M4	1	0	SYSCLK
CGU0	Cortex-M4	1	1	SYSCLK
CNT0	Cortex-M4	0	0	SYSCLK
CPTMR0	Cortex-M4	1	0	SYSCLK
CRC0	Cortex-M4	1	0	SYSCLK
DACC0	Cortex-M4	1	0	SYSCLK
DMA4–13	Cortex-M4	0	0	SYSCLK
EFS0	Cortex-M4	1	1	SYSCLK
FFTB0	Cortex-M4	1	0	SYSCLK
HAE0	Cortex-M4	0	0	SYSCLK
HPPWMBST	Cortex-M4	0	0	SYSCLK
LBA	Cortex-M4	0	0	SYSCLK
MBOX0	Cortex-M4	1	0	SYSCLK

Table 3-10: MMR Wait States by Peripheral (Continued)

Peripheral	Space	Wait States		Units
		Read (RWS)	Write (WWS)	
OCU0	Cortex-M4	0	0	SYSCLK
PINT1–5	Cortex-M4	0	0	SYSCLK
PORTB–F	Cortex-M4	0	0	SYSCLK
PWM0–2	Cortex-M4	0	0	SYSCLK
RCU0	Cortex-M4	1	1	SYSCLK
SEC1	Cortex-M4	2	1	SYSCLK
SINC0	Cortex-M4	0	0	SYSCLK
SMC0	Cortex-M4	0	0	SYSCLK
SMPU1–2	Cortex-M4	1	1	SYSCLK
SPI1	Cortex-M4	1	0	SYSCLK
SPORT0	Cortex-M4	1	0	SYSCLK
SPU1	Cortex-M4	2	1	SYSCLK
SWU2–6	Cortex-M4	1	1	SYSCLK
TAPC	Cortex-M4	1	1	SYSCLK
TEPADS1	Cortex-M4	2	2	SYSCLK
TESYS1	Cortex-M4	1	1	SYSCLK
TIMER1	Cortex-M4	1	0	SYSCLK
TRU1	Cortex-M4	2	1	SYSCLK
TTU0	Cortex-M4	0	0	SYSCLK
TWI0	Cortex-M4	1	0	SYSCLK
UART1	Cortex-M4	0	0	SYSCLK
WDOG1	Cortex-M4	0	0	SYSCLK
CSTF0	Private Peripheral Bus	2	2	CCLK
CTI03	Private Peripheral Bus	4	4	CCLK
CXTMC0	Private Peripheral Bus	2	1	CCLK
DWT0	Private Peripheral Bus	0	0	CCLK
ETM0	Private Peripheral Bus	1	1	CCLK
FPB0	Private Peripheral Bus	0	0	CCLK
ITM0	Private Peripheral Bus	0	0	CCLK
M4P	Private Peripheral Bus	0	0	CCLK

Table 3-10: MMR Wait States by Peripheral (Continued)

Peripheral	Space	Wait States		Units
		Read (RWS)	Write (WWS)	
M4PFLASH0	Private Peripheral Bus	0	0	CCLK
MATH0	Private Peripheral Bus	0	0	CCLK
SCS0_DFR0	Private Peripheral Bus	0	0	CCLK
SCS0_NVIC	Private Peripheral Bus	0	0	CCLK
TRACE0	Private Peripheral Bus	1	1	CCLK

CM41X_M4 M4P Register Descriptions

ARM Cortex-M4 Platform (M4P) contains the following registers.

Table 3-11: CM41X_M4 M4P Register List

Name	Description
M4P_BROMERR	Boot ROM Error Register
M4P_BUSFLT	Bus Fault Error Information Register
M4P_SRAM_CFG	SRAM Configuration Register
M4P_SRAM_EEADDR_CORE	SRAM ECC Error Address (Core) Register
M4P_SRAM_EEADDR_DMA	SRAM ECC Error Address (DMA) Register
M4P_SRAM_INITDONE	SRAM Initialization Done Status Register
M4P_SRAM_RFRADDR	SRAM Refresh Address
M4P_SRAM_RFRPER	SRAM Refresh Period Register
M4P_STCALIB	SysTick Calibration Register

Boot ROM Error Register

The M4P_BROMERR register is not typically used for user applications. It sets or clears the BCODE_ERR interrupt source and may raise the corresponding fault in SEC1. This fault (and its associated interrupt number) are reserved for the use of the boot ROM.

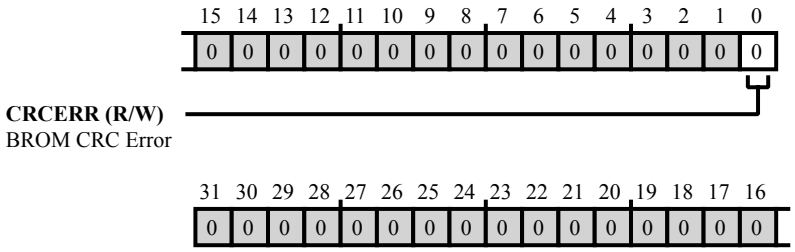


Figure 3-3: M4P_BROMERR Register Diagram

Table 3-12: M4P_BROMERR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	CRCERR	BROM CRC Error. The M4P_BROMERR.CRCERR bit sets or clears the BCODE_ERR interrupt. The SEC fault which corresponds to this interrupt code should be used only by the logic boot ROM program. This fault indicates the program has detected a CRC error with the Boot ROM array, or other fault in the initial boot process. The logic Boot ROM asserts one or more diagnostic codes for the fault reason(s) in the SYSBLK_LROM_STAT register. Note that because SRAM and the processor stack are not initialized at the time these faults are detected, interrupts are not safe to assert. Accordingly, the logic boot ROM does not assert the BCODE_ERR interrupt line; instead it asserts the corresponding fault in the SEC1 unit directly using SEC1 registers.

Bus Fault Error Information Register

The M4P_BUSFLT register captures the status and address of bus fault errors resulting from inexact posted writes by the Cortex to system space. Posted writes are enabled by the M4P_SRAM_CFG.POSTWR bit. The M4P_BUSFLT.STAT bit drives the M4P_BUS_FAULT interrupt, and is set on the detection of a posted write bus fault, and is cleared if written with a 1.

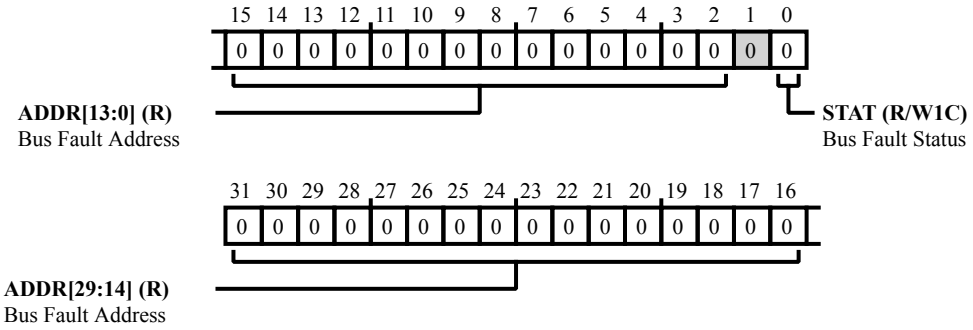


Figure 3-4: M4P_BUSFLT Register Diagram

Table 3-13: M4P_BUSFLT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:2 (R/NW)	ADDR	Bus Fault Address. The M4P_BUSFLT.ADDR bit field provides the address of bus fault errors resulting from inexact posted writes by the Cortex to system space.
0 (R/W1C)	STAT	Bus Fault Status. The M4P_BUSFLT.STAT bit provides the status of bus fault errors resulting from inexact posted writes by the Cortex to system space.

SRAM Configuration Register

The M4P_SRAM_CFG register selects the SRAM configuration.

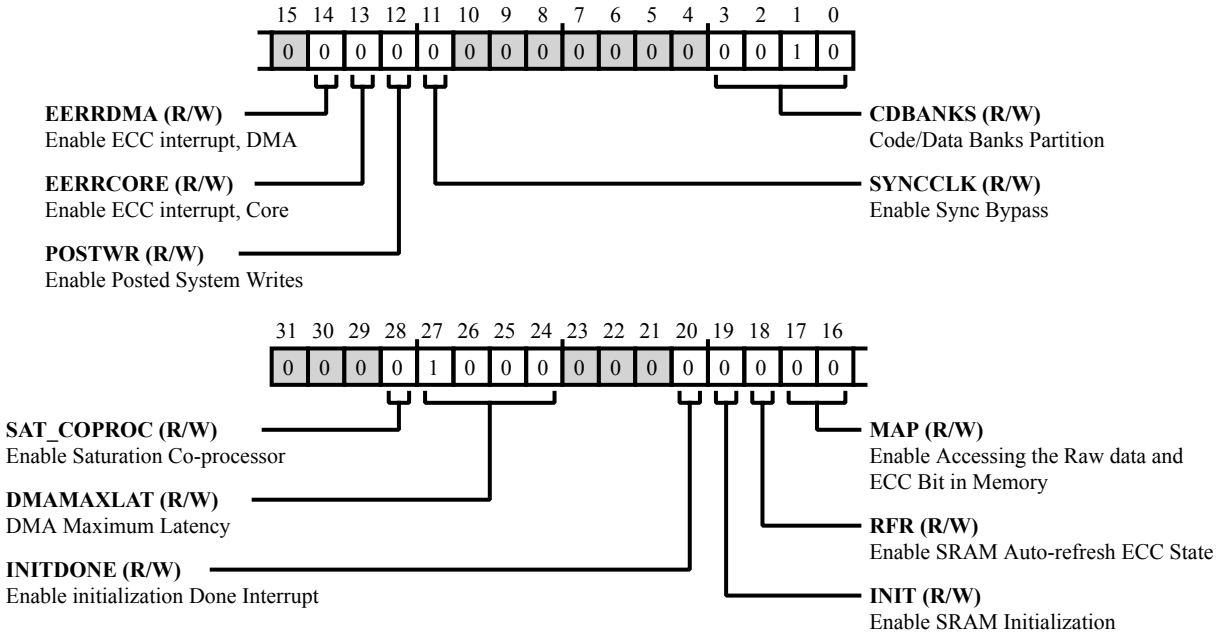


Figure 3-5: M4P_SRAM_CFG Register Diagram

Table 3-14: M4P_SRAM_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
28 (R/W)	SAT_COPROC	Enable Saturation Co-processor. The M4P_SRAM_CFG.SAT_COPROC bit enables saturation as described in the Floating-Point Saturation Unit (FSAT) chapter.
27:24 (R/W)	DMAMAXLAT	DMA Maximum Latency. The M4P_SRAM_CFG.DMAMAXLAT field controls the DMA maximum latency. When the DMA stalls for this number of cycles, the DMA access is moved to the highest priority. If DMA access is performed on mem blocks actively used for code/data, this guarantees the DMA is not locked-out of the memory block.
20 (R/W)	INITDONE	Enable initialization Done Interrupt. The M4P_SRAM_CFG.INITDONE bit enables an interrupt on an initialization-complete event.

Table 3-14: M4P_SRAM_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	INIT	Enable SRAM Initialization. When the `M4P_SRAM_CFG.INIT` bit is set to 0x1 the SRAM memory initialization to all zeros data will start; initializes ECC states. Reads 1 while initialization is pending. Accesses occur at highest priority, blocking any core or DMA accesses (causing access stalls).	
18 (R/W)	RFR	Enable SRAM Auto-refresh ECC State. The `M4P_SRAM_CFG.RFR` bit enables the M4 SRAM background ECC scrubber.	
17:16 (R/W)	MAP	Enable Accessing the Raw data and ECC Bit in Memory. The `M4P_SRAM_CFG.MAP` bit field exercises the SRAM's ECC error detection mechanisms. Note that when MAP is activated, all SRAM accesses (including instruction fetches and stack operations) are affected, so it is advised to operate this feature from code resident outside SRAM, for example in Flash.	
		0	Normal Operation ECC parity generated on writes and deposited in array with data; corrected data returned on reads.
		2	Map Data ECC generation is bypassed; writes are stored to the 32-bit data only, prior stored ECC is not updated. Can be used to deposit data indicating single- or multiple-bit errors. On reads, ECC checking is disabled, and the raw 32-bit array data is returned without correction; no errors or warnings are generated. (Note: access sizes 8, 16, and 32 bits are supported.)
		3	Map ECC ECC generation is bypassed; bits [6:0] of writes are stored to the ECC array; data array is not updated. Can be used to deposit ECC states indicating single- or multiple-bit errors. On reads, the ECC bits are returned on data bits [6:0] of the read data value. (Note: access sizes 8, 16, 32 are supported.)
14 (R/W)	EERRDMA	Enable ECC interrupt, DMA. The `M4P_SRAM_CFG.EERRDMA` bit enables the DMA ECC interrupt when set to 1.	
13 (R/W)	EERRCORE	Enable ECC interrupt, Core. The `M4P_SRAM_CFG.EERRCORE` bit enables the Core ECC interrupt when set to 1.	

Table 3-14: M4P_SRAM_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	POSTWR	Enable Posted System Writes. The `M4P_SRAM_CFG.POSTWR` bit controls the behavior of non-exclusive writes by the Cortex to system space (outside the M4P platform). This allows control of the trade-off between performance and precise error detection. For more information, see the "SRAM Posted System Writes (NormSysWrite versus PostSysWrite)" section.	
		0	Normal System Writes Non-exclusive Cortex writes to system space are performed with normal system fabric (bus interconnect) write transactions.
		1	Posted System Writes Non-exclusive Cortex writes to system space are posted.
11 (R/W)	SYNCCLK	Enable Sync Bypass. The `M4P_SRAM_CFG.SYNCCLK` bit configures the M4P block to treat the core clock and system clock as a synchronous clock, and bypass the clock synchronization, decreasing MMR access latency.	
3:0 (R/W)	CDBANKS	Code/Data Banks Partition. The `M4P_SRAM_CFG.CDBANKS` field controls the partitioning of the main SRAM resources between the CODE address range and the SRAM (DATA) address range. The `M4P_SRAM_CFG.CDBANKS` field indicates how many of the 32KByte Config Banks of SRAM are allocated to the CODE region; the remaining available banks are allocated to the SRAM (DATA) region. When the `M4P_SRAM_CFG.CDBANKS` field is changed, the contents of any banks which are deducted from a given region are lost. The contents of any banks which are added to a given region are undefined, but should be zeroed immediately for security purposes.	
		0	No CODE Config Banks: all assigned to Data
		1	1 CODE config Bank, 4 DATA config banks
		2	2 CODE Config Banks, 3 DATA config banks
		3	3 CODE config Banks, 2 DATA config banks
		4	4 CODE Config Banks, 1 DATA config bank
		5	5 CODE Config Banks, 0 DATA config banks

SRAM ECC Error Address (Core) Register

The M4P_SRAM_EEADDR_CORE register displays the status of ECC errors detected in the main SRAM resulting from transactions initiated by the Cortex M4 Core or auto-refresh logic. The register captures the address of the first ECC error(s) detected in any aligned 32-bit word since the last time the corresponding ECC error status was cleared (by writing a 1 to M4P_SRAM_EEADDR_CORE.EERR). If the core ECC error interrupts are enabled by M4P_SRAM_CFG.EERRCORE, and an ECC error is detected on a core interface or refresh access, then an SRAM_EEIRQ_CODE interrupt will be asserted.

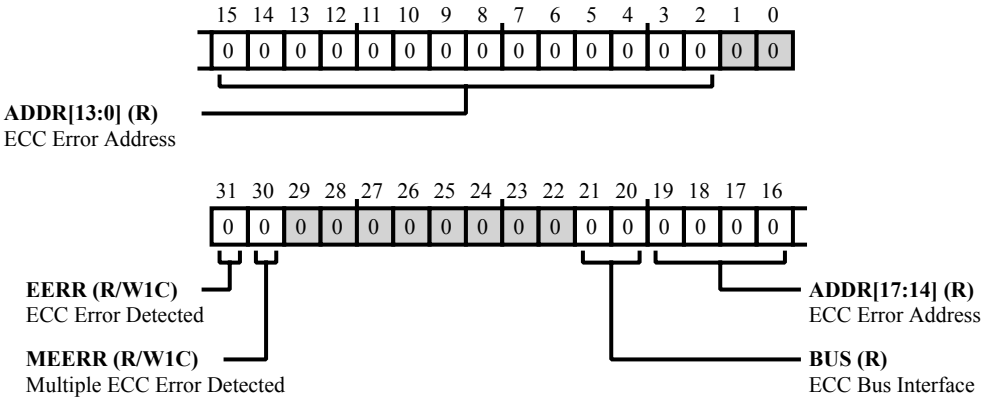


Figure 3-6: M4P_SRAM_EEADDR_CORE Register Diagram

Table 3-15: M4P_SRAM_EEADDR_CORE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	EERR	ECC Error Detected. The M4P_SRAM_EEADDR_CORE.EERR bit indicates one or more uncorrectable ECC errors has been detected.	
30 (R/W1C)	MEERR	Multiple ECC Error Detected. The M4P_SRAM_EEADDR_CORE.MEERR bit indicates more than one uncorrectable ECC error has been detected.	
21:20 (R/NW)	BUS	ECC Bus Interface. The M4P_SRAM_EEADDR_CORE.BUS bits indicate which Cortex bus initiated the access which caused detection of an ECC error.	
		0	ECC Error from I-CODE Interface
		1	ECC Error from D-Code Interface
		2	ECC Error from SYS Interface
19:2 (R/NW)	ADDR	ECC Error Address. The ADDR bit field holds the address offset of the most recently detected ECC error, relative to the base of the associated address space (CODE or DATA).	

SRAM ECC Error Address (DMA) Register

The M4P_SRAM_EEADDR_DMA register displays the status of EEC errors detected in the main SRAM resulting from transactions initiated by the DMA. The register captures the address of the first ECC error(s) detected in any aligned 32-bit word since the last time the corresponding ECC error status was cleared (by writing a 1 to M4P_SRAM_EEADDR_DMA.EERR). If DMA ECC error interrupts are enabled by M4P_SRAM_CFG.EERRDMA, and an ECC error is detected on the DMA interface, then an SRAM_EEIRQ_DMA interrupt will be asserted.

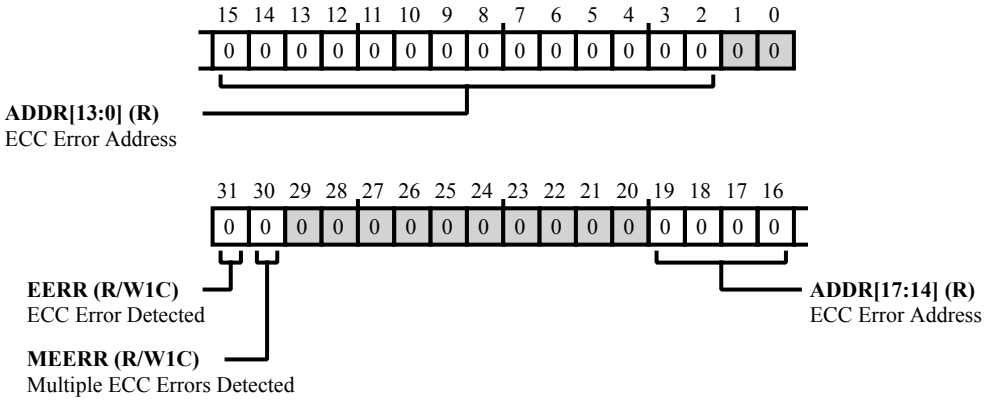


Figure 3-7: M4P_SRAM_EEADDR_DMA Register Diagram

Table 3-16: M4P_SRAM_EEADDR_DMA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	EERR	ECC Error Detected. The M4P_SRAM_EEADDR_DMA.EERR bit indicates that one or more uncorrectable ECC errors has been detected on accesses initiated by DMA.
30 (R/W1C)	MEERR	Multiple ECC Errors Detected. The M4P_SRAM_EEADDR_DMA.MEERR bit indicates that more than one uncorrectable ECC error has been detected on accesses initiated by DMA.
19:2 (R/NW)	ADDR	ECC Error Address. The ADDR bit field holds the address offset of the most recently detected ECC error, relative to the base of the associated address space (CODE or DATA).

SRAM Initialization Done Status Register

The M4P_SRAM_INITDONE register indicates the status of pending SRAM initialization.

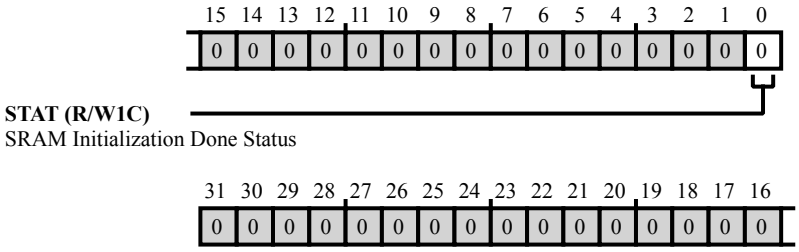


Figure 3-8: M4P_SRAM_INITDONE Register Diagram

Table 3-17: M4P_SRAM_INITDONE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1C)	STAT	SRAM Initialization Done Status. The M4P_SRAM_INITDONE.STAT bit indicates that the SRAM initialization is complete. Writing 0x1 to this register bit clears the status field to 0x0.

SRAM Refresh Address

The M4P_SRAM_RFRADDR register indicates the offset of the next memory address which will be accessed by the background memory refresh hardware. This is an offset relative to the base of Main SRAM's address space, without regard to the CODE/DATA partition.

Do not assign the refresh address outside the range of populated memory.

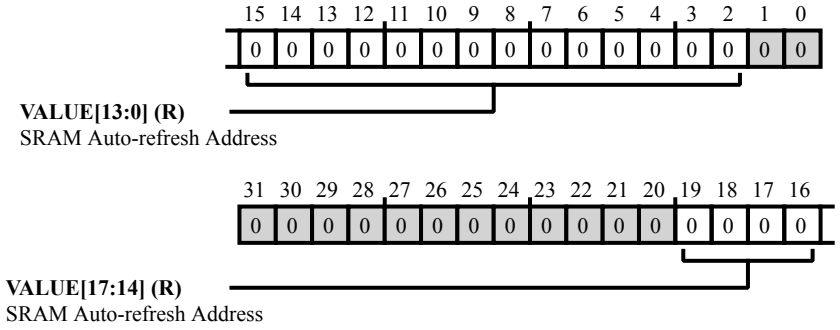


Figure 3-9: M4P_SRAM_RFRADDR Register Diagram

Table 3-18: M4P_SRAM_RFRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:2 (R/NW)	VALUE	SRAM Auto-refresh Address. The M4P_SRAM_RFRADDR.VALUE bits indicate the next address in main SRAM to be refreshed.

SRAM Refresh Period Register

The M4P_SRAM_RFRPER register controls the time period between ECC refresh access attempts.

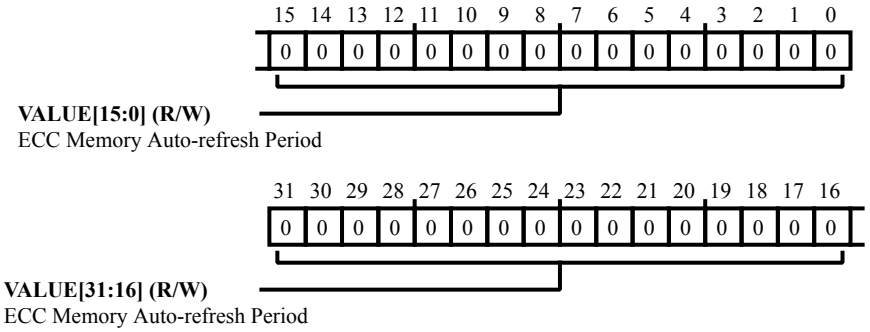


Figure 3-10: M4P_SRAM_RFRPER Register Diagram

Table 3-19: M4P_SRAM_RFRPER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	ECC Memory Auto-refresh Period. The M4P_SRAM_RFRPER.VALUE bits hold the 32-bit ECC refresh counter period. This is the number of SYSCLKs between successive requests for refresh, when refresh enable M4P_SRAM_CFG.RFR is set to 1. Minimum value is 64 (0x40); values less than 0x40 will be treated as 0x40. Maximum value is 0xFFFF_FFFF.

SysTick Calibration Register

The M4P_STCALIB register allows the programmer to define the calibration value for the ARM Cortex-M4 processor's SysTick timer, according to the frequency of the installed crystal and the divisor settings of the CGU and PLL. This allows the M4 to derive "real-time" from the system oscillator and pll values.

The value programmed into this register appears in the ARM SYST_CALIB read-only register. See the ARM Cortex-M4 manual for more information.

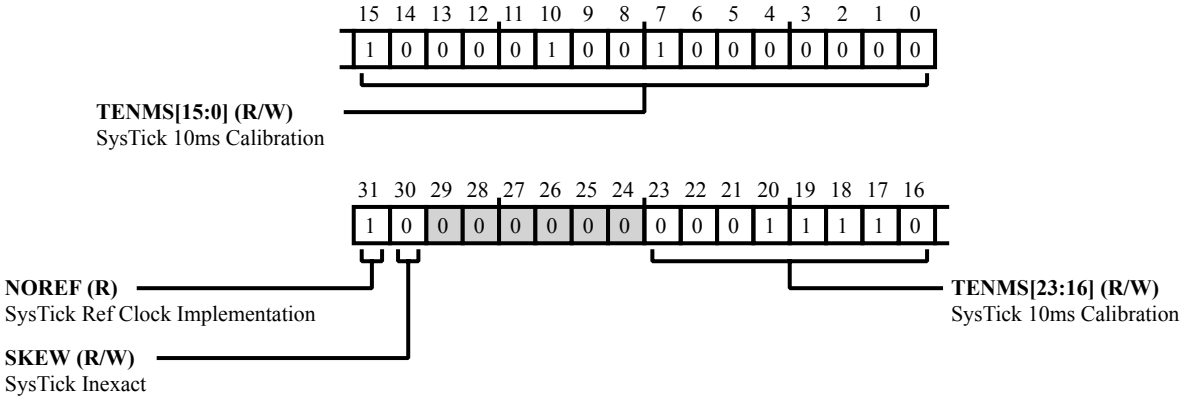


Figure 3-11: M4P_STCALIB Register Diagram

Table 3-20: M4P_STCALIB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/NW)	NOREF	SysTick Ref Clock Implementation. The M4P_STCALIB.NOREF indicates whether an external SysTick reference clock is implemented. In the ADI M4 Platform, no SysTick external reference clock is implemented. The processor internal reference clock is used for SYSTICK.	
		0	SYSTICK External Reference Clock is Implemented
		1	SYSTICK External Reference Clock is Not Implemented
30 (R/W)	SKEW	SysTick Inexact. The M4P_STCALIB.SKEW bit is set to 1 if the calibration value specified by M4P_STCALIB.TENMS does not provide an exact multiple of 10ms. Otherwise, set this bit to 0. For example, the frequency of a 166.66 (6 repeating) MHz core clock corresponds to a TENMS value of 1,666,666.66, which is not an exact integer. So, in that case, the M4P_STCALIB.SKEW bit would be set to 1.	
23:0 (R/W)	TENMS	SysTick 10ms Calibration. The M4P_STCALIB.TENMS bit is set to an integer 24-bit value usable to compute a 10ms delay from the user-programmed frequency of the M4P core clock. For example, for a 200MHz core clock, set this value to (200MHz * 10ms) = 24'd2_000_000 = 24'h1e_8480.	

4 Security

The ADSP-CM41x implements various user-controlled security features to protect the contents of the chip memory from unauthorized access.

Security Features

The security infrastructure provides the following features.

- Built-in support to protect processor memory content via Flash based passwords (Key).
- Protection from Flash reads and writes.
- The password (Key) can be updated via UART BOOT mode without using the debugger. This allows field upgrades of the Key as well as the application.
- The flash allows for physical separation of the user code area and the security areas of the flash memories.
- Programmable 128-bit bit lock on backdoor debug access.

Security Functional Description

The following sections provide information on the elements required to provide a secure operating environment.

Security States

The security states of the ADSP-CM41x are explained in this section. Below are the truth tables for access protection via a debugger and the UART. Before using the security features these states must be understood.

The features allow protection the part from un-authorized accesses from the debugger and UART boot connectivity paths. This is achieved via a 128-bit Secret User Key that is programmed in the Flash Info Blocks. The program needs to provide the correct/known key in order to access the chip memory via the debugger OR via UART boot. A Contrast Key of exact value (refer to Information Memory) must be in place to Lock the part correctly, along with the User Key.

NOTE: Refer to the Boot Chapter for information on how use the UART boot.

There are ID fields required for debuggers to identify the part and the debugger needs to be able to clear the flash. The following registers available to the debugger even when security is locked down are described in Debugger Access to a Locked Part.

Table 4-1: Truth Table for Security State for Access via Debugger

Security State	Flash Password (User Key)	User Key match via JTAG and SW Debug	Access via DEBUGGER
Unlocked_UNINIT_KEY	Uninitialized Key (Contrast is not programmed). Default State from factory	X	Yes • Debugger Connects • Debugger can access entire memory, with restrictions on Flash. • Flash firewalled: Only Contrast Key can be programmed
Unlocked_ BLANK_KEY	Blank Key (Contrast Key is programmed)	X	Yes • Debugger can access entire memory • No restrictions on Flash.
Unlocked_ ECC_FAIL_KEY	Invalid Key (ECC fail on User / Contrast)	X	Yes • Debugger Connects • Debugger can access entire memory, restrictions on Flash. • Flash firewalled. • Mass Erase
Locked	Programmed Key	1	Yes • Debugger can access entire memory • No restrictions on Flash.
Locked	Programmed Key	0	No • Debugger completely restricted.

Table 4-2: Truth table for Security State for access via UART BOOT

Security State	Flash Password (User Key)	User Key match via UART	Commands via UART
Unlocked_UNINIT_KEY	Uninitialized Key (Contrast is not programmed). Default State from factory	X	• Mass Erase • Reset • Write Contrast Key

Table 4-2: Truth table for Security State for access via UART BOOT (Continued)

Security State	Flash Password (User Key)	User Key match via UART	Commands via UART
Unlocked_ BLANK_KEY	Blank Key (Contrast Key is programmed)	X	• All Commands
Unlocked_ ECC_FAIL_KEY	Invalid Key (ECC fail)	X	• Mass Erase • Reset
Locked	Programmed Key	1	• All Commands
Locked	Programmed Key	0	• Mass Erase • Reset

Unlocked_UNINIT_KEY

This is default status of the processor where the Flash memory is in an all erased state which includes the User code locations and the Info Blocks. In this state, both the Contrast Key and the User Key are not programmed.

The programmer needs to program the Contrast Key in this state so that the User Key can be programmed next. Flash is said to be in a state of Fire-wall and only the Contrast Key can be programmed.

Unlocked_BLANK_KEY

In this state, program the User Key so that the part can be locked. If the system requires full access via a debugger as well as the UART boot, this key should not be programmed and the state should be left open.

Unlocked_ECC_FAIL_KEY

In this state, an ECC error is detected and the program needs to Mass Erase the Flash including its Info Blocks locations. Once they are erased, the state reverts back to the Unlocked_UNINIT_KEY state.

Locked

In this state, the processor security state is said to be Locked, and neither a debugger nor a UART boot can access the chip memory without knowing the correct User key. For the processor to be in the Lock state, the program must perform a Reset operation (a System reset or a Hard reset) after programming the keys.

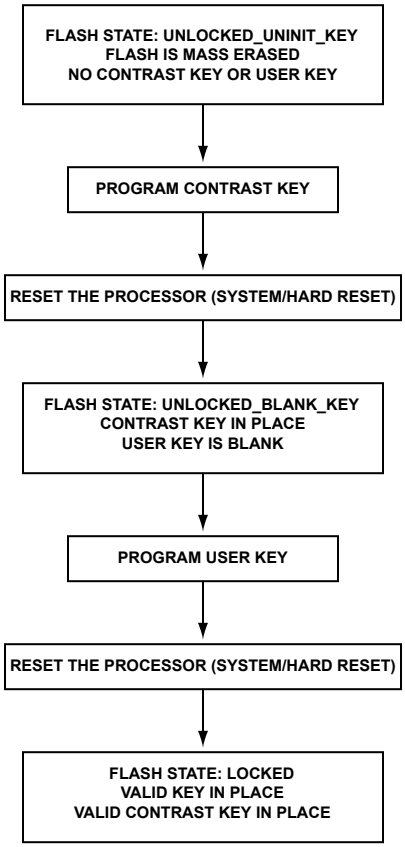


Figure 4-1: Security Enable Steps

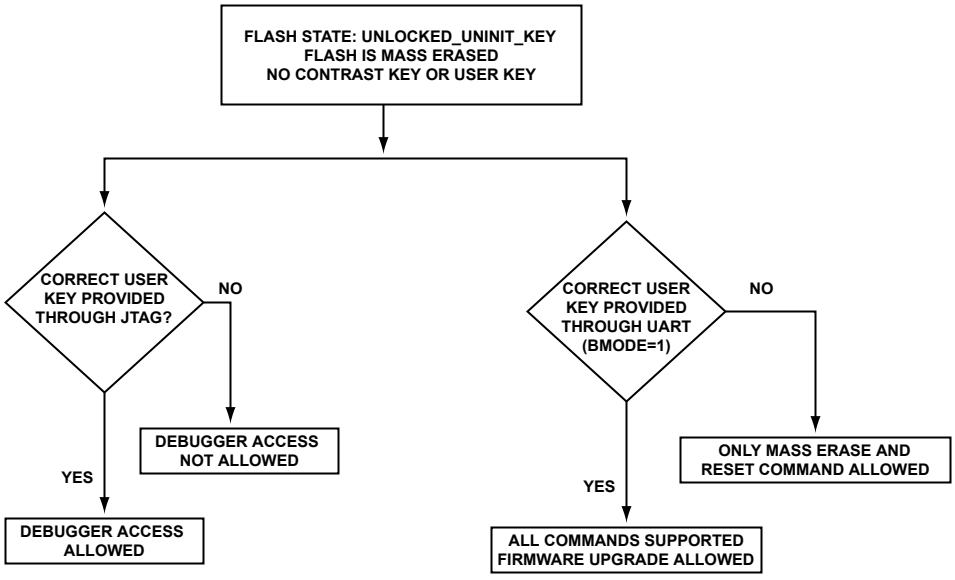


Figure 4-2: Security State Machine for Security Check

Architectural Concepts

Flash Info Block Structure

The following locations in the Flash Info Block store information required for security.

Table 4-3: Flash Info Block 0 Security Fields

Flash Block 0	Address Offset			
Address Base	0xC	0x8	0x4	0x0
0x1180_0FF0	SDBGKEY3	SDBGKEY2	SDBGKEY_CON_3	SDBGKEY_CON_2
0x1180_0FE0	SDBGKEY1	SDBGKEY0	SDBGKEY_CON_1	SDBGKEY_CON_0
0x1180_0FA0	SDBGKEYID3	SDBGKEYID2	SDBGKEYID1	SDBGKEYID0

Table 4-4: Flash Info Block 1 Security Fields

Flash Block 1	Address Offset			
Address Base	0xC	0x8	0x4	0x0
0x1188_0FF0	SDBGKEY3	SDBGKEY2	SDBGKEY_CON_3	SDBGKEY_CON_2
0x1188_0FE0	SDBGKEY1	SDBGKEY0	SDBGKEY_CON_1	SDBGKEY_CON_0

The SDBGKEY_CON0-3 fields in the Info Blocks contain the 128-bit Contrast Key. Note that both Flash Info Blocks 0 and 1 must have identical values programmed in their SDBGKEY_CON_n fields, there must be no difference between the values programmed in one Info Block and the other.

Table 4-5: Security Contrast Keys

Contrast Key Field	Value
SDBGKEY_CON0	0xAF57D151
SDBGKEY_CON1	0xE1F65DBA
SDBGKEY_CON2	0xD4CE466C
SDBGKEY_CON3	0xB0E87CD6

NOTE: The Contrast Key values should be exactly the same as above for the part to be in a proper secure state.

The SDBGKEY_CON0 through SDBGKEY_CON3 registers contain the 128-bit secure debug key. If this is not programmed (1s), the processor is unsecured (debug is unrestricted). If this key is programmed (not all 1s), the processor is secure and cannot be debugged without a matching key.

Note that both Flash Info Blocks 0 and 1 must have identical values programmed in their SDBGKEYn fields, there must be no difference between the values programmed in one Info Block and the other.

An SDBGKEY value of all 0’s could indicate a low-voltage attack or device failure. All 0's is treated by the hardware as an invalid key. It is recommended users have a significant number of 0’s and 1’s in a pseudo-random pattern throughout the 128 bit code for maximum protection.

The SDBGKEYID_CON0 through SDBGKEYID_CON3 registers contain the externally-visible 128-bit KEY ID. This contents of this field is user-defined and allows the user to identify or serialize an individual part, e.g. for association with a specific SDBGKEY.

The Logic ROM code that runs initially on a power-on reset, is tasked with copying the SDBGKEYn values from the flash info block to the TAPC_SDBGKEY0 through TAPC_SDBGKEY3 registers. To unlock debug access to the system, the correct SDBGKEYCMPn key of 128 bits needs to be written into the TAPC_SDBGKEYCMP0 through TAPC_SDBGKEYCMP3 registers. The TAPC_SDBGKEYCMPn register access can be performed through the JTAG scan and is usually accomplished when connecting a debugger to the part. The TAPC_SDBGKEY0 through TAPC_SDBGKEY3 and TAPC_SDBGKEYCMP0 through TAPC_SDBGKEYCMP3 registers are compared in the Logic ROM and access is given to debugger only if they match. The TAPC_SDBGKEY0 through TAPC_SDBGKEY3 registers are not readable by the debugger unless the keys are matched.

Flash Firewalled State

This attribute defines access control to the contents of the flash. The firewall does not prevent the flash from being mass-erased.

When firewalled, it is not possible to perform the following.

- Read the flash either by load-memory instructions or by MMR commands
- Fetch code (execute) from the flash (generates a Controlled Bus Fault Exception)
- Erase individual pages within the flash
- Write any locations within the flash, except the flash key contrast locations in the info blocks
- An Erase Reference Cell operation

When firewalled, it is possible to perform the following.

- A Mass Erase (with or without info block).
- A blind read of the entire flash memory space (no data returned) and detect any ECC errors within the flash arrays.

Debugger Access to a Locked Part

User debug security features either allow or restrict debug access. Debug security is independent of the flash firewall. Debug security mainly limits the debug access to MMR and memory space. It may also restrict other debug interface features (like JTAG pin boundary scan) that have system ramifications and are only intended for secure users. See the JTAG/SWD chapters for details.

Full debug access for unlocked part. Access is granted to the entire part outside the flash. The ability to access the flash is determined by the flash firewall state.

Restricted debug access for a locked part. Debug access to the portions of the chip outside of the flash is restricted to a short set of registers below. The ability to access the flash is determined by the flash firewall state. The registers consists of:

- The TAPC_SDBGKEYCMP0 through TAPC_SDBGKEYCMP3 input registers in the JTAG TAPC
- All JTAG Instruction and data registers
- The Flash controller registers required to perform a Mass Erase (all other functions which may be present in those registers are disabled except when full debug access is granted).

Debugger may NEVER access the internal debug key registers (JTAG) SDBGKEY(3-0)

Debugger may WRITE the following registers while "unsecure"

Module	Registers
JTAG	DBGKEY(3-0)
JTAG	RCMSG
M4-SYS	NVWR CTL
M4-LOCAL	FLASH(1-0) CTL
M4-LOCAL	FLASH(1-0) ADDR
M4-LOCAL	FLASH(1-0) STAT
M4-LOCAL	FLASH(1-0) CMDKEY
M4-LOCAL	FLASH(1-0) CMD (only flash command allowed is ERASE)
M4-LOCAL	FLASH0TIMPGM
M4-LOCAL	FLASH0TIMERASE
M4-LOCAL	FLASH0TIMNVSTR

Debugger may READ the following registers while "unsecure"

Module	Registers
JTAG	SDBGKEY_STAT
JTAG	SDBGKEY(3-0)
JTAG	IDCODE
JTAG	USERCODE
JTAG	RCMSG
M4-SYS	NVWR CTL
M4-SYS	CHIPID
M4-SYS	ADIID
M4-SYS	LROM_STAT

Module	Registers
M4-LOCAL	FLASH(1-0) CTL
M4-LOCAL	FLASH(1-0) ADDR
M4-LOCAL	FLASH(1-0) STAT
M4-LOCAL	FLASH(1-0) CMDKEY
M4-LOCAL	FLASH(1-0) CMD (only flash command allowed is ERASE)
M4-LOCAL	FLASH0TIMPGM
M4-LOCAL	FLASH0TIMERASE
M4-LOCAL	FLASH0TIMNVSTR
M4-PPB	Entire DBGROM, with all debug ID information

NOTE: If the debug attachment is through the ARM Cortex-M0 instead of the ARM Cortex-M4, the flash control registers may not be accessed.

The ARM Cortex-M0 cannot physically access the ARM Cortex-M4 system MMR space where the flash control registers reside.

5 System Crossbars (SCB)

A modern system on a chip (SoCs) contains multi-cores, memory controllers, security blocks, and other high speed peripherals. As system integration increases, SoCs need to provide bus connectivity that allows better throughput to reduce performance bottlenecks. While traditional point-to-point connection buses have performed well in smaller systems, there is a need to use advanced switching based bus architectures for efficient handling of data transfer between multiplicity of data sources and sinks in the system. Additionally, mixing various traffic types in the same SoC (for example control, communication over peripherals and computing) while sharing the same bus resources, create different requirements from the Quality of Service (QoS) perspective.

The system crossbars (SCB) are the fundamental building blocks of a switch-fabric style for (on-chip) system bus interconnection. The SCBs connect system bus masters to system bus slaves, providing concurrent data transfer between multiple bus masters and multiple bus slaves. The SCB architecture addresses the challenges described above. The SCB provides sustainable throughput for simultaneous transactions in the system with configurable quality of service for each type of transaction (traffic) as required. A hierarchical model, built from multiple SCBs, provides a power and area efficient system interconnect, which satisfies the performance and flexibility requirements of a specific system.

SCB Features

The SCBs provide the following features:

- Efficient, pipelined bus transfer protocol for sustained throughput
- Full-duplex bus operation for flexibility and reduced latency
- Concurrent bus transfer support to allow multiple bus masters to access bus slaves simultaneously
- Simple priority-based QoS based arbitration model

SCB Functional Description

The following sections provide a functional description of the SCB.

- SCB Architectural Concepts

CM41X_M4 SCB Register List

The system cross bar (SCB), which is often referred to as the system interconnect fabric, is a collection of interconnection units connecting system masters to slave memory spaces. A set of registers govern SCB operations. For more information on SCB functionality, see the SCB register descriptions.

Table 5-1: CM41X_M4 SCB Register List

Name	Description
SCB_AFE0_RQOS	AFE0 Read Quality of Service Register
SCB_AFE0_WQOS	AFE0 Write Quality of Service Register
SCB_M4_RQOS	M4 Core Read Quality of Service Register
SCB_M4_WQOS	M4 Core Write Quality of Service Register
SCB_DMA0_RQOS	DMA0 Read Quality of Service Register
SCB_DMA0_WQOS	DMA0 Write Quality of Service Register
SCB_DMA1_RQOS	DMA1 Read Quality of Service Register
SCB_DMA1_WQOS	DMA1 Write Quality of Service Register
SCB_DMA2_RQOS	DMA2 Read Quality of Service Register
SCB_DMA2_WQOS	DMA2 Write Quality of Service Register
SCB_DMA3_RQOS	DMA3 Read Quality of Service Register
SCB_DMA3_WQOS	DMA3 Write Quality of Service Register
SCB_DMA4_RQOS	DMA4 Read Quality of Service Register
SCB_DMA4_WQOS	DMA4 Write Quality of Service Register
SCB_DMA5_RQOS	DMA5 Read Quality of Service Register
SCB_DMA5_WQOS	DMA5 Write Quality of Service Register
SCB_DMA6_RQOS	DMA6 Read Quality of Service Register
SCB_DMA6_WQOS	DMA6 Write Quality of Service Register
SCB_DMA7_RQOS	DMA7 Read Quality of Service Register
SCB_DMA7_WQOS	DMA7 Write Quality of Service Register
SCB_DMA8_RQOS	DMA8 Read Quality of Service Register
SCB_DMA8_WQOS	DMA8 Write Quality of Service Register
SCB_DMA9_RQOS	DMA9 Read Quality of Service Register
SCB_DMA9_WQOS	DMA9 Write Quality of Service Register
SCB_FFT0_RQOS	FFT0 Read Quality of Service Register
SCB_FFT0_WQOS	FFT0 Write Quality of Service Register
SCB_SINC0_RQOS	SINC0 Read Quality of Service Register

Table 5-1: CM41X_M4 SCB Register List (Continued)

Name	Description
SCB_SINC0_WQOS	SINC0 Write Quality of Service Register
SCB_M0_RQOS	M0 Core Read Quality of Service Register
SCB_M0_WQOS	M0 Core Write Quality of Service Register

SCB Architectural Concepts

This section describes the components of the SCB and the modules connected to it. The basic elements in the SCB are SCB masters, slaves master interfaces, and slave interfaces.

Masters

The controllers in the system that raise the data request in the form of a read/write transaction on the SCB are called masters. The system bus masters include peripheral Direct Memory Access (DMA) channels. These include the Serial Port (SPORT) DMA, and SPI DMAs, among others. Also included are the Memory-to-Memory DMA channels (MDMA), and the processor cores.

Slaves

Slaves are SCB connections that are responding to transfer requests. Slaves include MMR registers, memory units, and various peripherals depending upon individual configurations. Each system slave has its own latencies and response times.

MDMA Pairs

DMA 14 and 13 may operate in a paired master/slave configuration to efficiently transfer data between 2 memory regions, with optional CRC computation.

SCB Block Diagram

The SCB architectural model is shown in the ***SCB Overview*** figure. This figure shows a high level representation of a basic SCB connecting n Slaves to x Masters. A variable number of masters connect to a variable number of slaves in each SCB. In this example, all SIs connect to all MIs as indicated by the lines connecting them.

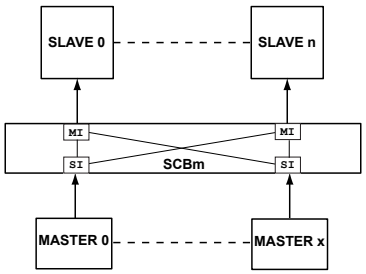


Figure 5-1: SCB Overview

Hierarchy Block Diagram

A system interconnect built from multiple SCBs in a hierarchical model is illustrated in the ***SCB Hierarchy Overview*** figure. The system master node level SCBs master connects multiple SIs to a single MI, which in turn connects to an SI of the system slave level node SCB.

As discussed above, all the masters in the system are distributed across different SCBs. A given SCB at system master node level connects directly to the system masters. These SCBs connect to SCB0 through its SIs forming a hierarchal structure. While a master has to access any slave, its first access goes through the SCB it is connected to, and then through SCB0, to reach its intended slave. This simplifies the connecting hardware in the basic SCB block by limiting the masters. Care must be taken when sharing masters to allow adequate throughput for their individual data transfer requirements.

In this example, all SIs are connected to all MIs.

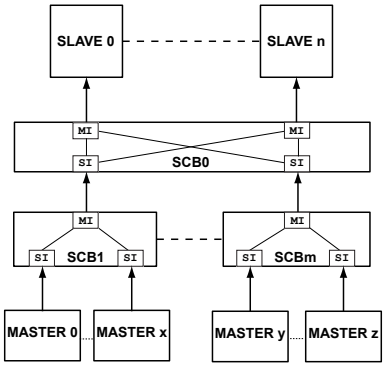


Figure 5-2: SCB Hierarchy Overview

NOTE: For an overall diagram of all SCB interconnections, see the ADSP-CM41x Block Diagram.

ADSP-CM41x Block Diagram

The ***ADSP-CM41x SCB Block Diagram*** figure shows the SCB block diagram for the ADSP-CM41x processors. The line colors are provided to better indentify block connections.

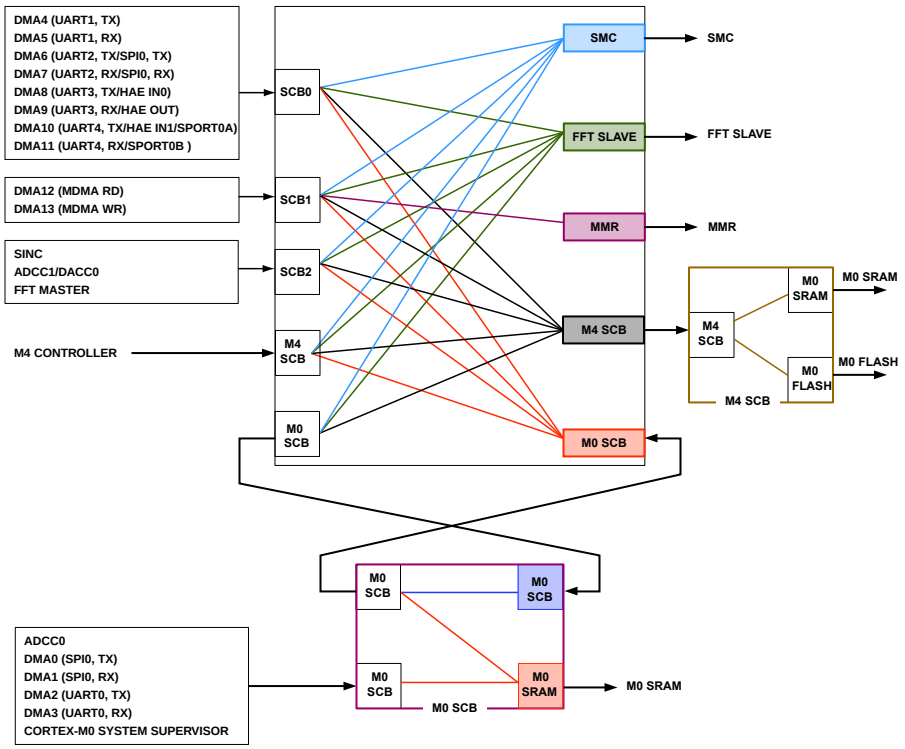


Figure 5-3: ADSP-CM41x SCB Block Diagram

While this figure is useful just for the overview it provides, it is also useful to observe the following relationships that are highlighted.

- The hierarchy of SCBs manages system bus interconnections, multiplexing, and arbitration among the cores and peripherals on the processor.
- The SCBs connections support DMA channels for some peripherals and support dedicated connections for others. The ADCC, SINC, DACC and FFT have their own DMA engines.
- The peripherals (and their SCBs) are in the SCLK clock domain. SCB0 is in the SYSCLK domain. The processor core is the CCLK clock domain. Synchronizations across clock domains affect SCB performance.
- Each peripheral has a latency for access across the SCB. The latency varies with the nature of the peripheral. Also, the number of active peripherals (especially for cases where multiple peripherals are active on a shared SCB) affects SCB performance.

The following are definitions of some of the acronyms that appear in the figure:

DMA0-DMA13

Indicate DMA channels for peripherals supporting DMA transfers.

SCB1 – SCB3, M0SCB, M4SCB

Indicate SCB interfaces, connecting the system bus masters and slaves.

SCLK0, SYSCLK, CCLK

Indicate clock domains in which the specific SCBs operate. For more information on clock domains, see the Clock Generation Unit (CGU) chapter and the product data sheet.

L1

Indicates the on-core (L1) internal memory.

SMC

Indicates the static memory controller (SMC) interface.

MMR

Indicates the system memory-mapped register interface.

SCB Bus Master IDs

The SCB bus master ID tables indicate which masters are connected to each of the slave ports of the SCBs. The following tables indicate the precise value of the ID as seen by the slave. These values are useful for SWU programming. The masters listed in the tables correspond to the QoS registers mentioned in the ADSP-CM41x SCB Register List table in their correct order.

- MASTER: CONT_MST is the ARM Cortex-M4 controller as memory access master and controls load/stores by the ARM Cortex-M4 core.
- MASTER: SUPER_MST is the ARM Cortex-M0 supervisor subsystem. This acts as the memory access master into the main system fabric and includes accesses by the ARM Cortex-M0 core as well as by DMA channels DMA0 through DMA3. The bits 'BBBB' indicate the master ID within the ARM Cortex-M0 subsystem, from the ***System Master IDs for SMPU0*** table.
- MASTER: SUPER_SLV is the ARM Cortex-M0 supervisor subsystem's slave port from the ARM Cortex-M4 controller subsystem fabric. The bits 'CCCCCCCCC' indicate the master ID within the ARM Cortex-M4 subsystem, from the ***System Master IDs for SMPU2 and SMPU1*** table.
- SLAVE: CONT_SLV is the ARM Cortex-M4 controller slave memory spaces which are ARM Cortex-M4 SRAM and flash.
- SLAVE: SUPER_SRAM is the ARM Cortex-M0 supervisor slave memory space. The ARM Cortex-M0 SRAM is the only such space.

The characters A, B, C and D are used in the following tables and have the following meanings.

- A – From DMA engine
- B – 4 bottom bits of 12-bit CONT_MST field
- C – All 9 bits of SUPER_SLV field

- D – All 9 bits of CONT_SLV field

Table 5-2: ADSP-CM41x SCB0, SCB1, SCB2 Bus Master IDs

	SLAVES				
MASTERS	**SMC**	**FFT**	**MMR**	**CONT_SLV**	**SUPER_SLV**
DMA4: UART1_TX	9b00000A000	9b00000A000	N/A	9b00000A000	9b00000A000
DMA5: UART1_RX	9b00000A001	9b00000A001	N/A	9b00000A001	9b00000A001
DMA6: SPI1_TX/UART2_TX	9b00000A010	9b00000A010	9b00000A010	9b00000A010	9b00000A010
DMA7: SPI1_RX/UART2_RX	9b00000A011	9b00000A011	9b00000A011	9b00000A011	9b00000A011
DMA8: HAE_IN0/UART3_TX	9b00000A100	9b00000A100	9b00000A100	9b00000A100	9b00000A100
DMA9: HAE_OUT/ UART3_RX	9b00000A101	9b00000A101	9b00000A101	9b00000A101	9b00000A101
DMA10: HAE_IN1/SPORT0A/ UART4_TX	9b00000A110	9b00000A110	9b00000A110	9b00000A110	9b00000A110
DMA11: SPORT0B/ UART4_RX	9b00000A111	9b00000A111	9b00000A111	9b00000A111	9b00000A111
DMA12: MDMA0_RD	9b01000A000	9b01000A000	9b01000A000	9b01000A000	9b01000A000
DMA13: MDMA0_WR	9b01000A001	9b01000A001	YES	9b01000A001	9b01000A001
SINC	9b10000A000	9b10000A000	N/A	9b10000A000	9b10000A000
AFE1: ADCC1 + DACC0	9b10000A001	9b10000A001	N/A	9b10000A001	9b10000A001
FFTB	9b10000A010	9b10000A010	N/A	9b10000A010	9b10000A010
CONT_MST	9b110000011	9b110000011	N/A	N/A	9b110000011
SUPER_MST	9b11BBBB100	9b11BBBB100	N/A	9b11BBBB100	N/A

Table 5-3: ADSP-CM41x M0 SCB Bus Master IDs

Master	**CONT_MST**	**SUPER_SRAM**
DMA0: SPI0_TX	12b00000000A000	12b00000000A000
DMA1: SPI0_RX	12b00000000A001	12b00000000A001
DMA2: UART0_TX	12b00000000A010	12b00000000A010
DMA3: UART0_RX	12b00000000A011	12b00000000A011
AFE0: ADCC0	12b00000000A100	12b00000000A100
SUPER_SLV	12bCCCCCCCCCC101	12bCCCCCCCCCC101
M0_CORE	12b000000000110	N/A

Table 5-4: ADSP-CM41x M4 SCB Bus Master IDs

Master	CONT_SRAM	CONT_FLASH
CONT_SLV	9bDDDDDDDDD	NA-AHB

System Crossbars

The System Crossbars (SCB) are the fundamental building blocks of the system bus interconnect. The SCB (often referred to as the system interconnect fabric), is a collection of inter-connection units connecting system masters to slave memory spaces. The SCB connects one or more master devices to one or more memory-mapped slave devices. Each connected master can be a core that originates an SCB transaction, or a master interface of an upstream SCB cascaded interconnect. Each connected slave can be the final target of an SCB transaction or a slave interface of a downstream cascaded SCB interconnect (forming a hierarchy of SCBs).

Each SCB that has multiple masters and slaves share the total bandwidth of the SCB. (In a M:N configuration where M masters are connected to N slaves through the SCBx.)

The SCB provides separate channels for reads and writes. Read and write accesses through a given SCB do not share bandwidth. All the SCBs are 32 bits wide and run at SCLK speed, and can provide a bandwidth of up to 400 Mbytes per second for reads and writes separately (when SCLK = 100 MHz). Only SCB0, which is the major SCB in the SCB hierarchy, has the multiple paths between multiple master and slave interfaces.

The processor does not contain any high transfer rate external peripherals. There should be adequate bandwith for all peripherals. Programs should use care with MDMA priority settings; the wrong settings can use significant SCB resources during internal memory transfers. L1 memories are 32-bit ECC protected and all major busses are 32 bits wide. 32-bit transfers are the most efficient for use in the product. 16-bit and 8-bit transfers are fully supported, just use bandwidth less efficiently.

Since the SCBs support burst transfers, it is important to configure the requesting master appropriately to make best use of available SCB bandwidth. For a DMA master, choosing the appropriate `DMA_CFG.MSIZE` value, is important from both a functional and a performance perspective. The value in the `DMA_CFG.PSIZE` bit field determines the width of the peripheral bus in use. It can be configured to 1-byte, 2-bytes, or 4-bytes. The `DMA_CFG.MSIZE` value determines the actual size of the SCB bus in use. It also determines the minimum number of bytes which are transferred from or to memory corresponding to a single DMA request or grant. The transfer can be 1-, 2-, 4-, 8-, 16-, or 32-bytes. If the `DMA_CFG.MSIZE` value is greater than the SCB bus width, the SCB performs burst transfers according to the width defined in `DMA_CFG.MSIZE`. When `DMA_CFG.MSIZE` is less than the SCB bus width, bursting is not supported and partial bus use results.

Each of the SCB unit in the fabric consists of N Slave interfaces (MSTn). Each of these interfaces has controls for read quality of service, write quality of service, and functional mode. A subset of these matrices includes controls for IB (Interface Block) sync mode, and bus functional mode. For more details on IB, see the clock domain synchronization section.

ADSP-CM41x SCB Arbitration

The SCB uses QoS based arbitration to prioritize each slave-to-master interface (master interface) and each master-to-slave interface (slave interface). Each slave has a quality of service (QoS) programmable feature that affects arbitration. This allows to increase the priority of a specific master and allow the chances of it getting higher priority when multiple masters are requesting over same SCB.

The slave interface of the crossbar (where the masters such as DMA connect to) performs few functions and arbitration is one of them. There are programmable register to decide the QoS for each master. The programmable QoS registers can be viewed as being associated with SCBx. There is separate Quality of service control for read (RQoS) and write (WQoS). For example, the QoS registers for DMA0-5, SINC can be viewed as residing in SCB4. Whenever a transaction is received at DMA0, the programmed QoS value is associated with that transaction and is arbitrated with the rest of the masters at SCB4. Consider the following situation.

- At SCB1, masters (1, 2, and 3) have RQOS values of (6, 4, and 2, respectively).
- At SCB2, masters (4, 5, and 6) have RQOS values of (12, 13, and 1, respectively).
- Master 1 wins at SCB1, and master 5 wins at SCB2, as defined by each having the largest RQOS value for its respective group.
- In a perfect competition at SCB0 masters 4 and 5 had the highest overall RQOS values, and they compete for arbitration directly at SCB0. Because of the mini-SCBs, master 1 which has a much lower RQOS value, is able to win against master 4 and make it all the way to SCB0.

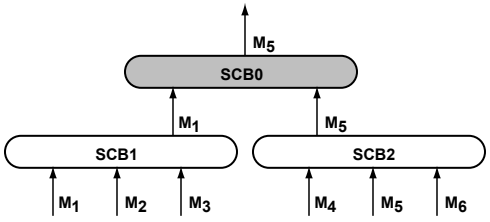


Figure 5-4: SCB Arbitration

NOTE: In most cases, the default RQOS/WQOS register values should be sufficient to achieve optimum throughput. For more information, see ADSP-CM41x SCB Programming Model.

Table 5-5: ADSP-CM41x Bus Master Reset QoS Values

MasterID	Master	Read_qos_reset_value	Write_qos_reset_value
0	DMA4: UART1_TX	1	1
1	DMA5: UART1_RX	1	1
2	DMA6: SPI1_TX/UART2_TX	1	1
3	DMA7: SPI1_RX/UART2_RX	1	1
4	DMA8: HAE_IN0/UART3_TX	1	1

Table 5-5: ADSP-CM41x Bus Master Reset QoS Values (Continued)

MasterID	Master	Read_qos_reset_value	Write_qos_reset_value
5	DMA9: HAE_OUT/UART3_RX	1	1
6	DMA10: HAE_IN1/SPORT0A/ UART4_TX	1	1
7	DMA11: SPORT0B/UART4_RX	1	1
8	DMA12: MDMA0_RD	0	0
9	DMA13: MDMA0_WR	0	0
10	ADCC1 + DACC0	1	1
11	FFTB	1	1
12	Controller MST	2	2
13	Supervisor MST	3	3
14	SINC	1	1

Clock Domain Synchronization

These registers perform clock domain crossing synchronization from CCLK to SCLK. The configuration of these registers depends on the CCLK and SCLK configuration.

ADSP-CM41x SCB Programming Model

As previously noted, the SCB arbitration model for the master and slave SCBs is QoS based. Each slave interface has a QoS value (or priority) associated with its read and write channels. These QoS selections are 4-bit values, which are present in the read QoS register and write QoS register register of each SCB master.

At the entry point to the infrastructure, all transactions are allocated a programmable, local QoS value. The arbitration of the transaction throughout the infrastructure uses this QoS. At any arbitration node, a fixed priority exists for transactions with a different QoS. The highest value has the highest priority. If there are coincident transactions at an arbitration node with the same QoS value that require arbitration, then the network uses a least recently used (LRU) algorithm. At each switch, the master with the highest QoS gains access, and that switch output takes the QoS value of the winner for that transaction. At the next switch slave interface, that master uses the QoS value of the winner.

SCB fabric registers occupy 1M byte of address space. The QoS registers in this space have values from 0 (lowest priority) to 15 (highest priority).

NOTE: Because the SCB arbitration of the processor is fixed (not programmable), these SCBs do not have slot numbers for modifying read/write arbitration settings.

NOTE: The MDMA is set to a default lower priority due to its potential for short term high demand. The processor SCB should have adequate bandwidth for configured system peripherals.

FIFO Synchronization

The FIFO associated with every channel is implemented to support clock domain crossing functionality.

The synchronization scheme used in the FIFO can be changed in the FIFO Sync_mode register. By default, FIFO is a pure asynchronous FIFO. Programming can reduce register access latency while CCLK:SYSCLK frequency ratio is n:1(n integer) or 1:1. Program the FIFO mode register to the respective values in the sync mode bits of the FIFO Sync mode register as follows:

- 0 = Sync 1:1
- 1 = Sync n:1
- 4 = async

Table 5-6: FIFO Sync Modes and Actions

Original Mode	Required Mode	Action
ASYNC	Sync 1:1 or n:1	Change the clocks, then change the register
Sync 1:1 or n:1	ASYNC	Change the register, then change the clocks

NOTE: This synchronization feature is applicable only for CCLK:SYSCLK ratios of 1:1 and n:1, but the feature is not applicable for ratios of m:n. For example, the synchronization feature is applicable for a CCLK:SYSCLK ratio of 200 MHz:100 MHz. This feature is not applicable for a ratio of 250 MHz:100 MHz.

CM41X_M4 SCB Register Descriptions

System Crossbar (SCB) contains the following registers.

Table 5-7: CM41X_M4 SCB Register List

Name	Description
SCB_AFE0_RQOS	AFE0 Read Quality of Service Register
SCB_AFE0_WQOS	AFE0 Write Quality of Service Register
SCB_M4_RQOS	M4 Core Read Quality of Service Register
SCB_M4_WQOS	M4 Core Write Quality of Service Register
SCB_DMA0_RQOS	DMA0 Read Quality of Service Register
SCB_DMA0_WQOS	DMA0 Write Quality of Service Register
SCB_DMA1_RQOS	DMA1 Read Quality of Service Register
SCB_DMA1_WQOS	DMA1 Write Quality of Service Register
SCB_DMA2_RQOS	DMA2 Read Quality of Service Register
SCB_DMA2_WQOS	DMA2 Write Quality of Service Register

Table 5-7: CM41X_M4 SCB Register List (Continued)

Name	Description
SCB_DMA3_RQOS	DMA3 Read Quality of Service Register
SCB_DMA3_WQOS	DMA3 Write Quality of Service Register
SCB_DMA4_RQOS	DMA4 Read Quality of Service Register
SCB_DMA4_WQOS	DMA4 Write Quality of Service Register
SCB_DMA5_RQOS	DMA5 Read Quality of Service Register
SCB_DMA5_WQOS	DMA5 Write Quality of Service Register
SCB_DMA6_RQOS	DMA6 Read Quality of Service Register
SCB_DMA6_WQOS	DMA6 Write Quality of Service Register
SCB_DMA7_RQOS	DMA7 Read Quality of Service Register
SCB_DMA7_WQOS	DMA7 Write Quality of Service Register
SCB_DMA8_RQOS	DMA8 Read Quality of Service Register
SCB_DMA8_WQOS	DMA8 Write Quality of Service Register
SCB_DMA9_RQOS	DMA9 Read Quality of Service Register
SCB_DMA9_WQOS	DMA9 Write Quality of Service Register
SCB_FFT0_RQOS	FFT0 Read Quality of Service Register
SCB_FFT0_WQOS	FFT0 Write Quality of Service Register
SCB_SINC0_RQOS	SINC0 Read Quality of Service Register
SCB_SINC0_WQOS	SINC0 Write Quality of Service Register
SCB_M0_RQOS	M0 Core Read Quality of Service Register
SCB_M0_WQOS	M0 Core Write Quality of Service Register

AFE0 Read Quality of Service Register

The SCB_AFE0_RQOS register selects the QoS value for AFE0 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

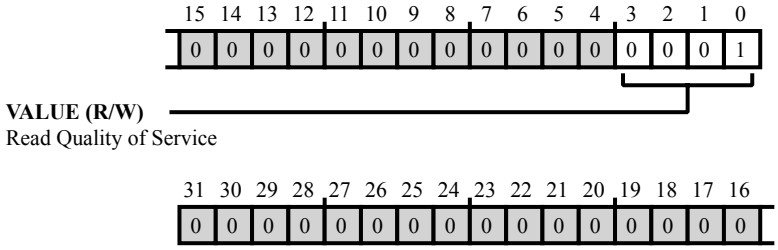


Figure 5-5: SCB_AFE0_RQOS Register Diagram

Table 5-8: SCB_AFE0_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_AFE0_RQOS.VALUE bit field holds the programmable QoS value for the AFE0 read channel.

AFE0 Write Quality of Service Register

The SCB_AFE0_WQOS register selects the QoS value for AFE0 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

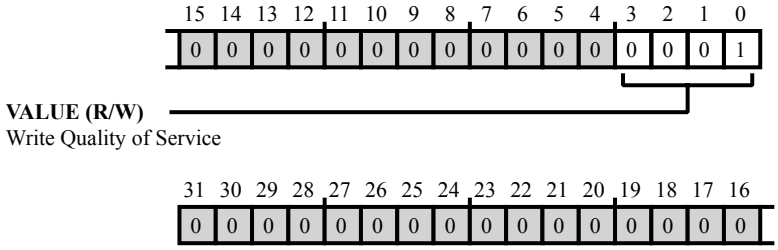


Figure 5-6: SCB_AFE0_WQOS Register Diagram

Table 5-9: SCB_AFE0_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_AFE0_WQOS.VALUE bit field holds the programmable QoS value for the AFE0 write channel.

M4 Core Read Quality of Service Register

The SCB_M4_RQOS register selects the QoS value for the Cortex-M4 core read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

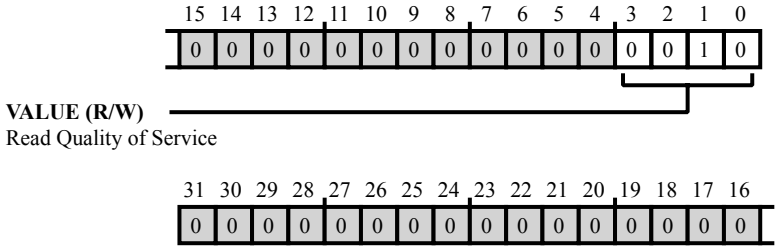


Figure 5-7: SCB_M4_RQOS Register Diagram

Table 5-10: SCB_M4_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service.

M4 Core Write Quality of Service Register

The SCB_M4_WQOS register selects the QoS value for the Cortex-M4 core write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

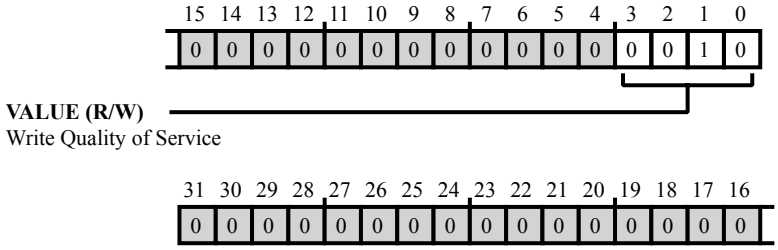


Figure 5-8: SCB_M4_WQOS Register Diagram

Table 5-11: SCB_M4_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service.

DMA0 Read Quality of Service Register

The SCB_DMA0_RQOS register selects the QoS value for the DMA2 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

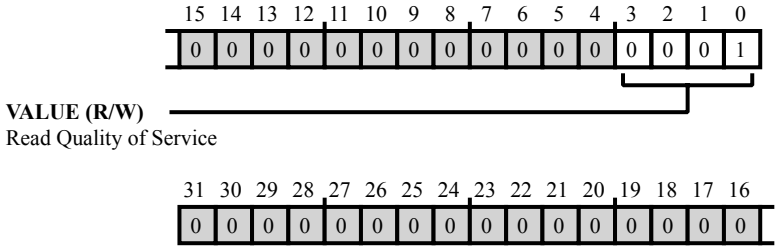


Figure 5-9: SCB_DMA0_RQOS Register Diagram

Table 5-12: SCB_DMA0_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA0_RQOS.VALUE bit field holds the programmable QoS value for the DMA0 read channel.

DMA0 Write Quality of Service Register

The SCB_DMA0_WQOS register selects the QoS value for the DMA0 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

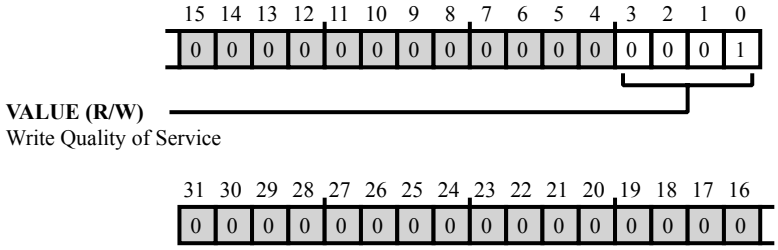


Figure 5-10: SCB_DMA0_WQOS Register Diagram

Table 5-13: SCB_DMA0_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA0_WQOS.VALUE bit field holds the programmable QoS value for the DMA0 write channel.

DMA1 Read Quality of Service Register

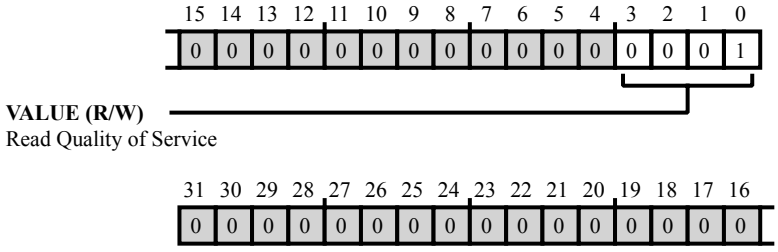


Figure 5-11: SCB_DMA1_RQOS Register Diagram

Table 5-14: SCB_DMA1_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The `SCB_DMA1_RQOS.VALUE` bit field holds the programmable QoS value for the DMA1 read channel.

DMA1 Write Quality of Service Register

The SCB_DMA1_WQOS register selects the QoS value for the DMA1 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

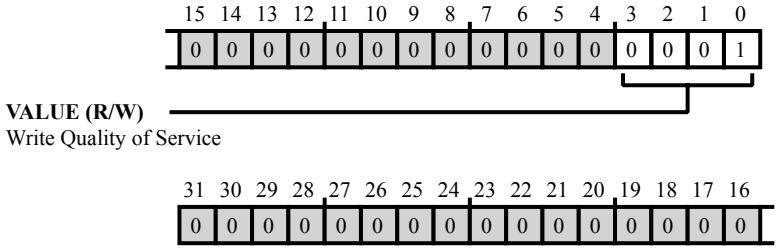


Figure 5-12: SCB_DMA1_WQOS Register Diagram

Table 5-15: SCB_DMA1_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA1_WQOS.VALUE bit field holds the programmable QoS value for the DMA1 write channel.

DMA2 Read Quality of Service Register

The SCB_DMA2_RQOS register selects the QoS value for the DMA2 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

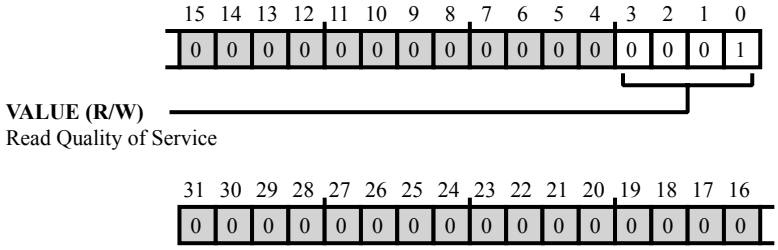


Figure 5-13: SCB_DMA2_RQOS Register Diagram

Table 5-16: SCB_DMA2_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA2_RQOS.VALUE bit field holds the programmable QoS value for the DMA2 read channel.

DMA2 Write Quality of Service Register

The SCB_DMA2_WQOS register selects the QoS value for the DMA2 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

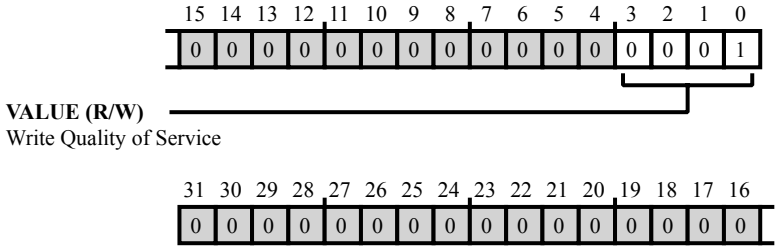


Figure 5-14: SCB_DMA2_WQOS Register Diagram

Table 5-17: SCB_DMA2_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA2_WQOS.VALUE bit field holds the programmable QoS value for the DMA2 write channel.

DMA3 Read Quality of Service Register

The SCB_DMA3_RQOS register selects the QoS value for the DMA3 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

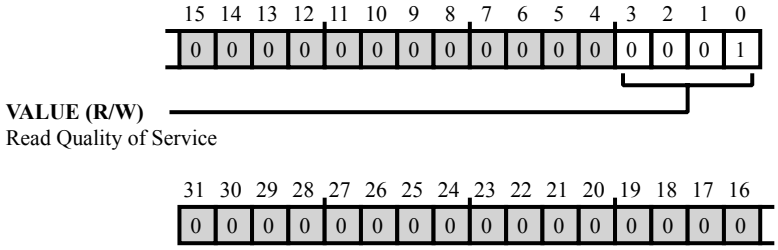


Figure 5-15: SCB_DMA3_RQOS Register Diagram

Table 5-18: SCB_DMA3_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA3_RQOS.VALUE bit field holds the programmable QoS value for the DMA3 read channel.

DMA3 Write Quality of Service Register

The SCB_DMA3_WQOS register selects the QoS value for the DMA3 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

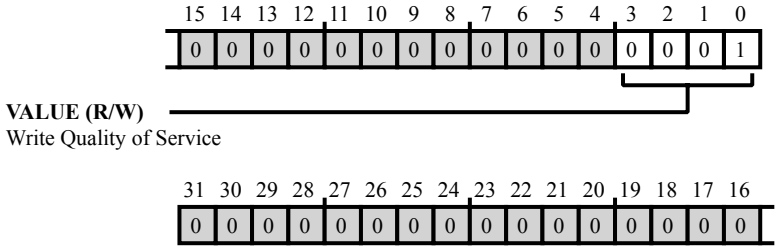


Figure 5-16: SCB_DMA3_WQOS Register Diagram

Table 5-19: SCB_DMA3_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA3_WQOS.VALUE bit field holds the programmable QoS value for the DMA3 write channel.

DMA4 Read Quality of Service Register

The SCB_DMA4_RQOS register selects the QoS value for the DMA4 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

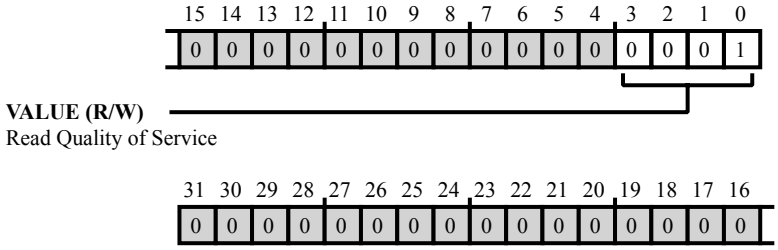


Figure 5-17: SCB_DMA4_RQOS Register Diagram

Table 5-20: SCB_DMA4_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA4_RQOS.VALUE bit field holds the programmable QoS value for the DMA4 read channel.

DMA4 Write Quality of Service Register

The SCB_DMA4_WQOS register selects the QoS value for the DMA4 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

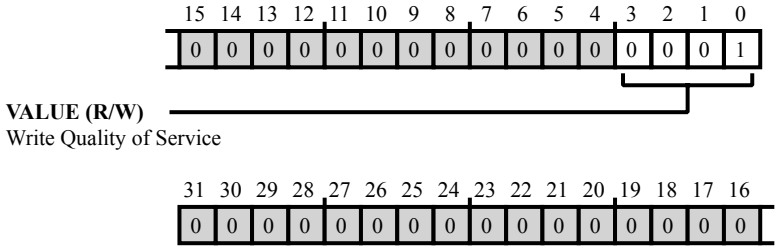


Figure 5-18: SCB_DMA4_WQOS Register Diagram

Table 5-21: SCB_DMA4_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA4_WQOS.VALUE bit field holds the programmable QoS value for the DMA4 write channel.

DMA5 Read Quality of Service Register

The SCB_DMA5_RQOS register selects the QoS value for the DMA5 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

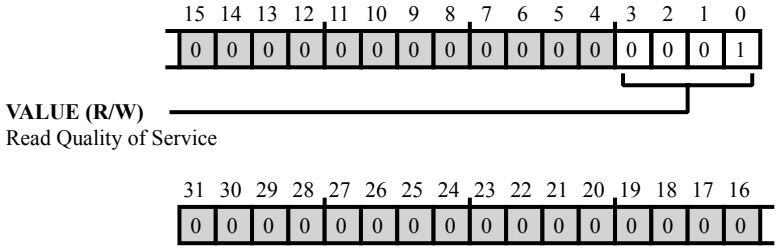


Figure 5-19: SCB_DMA5_RQOS Register Diagram

Table 5-22: SCB_DMA5_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA5_RQOS.VALUE bit field holds the programmable QoS value for the DMA5 read channel.

DMA5 Write Quality of Service Register

The SCB_DMA5_WQOS register selects the QoS value for the DMA5 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

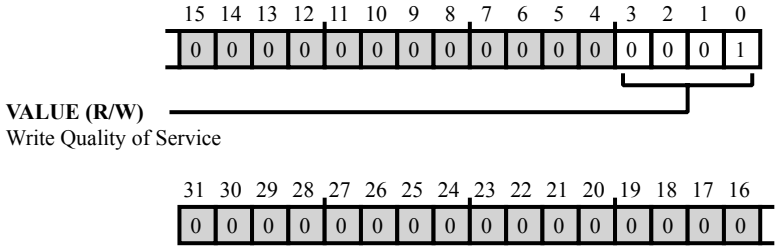


Figure 5-20: SCB_DMA5_WQOS Register Diagram

Table 5-23: SCB_DMA5_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA5_WQOS.VALUE bit field holds the programmable QoS value for the DMA5 write channel.

DMA6 Read Quality of Service Register

The SCB_DMA6_RQOS register selects the QoS value for the DMA6 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

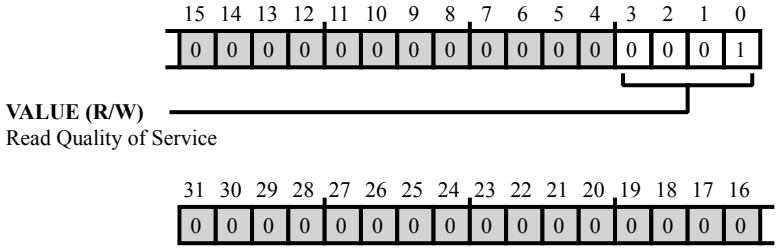


Figure 5-21: SCB_DMA6_RQOS Register Diagram

Table 5-24: SCB_DMA6_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA6_RQOS.VALUE bit field holds the programmable QoS value for the DMA6 read channel.

DMA6 Write Quality of Service Register

The SCB_DMA6_WQOS register selects the QoS value for the DMA6 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

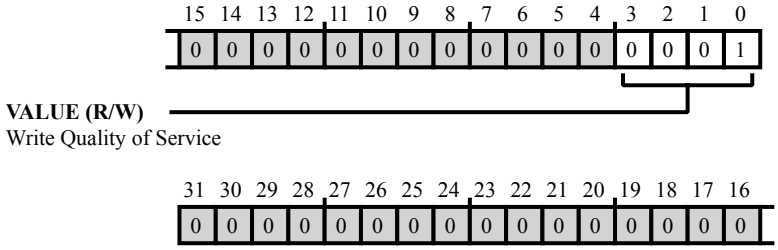


Figure 5-22: SCB_DMA6_WQOS Register Diagram

Table 5-25: SCB_DMA6_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA6_WQOS.VALUE bit field holds the programmable QoS value for the DMA6 write channel.

DMA7 Read Quality of Service Register

The SCB_DMA7_RQOS register selects the QoS value for the DMA7 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

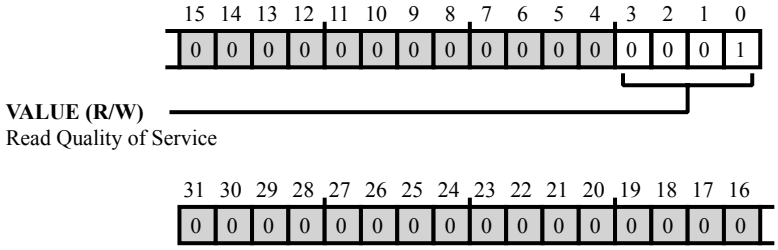


Figure 5-23: SCB_DMA7_RQOS Register Diagram

Table 5-26: SCB_DMA7_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA7_RQOS.VALUE bit field holds the programmable QoS value for the DMA7 read channel.

DMA7 Write Quality of Service Register

The SCB_DMA7_WQOS register selects the QoS value for the DMA7 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

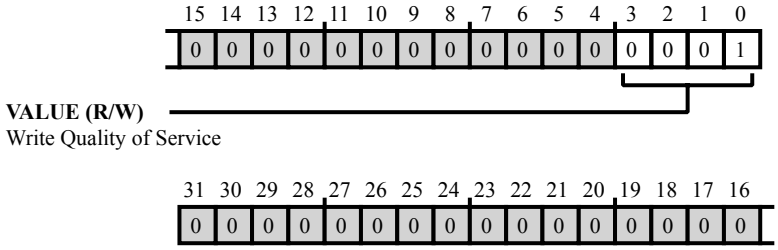


Figure 5-24: SCB_DMA7_WQOS Register Diagram

Table 5-27: SCB_DMA7_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA7_WQOS.VALUE bit field holds the programmable QoS value for the DMA7 write channel.

DMA8 Read Quality of Service Register

The SCB_DMA8_RQOS register selects the QoS value for the DMA8 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

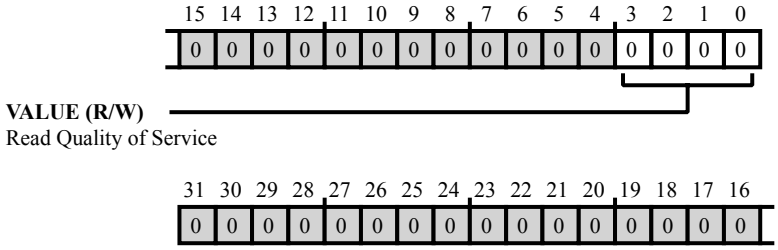


Figure 5-25: SCB_DMA8_RQOS Register Diagram

Table 5-28: SCB_DMA8_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA8_RQOS.VALUE bit field holds the programmable QoS value for the DMA8 read channel.

DMA8 Write Quality of Service Register

The SCB_DMA8_WQOS register selects the QoS value for the DMA8 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

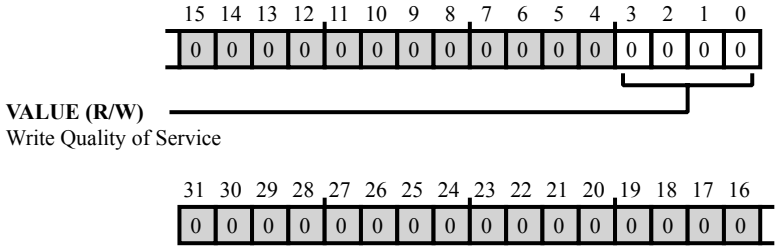


Figure 5-26: SCB_DMA8_WQOS Register Diagram

Table 5-29: SCB_DMA8_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA8_WQOS.VALUE bit field holds the programmable QoS value for the DMA8 write channel.

DMA9 Read Quality of Service Register

The SCB_DMA9_RQOS register selects the QoS value for the DMA9 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

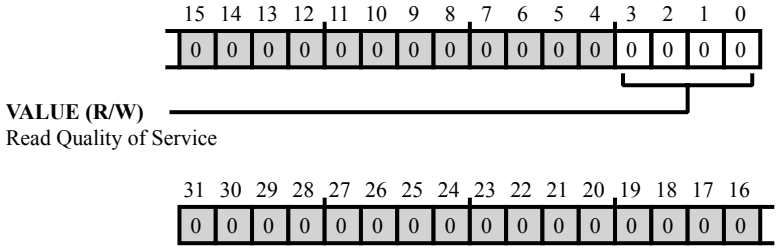


Figure 5-27: SCB_DMA9_RQOS Register Diagram

Table 5-30: SCB_DMA9_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_DMA9_RQOS.VALUE bit field holds the programmable QoS value for the DMA9 read channel.

DMA9 Write Quality of Service Register

The SCB_DMA9_WQOS register selects the QoS value for the DMA9 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

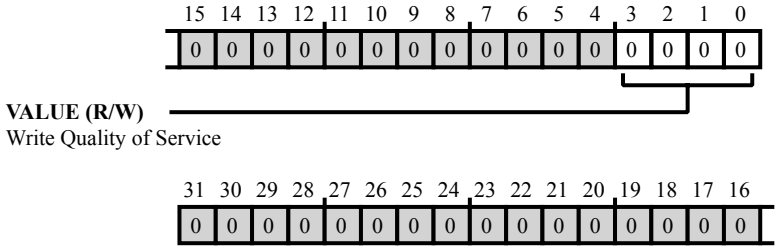


Figure 5-28: SCB_DMA9_WQOS Register Diagram

Table 5-31: SCB_DMA9_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_DMA9_WQOS.VALUE bit field holds the programmable QoS value for the DMA9 write channel.

FFT0 Read Quality of Service Register

The SCB_FFT0_RQOS register selects the QoS value for the FFTB0 read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

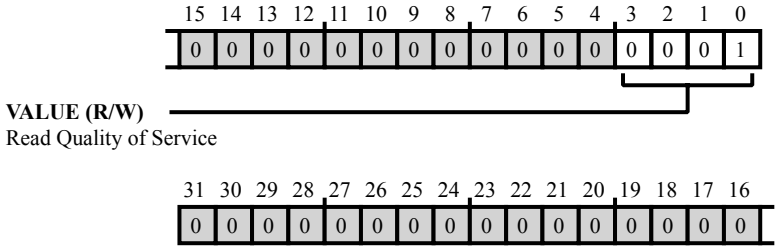


Figure 5-29: SCB_FFT0_RQOS Register Diagram

Table 5-32: SCB_FFT0_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_FFT0_RQOS.VALUE bit field holds the programmable QoS value for the FFT0 read channel.

FFT0 Write Quality of Service Register

The SCB_FFT0_WQOS register selects the QoS value for the FFTB0 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

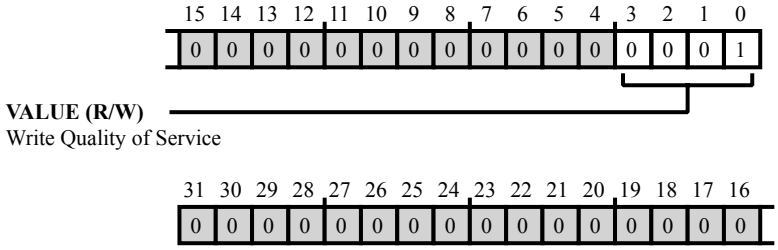


Figure 5-30: SCB_FFT0_WQOS Register Diagram

Table 5-33: SCB_FFT0_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_FFT0_WQOS.VALUE bit field holds the programmable QoS value for the FFT0 write channel.

SINC0 Read Quality of Service Register

The SCB_SINC0_RQOS register selects the QoS value for the SINC0 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

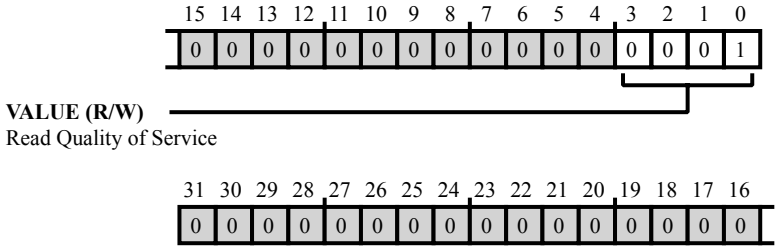


Figure 5-31: SCB_SINC0_RQOS Register Diagram

Table 5-34: SCB_SINC0_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service. The SCB_SINC0_RQOS.VALUE bit field holds the programmable QoS value for the SINC0 read channel.

SINC0 Write Quality of Service Register

The SCB_SINC0_WQOS register selects the QoS value for the SINC0 write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

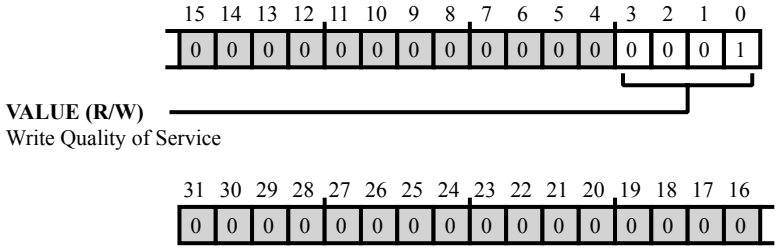


Figure 5-32: SCB_SINC0_WQOS Register Diagram

Table 5-35: SCB_SINC0_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service. The SCB_SINC0_WQOS.VALUE bit field holds the programmable QoS value for the SINC0 write channel.

M0 Core Read Quality of Service Register

The SCB_M0_RQOS register selects the QoS value for the Cortex-M0 core read channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

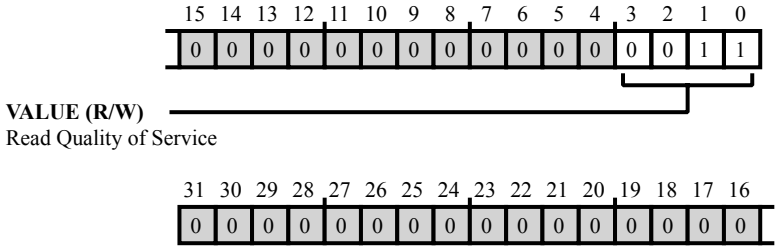


Figure 5-33: SCB_M0_RQOS Register Diagram

Table 5-36: SCB_M0_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read Quality of Service.

M0 Core Write Quality of Service Register

The SCB_M0_WQOS register selects the QoS value for the Cortex-M0 core write channel. The arbitration of transactions throughout the infrastructure uses this QoS value. At any arbitration node, the master with highest QoS value gains access.

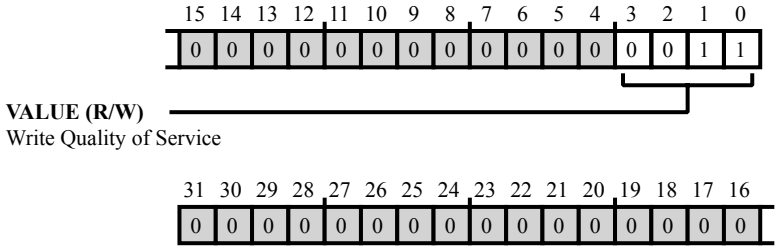


Figure 5-34: SCB_M0_WQOS Register Diagram

Table 5-37: SCB_M0_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write Quality of Service.

6 Clock Generation Unit (CGU)

The Clock Generation Unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a master clock that runs at a frequency that is a multiple of the CLKIN input clock frequency. The PCU divides down the master clock to generate various system clocks and synchronization signals.

NOTE: Unless otherwise noted, all references to CLKIN or SYS_CLKIN in this chapter correspond to the SYS_CLKIN0 pin on the processor.

CGU Features

The CGU module supports the following features:

- Provides smooth transitions from the current clock condition to a new condition with PLL logic and executes the changes to clocks due to register programming
- Provides PLL and clock domain status reporting for event management
- Supports the capability to bypass the PLL for power savings
- Manages power dynamically through software, allowing the dynamic control of the core clock frequency (f_{CCLK}) of the processor
- Controls clock gating of core and system clocks

NOTE: For more information about processor-specific CGU features, see the processor data sheet.

CGU Functional Description

The CGU generates all on-chip clocks and synchronization signals based on the programmed PLL multiplication factor and dividers. The CGU provides the following functionality.

Change the PLL Clock Frequency

The CGU allows programs to change the PLL clock frequency by writing new values to bits in the control register. Any time the PLL relocks, the CGU aligns all core and system clocks.

Change Other Clock Frequencies

The CGU allows programs to change the CCLKn, SYSCLK, SCLKn, DCLK, and OCLK frequencies by writing values to the CGU_DIV register. Any time the clock frequency is changed, the OCLK, CCLKn, SYSCLK, DCLK, and SCLKn clocks exit the frequency change sequence aligned.

Perform Clock Alignment

The CGU can align all clocks by writing to the CGU_DIV register. This function aligns all PLL-based clocks.

For more information on these functions, see the CGU Programming Model section.

CM41X_M4 CGU Register List

The clock generation unit (CGU) includes the phase locked loop (PLL) and the PLL control unit (PCU). The PLL generates a clock, running at a frequency that is a multiple of the CLKIN input clock's frequency. The CGU also generates all on-chip clocks and synchronization signals. The PCU permits application software control of the PLL's operation. A set of registers govern CGU operations. For more information on CGU functionality, see the CGU register descriptions.

Table 6-1: CM41X_M4 CGU Register List

Name	Description
CGU_CCBF_DIS	Core Clock Buffer Disable Register
CGU_CCBF_STAT	Core Clock Buffer Status Register
CGU_CLKOUTSEL	CLKOUT Select Register
CGU_CTL	Control Register
CGU_DIV	Clocks Divisor Register
CGU_OSCWDCTL	Oscillator Watchdog Register
CGU_PLLCTL	PLL Control Register
CGU_REVID	Revision ID Register
CGU_SCBF_DIS	System Clock Buffer Disable Register
CGU_SCBF_STAT	System Clock Buffer Status Register
CGU_STAT	Status Register
CGU_TSCOUNT0	Time Stamp Counter 32 LSB Register
CGU_TSCOUNT1	Time Stamp Counter 32 MSB Register
CGU_TSCTL	Time Stamp Control Register
CGU_TSVALUE0	Time Stamp Counter Initial 32 LSB Value Register
CGU_TSVALUE1	Time Stamp Counter Initial MSB Value Register

CM41X_M0 CGU Interrupt List

Table 6-2: CM41X_M0 CGU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
0	Reserved	Reserved	Reserved	Reserved
0	CGU0_OSCW_INT	CGU0 Oscillator Watchdog Interrupt	Level	
7	CGU0_EVT	CGU0 Event	Edge	

CM41X_M4 CGU Interrupt List

Table 6-3: CM41X_M4 CGU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
2	Reserved	Reserved	Reserved	Reserved
3	CGU0_OSCW_INT	CGU0 Oscillator Watchdog Interrupt	Level	
30	CGU0_EVT	CGU0 Event	Edge	

CM41X_M0 CGU Trigger List

Table 6-4: CM41X_M0 CGU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
3	CGU0_EVT	CGU0 Event	Edge

Table 6-5: CM41X_M0 CGU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

CM41X_M4 CGU Trigger List

Table 6-6: CM41X_M4 CGU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
3	CGU0_EVT	CGU0 Event	Edge

Table 6-7: CM41X_M4 CGU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

CGU Definitions

DPM

The dynamic power management (DPM) works with the CGU to provide flexible power dissipation modes for the processor.

PCU

The PLL control unit (PCU) in the CGU controls PLL operations. All the MMR registers of the CGU are implemented in this unit.

PLL

The phase-locked loop (PLL) operates within the CGU.

RCU

The reset control unit (RCU) provides input to the CGU to manage clocks during processor reset.

CGU

The clock generation unit (CGU) is comprised of the PLL and PCU.

CGU PLL Block Diagram

The ***CGU PLL Block Diagram*** provides a top-level block diagram of the phase locked loop (PLL). The main blocks of the PLL are the phase/frequency detector (PFD), the charge pump, the loop filter, and the voltage controlled oscillator (VCO). The VCO multiplies the `SYS_CLKIN0` input to a higher frequency.

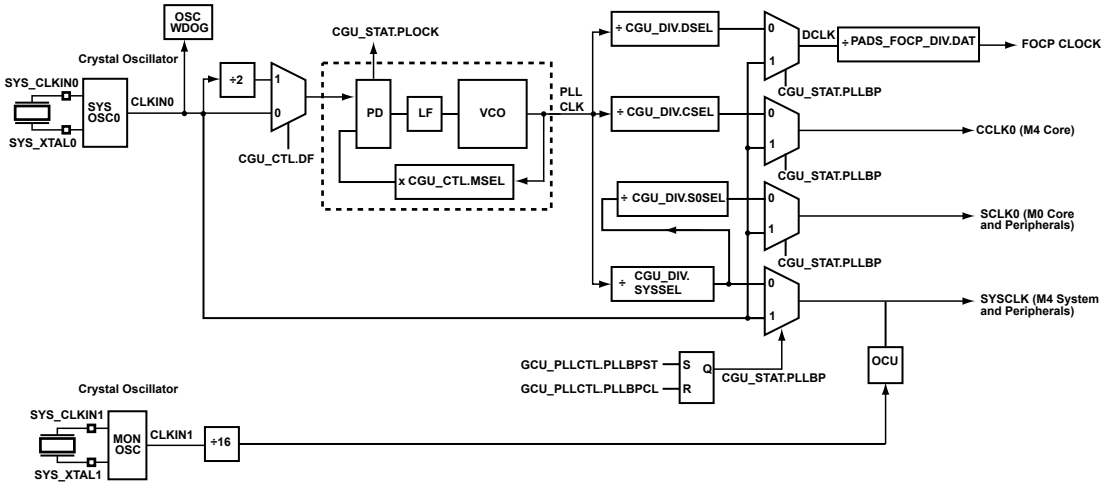


Figure 6-1: CGU PLL Block Diagram

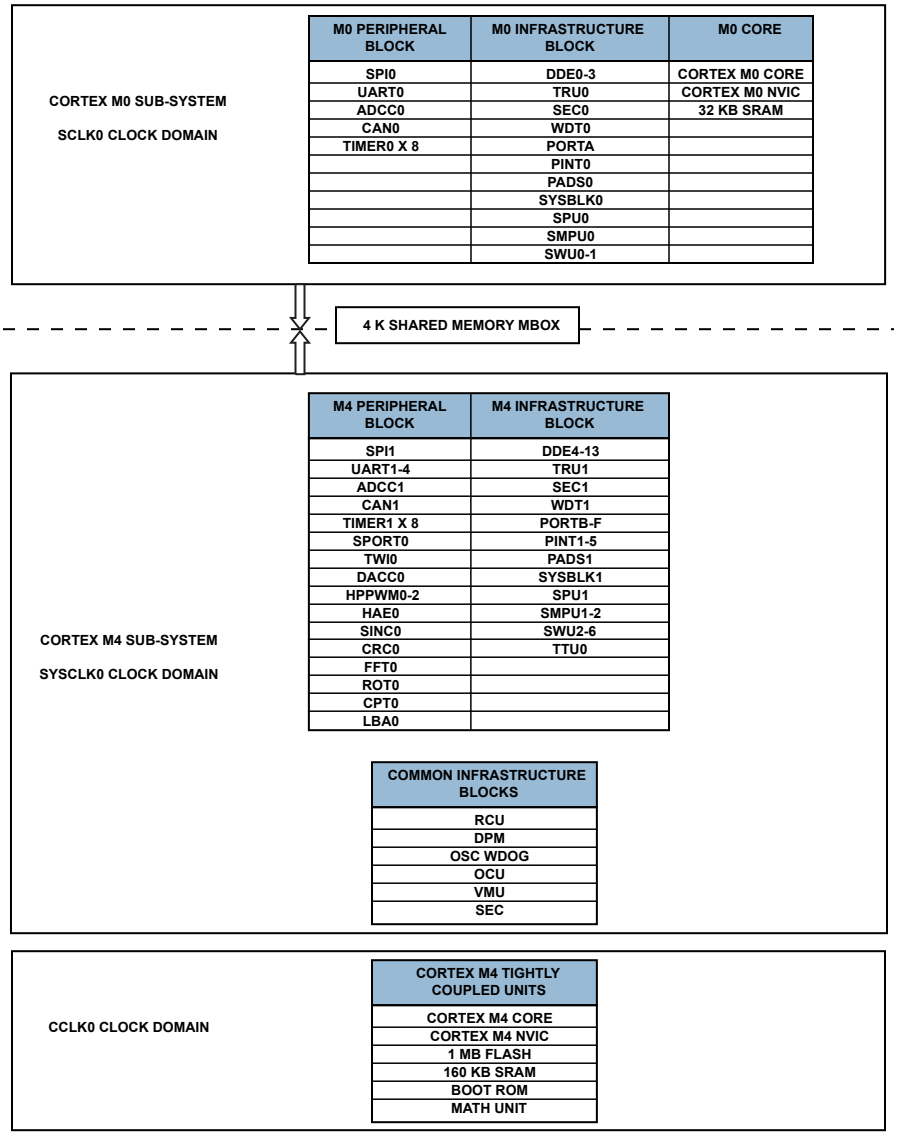


Figure 6-2: ADSP-CM41x Clock Domains (Detail)

The output of these blocks is called *PLLCLK*. The PLLCLK is divided to form CCLK, SYSCLK, FOCP_CLK and DCLK.

NOTE: On ADSP-CM41x processors, the SYSCLK signal is equivalent to the SCLKx signals mentioned throughout this book. The CCLKn signal is equivalent to the CCLK0 signal mentioned throughout this manual.

The ***SYS_CLKOUT Generation*** figure is a conceptual representation of the CLKOUT module. As shown in the ***CGU PLL Block Diagram***, many clocks that originate from the CGU block are available on the `SYS_CLKOUT` output pin. The `CGU_CLKOUTSEL.CLKOUTSEL` bit controls the clock outputs selection on the `SYS_CLKOUT` pin.

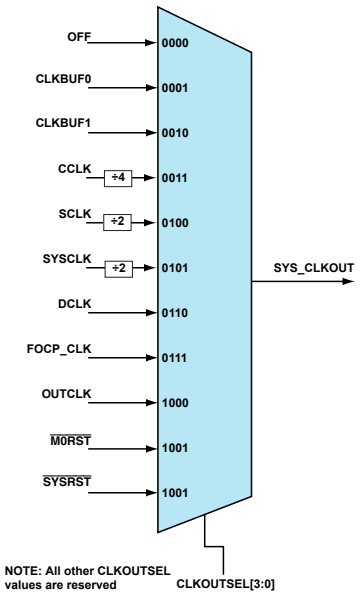


Figure 6-3: SYS_CLKOUT Generation

CGU Operating Modes

The CGU does not have configurable operating modes, but CGU operations affect the operating modes of other modules. Some CGU operation issues that affect operation of other modules include the following:

- The PLL of the CGU operates in either normal mode (CGU clock divisors applied) or bypass mode (CGU PLL is bypassed and clock divisors ignored).
- The SCB uses the CGU for clock synchronization across clock domains. For more information, see System Crossbars (SCB).
- The DPM uses the CGU for clock management as power state transitions occur. For more information, see the Dynamic Power Management (DPM) chapter.
- The CGU uses clock gating control to obtain flexible low-power modes.

CGU Power-up Sequence

See the product data sheet for exact power-up requirements. The processor is configured to come up in clock bypass mode. The programs is required to configure full speed clocks and safety monitors. CLKIN0 and all supplies should be stable before the `SYS_HWRST` signal is deasserted. CLKIN1 should be stable before operation of OCU safety unit.

CGU Event Control

The CGU generates an event or error for several different reasons.

CGU Events

After a frequency change, a CGU event indicates that the PLL has locked and clocks are synchronized. If a core was idled while changing frequencies, the CGU can use an event interrupt to break the core idle. While in active mode, a CGU event indicates that the PLL has locked.

CGU Errors

A CGU error occurs under following conditions:

- A write access to the CGU_DIV register triggers an alignment sequence while the PLL is locked and is still aligning the clocks.

  The CGU_STAT.WDIVERR bit state indicates this error. If this error occurs, clear the CGU_STAT.WDIVERR bit and rewrite the desired values to the CGU_DIV register.

- A change to the CGU_DIV register occurs while the PLL is locked and is still aligning the clocks

  The CGU_STAT.WDIVERR bit state indicates this error. If this error occurs, clear the CGU_STAT.WDIVERR bit and rewrite the desired values to the CGU_DIV register.

- A write access to the CGU_CTL.DF bit field occurs or a write access to the CGU_CTL.MSEL bit field occurs while the PLL is locking.

  The CGU_STAT.WDFMSERR bit state indicates this error. If this error occurs, wait until the PLL has finished locking, clear the error, and rewrite the desired value change.

- A clock divisor value error occurs when the *CCLK* divisor is greater than the *SYSCLK* divisor. For example, the CGU_DIV.CSEL is greater than CGU_DIV.SYSSEL.

  The CGU_STAT.WDIVERR bit state indicates this error. If this error occurs, clear it. The CGU writes new values to the CGU_DIV.CSEL bit field, so the field is less than or equal to the CGU_DIV.SYSSEL bit field value.

The CGU monitors changes to the following fields:

- CCLK Divisor - CGU_DIV.CSEL
- SYSCLK Divisor - CGU_DIV.SYSSEL

CGU Generated Bus Errors

The CGU generates a bus error when a read or write transaction is attempted to an unused address within the CGU address range. It also generates a bus error when a misaligned access is made to a CGU register. In addition to the bus error, the CGU_STAT.ADDRERR bit is set. If a write to a write-protected CGU register is attempted, the CGU generates a bus error. In addition, the CGU_STAT.LWERR bit is set.

Oscillator Watchdog

The oscillator watchdog detects the absence of input clock transitions and provides a fault warning through the SYS_FAULT pin. To detect harmonic or subharmonic crystal oscillator behavior, the watchdog (under programmable control) also detects and reports input oscillator frequencies above and below the specified limits. Use an internal asynchronous, local 1 MHz oscillator combined with a series of programmable counters for this detection. Set the CGU_OSCWDCTL.MONDIS bit and clear the CGU_OSCWDCTL.FAULTEN bit to optionally disable all the input clock monitor and fault detection functions.

Set the CGU_OSCWDCTL.HODEN bit to enable harmonic oscillation detection. The CGU uses the CGU_OSCWDCTL.HODF bit field to indicate the desired lower fail limit (in MHz) for the harmonic oscillation detection. The upper limit is always twice the lower limit.

The ***HODF Settings for Different Input Clock Frequencies*** table shows an example of the CGU_OSCWDCTL.HODF bit settings for different input clock frequencies.

Table 6-8: HODF Settings for Different Input Clock Frequencies

CGU_OSCWDCTL.HODF[5:0]	**Subharmonic Frequency (MHz)**	**Nom. Lower Fail Limit (MHz)**	**Input Clock Frequency (MHz)**	**Nom. Upper Fail Limit (MHz)**	**Second Harmonic Frequency (MHz)**
14	10	14	20	28	40
21	15	21	30	42	60

The CGU uses the CGU_OSCWDCTL.BOUF (Bad Oscillator Upper Frequency limit) asynchronous control bit field to indicate the desired upper fail limit for the bad oscillation detection. Set the CGU_OSCWDCTL.BOUEN bit to enable upper-limit bad oscillation detection. A bad oscillation detection condition signals a fault before any processor operations occur. This detection occurs (even in bypass mode) whenever a clock frequency exceeds the specifications.

The CGU_OSCWDCTL.BOUF =0 operation starts with a target of 32 MHz and each additional LSB increases the frequency test limit by 2 MHz. For example:

Target Upper Frequency Limit = CGU_OSCWDCTL.BOUF × 2 MHz + 32 MHz

The CGU_STAT.OSCWDSTATFC status bits indicate the nature of the fault.

The ***Fault Map*** table shows the fault values.

Table 6-9: Fault Map

CGU_STAT.OSCWDSTATFC Bitfield Values	**Fault Type**
0	No Fault
1	No Input Clock
2	Subharmonic CLKIN
3	Harmonic CLKIN
4	No AUX_CLK

Table 6-9: Fault Map (Continued)

CGU_STAT.OSCWDSTATFC Bitfield Values	**Fault Type**
5	CLKIN > Upper Freq Limit (BOUF)
6	Reserved
7	Multiple Limit Faults

There is a priority to the faults given in the case of multiple fault errors. The highest priority is given to No Input clock followed by No AUX_CLK. The other three fault cases share the lowest priority. Multiple limit faults are asserted if more than one type of subharmonic CLKIN, harmonic CLKIN, or BOUF faults are observed.

NOTE: All the CGU_STAT.OSCWDSTATFC faults other than the absence of AUX_CLK (for example, CGU_STAT.OSCWDSTATFC =4) are not reliable and used for debug only.

Program and enable the OSCWDOG to match the actual crystal, before bringing the PLL out of bypass.

Clock Not Good Reset

The CGU uses the CGU_OSCWDCTL.CNGEN bit to enable the detection of the clock fault. A CLK_NOT_GOOD internal signal is generated on a faulty clock condition detected by the Oscillator Watchdog. The CLK_NOT_GOOD signal can also be generated by the Oscillator Comparator Unit as well as through the Triggers TRGS_SYS_FRC_1MS_RESET_n trigger slave inputs. The CGU_OSCWDCTL.CNGEN bit has to be enabled in both of these cases.

The CLK_NOT_GOOD output signal is intended to be used as a backup safety mechanism for when a clock fault is detected and no system response to the fault occurs within approximately 1ms. The CLK_NOT_GOOD output flag is sticky. If the CLOCK_NOT_GOOD signal is generated (and resets the part) it remains in this state until a hardware pin reset occurs. While CLK_NOT_GOOD is in this sticky asserted state, then the Fault output also remains sticky, even if the fault condition somehow corrects itself as part of the system reset action caused by the CLK_NOT_GOOD signal going active.

It is assumed that the SOC using the oscillator function has two external pins to indicate a fault state (SYS_FAULT) and a reset state (SYS_RESOUT) of the SOC to the external world. If any CLKIN fault type occurs, then a SYS_HWRST SYS_HW_FAULT activates. If no action is taken externally in ~1ms, and the CGU_OSCWDCTL.CNGEN bit remains high, then the CLOCK_NOT_GOOD signal activates and resets the part. The SYS_RESOUT pin and that SYS_FAULT pin remain active until a hardware reset is applied to clear them and potentially attempt a reboot (assuming the fault condition has been corrected).

GPIO Pin Safe State

The Voltage Monitoring Unit (VMU) can be configured to assert the Pin Safe State on a clock fault detected by the OSCWDOG. The PADS_VMU_TRIPEN.OSC0FAULT and PADS_VMU_TRIPEN.OSC0CLK bits are used to enable the pin safe state trips for the OSCWDOG faults.

CGU Programming Model

The programming model for the CGU involves the various mode configuration techniques.

Configuring CGU Modes

Use the following procedures to configure the clocks and PLL.

NOTE: The program needs to clear the CGU_STAT.CLKSALGN bit before changing clocks. The following sequence is executed once, inside the application, after coming out of reset.

```
*pREG_CGU0_PLLCTL |= BITM_CGU_PLLCTL_PLLBPCL; // come out of bypass and enter
Full ON
while( (pADI_CGU0 ->STAT & 0xF) != 0x5 ) { } // poll
// now clocks are running with hardware default divisors.
// now program can change frequencies If desired the program can put the PLL
again into bypass.
```

Changing Clock Frequencies

Applications change clock frequencies in two ways. The first way is modifying the PLL multiplication value by writing to the CGU_CTL register and the second is modifying the clock dividers by writing to the CGU_DIV register. Both actions have different implications even if the frequencies of the final clock are the same. Write accesses to change the CGU_CTL.DF or CGU_CTL.MSEL bit fields while the PLL is locking set the CGU_STAT.WDFMSERR error bit. The CGU_STAT.WDIVERR error bit is set when one of following accesses is attempted while the PLL is locked, but still aligning the clocks:

- A write access to the CGU_DIV to trigger an alignment sequence.
- A write access to the CGU_DIV to change the CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, or CGU_DIV.DSEL bits.

Read-after-write accesses to these registers return the new value, even if the frequency of the clock change is still in-progress.

Modifying the PLL multiplier requires the PLL to relock. Once the PLL locks, the CGU synchronizes the clocks. Changes to the CGU_CTL.DF or CGU_CTL.MSEL bit field result in bypassing the PLL. By setting the CGU_CTL.WFI bit, programs force the PLL to wait for all the cores to return to their idle or reset states before the frequency changes . If necessary, clear the CGU_DIV.UPDT bit to avoid multiple clock alignment sequences. If the CGU_DIV register is not updated, the CGU uses the current values to determine the frequencies of the clock. It is the programs responsibility to guarantee that the new CGU_CTL.DF or CGU_CTL.MSEL and CGU_DIV combinations are legal.

Changing the PLL Clock Frequency

To change the phase-locked loop clock (*PLLCLK*) frequency, write new values to the CGU_CTL.MSEL field or CGU_CTL.DF field. Any time the PLL relocks, all core and system clocks are aligned.

1. Read CGU_STAT register and verify that:
    a. The CGU_STAT.PLLEN bit =1 (PLL enabled)
    b. The CGU_STAT.PLOCK bit =1 (PLL is not locking)
    c. The CGU_STAT.CLKSALGN bit =0 (clocks aligned)
2. Write the desired values to the clock divisor select fields of the CGU_DIV register with the CGU_DIV.UPDT bit =0.
3. Write the desired values to the CGU_CTL.DF and CGU_CTL.MSEL fields.
    a. To change the PLL frequency while the cores are idle, write to the CGU_CTL register with the CGU_CTL.WFI bit =1.
    b. To change the PLL frequency while the cores are active, write to the CGU_CTL register with the CGU_CTL.WFI bit =0.

This sequence performs these actions:

1. Updates the corresponding CGU registers.
2. Bypasses the PLL.
3. Makes the PLL lock to the new values in the CGU_CTL.MSEL or CGU_CTL.DF fields.
4. Changes the clock frequencies.
5. Exits the PLL bypass with all clocks aligned.

When exiting the PLL bypass state, a CGU event occurs.

The CGU_STAT register exits this sequence with the CGU_STAT.PLLEN bit =1, the CGU_STAT.PLOCK bit =1, the CGU_STAT.PLLBP bit =0, and the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.PLOCK bit, CGU_STAT.PLLBP bit, and CGU_STAT.CLKSALGN bit to discover when the PLL is locked and the clocks are aligned.

Changing the frequency of the PLL is allowed while the PLL is bypassed. In this case the new PLLCLK frequency is not used until the PLL is no longer bypassed.

Changing the CCLKn or SYSCLK Frequency Without Modifying the PLLCLK Frequency

To change the clock frequencies, write new values to CGU_DIV.CSEL or CGU_DIV.SYSSEL bits. The frequency change occurs only when the PLL is not bypassed. Any time the *CCLKn* or *SYSCLK* clock frequencies are changed, they exit the frequency change sequence aligned.

1. Read the CGU_STAT register to verify that the CGU_STAT.CLKSALGN bit =0 (clocks aligned).
2. Write the desired CGU_DIV.CSEL, CGU_DIV.SYSSEL, and CGU_DIV.OSEL bit field values with the CGU_DIV.UPDT bit = 1.

ADDITIONAL INFORMATION: This write updates the CGU_DIV register, changes the *SCLKn* and *SYSCLK* frequencies, and aligns the clocks. When the clocks are aligned, a CGU event occurs.

The CGU_STAT register exits this sequence with the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.CLKSALGN bit to discover when the clocks are aligned. Any write attempt to change the CGU_DIV.S0SEL or CGU_DIV.S1SEL bit fields while CGU_STAT.CLKSALGN bit =1 (clocks alignment in progress) triggers an MMR access bus error and the CGU_DIV register is not modified.

Programming the *SYSCLK* frequency to a higher value than *CCLKn* also triggers an MMR access bus error and the CGU_DIV register is not modified.

Changing the OCLK Frequency

To change the OCLK clock frequency, write a new CGU_DIV.OSEL bit value. Any time the OCLK clock frequency is changed, the OCLK, CCLKn, SYSCLK, and SCLKn clocks exit the frequency change sequence aligned.

1. Read the CGU_STAT register to verify that the CGU_STAT.CLKSALGN bit =0 (clocks aligned).
2. Write the desired CGU_DIV.OSEL value with the CGU_DIV.UPDT bit =1. This write updates the CGU_DIV register, changes the OCLK frequency, and aligns all clocks except OCLK.

The CGU_STAT register exits this sequence with the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.CLKSALGN bit to discover when the clocks are aligned. Any write attempt to change the CGU_DIV.DSEL field while the CGU_STAT.CLKSALGN bit =1 (clock alignment in progress) triggers an MMR access bus error and the CGU_DIV is not modified. When the clocks are aligned, a CGU event occurs.

Writing to the CGU_DIV.OSEL bit field is allowed while the processor is in active (PLL bypassed) mode. In this case the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Aligning All Clocks

To align the clocks, write 1 to the CGU_DIV.ALGN bit. The frequency can also be changed, if necessary. The clocks aligned include:

- CCLKn
- SYSCLK
- SCLKn
- DCLK
- OCLK

1. Read the CGU_STAT register to verify that CGU_STAT.CLKSALGN bit =0 (clocks aligned).
2. Write 1 to the CGU_DIV.ALGN bit. All other fields can change.

ADDITIONAL INFORMATION: This write does not alter the CGU_DIV register unless one of the clock-select fields is modified. When the clocks are aligned, a CGU event occurs.

The CGU_STAT register exits this sequence with the CGU_STAT.CLKSALGN bit =0. Poll the CGU_STAT.CLKSALGN bit to discover when the clocks are aligned. Any write to the CGU_DIV register intended to align clocks or to change a clock select field while the CGU_STAT.CLKSALGN bit =1 (clocks alignment in progress) triggers an MMR access bus error. In this case, the CGU_DIV register is not modified.

Writing 1 to the CGU_DIV.ALGN bit has no effect while the processor is in active (PLL bypassed) mode.

Valid Clock Multiplier Settings

Processor operations depend on valid settings in the CGU_CTL and CGU_DIV registers. These registers control the clock multiplier and divisor values. Set these registers such that the minimum and maximum clocks specified in the data sheet are not violated. All other clock specifications in the data sheet must also be adhered to for correct operation of the processor.

CM41X_M4 CGU Register Descriptions

Clock Generation Unit (CGU) contains the following registers.

Table 6-10: CM41X_M4 CGU Register List

Name	Description
CGU_CCBF_DIS	Core Clock Buffer Disable Register
CGU_CCBF_STAT	Core Clock Buffer Status Register
CGU_CLKOUTSEL	CLKOUT Select Register
CGU_CTL	Control Register
CGU_DIV	Clocks Divisor Register
CGU_OSCWDCTL	Oscillator Watchdog Register
CGU_PLLCTL	PLL Control Register
CGU_REVID	Revision ID Register
CGU_SCBF_DIS	System Clock Buffer Disable Register
CGU_SCBF_STAT	System Clock Buffer Status Register
CGU_STAT	Status Register
CGU_TSCOUNT0	Time Stamp Counter 32 LSB Register
CGU_TSCOUNT1	Time Stamp Counter 32 MSB Register
CGU_TSCTL	Time Stamp Control Register
CGU_TSVALUE0	Time Stamp Counter Initial 32 LSB Value Register
CGU_TSVALUE1	Time Stamp Counter Initial MSB Value Register

Core Clock Buffer Disable Register

The CGU_CCBF_DIS register controls each core's clock buffer to determine if the CCLK is enabled.

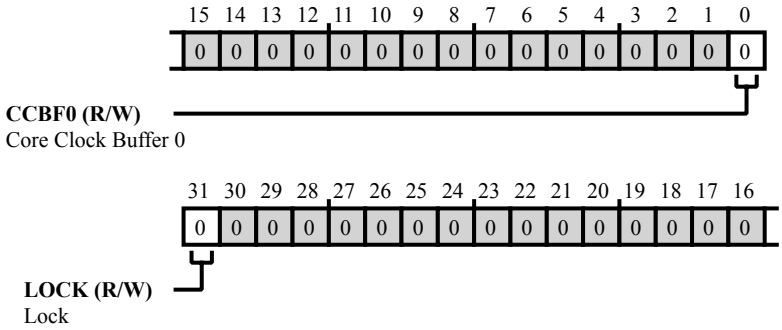


Figure 6-4: CGU_CCBF_DIS Register Diagram

Table 6-11: CGU_CCBF_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If set (=1) the CGU_CCBF_DIS.LOCK bit locks the CGU_CCBF_DIS register.	
		0	Unlock register
		1	Lock register
0 (R/W)	CCBF0	Core Clock Buffer 0. The CGU_CCBF_DIS.CCBF0 bit enables (=0) or disables (=1) CCLK0s buffer.	
		0	Enable buffer
		1	Disable buffer

Core Clock Buffer Status Register

The CGU_CCBF_STAT register shows which core clock buffer(s) are disabled. For example clearing the CGU_CCBF_DIS.CCBF0 bit clears the CGU_CCBF_STAT.CCBF0 bit after a number of cycles. To guarantee that the correct value is read, this register should be read twice and the second result used.

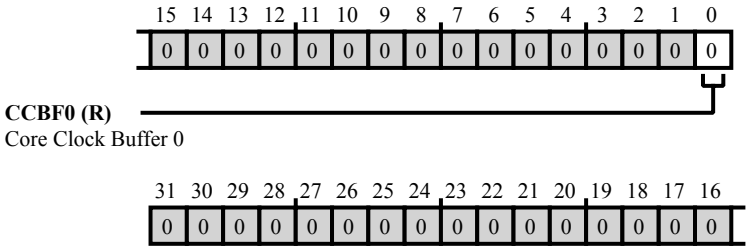


Figure 6-5: CGU_CCBF_STAT Register Diagram

Table 6-12: CGU_CCBF_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	CCBF0	Core Clock Buffer 0. The CGU_CCBF_STAT.CCBF0 bit reports the status of the CGU_CCBF_DIS.CCBF0 bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled

CLKOUT Select Register

The CGU_CLKOUTSEL selects the signal that the CGU drives through the CLKOUT multiplexer.

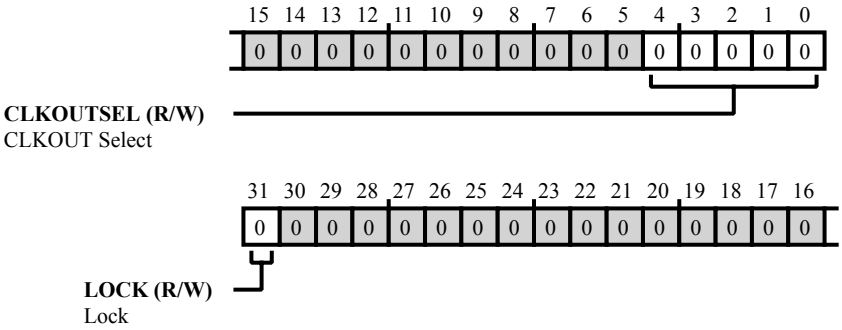


Figure 6-6: CGU_CLKOUTSEL Register Diagram

Table 6-13: CGU_CLKOUTSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the CGU_CLKOUTSEL.LOCK bit is set, the CGU_CLKOUTSEL register is read only (locked).	
		0	Unlock
		1	Lock
4:0 (R/W)	CLKOUTSEL	CLKOUT Select. The CGU_CLKOUTSEL.CLKOUTSEL selects the signal that the CGU drives through the CLKOUT pin multiplexer. CGU_CLKOUTSEL.CLKOUTSEL values 0x0A to 0x1F are reserved.	
		0	GND
		1	CLKBUF0 - CLKIN from System Oscillator 0
		2	CLKBUF1 - CLKIN from System Oscillator 1
		3	CCLK - CCLK0 from CGU (divided by 4)
		4	SCLK - SCLK0 from CGU (divided by 2)
		5	SYSCLK - SYSCLK from CGU (divided by 2)
		6	DCLK from CGU. Used to generate FOCP_CLK
		7	AFE FOCP comparator clock
		8	OUTCLK/PLLCLK - OCLK from CGU
		9	M0RSTB - RCU_CRES[0] from RCU

Table 6-13: CGU_CLKOUTSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		10	Reserved
		11	Reserved
		12	Reserved
		13	Reserved
		14	Reserved
		15	Reserved

Control Register

The CGU_CTL controls the clock generation divisors for SYS_CLKIN and the PLL. Read after write accesses to the CGU_CTL register returns the new value even if the clock's frequency change is still in progress.

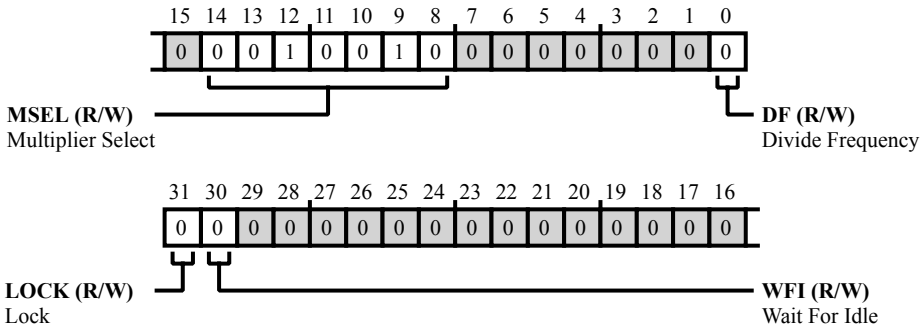


Figure 6-7: CGU_CTL Register Diagram

Table 6-14: CGU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the CGU_CTL.LOCK bit is set, the CGU_CTL register is read only (locked).	
		0	Unlock
		1	Lock
30 (R/W)	WFI	Wait For Idle. Modifying the PLL multiplier requires the PLL to re-lock and once the PLL locks, clocks have to be synchronized. Changes to the CGU_CTL.MSEL and the CGU_CTL.DF bit values results in bypassing the PLL. The CGU_CTL.WFI bit forces the PLL to wait for all processor cores to be in an idle or reset state before changing frequencies as a result of changes to the CGU_CTL.MSEL or CGU_CTL.DF bits. Write accesses to the CGU_CTL to change the CGU_CTL.DF or CGU_CTL.MSEL bit values while the PLL is locking sets the CGU_STAT.WDFMSERR bit.	
		0	Update Immediately
		1	Wait for Idle

Table 6-14: CGU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14:8 (R/W)	MSEL	Multiplier Select. The `CGU_CTL.MSEL` bit field selects the multiplier in the PLLCLK equation: PLLCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL Where the value of MSEL is between 1 and 127.	
		0	MSEL = 128
		1-127	MSEL = 1 to 127
0 (R/W)	DF	Divide Frequency. The `CGU_CTL.DF` bit selects whether or not the CLKIN input is divided by two before being passed to the PLL.	
		0	Pass OSC_CLKIN to PLL
		1	Pass OSC_CLKIN/2 to PLL

Clocks Divisor Register

The CGU_DIV register controls clock divisors for core clocks, system clocks, external (off core) memory clocks, and output clock. Read after write accesses to the CGU_DIV register returns the new value even if the clock's frequency change is still in progress.

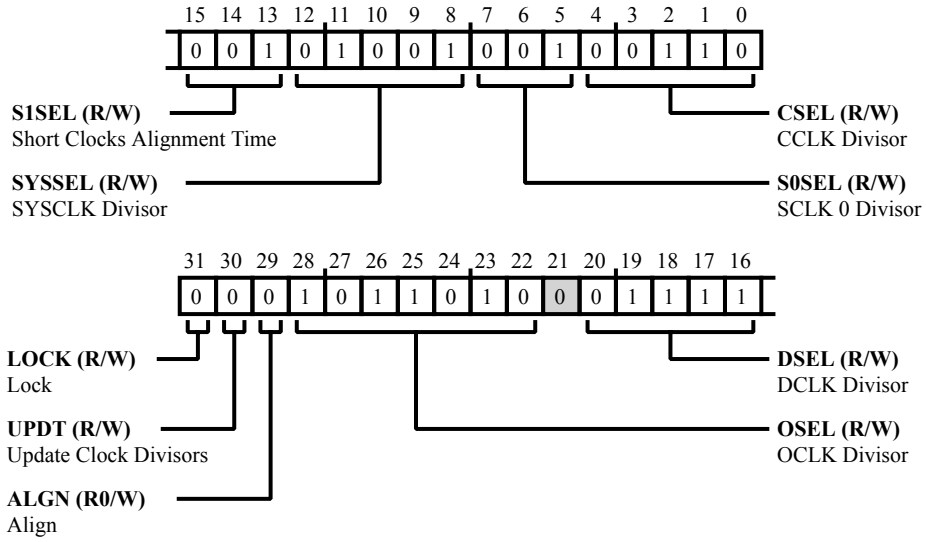


Figure 6-8: CGU_DIV Register Diagram

Table 6-15: CGU_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the CGU_DIV.LOCK bit is set, the CGU_DIV register is read only (locked).	
		0	Unlock
		1	Lock
30 (R/W)	UPDT	Update Clock Divisors. The CGU_DIV.UPDT controls whether the CGU drives new CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, and CGU_DIV.OSEL values to PLL after CGU_DIV register update.	
		0	No PLL Update
		1	Drive Updated SEL Values to PLL

Table 6-15: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R0/W)	ALGN	Align. The CGU_DIV.ALGN directs the CGU to align the PLL-based clocks. The divisor selections (CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, and/or CGU_DIV.OSEL) do not have to change.	
		0	No Action
		1	Align PLL Clocks
28:22 (R/W)	OSEL	OCLK Divisor. The CGU_DIV.OSEL selects the divisor in the OCLK equation: OCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL / CGU_DIV.OSEL Where the value of CGU_DIV.OSEL is between 1 and 127.	
		0	OSEL = 128
		1-127	OSEL = 1 to 127
20:16 (R/W)	DSEL	DCLK Divisor. The CGU_DIV.DSEL selects the divisor in the DCLK equation: DCLK frequency = (SYS_CLKIN frequency/(DF+1)) MSEL/CGU_DIV.DSEL Where the value of CGU_DIV.DSEL is between 1 and 31.	
		0	DSEL = 32
		1-31	DSEL = 1 to 31
15:13 (R/W)	S1SEL	Short Clocks Alignment Time. The CGU_DIV.S1SEL Determines if the time it takes clocks to align is short or long.	
		0	Long Clocks Alignment Time
		1	Short Clocks Alignment Time
		2-7	Reserved
12:8 (R/W)	SYSSEL	SYSCLK Divisor. The CGU_DIV.SYSSEL selects the divisor in the SYSCLK equation: SYSCLK frequency = (SYS_CLKIN frequency/(DF+1)) MSEL/CGU_DIV.SYSSEL Where the value of CGU_DIV.SYSSEL is between 1 and 31.	
		0	SYSSEL = 32
		1-31	SYSSEL = 1 to 31

Table 6-15: CGU_DIV Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:5 (R/W)	S0SEL	SCLK 0 Divisor. The `CGU_DIV.S0SEL` selects the divisor in the SCLK0 equation: SCLK0 frequency = (SYSCLK frequency) / `CGU_DIV.S0SEL` Where the value of `CGU_DIV.S0SEL` is between 1 and 7.	
		0	S0SEL = 8
		1-7	S0SEL = 1 to 7
4:0 (R/W)	CSEL	CCLK Divisor. The `CGU_DIV.CSEL` selects the divisor in the CCLK equation: CCLK frequency = (SYS_CLKIN frequency / (DF+1)) * MSEL / `CGU_DIV.CSEL` Where the value of `CGU_DIV.CSEL` is between 1 and 31.	
		0	CSEL = 32
		1-31	CSEL= 1 to 31

Oscillator Watchdog Register

The CGU_OSCWDCTL register configures the CGU to allow the detection of the absence of input clock transitions and provides a fault warning via the SYS_FAULT pin. The CGU_OSCWDCTL register also detects and reports input oscillator frequencies above and below specified limits, in order to specifically detect harmonic or sub-harmonic crystal oscillator behavior. This detection is achieved by using an internal asynchronous, local 1 MHz oscillator combined with a series of programmable counters.

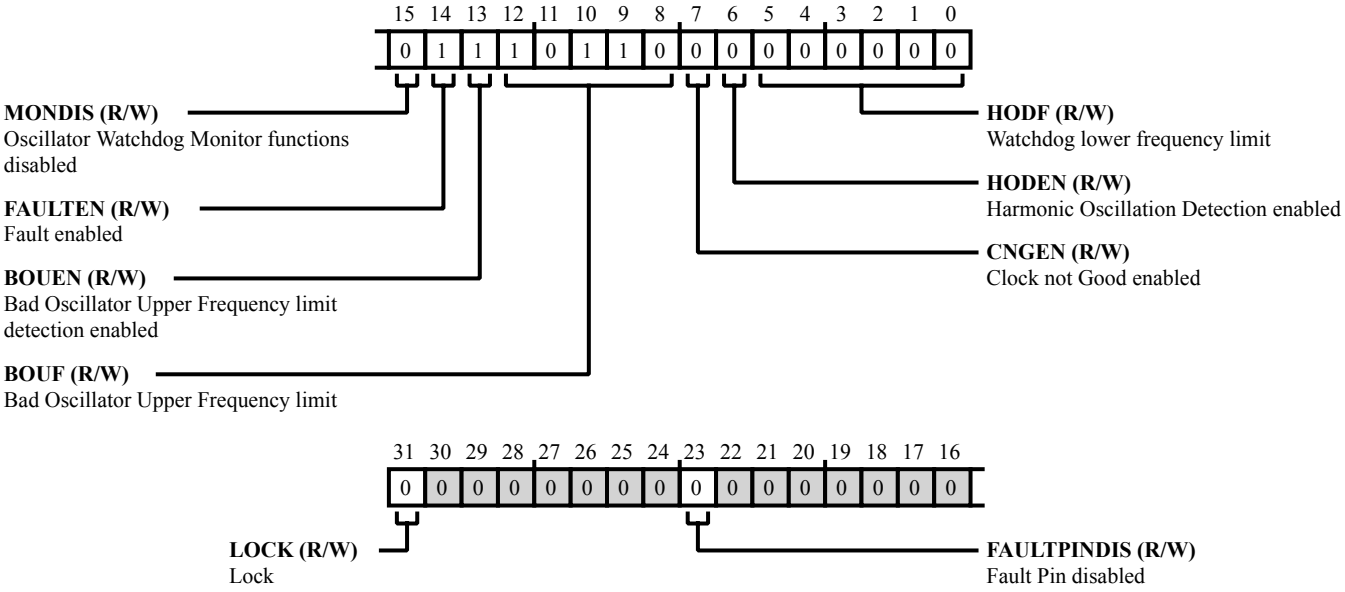


Figure 6-9: CGU_OSCWDCTL Register Diagram

Table 6-16: CGU_OSCWDCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock. If the global lock bit is set and theCGU_OSCWDCTL.LOCK bit is set, the CGU_OSCWDCTL register is read only (locked).
23 (R/W)	FAULTPINDIS	Fault Pin disabled. The CGU_OSCWDCTL.FAULTPINDIS bit disables pin fault detection.
15 (R/W)	MONDIS	Oscillator Watchdog Monitor functions disabled. The CGU_OSCWDCTL.MONDIS bit disables all the input clock monitor and fault detection functions.
14 (R/W)	FAULTEN	Fault enabled. The CGU_OSCWDCTL.FAULTEN bit enables fault detection.
13 (R/W)	BOUEN	Bad Oscillator Upper Frequency limit detection enabled. The CGU_OSCWDCTL.BOUEN bit enables upper limit bad oscillation detection.

Table 6-16: CGU_OSCWDCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
12:8 (R/W)	BOUF	Bad Oscillator Upper Frequency limit. The `CGU_OSCWDCTL.BOUF` bits indicate the desired upper fail limit for the bad oscillation detection.
7 (R/W)	CNGEN	Clock not Good enabled. The `CGU_OSCWDCTL.CNGEN` bit enables the detection of an oscillator watchdog clock fault.
6 (R/W)	HODEN	Harmonic Oscillation Detection enabled. The `CGU_OSCWDCTL.HODEN` bit enables harmonic oscillation detection.
5:0 (R/W)	HODF	Watchdog lower frequency limit. The `CGU_OSCWDCTL.HODF` bit field is used to indicate the desired lower fail limit for the harmonic oscillation detection in MHz.

PLL Control Register

The CGU_PLLCTL register contains bits that enable and disable the PLL as well as control its function.

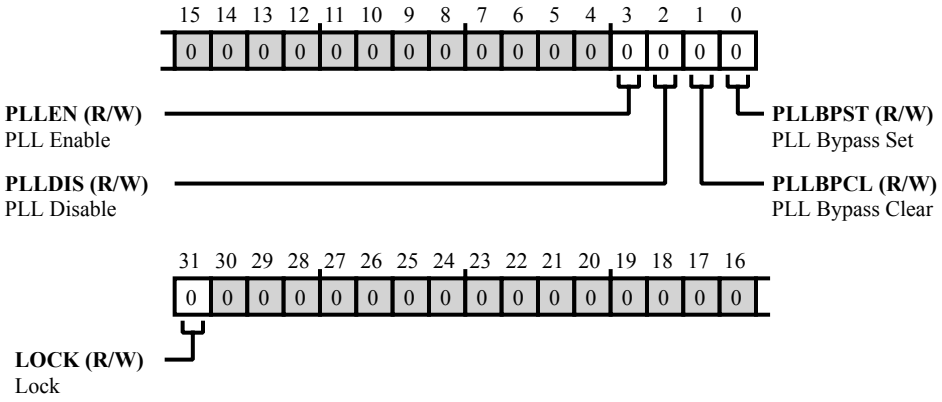


Figure 6-10: CGU_PLLCTL Register Diagram

Table 6-17: CGU_PLLCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. Setting (=1) the CGU_PLLCTL.LOCK bit locks access to the CGU_PLLCTL register.	
		0	Unlock register
		1	Lock register
3 (R/W)	PLLEN	PLL Enable. Setting (=1) the CGU_PLLCTL.PLLEN bit enables the PLL.	
		0	No action
		1	Enable PLL
2 (R/W)	PLLDIS	PLL Disable. Setting (=1) the CGU_PLLCTL.PLLDIS bit disables the PLL.	
		0	No action
		1	Disable PLL
1 (R/W)	PLLBPCL	PLL Bypass Clear. Setting (=1) the CGU_PLLCTL.PLLBPCL bit takes the PLL out of bypass mode.	
		0	No action
		1	Exit bypass mode

Table 6-17: CGU_PLLCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	PLLBPST	PLL Bypass Set. Setting (=1) the `CGU_PLLCTL.PLLBPST` bit bypasses the PLL and all the clocks run on CLKIN.	
		0	Use PLL
		1	Bypass PLL

Revision ID Register

The CGU_REVID register reports the version of the CGU.

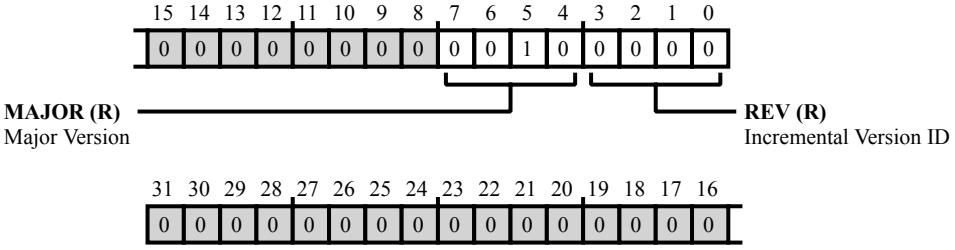


Figure 6-11: CGU_REVID Register Diagram

Table 6-18: CGU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version.
3:0 (R/NW)	REV	Incremental Version ID.

System Clock Buffer Disable Register

The CGU_SCBF_DIS register controls each system's clock buffer to determine if the SCLKn buffer is enabled.

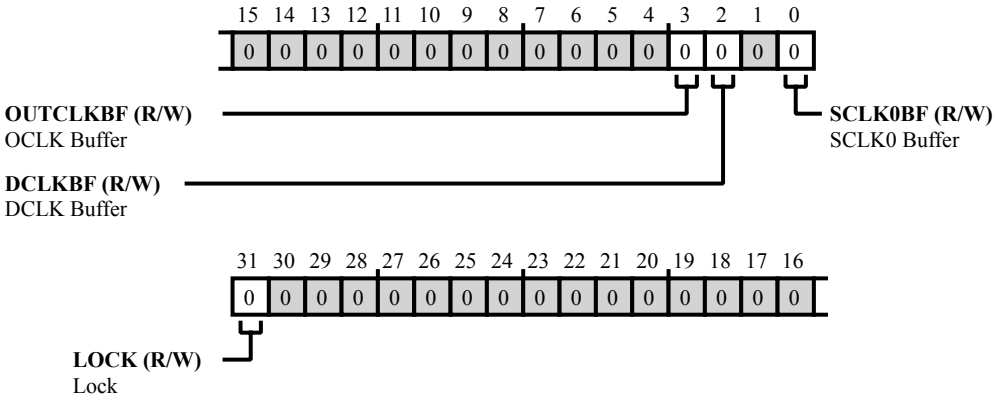


Figure 6-12: CGU_SCBF_DIS Register Diagram

Table 6-19: CGU_SCBF_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. The CGU_SCBF_DIS.LOCK bit allows writes to the CGU_SCBF_DIS register when cleared (=0) or blocks writes if set (=1) and the SPU_CTL.GLCK bit is set.	
		0	Unlock register
		1	Lock register
3 (R/W)	OUTCLKBF	OCLK Buffer. The CGU_SCBF_DIS.OUTCLKBF bit enables (=0, default) or disables (=1) OCLKs buffer.	
		0	Enable buffer
		1	Disable buffer
2 (R/W)	DCLKBF	DCLK Buffer. The CGU_SCBF_DIS.DCLKBF bit enables (=0, default) or disables (=1) DCLKs buffer.	
		0	Enable buffer
		1	Disable buffer

Table 6-19: CGU_SCBF_DIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	SCLK0BF	SCLK0 Buffer. The `CGU_SCBF_DIS.SCLK0BF` bit enables (=0, default) or disables (=1) SCLK0s buffer.	
		0	Enable buffer
		1	Disable buffer

System Clock Buffer Status Register

The CGU_SCBF_STAT register shows which system clock buffer(s) are disabled. For example clearing the CGU_CCBF_DIS.CCBF0 bit clears the CGU_SCBF_STAT.SCLK0BF bit after a number of cycles. To guarantee that the correct value is read, this register should be read twice and the second result used.

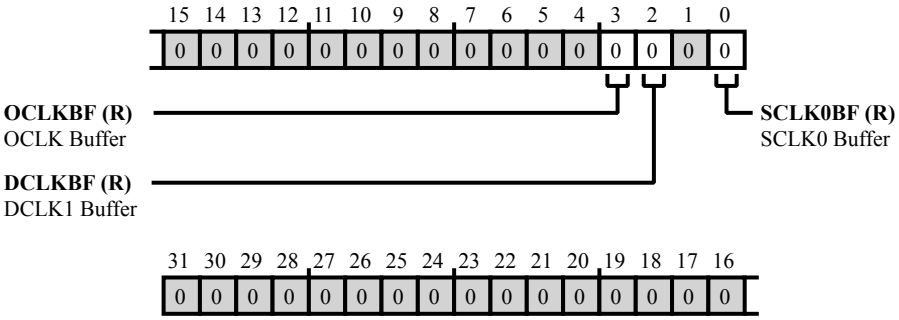


Figure 6-13: CGU_SCBF_STAT Register Diagram

Table 6-20: CGU_SCBF_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	OCLKBF	OCLK Buffer. The CGU_SCBF_STAT.OCLKBF bit reports the status of the CGU_SCBF_DIS.OUTCLKBF bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled
2 (R/NW)	DCLKBF	DCLK1 Buffer. The CGU_SCBF_STAT.DCLKBF bit reports the status of the CGU_SCBF_DIS.DCLKBF bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled
0 (R/NW)	SCLK0BF	SCLK0 Buffer. The CGU_SCBF_STAT.SCLK0BF bit reports the status of the CGU_SCBF_DIS.SCLK0BF bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled

Status Register

The CGU_STAT register reflects the PLL status and errors detected during the PLL configuration. This register may be cleared asynchronously by a reset signal from the RCU module. All bits---except those defined as W1C (write-1-to-clear)---are read only.

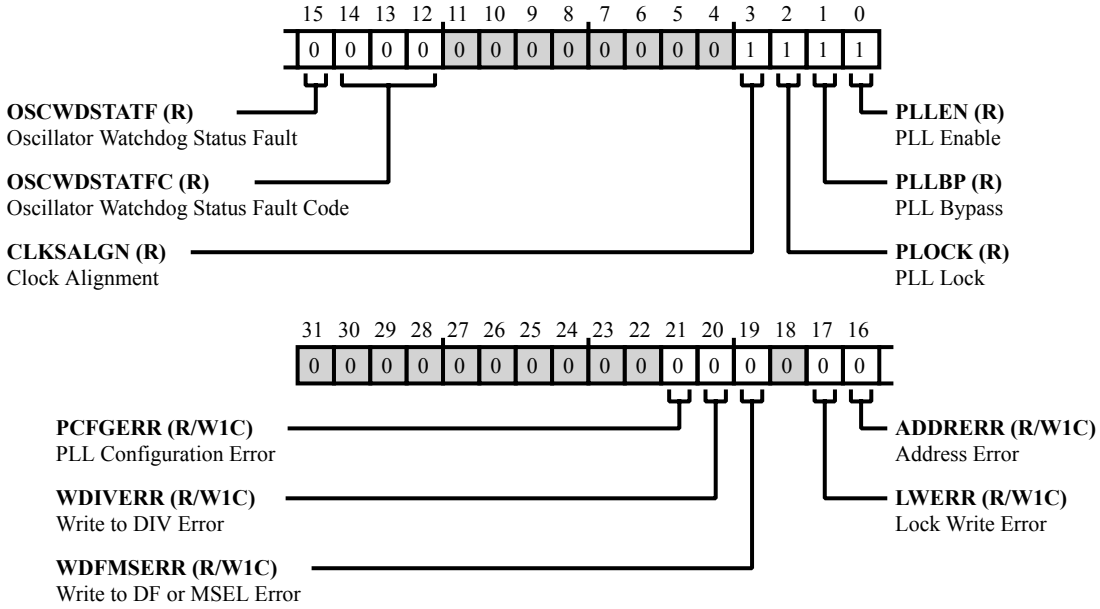


Figure 6-14: CGU_STAT Register Diagram

Table 6-21: CGU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W1C)	PCFGERR	PLL Configuration Error. If the CGU_PLLCTL.PLLBPST and the CGU_PLLCTL.PLLBPCL bits are set (=1) simultaneously or the CGU_PLLCTL.PLLDIS bit was set (=1) in full-on mode or while trying to enter full-on mode (CGU_PLLCTL.PLLBPCL =1), the CGU_STAT.PCFGERR bit triggers the bus error.	
		0	No Error
		1	Configuration Error

Table 6-21: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	WDIVERR	Write to DIV Error. The CGU_STAT.WDIVERR bit indicates a write access to the CGU_DIV register (to trigger an alignment sequence or to change the CGU_DIV.CSEL, CGU_DIV.SYSSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, or CGU_DIV.DSEL bit values) while the PLL is locked, but still aligning the clocks. Read after write accesses to the CGU_STAT and CGU_DIV registers return the new value even if the clock frequency change is still in progress.	
		0	No Error
		1	Write DIV Error
19 (R/W1C)	WDFMSERR	Write to DF or MSEL Error. The CGU_STAT.WDFMSERR bit indicates a write access to the CGU_CTL register to change the CGU_CTL.DF or CGU_CTL.MSEL bit values while the PLL is locking.	
		0	No Error
		1	Write DF/MSEL Error
17 (R/W1C)	LWERR	Lock Write Error. The CGU_STAT.LWERR bit indicates an attempt to write to write-protected (locked) CGU registers. The CGU issues a bus error for this condition.	
		0	No Error
		1	Lock Write Error
16 (R/W1C)	ADDRERR	Address Error. The CGU_STAT.ADDRERR bit indicates an attempt to make a read or write access to unimplemented addresses or accesses are non-aligned. The CGU issues a bus error for this condition.	
		0	No Error
		1	Address Error
15 (R/NW)	OSCWDSTATF	Oscillator Watchdog Status Fault. The CGU_STAT.OSCWDSTATF bit indicates a fault in the oscillator watchdog (CGU's OSC_WDSTAT[1:0]) input pins.	
		0	No Fault
		1	Fault
14:12 (R/NW)	OSCWDSTATFC	Oscillator Watchdog Status Fault Code. The CGU_STAT.OSCWDSTATFC bit field indicates the nature of the fault in the oscillator watchdog (CGU's OSC_WDSTAT[1:0]) input pins.	
		0	No Fault
		1	No Input Clock

Table 6-21: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	Subharmonic CLKIN
		3	Harmonic CLKIN
		4	No AUX_CLK
		5	CLKIN > Upper Frequency Limit (BOUF)
		6	Reserved
		7	Multiple Limit Faults
3 (R/NW)	CLKSALGN	Clock Alignment. The CGU_STAT.CLKSALGN bit indicates whether a clock alignment sequence is in progress. This bit is set when clocks alignment is required by changes to CGU_DIV.CSEL, CGU_DIV.S0SEL, CGU_DIV.S1SEL, CGU_DIV.DSEL, or CGU_DIV.OSEL. The CGU_STAT.CLKSALGN bit is cleared when clocks are aligned. Note that (after a PLL frequency change in active state) the CGU_STAT.CLKSALGN bit may indicate that clocks are not aligned even though the clocks are aligned (all clocks are aligned and running at CLKIN frequency).	
		0	Clocks are Aligned
		1	Clocks not Aligned (alignment in progress)
2 (R/NW)	PLOCK	PLL Lock. The CGU_STAT.PLOCK bit indicates whether the PLL is locked. This bit is set when the PLL locks (PLL lock counter end-of-count). The CGU_STAT.PLOCK bit is cleared when requested PLL frequency change (for PLL reset, PLL disable-to-enable transition, or a change to the CGU_CTL.MSEL or CGU_CTL.DF values) is in progress.	
		0	PLL not Locked (PLL frequency change in progress)
		1	PLL Locked
1 (R/NW)	PLLBP	PLL Bypass. The CGU_STAT.PLLBP bit indicates whether the PLL is bypassed. The default value for the CGU_STAT.PLLBP bit is determined by the bypass strap pin.	
		0	PLL not Bypassed
		1	PLL Bypassed
0 (R/NW)	PLLEN	PLL Enable. The CGU_STAT.PLLEN bit indicates whether the PLL is enabled.	
		0	Disabled
		1	Enabled

Time Stamp Counter 32 LSB Register

The CGU_TSCOUNT0 register address is used to read the CoreSight time stamp counter LSB 32-bit (bits [31:0]) value.

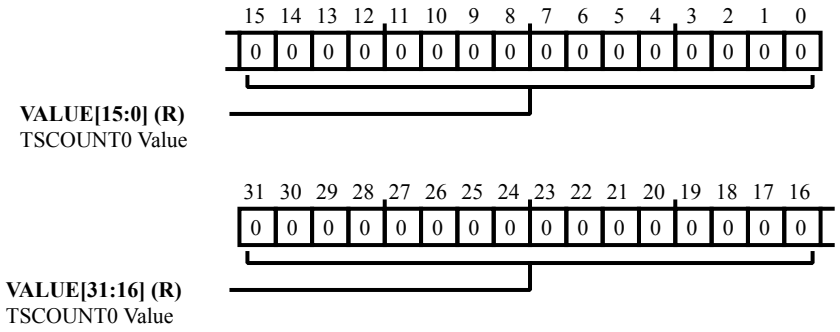


Figure 6-15: CGU_TSCOUNT0 Register Diagram

Table 6-22: CGU_TSCOUNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	TSCOUNT0 Value. The CGU_TSCOUNT0.VALUE bit field holds the time stamp counter 32 LSBs.

Time Stamp Counter 32 MSB Register

The CGU_TSCOUNT1 register address is used to read the CoreSight time stamp counter MSB 32-bit (bits [63:32]) value.

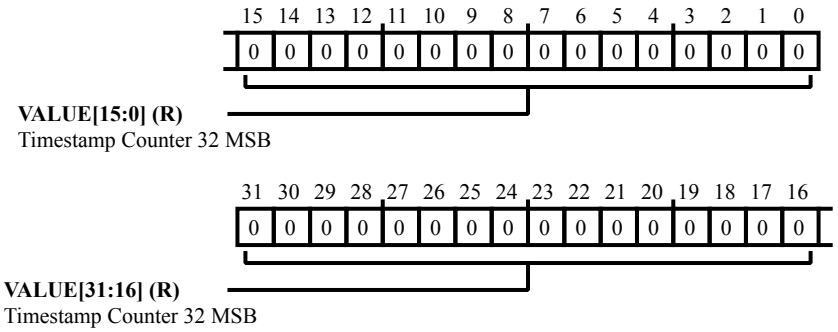


Figure 6-16: CGU_TSCOUNT1 Register Diagram

Table 6-23: CGU_TSCOUNT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Timestamp Counter 32 MSB. The CGU_TSCOUNT1.VALUE bit field holds the time stamp counter 32 MSBs.

Time Stamp Control Register

The CGU_TSCTL register controls the operation of the CoreSight time stamp counter.

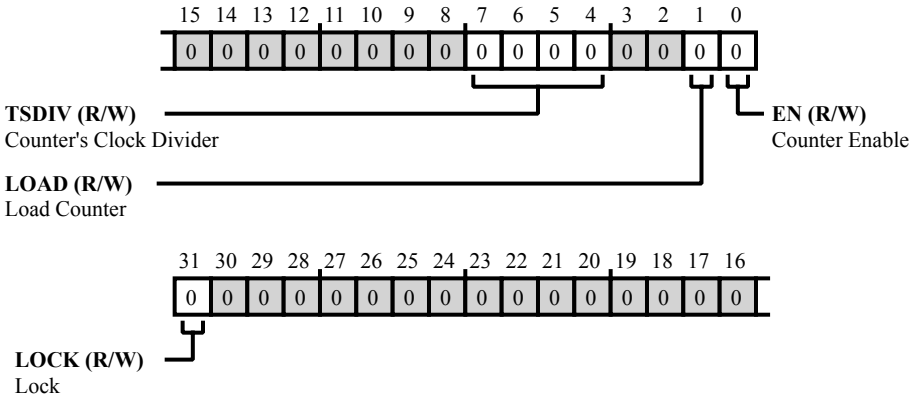


Figure 6-17: CGU_TSCTL Register Diagram

Table 6-24: CGU_TSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. Setting the CGU_TSCTL.LOCK bit locks this register.	
		0	Unlock
		1	Lock
7:4 (R/W)	TSDIV	Counter's Clock Divider. The CGU_TSCTL.TSDIV bit field divides SYSCLK by 2^{TSDIV}.	
		0-15	Divides SYSCLK by 2^{TSDIV}
1 (R/W)	LOAD	Load Counter. Writing one to the CGU_TSCTL.LOAD bit causes CoreSight time stamp counter to be loaded from the CGU_TSVALUE0 and CGU_TSVALUE1 registers.	
		0	Always read as "0"
0 (R/W)	EN	Counter Enable. The CGU_TSCTL.EN bit enables or disables the CoreSight time stamp counter.	
		0	Counter Disabled
		1	Counter Enabled

Time Stamp Counter Initial 32 LSB Value Register

The CGU_TSVALUE0 register holds the least significant bits (bits [31:0]) value that is initially loaded to the CoreSight time stamp counter.

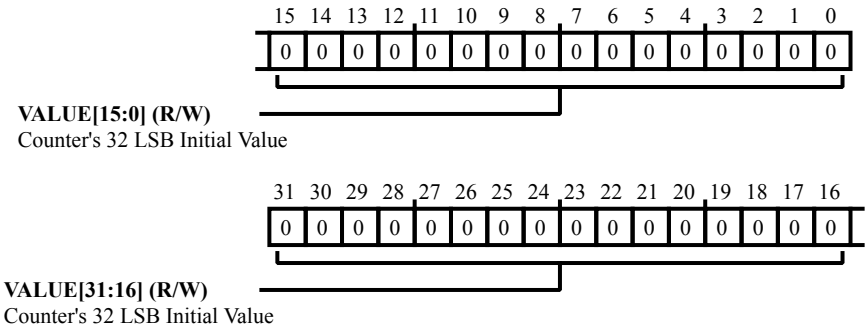


Figure 6-18: CGU_TSVALUE0 Register Diagram

Table 6-25: CGU_TSVALUE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Counter's 32 LSB Initial Value. The CGU_TSVALUE0.VALUE bit field holds the LSBs value that is initially loaded to the CoreSight time stamp counter.

Time Stamp Counter Initial MSB Value Register

The CGU_TSVALUE1 register holds the most significant bits (bits [63:32]) value that is initially loaded to the CoreSight time stamp counter.

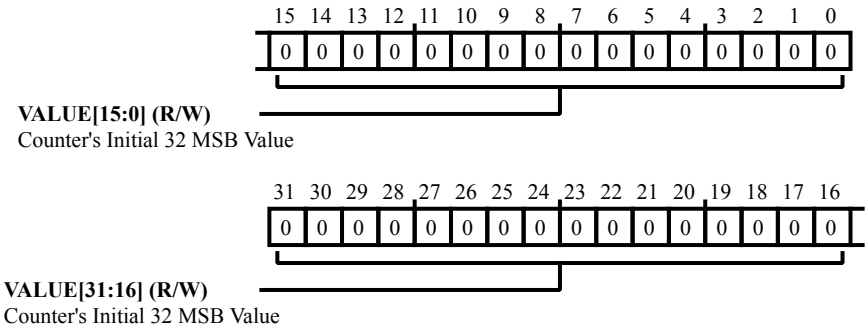


Figure 6-19: CGU_TSVALUE1 Register Diagram

Table 6-26: CGU_TSVALUE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Counter's Initial 32 MSB Value. The CGU_TSVALUE1.VALUE bit field holds the MSBs value that is initially loaded to the CoreSight time stamp counter.

7 Oscillator Comparator Unit (OCU)

The OCU monitors the frequency of the main derived system clock. The OCU uses an additional oscillator input (see the data sheet for specifications) to monitor the internal derived system clocks for excessive drift or out-of-bounds operation. It also provides for additional dead-clock monitoring beyond the basic OSCMON. The OSCMON is used for GROSS errors, this block is used to measure smaller errors.

The OCU functions are controlled through control and status registers accessible through the peripheral bus interface.

NOTE: Clock errors detected by the OCU may indicate the processor is operating out-of-range or has no clocking. A processor response to interrupt assertions may not occur. The system allows this fault output to assert the chip fault output pin and request external assistance.

OCU Features

The OCU is controlled through the memory-mapped registers which are accessed by the peripheral bus slave port. The OCU System Clock Monitor has the following features

- A system clock frequency monitor that is implemented using a counter and separate low-frequency oscillator (LFO)
- A wide counter and programmable measurement period that supports high-frequency resolution.
- Continuous measurement mode that automatically compares the measured frequency against stored limits and signals a fault when the limits are exceeded
- Manual measurement mode under software control
- Dead clock monitoring. Both input and LFO clocks are monitored for gross frequency error. The frequency limits that are used to define dead clocks are user-programmable.
- A fault interrupt pin that connects to the system event controller. Fault status and control bits control fault reporting.
- An asynchronous fault output pin that is to report a catastrophic clock failure

OCU Functional Description

The OCU monitors the frequency of the input clock using an additional low-frequency oscillator input. If the combined frequency drift of the two clocks exceeds the specification, a fault is issued.

CM41X_M4 OCU Register List

Table 7-1: CM41X_M4 OCU Register List

Name	Description
OCU_CLKCNT	Frequency Measured Count Register
OCU_CMONCNT	CLKIN Monitor Reference Count Register
OCU_CTL	Control Register
OCU_LMONCNT	LFO Monitor Reference Count Register
OCU_MAXCNT	Maximum Count Register
OCU_MINCNT	Minimum Limit Register
OCU_REFCNT	Reference Count Register
OCU_STAT	Status Register

Additional OCU Registers

The OCU module contains an additional register (PADS_MONOSC_CFG) that provides configuration status.

CM41X_M4 OCU Interrupt List

Table 7-2: CM41X_M4 OCU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
4	OCU0_ERR	OCU0 Freq. out-of-range, or dead clock, or counter overflow detected	Level	

OCU Block Diagram

The ***Timer Block Diagram*** shows the functional blocks within the OCU.

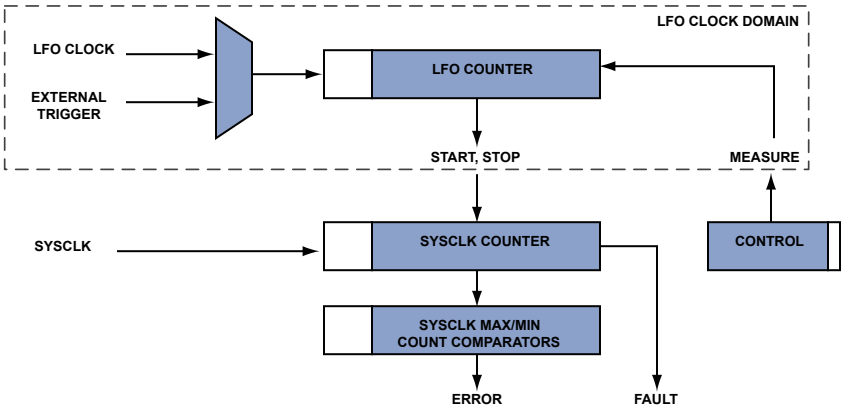


Figure 7-1: Timer Block Diagram

The measurement counter is located in the SYSCLK domain. It measures the time interval in SYSCLK cycles between start and stop events. These events originate in the LFO clock domain. The events indicate when the low-frequency oscillator (LFO) time-base counter begins and ends a count of COUNT_TIME cycles of the reference clock (either the LFO clock or the external calibration trigger input.) The resulting count of SYSCLK cycles can be read in the OCU_CLKCNT register.

Input Clock Frequency Measurement

The OCU measures the input clock frequency by counting input clocks for a predetermined period. The final counter result is automatically loaded into the OCU_CLKCNT register. A programmable number of cycles of a separate low-frequency oscillator (LFO) determines the counting period. The number of LFO cycles used for an input clock frequency measurement is located in the OCU_REFCNT register. Measurement cycles can be run continuously. They can also be run manually. Bits in the OCU_CTL register control this behavior.

Performing a Frequency Measurement

A frequency measurement is started by setting the OCU_CTL.MEASURE bit. The measurement does not start if the OCU_CTL.MEASURE bit is already set (=1) or the OCU_STAT.BUSY bit =1. Starting a measurement cycle causes the input clock counter and the OCU_CLKCNT register to be cleared. The LFO reference counter is loaded from the OCU_REFCNT register. The OCU_STAT.BUSY status bit is set. The fault status bits are unaffected.

While the LFO reference counter counts down, the input clock counter counts up. The counter run signal is synchronized from the slower LFO clock domain to the faster input clock domain. This synchronization minimizes the delay and the counter error due to synchronization.

When the reference count reaches zero, it stops counting. The input clock counter stops counting after a slow-to-fast synchronization delay. Because the start and stop signals are synchronized in the same direction, the input counter error due to synchronization is ±1.

When counting completes, the OCU_CLKCNT register is updated with the final input clock count. Then, the OCU_STAT.BUSY bit is cleared.

Comparing Measured Frequency Against Limits

The OCU_MINCNT and OCU_MAXCNT are user-programmable limit registers which are used to set bounds on the acceptable frequency error. At the end of a measurement cycle, the input clock counter is compared to each limit. If the measured frequency is outside the specified range, the OCU_STAT.FREQ_FAULT status bit is set.

NOTE: Changing the contents of the limit registers during a measurement cycle can have unpredictable effects on the fault detection.

Automatic Frequency Monitoring

Setting the OCU_CTL.AUTO_EN control bit enables continuous automatic frequency fault detection. A new measurement cycle is started a few LFO clocks after the current cycle completes. All other functions are identical between the manual and automatic modes.

Dead Clock Monitoring

The input clock and the LFO clock are monitored for gross frequency errors. This monitoring is accomplished by synchronizing the status handshake signals across the clock domains. The reference clock domain sends a request which is synchronized to the target clock domain, which in turn sends an acknowledge signal back to the reference clock domain. If the handshake is not completed by a user-specified number of clock cycles, a dead clock error signal is asserted.

The LFO clock monitors the input clock. The handshake timeout value is contained in the OCU_CMONCNT register, which specifies a number of LFO clock cycles. A dead input clock error sets the OCU_STAT.CMON_FAULT sticky status bit.

The input clock monitors the LFO clock. The handshake timeout value is contained in the OCU_LMONCNT register, which specifies a number of input clock cycles. A dead LFO clock error sets the OCU_STAT.LMON_FAULT sticky status bit.

The dead clock monitoring operations are only run when the OCU_CTL.AUTO_EN control bit is set.

Control and Status

All the control bits are contained in the OCU_CTL register. All the status bits are contained in the OCU_STAT register.

The OCU_CTL.MEASURE bit starts a frequency measurement cycle when it transitions from 0 to1. If this bit is set during a running measurement cycle, the measurement cycle is not interrupted and no new measurement cycle starts after the current one completes. Wait until the OCU_STAT.BUSY bit is clear (=0) before starting a new measurement cycle.

The OCU_CTL.AUTO_EN bit enables continuous automatic measurement cycles. This bit can be modified at any time. Clearing it does not abort a measurement cycle. When OCU_CTL.AUTO_EN is high, the OCU_CTL.MEASURE bit has no effect.

The OCU_CTL.FREQ_FAULT_EN bit allows the OCU_STAT.FREQ_FAULT bit to be OR'ed into the OCU_FAULT output. The OCU_CTL.CMON_FAULT_EN and OCU_CTL.LMON_FAULT_EN bits provide the same behavior for the OCU_STAT.CMON_FAULT and OCU_STAT.LMON_FAULT bits, respectively. The OCU_CTL.DEAD_CLOCK_EN bit gates the OCU_DEAD_CLOCK output.

The OCU_STAT.FREQ_FAULT bit is set when a measurement cycle completes and the clock counter is outside the limits set by the OCU_MINCNT and OCU_MAXCNT registers. This is sticky write-one-to-clear bit. It is also cleared when the OCU_rstb signal is asserted.

Faults in the respective dead clock monitors set the OCU_STAT.CMON_FAULT and OCU_STAT.LMON_FAULT bits. They are cleared in the same way as the OCU_STAT.FREQ_FAULT bit.

The OCU_STAT.OVERFLOW sticky status bit is set when the input clock counter overflows during a frequency measurement. It is cleared in the same way as the OCU_STAT.FREQ_FAULT bit. The OCU_STAT.OVERFLOW bit can also be OR'ed into the ocu_fault output. It is masked by the OCU_CTL.FREQ_FAULT_EN bit.

Reset Behavior

All OCU registers are reset to default states on the assertion of the SYS_HWRST signal. The measurement state machine is reset to the idle state. The asynchronous fault output is deasserted. The peripheral bus interface state is reset by the peripheral-specific reset.

MMR Interface

The OCU memory mapped registers are accessed through the SCB slave interface. The OCU occupies a 4 KB memory space. All system crossbar accesses to the OCU are interpreted as 32-bit aligned reads (the processor and OCU only support 32-bit accesses). The two address LSBs are ignored. For writes, byte resolution is supported through the SCB write strobes. Reads and writes to reserved addresses return a slave error. The assertion of the SCB reset asynchronously resets the bus interface state. The SCB reset does not affect any OCU register state, unless a write was in-progress at the time. In that case, the state of the addressed register is unspecified.

Outputs

The OCU produces two control outputs. The OCU_ERR signal goes active high to indicate a fault in the frequency measurement, or one of the dead clock monitor operations. The propagation of the fault status to this output is controlled by several enables.

The OCU_DEAD_CLOCK signal goes active LOW when a dead clock fault is detected on the input clock. It is deasserted HIGH when the OCU block is reset.

Clock Domains and Synchronization

The clock counter is clocked in the input clock domain. The time base counter is clocked in the LFO clock domain. In either case, the respective count enable operations are synchronized to the clock domain of each counter. There is a ±1 uncertainty due to the synchronization.

The measurement state machine is run in the LFO clock domain. This allows the input clock measurement to run for an exact number of LFO (slow) clocks. The synchronization error of the clock counter is small due to the higher frequency of the input clock. To ensure correct operation of the frequency measurement system, the input clock must be at least twice the frequency of the LFO clock. This frequency ratio allows correct functioning in the presence of a frequency drift that is well above the expected fault limits.

There are no clock ratio requirements for the dead clock monitoring operations because they use a full handshake across the clock boundary.

Critical Startup and Shutdown

While an OCU fault can signal a critical system error that requires shutdown, the OCU needs no critical handling. To avoid spurious errors, ensure that the clocks used by the OCU are stable before enabling continuous monitoring.

OCU Architectural Concepts

The following sections describe the OCU architectural concepts including frequency measurement, dead clock detection, calibration, clock gating and jitter.

Clock Frequency Measurement

The frequency accuracy of the input clock must be ensured within a certain tolerance. This is accomplished by counting the number of input clocks within a predefined period of time. This clock count is then compared to maximum and minimum count values which are based on the acceptable frequency error. The amount of measurement time must be large enough to provide the necessary frequency resolution, but small enough that the system can issue a fault quickly enough to take appropriate action.

The time base is used establish the measurement time is provided by an additional low frequency oscillator (LFO) input. Since this oscillator is not a perfect time base, the measured input clock frequency error is actually the sum of the frequency errors of the input clock and the LFO. If the drift of each oscillator is in the direction opposite to the other, the input clock error appears larger than it is. If the drift is in the same direction, the error appears smaller. Thus, the LFO error can mask the input clock error. There is no solution to resolve this potential problem. It is possible that the system designer can choose crystals which are known to drift in opposite directions.

The frequency offset from nominal can be measured and compensated for in the factory using calibration. The remaining frequency error is due to temperature and aging effects. These errors cannot be removed, and set a bound on the accuracy of the frequency measurement.

The inherent measurement tolerance of the OCU is ±2 input clocks in any complete measurement sequence. The ***Inherent Measurement Tolerance*** figure illustrates the tolerance (input clock is called SYSCLK).

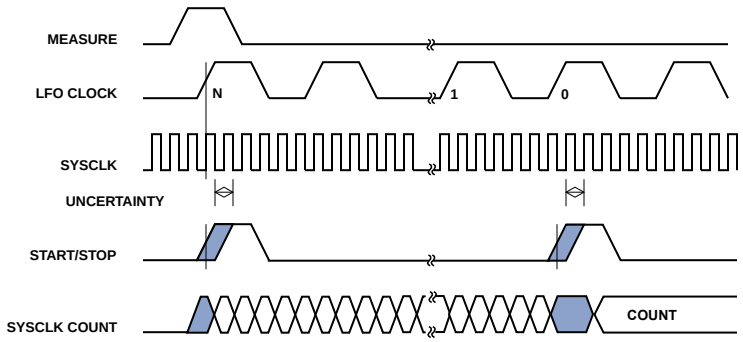


Figure 7-2: Inherent Measurement Tolerance

Clock Frequency Measurement Example

A frequency measurement depends on the following quantities:

- f_{SYS} - SYSCLK frequency; for example, 100 MHz
- f_{LFO} - LFO reference frequency; for example, 32.768 KHz
- $ftol_{SYS}$ - expected frequency tolerance of f_{SYS}
- $ftol_{LFO}$ - expected frequency tolerance of f_{LFO}
- COUNT_TIME - number of LFO cycles in the counting interval; for example, 32768 cycles = 1 second

The absolute time that elapses during COUNT_TIME LFO cycles is:

$$T = (COUNT\_TIME / f_{LFO}) \times (1 \pm ftol_{LFO}) \text{ seconds}$$

The SYSCLK count that is captured during a complete measurement, given the inherent ±2 SYSCLK cycle measurement uncertainty, is:

$$COUNT = int(T \times f_{SYS}) \times (1 \pm ftol_{SYS}) \pm 2 \text{ counts}$$

$$= int((COUNT\_TIME \times (f_{SYS} / f_{LFO}) \times (1 \pm ftol_{LFO}) \times (1 \pm ftol_{SYS})) \pm 2$$

For the frequencies and counting interval in the above example, the expected range of COUNT in the measurement is:

$$COUNT = int(32768 \times (10^8 / 32768) \times (1 \pm ftol_{LFO}) \times (1 \pm ftol_{SYS})) \pm 2$$

$$= int(10^8 \times (1 \pm ftol_{LFO}) \times (1 \pm ftol_{SYS})) \pm 200$$

If, for example, the frequency tolerances of the LFO and SYSCLK time bases were 100 ppm = 10^{-4} and 200 ppm = 2×10^{-4} respectively, then the expected range of COUNT is:

$$COUNT_{max} = int(10^8 \times (1 + ftol_{LFO}) \times (1 + ftol_{SYS})) + 2$$

$$= int(10^8 \times (1 + 10\text{-}4) \times (1 + 2\times10^{-4})) + 2$$

$$= int(100,000,000 \times 1.0001 \times 1.0002) +2$$

$= 100,030,002 + 2 = 100,030,004$

$COUNT_{min} = int(10^8 \times (1 - ftol_{LFO}) \times (1 - ftol_{SYS})) - 2$

$= int(100,000,000 \times 0.9999 \times 0.9998) -2$

$= 99,970,002 -2 = 99,970,000$

As can be seen from this example, the measurement tolerance of the OCU (±2 SYSCLKs) can be made negligible if sufficient time is allowed for the measurement interval. This measurement tolerance can be expressed as:

$tol_{OCU} = 2 \times (f_{LFO} / f_{SYS})/ COUNT\_TIME$

$\sim= 2 / (T \times f_{SYS})$

For T=1 second and f_{SYS} = 100 MHz, this is 0.02 ppm.

For T=10 milliseconds and f_{SYS} = 100 MHz, this is 2 ppm.

Dead Clock Detection

The OCU can detect dead or malfunctioning clocks on both the input clock and the LFO clock. A dead clock is a clock that is not switching, or is switching at a rate that is far outside the expected nominal frequency.

To detect a dead clock, a status request or acknowledge handshake is used. The request is synchronized from the reference clock domain to the target clock domain, and the acknowledge is synchronized back in the other direction. When the status request is issued, a timer begins counting down. If the synchronized acknowledge is not received before the timer reaches zero, a fault is asserted. The timer period is specified in a user-programmable register.

The input clock and the LFO clock monitor each other. The LFO serves as the reference for the input clock monitor. The input clock serves as the reference for the LFO clock monitor. This scheme is not robust for faults that affect both clocks.

Dead clock errors are captured in sticky status bits, one per monitored clock. They can be signaled to the system event controller. Dead input clock faults are also optionally signaled through a separate asynchronous output port which is not synchronized to either the input or system clocks. This asynchronous output handles the case in which the input clock *is* the system clock. In this case, the system event controller cannot be assumed to be functional.

The reference clock period determines the timeout delay of the dead clock monitor operations. This delay determines the frequency resolution of the monitor operations.

Calibration

For a given clock measurement time, there is an ideal clock count. This count assumes that a perfectly accurate clock exists. In reality, the two clock sources, a system clock and an LFO, have frequency errors from the nominal specifications. The combined frequency error can be measured by running a manual measurement cycle under controlled temperature and voltage conditions. For correct fault detection, the deviation from the ideal count must be included in the stored count limits. A manual measurement cycle is started by setting the MANUAL control bit (=1). Extra manual measurement cycles are disabled until that bit is brought low again.

The `OCU_STAT.BUSY` bit indicates that the measurement cycle is in-progress.

This method measures the combined frequency error. If the system clock frequency error can be measured in another way, then the LFO frequency error can be derived from OCU calibration data.

Clock Jitter

Jitter on both the input clock and the reference clock creates a frequency measurement error. The magnitude of the error depends on the magnitude and frequency of the clock jitter, as well as the length of the measurement period. High frequency jitter is averaged out over the measurement interval. Lower frequency clock drift is more likely to introduce frequency error, particularly when the measurement interval is short. In general, maximize the measurement interval to minimize the measurement error due to clock jitter. Jitter does not affect dead clock monitoring, assuming there is a reasonable reference counter interval.

OCU Operating Modes

By default, the OCU is in manual measurement mode. A continuous measurement mode can be enabled through the control register.

Manual Mode

In this mode, a clock measurement cycle is started when the `OCU_CTL.MEASURE` bit changes state from 0 to 1. Only one measurement cycle is run. The `OCU_CTL.MEASURE` bit must be cleared before another measurement cycle can be started. The LFO clock monitor cycle does not run while in this mode. Faults and errors are captured in sticky status bits in the `OCU_CTL` register.

Automatic Mode

In automatic mode, measurement cycles (including the LFO monitor cycle) are run continuously. Faults and errors are captured in the sticky status bits, as in manual mode.

OCU Event Control

The OCU generates an event output. This output is synchronous to the system clock. It is intended to be routed to the system event handler. Two asynchronous outputs exist to indicate clock faults in the absence of a functional system clock.

FAULT Synchronous Error

The error output signal `OCU_ERR` is an active HIGH level sensitive output. The fault event is triggered by one of the following:

- a frequency measurement out-of-range detection
- a dead clock detection on either dead clock monitor
- a frequency counter overflow

```
FAULT = ((FREQ_FAULT || OVERFLOW) && FREQ_FAULT_EN) || (CMON_FAULT &&
CMON_FAULT_EN) || (LMON_FAULT && LMON_FAULT_EN)
```

The event handler must clear the fault output by clearing the fault status bits.

Asynchronous Event Outputs

Two asynchronous outputs exist to communicate OCU events in the case where the system is unresponsive to synchronous events. The ocu_dead_clock signal indicates a dead clock condition on OCU_clkin. The OCU_CTL.DEAD_CLOCK_EN control bit gates this output. The ocu_async_fault output is the logic OR of the ocu_dead_clock output and the synchronous ocu_fault event output. Both these outputs are active LOW.

Unclocked Fault Response in System

The main purpose of the OCU is to measure the main internal system clock to a high-degree of accuracy. An error could indicate the system processors may be unable to properly respond to system clock problems. The system may be configured for the following responses in the absence of a reliable internal system clock.

Asserting the Fault pin. It is also possible to assert the Fault pin up on an OCU Fault due to asynchronous fault or a dead clock event, by programming the SYSBLK_FAULT_TRIPEN register.

GPIO Pin Safe state. The VMU can be configured to assert the Pin Safe State on a clock fault detected by the OCU, for both asynchronous faults and the dead clock state. The PADS_VMU_TRIPEN.OCU0FAULT and PADS_VMU_TRIPEN.OCU0CLK bits are used to enable the pin safe state trips for the OCU faults.

NOTE: The Fault pin is also asserted on a pin safe state generated from the OCU.

Clock Not Good reset. The SYSBLK_CLKNG_TRIPEN register can be used to program whether the CLKNG must be enabled for an Asynchronous Fault Trip or a Dead Clock. Once enabled and asserted, the chip would be put under reset and can only be recovered from its 'Safe State' by a hard reset or power cycle. Note that the CGU_OSCWDCTL.CNGEN must also be enabled in order for the safe reset condition to be forwarded from the OCU.

OCU Interrupt Signals

The OCU contains a level-sensitive active high interrupt called ocu_fault. This signal is intended to be connected to the system event controller or fault management unit.

Another output called ocu_dead_clock is an active low signal which is asynchronous to the input and APB clock domains. This signal can be used to indicate dead clock faults on the input clock in situations where the larger system is incapacitated due to a clock fault.

Both these outputs can be masked by bits in the OCU_CTL register. Refer to the functional description for more information.

OCU Programming Model

This section describes OCU interactions with other system functions. Software action is required to ensure that these interactions are safe.

Startup

Software must ensure that both the input clock and the LFO reference clock are running with a stable frequency before the OCU is enabled.

Interaction with CGU

The OCU expects to see continuously running clocks of a specific frequency. Clock generator (CGU) programming changes which alter the frequency of monitored clocks, or gate said clocks, can cause the OCU to generate spurious faults. Software must disable the OCU before modifying the CGU programming. Software must also ensure that the clocks are stable before enabling the OCU.

To disable the OCU prior to CGU programming changes:

1. Clear the `OCU_CTL.AUTO_EN` control bit to turn off continuous monitoring.
2. Poll the `OCU_STAT.BUSY` status bit until it is cleared.
3. Program the CGU.

CM41X_M4 OCU Register Descriptions

Oscillator Comparator Unit (OCU) contains the following registers.

Table 7-3: CM41X_M4 OCU Register List

Name	Description
OCU_CLKCNT	Frequency Measured Count Register
OCU_CMONCNT	CLKIN Monitor Reference Count Register
OCU_CTL	Control Register
OCU_LMONCNT	LFO Monitor Reference Count Register
OCU_MAXCNT	Maximum Count Register
OCU_MINCNT	Minimum Limit Register
OCU_REFCNT	Reference Count Register
OCU_STAT	Status Register

Frequency Measured Count Register

The OCU_CLKCNT register contains the final clock count after the count sequence has completed. The input clock counter is an up counter which starts at zero.

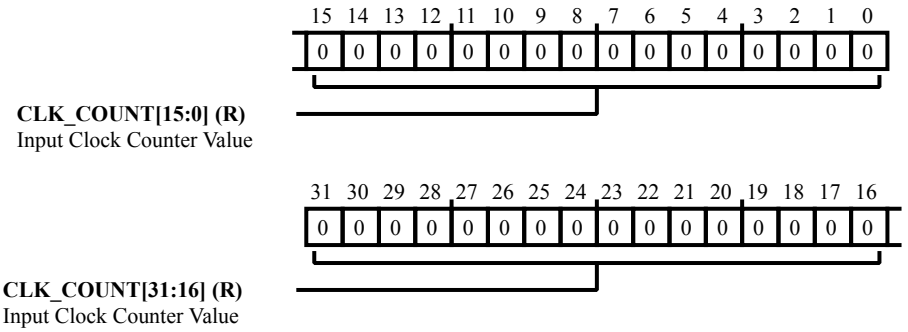


Figure 7-3: OCU_CLKCNT Register Diagram

Table 7-4: OCU_CLKCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CLK_COUNT	Input Clock Counter Value. The OCU_CLKCNT.CLK_COUNT bit field contains the clock counter value after a measurement.

CLKIN Monitor Reference Count Register

The OCU_CMONCNT register specifies the number of LFO periods in the OCU SYSCLK monitor cycle.

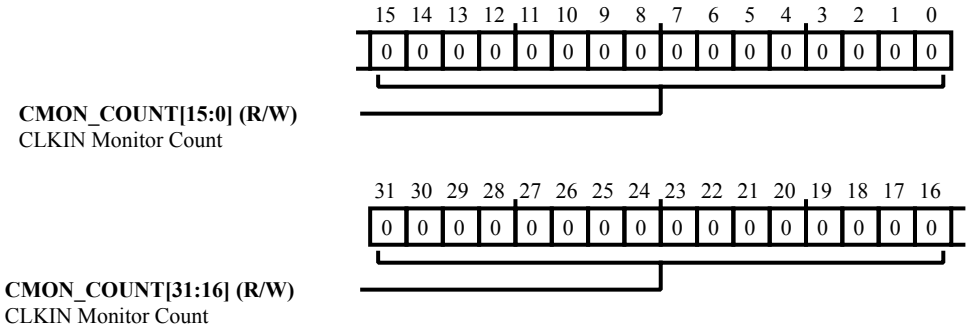


Figure 7-4: OCU_CMONCNT Register Diagram

Table 7-5: OCU_CMONCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CMON_COUNT	CLKIN Monitor Count. The OCU_CMONCNT.CMON_COUNT bit field specifies the number of LFO clocks in the input clock monitor cycle.

Control Register

The OCU_CTL register configures several interrupts as well as test and lock functions.

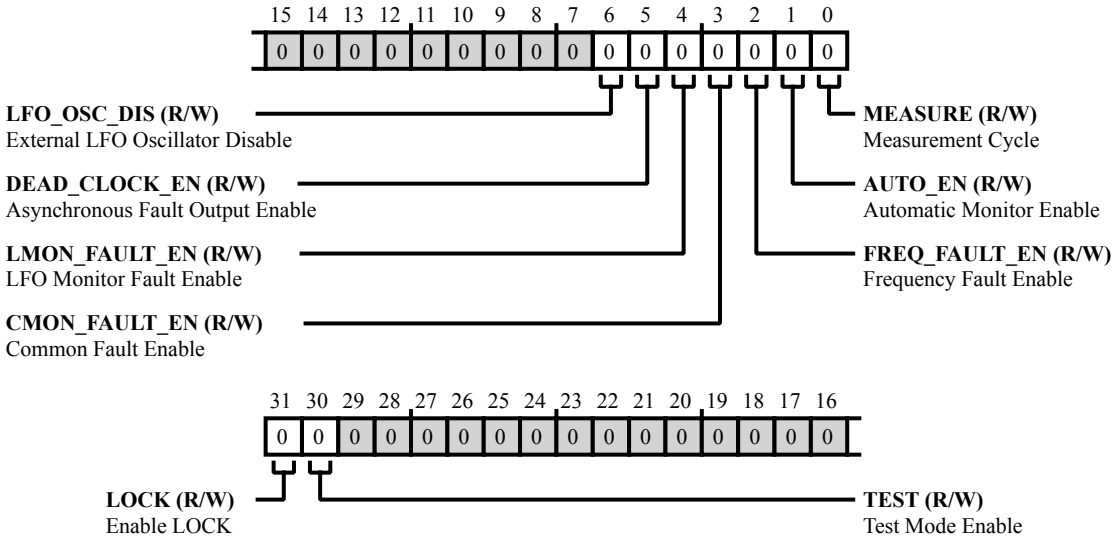


Figure 7-5: OCU_CTL Register Diagram

Table 7-6: OCU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Enable LOCK. When asserted the OCU_CTL.LOCK bit enables the register locking function.
30 (R/W)	TEST	Test Mode Enable. The OCU_CTL.TEST bit moves the counter overflow bit position from bit 31 to bit 8.
6 (R/W)	LFO_OSC_DIS	External LFO Oscillator Disable. The OCU_CTL.LFO_OSC_DIS bit drives ocu_osc_disable output signal. This bit has no other internal functionality.
5 (R/W)	DEAD_CLOCK_EN	Asynchronous Fault Output Enable. The OCU_CTL.DEAD_CLOCK_EN bit enables the ASYNC_FAULT output. This gates the ocu_async_fault output port.
4 (R/W)	LMON_FAULT_EN	LFO Monitor Fault Enable. The OCU_CTL.LMON_FAULT_EN bit enables the OCU_STAT.LMON_FAULT status onto interrupt output port.
3 (R/W)	CMON_FAULT_EN	Common Fault Enable. The OCU_CTL.CMON_FAULT_EN bit enables the OCU_STAT.CMON_FAULT status onto interrupt output port.

Table 7-6: OCU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	FREQ_FAULT_EN	Frequency Fault Enable. The `OCU_CTL.FREQ_FAULT_EN` bit enables the `OCU_STAT.FREQ_FAULT` status onto the interrupt output port.
1 (R/W)	AUTO_EN	Automatic Monitor Enable. The `OCU_CTL.AUTO_EN` bit enables automatic monitor mode. All monitors run continuously.
0 (R/W)	MEASURE	Measurement Cycle. The `OCU_CTL.MEASURE` bit controls whether to begin a manual measurement cycle on the low to high transition of this bit. This bit is ignored while in automatic mode.

LFO Monitor Reference Count Register

The OCU_LMONCNT register specifies the number of system clock periods in the OCU LFO monitor cycle.

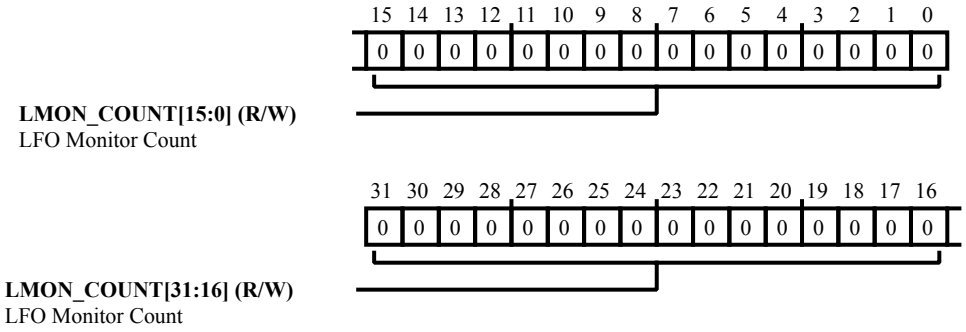


Figure 7-6: OCU_LMONCNT Register Diagram

Table 7-7: OCU_LMONCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	LMON_COUNT	LFO Monitor Count. The OCU_LMONCNT.LMON_COUNT bit field specifies the number of input clocks in the LFO monitor cycle.

Maximum Count Register

The OCU_MAXCNT register contains a software supplied upper count limit. If, on measurement, the OCU_CLKCNT register is more than this value, an out of bounds signal is generated.

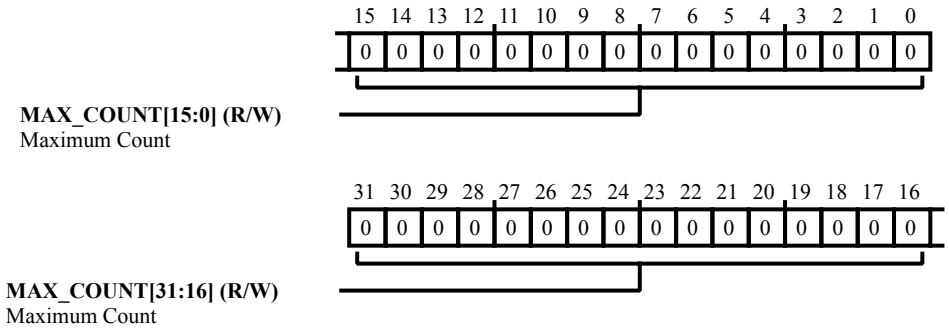


Figure 7-7: OCU_MAXCNT Register Diagram

Table 7-8: OCU_MAXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MAX_COUNT	Maximum Count. The OCU_MAXCNT.MAX_COUNT bit field contains the counter upper compare limit.

Minimum Limit Register

The OCU_MINCNT register contains a software supplied lower count limit. If, on measurement, the OCU_CLKCNT register is less than this value, an out of bounds signal is generated.

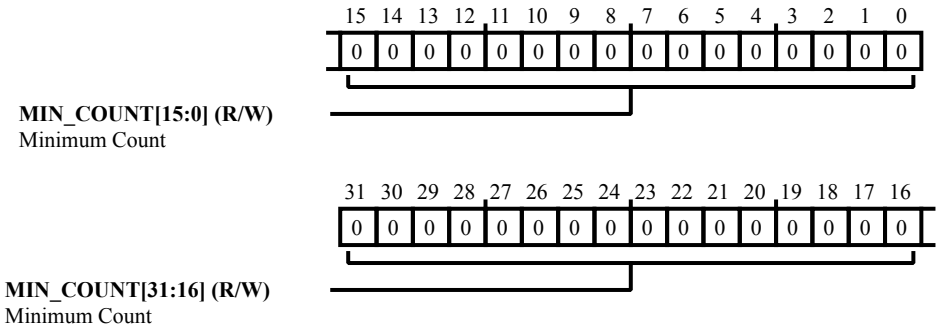


Figure 7-8: OCU_MINCNT Register Diagram

Table 7-9: OCU_MINCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MIN_COUNT	Minimum Count. The OCU_MINCNT.MIN_COUNT bit field contains the counter lower compare limit.

Reference Count Register

The OCU_REFCNT register specifies the number of LFO clock periods in a input clock measurement cycle

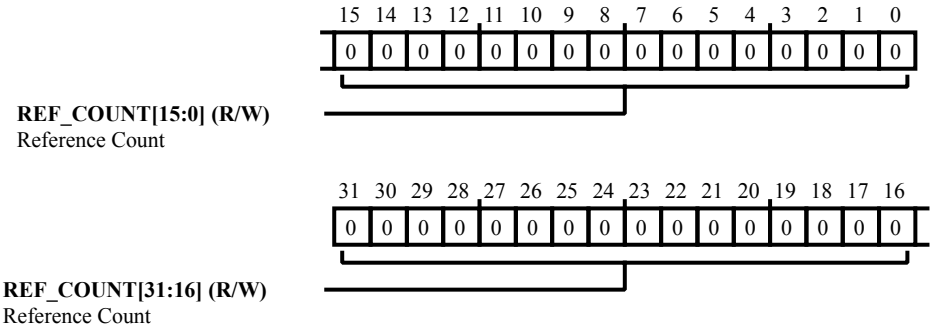


Figure 7-9: OCU_REFCNT Register Diagram

Table 7-10: OCU_REFCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	REF_COUNT	Reference Count. The OCU_REFCNT.REF_COUNT bit field specifies number of LFO clock periods in the input clock measure cycle.

Status Register

OCU status register

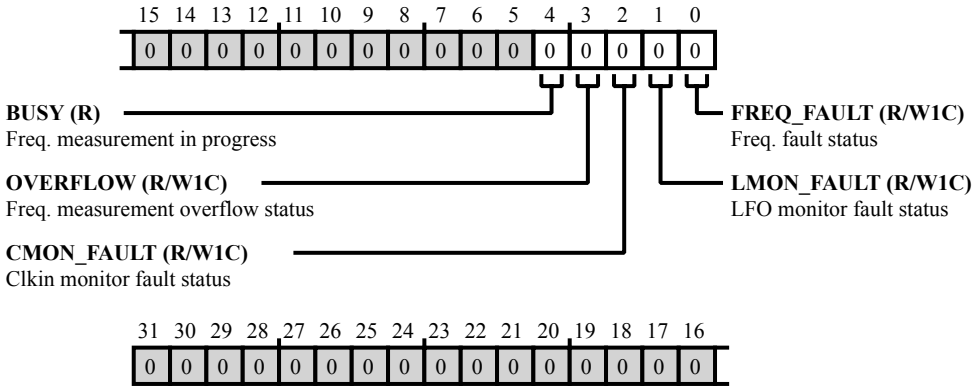


Figure 7-10: OCU_STAT Register Diagram

Table 7-11: OCU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/NW)	BUSY	Freq. measurement in progress. Input clock measurement cycle is in progress.
3 (R/W1C)	OVERFLOW	Freq. measurement overflow status. Sticky status bit set when the input clock counter overflows.
2 (R/W1C)	CMON_FAULT	Clkin monitor fault status. Sticky status bit set when a dead clock fault is detected on the input clock.
1 (R/W1C)	LMON_FAULT	LFO monitor fault status. Sticky status bit set when a dead clock fault is detected on LFO clock.
0 (R/W1C)	FREQ_FAULT	Freq. fault status. Sticky status bit set when a frequency fault is detected on input clock.

8 System Protection Unit (SPU)

The system protection unit (SPU) provides features for protecting system resources from errant reads and writes. A number of protection categories are available. Each category contains a set of registers for protection.

In a system with multiple system MMR masters, configurations of peripherals can be changed unintentionally leading to bad data or even system malfunctions. The peripherals are shared resources in the system. The SPU restricts access to certain MMRs, similar to the functionality of a semaphore.

SPU Features

The SPU has the following features:

- Write-protect and access protect system MMR from certain system masters and core masters.
- Simultaneously lock multiple peripheral configuration registers through a global lock mechanism.
- Write-protect and access protect and block access to its own write-protection registers from other system masters.

In the ADSP-CM41x, two instances of the SPU are provided.

- SPU1 controls the main peripheral bus in the SYSCLK domain.
- SPU0 controls the M0 Subsystem peripheral bus fabric in the SCLK0 domain.

There are three potential masters for both peripheral buses. The Cortex-M4 core, the Cortex-M0 core and the main system fabric.

SPU Functional Description

The following sections provide information on the function of the SPU.

CM41X_M4 SPU Register List

The System Protection Unit (SPU) provides a set of registers that can protect system resources from errant writes. The protection categories are global lock (protects configuration registers) and write protect register lock (protects the write protect register). For more information on SPU functionality, see the SPU register descriptions.

Table 8-1: CM41X_M4 SPU Register List

Name	Description
SPU_AP[n]	Access Protect Register n
SPU_CTL	Control Register
SPU_DEVID	Device Configuration Register
SPU_IADDR	Interrupt Address Register
SPU_IDTLS	Interrupt Details Register
SPU_STAT	Status Register
SPU_TIMEOUT	Timeout Register
SPU_WP[n]	Write Protect Register n

CM41X_M0 SPU Interrupt List

Table 8-2: CM41X_M0 SPU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
3	SPU0_TIMEOUT	SPU0 Timeout Interrupt	Level	
5	SPU1_TIMEOUT	SPU1 Timeout Interrupt	Level	
8	SPU0_INT	SPU0 Interrupt	Level	

CM41X_M4 SPU Interrupt List

Table 8-3: CM41X_M4 SPU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
12	SPU1_TIMEOUT	SPU1 Timeout Interrupt	Level	
27	SPU0_TIMEOUT	SPU0 Timeout Interrupt	Level	
45	SPU1_INT	SPU1 Interrupt	Level	
46	SPU0_INT	SPU0 Interrupt	Level	

Peripheral Register Write Protection

The SPU has a write-protection register (SPU_WP[n]) associated with each peripheral. Each of these write-protection registers has the exact same bits that correspond to a particular SMMR master (for example, Core 0, Core 1, MDMA). When the bits are set, the SPU locks the corresponding SMMR masters from accessing the register address space of the associated peripheral. The bits in the register can be cleared to allow access to the registers of the peripheral again. When the SPU intiates the write-protection register, any writes that are in-progress complete before the SPU blocks subsequent writes.

In the ***SPU Write Protect Registers*** figure, each write-protect register in the SPU is associated with a particular peripheral.

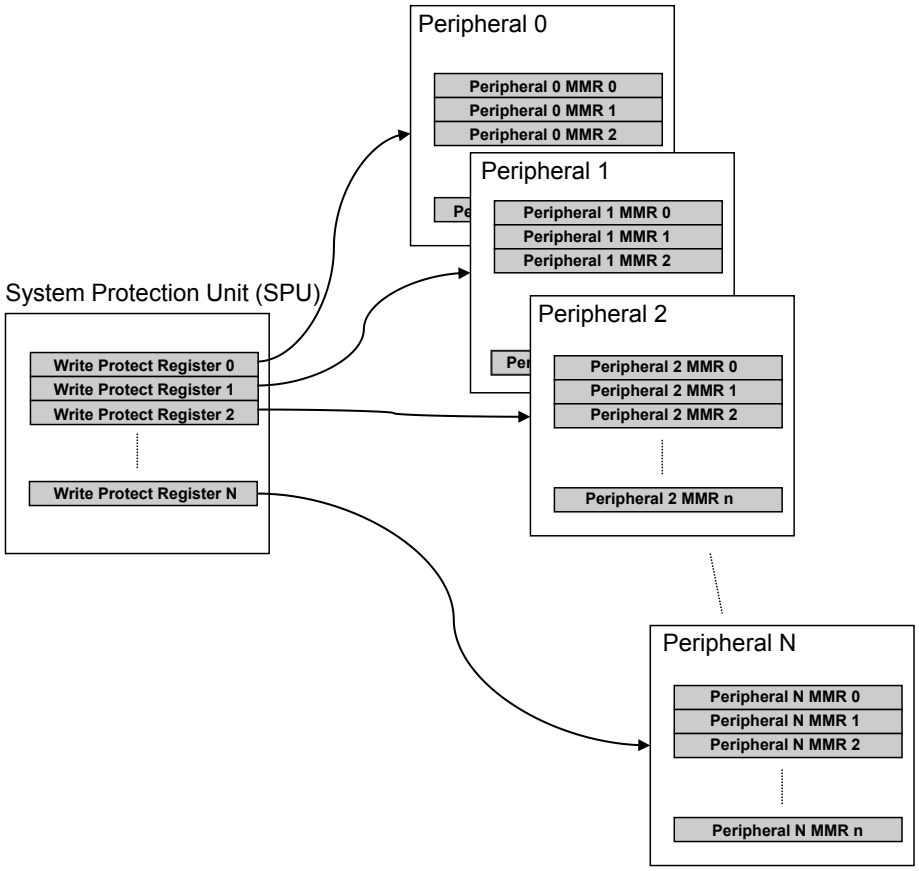


Figure 8-1: SPU Write Protect Registers

In the figure, a write-protect register in the SPU module blocks write-attempts to the MMR space of the associated peripheral. The bits in the write-protect register specify from which masters to block write-access.

NOTE: A SPU write protection register (`SPU_WP[n]`) exists for the SPU alone. If all defined bits are set in this register for the SPU, any configurations in the SPU are locked and cannot be changed. Only a system reset can restore access to the SPU.

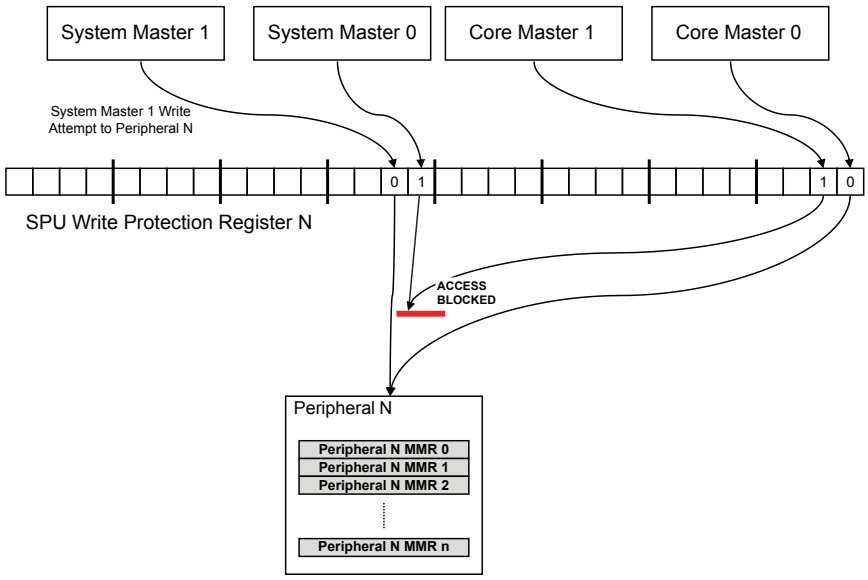


Figure 8-2: SPU Write-Protect Register Blocking Access from System Master 0 and Core Master 1

Global Locking

The SPU also has global locking capability. When enabled by setting `SPU_CTL.GLCK` bit field to a value other than 0xAD, a system-wide global lock signal is active. Some peripherals have a lock enable bit in their control register. When this bit is set, the peripheral recognizes the global lock signal and blocks further write-accesses to its own control register. Access to the configuration register of the peripheral is enabled when the global lock is turned off in the SPU.

The ***Global Locking*** figure is a conceptual diagram. The diagram shows how the SPU module (or any peripheral) blocks any write attempts to its control register when:

- The global lock signal from the SPU is active, and
- The global lock enable bit is set in the control register of the peripheral

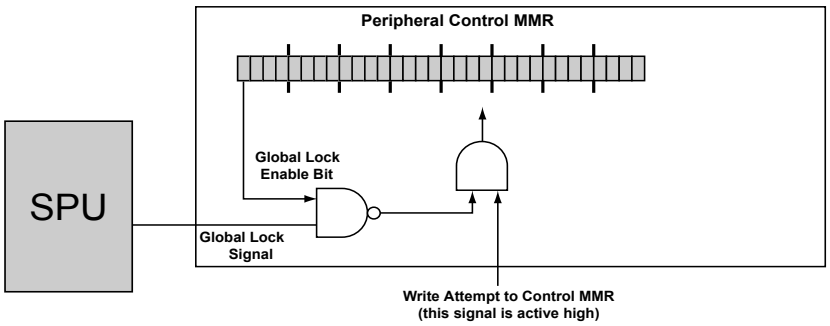


Figure 8-3: Global Locking

The SPU can write-protect its own registers. When the `SPU_CTL.WPLCK` bit is set and global locking is enabled, the SPU blocks accesses to the SPU write-protection registers. To enable write access to the write-protection registers in the SPU, disable the global locking.

Peripheral Register Access Protection

In the system, each access protect register (`SPU_AP[n]`) is assigned to a specific MMR address range associated with a specific peripheral. When the appropriate bits are set, reads and writes to the peripheral from a specific master are blocked and an error is returned to the master. For specific register assignments, see ADSP-CM41x Write-Protect and Secure Peripheral Registers.

SPU Block Diagram

The ***SPU System-Level Block Diagram*** shows a system-level block diagram of where the SPU is located in the system. It resides between the SMMR interface and the system crossbar. Depending on the configuration of the SPU write-protect registers, it can block access to some peripherals from certain SMMR masters.

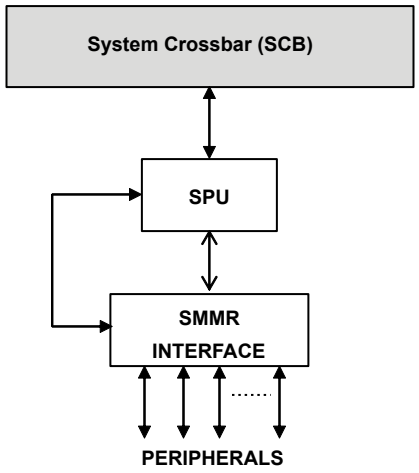


Figure 8-4: SPU System-Level Block Diagram

SPU Architectural Concepts

As shown in the block diagram, the SPU sits between the system crossbar (SCB) and the SMMR interface to the peripherals. The SPU gates any MMR access to any peripheral from any master that comes through the SCB. Depending on the configuration of the write-protection registers in the SPU, the SPU does or does not allow the MMR write to go through.

SPU Event Control

The system protection unit provides write-protection against MMRs peripherals and its own write-protect registers. If a write attempt is made to any locked MMR peripheral the SPU has write-protected, it blocks the write. The SPU generates a bus error to the master that attempted the write. That master does or does not generate an event, based on the returned error.

SPU Programming Model

The system protection unit (SPU) consists of write-protect and access-protect registers. Each one corresponds to a different peripheral instance. Bits in the write-protect registers correspond to system masters that can modify the MMR contents of the peripherals. By writing to these write-protect registers, the corresponding memory-mapped registers of the peripheral are write-protected against masters whose bits in the write-protect register are set.

The SPU globally locks the control register of the peripheral. Peripherals that support this feature have a lock enable bit in their control register. The peripheral blocks any additional write attempts to its control register from any master when:

- The global lock signal is active from the SPU, and
- The lock enable bit of the peripheral is set

If the lock enable bit of a peripheral is not set and the global lock signal is active, access to that control register of the peripheral is still allowed. To grant access again, disable the global lock signal from the SPU by writing the value 0xAD into the `SPU_CTL.GLCK` bit field.

Another protection mechanism that the SPU offers is write-protection against the write-protection registers. If the write protect register lock bit (`SPU_CTL.WPLCK`) is set and the global lock signal is active, writes to the write-protect registers of the SPU are blocked. To reenable access to the write-protect registers in the SPU, deactivate the global lock signal by writing 0xAD into the `SPU_CTL.GLCK` bit field.

Enabling and Disabling the SPU

The SPU is always operating. There are no bits to enable or disable the SPU. The SPU configuration can be updated at any time. Any ongoing transactions finish before a new configuration is in effect. By default, the SPU does not write-protect any of the MMRs.

Write-Protecting the SPU

The SPU is treated like any other peripheral in the system. As such, the SPU also has an associated write-protection register. If this write-protection register is configured to block all writes from all masters, any SPU configuration remains the same until the next system reset.

SPU Mode Configuration

The SPU can provide address range-wide protection by write-protecting the peripherals MMR address range from system MMR masters. It can also provide register wide protection using global locking. Peripherals that support this feature can enable it in their respective configuration register. When the SPU enables the global lock signal, all subsequent writes to the configuration register of the peripheral are blocked until the global lock signal is deasserted. Similarly, the write-protection registers of the SPU can be write-protected using the global lock signal as well. The SPU uses all these modes of operation together.

Locking Write-Protect Registers

Use the following steps to lock (write-protect) a register.

1. Set the SPU_CTL.WPLCK bit and configure the SPU_CTL.GLCK field to something other than 0xAD.

The SPU write-protect registers are blocked from further write accesses.

Protecting a Peripheral

Use the following procedure to protect a peripheral.

1. Determine which peripheral needs protection and locate the corresponding write-protect register (SPU_WP[n]) in the SPU. See the "Write-Protect and Secure Peripheral Registers" section.
2. Determine the SMMR masters from which the peripheral needs protection. Then, set the corresponding bit or bits in the write-protect register (SPU_WP[n]) for the peripheral. See the "Write-Protect and Secure Peripheral Registers" section.

After setting the write-protect register for the particular peripheral, the identified SMMR masters are blocked from writing to any MMR in the address space of the peripheral. This block remains in place until the bits in the write-protect register are cleared.

Access Protecting a Peripheral

Use the following procedure to access protect a peripheral. For more information, see ADSP-CM41x Write-Protect and Secure Peripheral Registers.

1. Determine which peripheral needs protection and locate the corresponding access-protect register (SPU_AP[n]) in the SPU.
2. Determine the SMMR masters from which the peripheral needs protection. Then, set the corresponding bit or bits in the write-protect register (SPU_AP[n]) for the peripheral. After setting the write-protect register for the particular peripheral, the identified SMMR masters are blocked from reading/writing to any MMR in the address space of the peripheral. This block remains in place until the bits in the access-protect register are cleared.

CM41X_M4 SPU Register Descriptions

System Protection Unit (SPU) contains the following registers.

Table 8-4: CM41X_M4 SPU Register List

Name	Description
SPU_AP[n]	Access Protect Register n
SPU_CTL	Control Register

Table 8-4: CM41X_M4 SPU Register List (Continued)

Name	Description
SPU_DEVID	Device Configuration Register
SPU_IADDR	Interrupt Address Register
SPU_IDTLS	Interrupt Details Register
SPU_STAT	Status Register
SPU_TIMEOUT	Timeout Register
SPU_WP[n]	Write Protect Register n

Access Protect Register n

In the system, each SPU_AP[n] register is assigned to a specific MMR address range associated with one peripheral. When the appropriate bits are set, read and writes access to the peripheral from a specific master are blocked and an error is returned to the master. For more information, see the processor specific additional information for the SPU_AP[n] register.

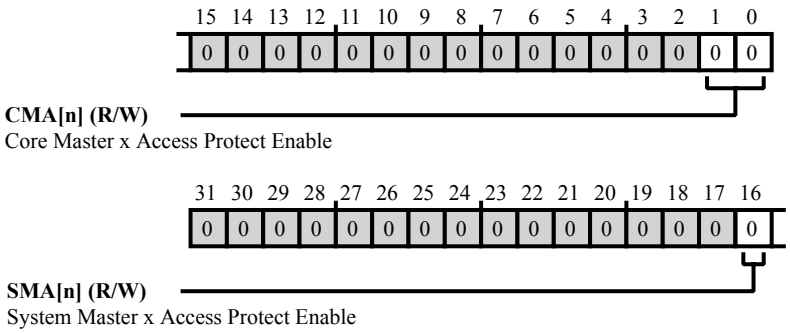


Figure 8-5: SPU_AP[n] Register Diagram

Table 8-5: SPU_AP[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	SMA[n]	System Master x Access Protect Enable. The bits in the SPU_AP[n].SMA[n] field correspond to different system masters in the system. When a particular bit is set in this field, the corresponding system master cannot read or write to the corresponding peripheral's MMR address space.
1:0 (R/W)	CMA[n]	Core Master x Access Protect Enable. The SPU_AP[n].CMA[n] bit field corresponds to different cores in the system. When a particular bit is set in this field, the corresponding core is not able to read or write to the corresponding peripheral's MMR address space. Core ID 0 = M0 Supervisor and Core ID 1 = M4 Controller.

Control Register

The SPU control register (SPU_CTL) provides a global lock for configuration registers as well as control for locking the write protect (SPU_WP[n]) registers. It also controls the generation of an interrupt to report blocked accesses.

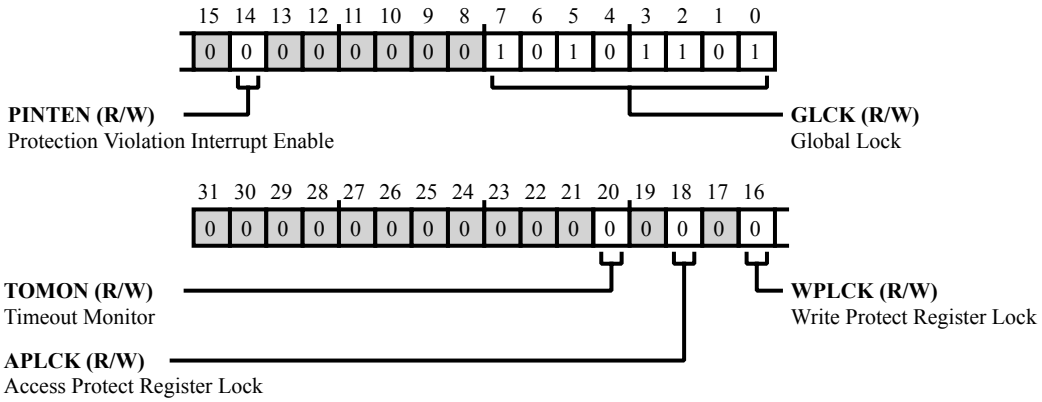


Figure 8-6: SPU_CTL Register Diagram

Table 8-6: SPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W)	TOMON	Timeout Monitor. The SPU_CTL.TOMON bit enables the timeout counter to monitor outstanding transaction timeout.	
		0	Disable
		1	Enable
18 (R/W)	APLCK	Access Protect Register Lock. When the SPU_CTL.APLCK bit is set in combination with the SPU_CTL.GLCK bit, writes to the access protect registers are blocked and return an error.	
		0	Disable
		1	Enable
16 (R/W)	WPLCK	Write Protect Register Lock. When the SPU_CTL.WPLCK bit is set in combination with the SPU_CTL.GLCK bit, writes to the SPU's write protect registers are blocked and return an error.	
		0	Disable
		1	Enable

Table 8-6: SPU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PINTEN	Protection Violation Interrupt Enable. When the `SPU_CTL.PINTEN` bit is set (=1), a block of any transaction according to the configured settings produces an interrupt.	
		0	Disable
		1	Enable
7:0 (R/W)	GLCK	Global Lock. The `SPU_CTL.GLCK` controls the global lock signal. The global lock signal provides register-based write protection. Writing 0xAD to this field disables the lock, and writing any other value enables the lock.	

Device Configuration Register

The SPU_DEVID register reports the hardware features that are supported on this device.

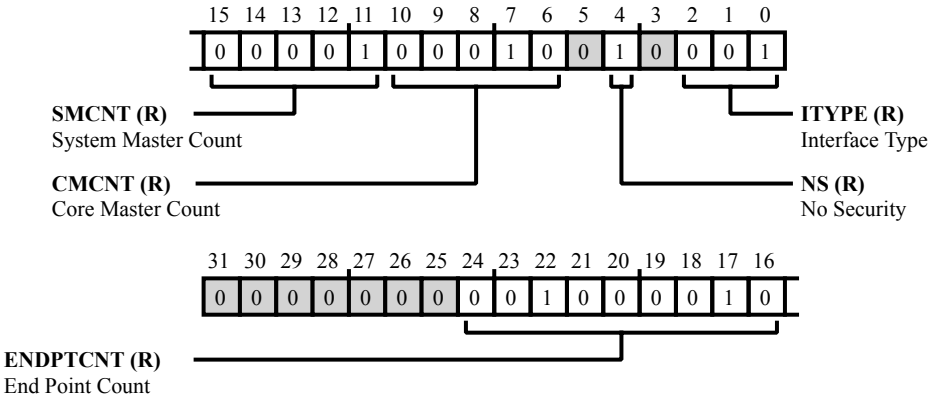


Figure 8-7: SPU_DEVID Register Diagram

Table 8-7: SPU_DEVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
24:16 (R/NW)	ENDPTCNT	End Point Count. The SPU_DEVID.ENDPTCNT bit field indicates the number of end points connected.	
15:11 (R/NW)	SMCNT	System Master Count. The SPU_DEVID.SMCNT bit field indicates the number of system masters connected.	
10:6 (R/NW)	CMCNT	Core Master Count. The SPU_DEVID.CMCNT bit field indicates the number of core masters connected.	
4 (R/NW)	NS	No Security. The SPU_DEVID.NS bit indicates the absence of security features. Registers related to security control (SPU_SECURE) and the message register are not available when security feature is absent.	
		0	Security
		1	No Security
2:0 (R/NW)	ITYPE	Interface Type. The SPU_DEVID.ITYPE bit indicates the interface type of the existing hardware implementation.	
		0	System Bus
		1	Peripheral Bus

Interrupt Address Register

The SPU violation address register (SPU_IADDR) reports the address of the first master that caused a security or protection violation. The value is set and remains sticky upon each assertion of VIRQ.

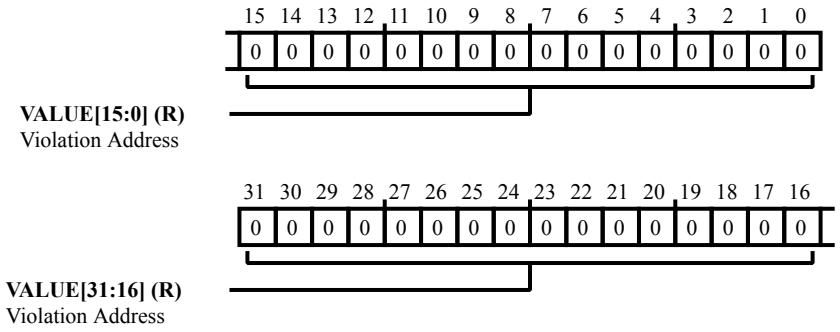


Figure 8-8: SPU_IADDR Register Diagram

Table 8-8: SPU_IADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Violation Address. The SPU_IADDR.VALUE reports the address of the first master that caused a security or protection violation. The value is set and remains sticky upon each assertion of VIRQ. Write any value to this field to clear.

Interrupt Details Register

The SPU interrupt details register (SPU_IDTLS) reports the details of the master that caused a security or protection violation. The value is set and remains sticky upon each assertion of VIRQ. If a second memory access violation occurs while the SPU_STAT.VIRQ bit is set, the SPU_IADDR and the SPU_IDTLS registers are not updated until the SPU_STAT.VIRQ bit is cleared.

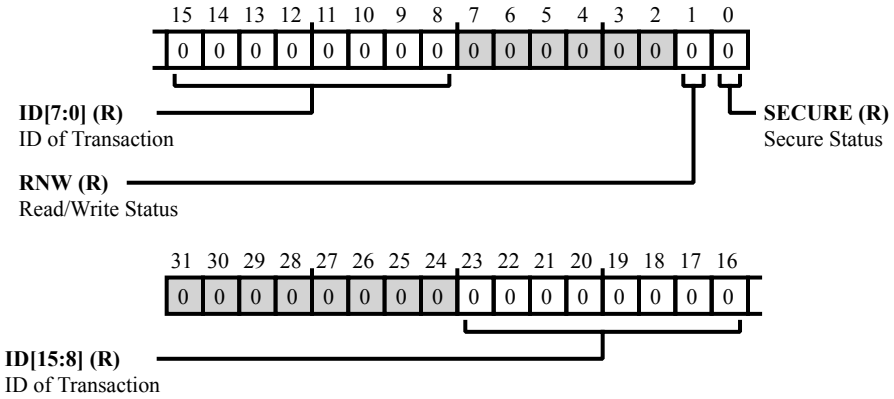


Figure 8-9: SPU_IDTLS Register Diagram

Table 8-9: SPU_IDTLS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:8 (R/NW)	ID	ID of Transaction. The SPU_IDTLS.ID reports the ID of the master that cause a security or protection violations.	
1 (R/NW)	RNW	Read/Write Status. The SPU_IDTLS.RNW reports the transaction type of the last signaled interrupt	
		0	Write
		1	Read
0 (R/NW)	SECURE	Secure Status. The SPU_IDTLS.SECURE reports the security status of the last interrupt	
		0	Non Secure
		1	Secure

Status Register

The SPU_STAT register indicates if there have been any errors, active interrupts and global lock status.

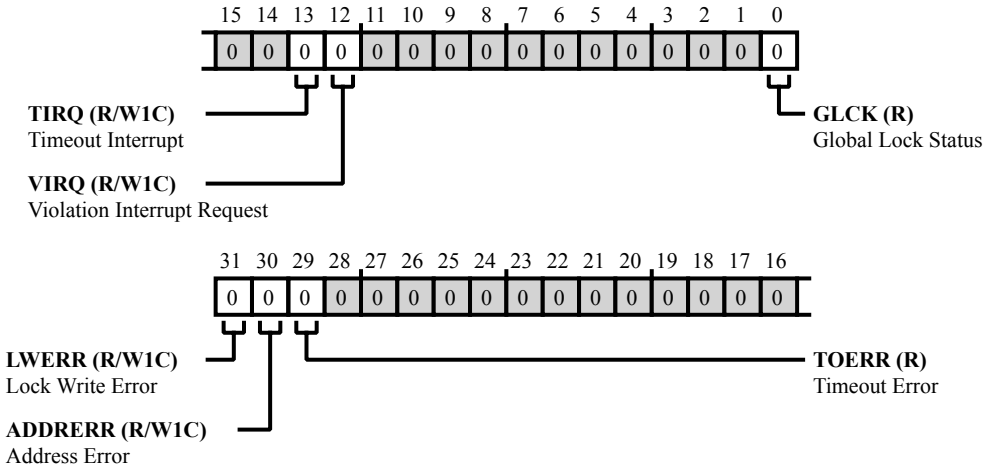


Figure 8-10: SPU_STAT Register Diagram

Table 8-10: SPU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	LWERR	Lock Write Error. The SPU_STAT.LWERR indicates whether there has been an attempted write to a register in the SPU with its lock bit (SPU_CTL.WPLCK or SCRLCK) set while SPU_CTL.GLCK was asserted. This bit is W1C.	
		0	Inactive
		1	Active
30 (R/W1C)	ADDRERR	Address Error. The SPU_STAT.ADDRERR indicates whether there has been an attempted write to a read-only register or an access an invalid address in the SPU MMR address range. This bit is W1C.	
		0	Inactive
		1	Active

Table 8-10: SPU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/NW)	TOERR	Timeout Error. The SPU_STAT.TOERR bit indicates a timeout transaction is pending. This bit is set by a timeout error. The SPU_STAT.TOERR bit should be cleared when the pending access is acknowledged by the target slave. Else, it is cleared when a subsequent access does not time out. In certain cases, when timeout is due to chip malfunctioning, subsequent access are not allowed to get through the SPU. Therefore, they cannot clear the bit. If the fault causes an infinite stall, the next access does not happen. The bit requires a full system reset to be cleared. If the fault causes a long, but finite stall, the next access happens and clears the bit.	
13 (R/W1C)	TIRQ	Timeout Interrupt. The SPU_STAT.TIRQ bit indicates a timeout transaction is pending. This bit is set by a timeout error. The SPU_STAT.TIRQ bit should be cleared when the pending access is acknowledged by the target slave.	
		0	Inactive
		1	Active
12 (R/W1C)	VIRQ	Violation Interrupt Request. The SPU_STAT.VIRQ bit indicates that a security and/or protection violation has been detected and interrupt asserted. This is a W1C bit.	
		0	Inactive
		1	Active
0 (R/NW)	GLCK	Global Lock Status. The SPU_STAT.GLCK indicates whether the global lock is enabled or disabled.	
		0	Disabled (global_lock=0)
		1	Enabled (global_lock=1)

Timeout Register

The SPU_TIMEOUT register contains the timeout counter value. Program the SPU_CTL.TOMON to enable transaction timeout monitoring. The TIRQ interrupt is triggered when the timeout period expires.

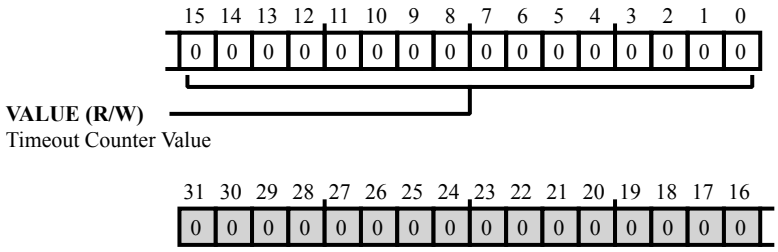


Figure 8-11: SPU_TIMEOUT Register Diagram

Table 8-11: SPU_TIMEOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timeout Counter Value. The SPU_TIMEOUT.VALUE bit field contains the timeout counter value. A read returns the programmed value only (not the current count).

Write Protect Register n

In the system, each SPU_WP[n] register is assigned to a specific MMR address range associated with one peripheral. When the appropriate bits are set, writes to the peripheral from a specific master are blocked and an error is returned to the master. For more information, see the processor specific additional information for the SPU_WP[n] register.

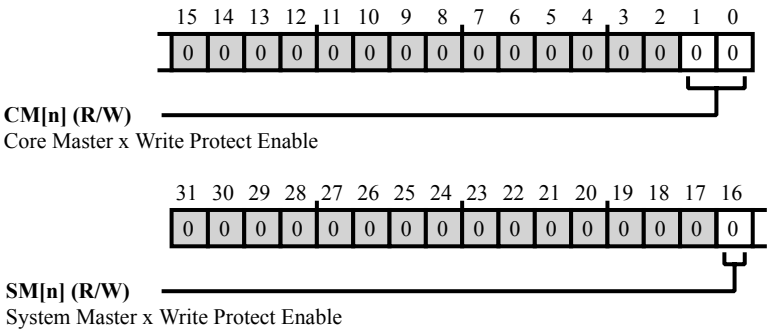


Figure 8-12: SPU_WP[n] Register Diagram

Table 8-12: SPU_WP[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	SM[n]	System Master x Write Protect Enable. The SPU_WP[n].SM[n] bits correspond to different system masters in the system. When a particular bit is set in this field, the corresponding system master cannot write to the corresponding peripheral's MMR address space. The write attempt is blocked by the SPU.
1:0 (R/W)	CM[n]	Core Master x Write Protect Enable. The SPU_WP[n].CM[n] bits correspond to different cores in the system. When a particular bit is set in this field, the corresponding core cannot write to the corresponding peripheral's MMR address space. The write attempt is blocked by the SPU. Core ID 0 = M0 Supervisor and Core ID 1 = M4 Controller.

ADSP-CM41x Write-Protect and Secure Peripheral Registers

The SPU consists of a collection of write-protect registers, each of which are associated with a specific peripheral or slave. The SPU also has a collection of secure peripheral registers which are also associated with specific peripherals. The SPU for the ADSP-CM41x is configured with 83 peripheral registers. The ***SPU_WPn Registers and SPU_SECUREPn and Related Peripherals*** table provides the write-protect register and secure peripheral number for each peripheral. The number for the peripheral correlates to both the write-protect register and the secure peripheral register.

Table 8-13: SPU0 Protection Registers Peripheral Assignment

Write Protection Register	Access Protection Register	Peripheral	Peripheral Description
REG_SPU0_WP2	REG_SPU0_AP2	PINT0	Pin Interrupt
REG_SPU0_WP3	REG_SPU0_AP3	PORTA	General Purpose Input/Output
REG_SPU0_WP4	REG_SPU0_AP4	SEC0	System Event Controller
REG_SPU0_WP5	REG_SPU0_AP5	WDOG0	Watch Dog Timer Unit
REG_SPU0_WP6	REG_SPU0_AP6	TRU0	Trigger Routing Unit
REG_SPU0_WP7	REG_SPU0_AP7	TIMER0	General-Purpose Timer Block
REG_SPU0_WP8	REG_SPU0_AP8	CAN0	Controller Area Network
REG_SPU0_WP9	REG_SPU0_AP9	SPI0	Serial Peripheral Interface
REG_SPU0_WP10	REG_SPU0_AP10	UART0	UART
REG_SPU0_WP11	REG_SPU0_AP11	ADCC0	ADC Controller
REG_SPU0_WP12	REG_SPU0_AP12	DMA0	DMA Channel
REG_SPU0_WP13	REG_SPU0_AP13	DMA1	DMA Channel
REG_SPU0_WP14	REG_SPU0_AP14	DMA2	DMA Channel
REG_SPU0_WP15	REG_SPU0_AP15	DMA3	DMA Channel
REG_SPU0_WP16	REG_SPU0_AP16	SMPU0	System Memo ry Protection Unit
REG_SPU0_WP17	REG_SPU0_AP17	SWU0	System Watchpoint Unit
REG_SPU0_WP18	REG_SPU0_AP18	SWU1	System Watchpoint Unit
REG_SPU0_WP19	REG_SPU0_AP19	SPU0	System Protection Unit

Table 8-14: SPU1 Protection Registers Peripheral Assignment

Write Protection Register	Access Protection Register	Peripheral	Peripheral Description
REG_SPU1_WP2	REG_SPU1_AP2	EMUID0	CHIP ID For ARM Emulators
REG_SPU1_WP3	REG_SPU1_AP3	PINT1	Pin Interrupt
REG_SPU1_WP4	REG_SPU1_AP4	PINT2	Pin Interrupt
REG_SPU1_WP5	REG_SPU1_AP5	PINT3	Pin Interrupt
REG_SPU1_WP6	REG_SPU1_AP6	PINT4	Pin Interrupt
REG_SPU1_WP7	REG_SPU1_AP7	PINT5	Pin Interrupt
REG_SPU1_WP8	REG_SPU1_AP8	PORTB	General Purpose Input/Output
REG_SPU1_WP9	REG_SPU1_AP9	PORTC	General Purpose Input/Output
REG_SPU1_WP10	REG_SPU1_AP10	PORTD	General Purpose Input/Output

Table 8-14: SPU1 Protection Registers Peripheral Assignment (Continued)

Write Protection Register	Access Protection Register	Peripheral	Peripheral Description
REG_SPU1_WP11	REG_SPU1_AP11	PORTE	General Purpose Input/Output
REG_SPU1_WP12	REG_SPU1_AP12	PORTF	General Purpose Input/Output
REG_SPU1_WP13	REG_SPU1_AP13	LBA	Logic Block Array
REG_SPU1_WP14	REG_SPU1_AP14	SEC1	System Event Control
REG_SPU1_WP15	REG_SPU1_AP15	WDOG1	Watchdog Timer
REG_SPU1_WP16	REG_SPU1_AP16	RCU0	Reset Control Unit
REG_SPU1_WP17	REG_SPU1_AP17	DPM0	Dynamic Power Management
REG_SPU1_WP18	REG_SPU1_AP18	CGU0	Clock Generation Unit
REG_SPU1_WP19	REG_SPU1_AP19	OCU0	Oscillator Control Unit
REG_SPU1_WP21	REG_SPU1_AP21	TAPC	
REG_SPU1_WP22	REG_SPU1_AP22	TRU1	Trigger Routing Unit
REG_SPU1_WP23	REG_SPU1_AP23	TIMER1	General-Purpose Timer
REG_SPU1_WP24	REG_SPU1_AP24	CNT	General-Purpose Counter
REG_SPU1_WP25	REG_SPU1_AP25	TWI0	Two-Wire Interface
REG_SPU1_WP26	REG_SPU1_AP26	CAN1	Controller Area Network
REG_SPU1_WP27	REG_SPU1_AP27	CPTMR0	Capture Timer
REG_SPU1_WP28	REG_SPU1_AP28	PWM0	Pulse-Width Modulator
REG_SPU1_WP29	REG_SPU1_AP29	PWM1	Pulse-Width Modulator
REG_SPU1_WP30	REG_SPU1_AP30	PWM2	Pulse-Width Modulator
REG_SPU1_WP32	REG_SPU1_AP32	TTU0	Trigger Timing Unit
REG_SPU1_WP33	REG_SPU1_AP33	SMCO	Static Memory Controller
REG_SPU1_WP34	REG_SPU1_AP34	SPI1	Serial Peripheral Interface
REG_SPU1_WP35	REG_SPU1_AP35	SPORT0	Serial Port
REG_SPU1_WP36	REG_SPU1_AP36	UART1	UART
REG_SPU1_WP37	REG_SPU1_AP37	UART2	UART
REG_SPU1_WP38	REG_SPU1_AP38	UART3	UART
REG_SPU1_WP39	REG_SPU1_AP39	UART4	UART
REG_SPU1_WP40	REG_SPU1_AP40	FFTB0	Fast Fourier Transform
REG_SPU1_WP41	REG_SPU1_AP41	SINC0	Sinus Cardinalis Filter
REG_SPU1_WP42	REG_SPU1_AP42	ADCC1	Analog-to-Digital Converter Controller
REG_SPU1_WP43	REG_SPU1_AP43	DACC0	Digital-to-Analog Converter Controller

Table 8-14: SPU1 Protection Registers Peripheral Assignment (Continued)

Write Protection Register	Access Protection Register	Peripheral	Peripheral Description
REG_SPU1_WP44	REG_SPU1_AP44	CRC0	Cyclic Redundancy Check
REG_SPU1_WP45	REG_SPU1_AP45	HAE0	Harmonic Analysis Engine
REG_SPU1_WP46	REG_SPU1_AP46	DMA4	DMA Channel
REG_SPU1_WP47	REG_SPU1_AP47	DMA5	DMA Channel
REG_SPU1_WP48	REG_SPU1_AP48	DMA6	DMA Channel
REG_SPU1_WP49	REG_SPU1_AP49	DMA7	DMA Channel
REG_SPU1_WP50	REG_SPU1_AP50	DMA8	DMA Channel
REG_SPU1_WP51	REG_SPU1_AP51	DMA9	DMA Channel
REG_SPU1_WP52	REG_SPU1_AP52	DMA10	DMA Channel
REG_SPU1_WP53	REG_SPU1_AP53	DMA11	DMA Channel
REG_SPU1_WP54	REG_SPU1_AP54	DMA12	DMA Channel
REG_SPU1_WP55	REG_SPU1_AP55	DMA13	DMA Channel
REG_SPU1_WP56	REG_SPU1_AP56	MBOX0_PORT0	MBOX Port 1 register map
REG_SPU1_WP57	REG_SPU1_AP57	MBOX0_PORT1	MBOX Port 0 register map
REG_SPU1_WP58	REG_SPU1_AP58	SMPU1	System Memory Protection Unit
REG_SPU1_WP59	REG_SPU1_AP59	SMPU2	System Memory Protection Unit
REG_SPU1_WP60	REG_SPU1_AP60	SWU2	System Watchpoint Unit
REG_SPU1_WP61	REG_SPU1_AP61	SWU3	System Watchpoint Unit
REG_SPU1_WP62	REG_SPU1_AP62	SWU4	System Watchpoint Unit
REG_SPU1_WP63	REG_SPU1_AP62	SWU5	System Watchpoint Unit
REG_SPU1_WP64	REG_SPU1_AP62	SWU6	System Watchpoint Unit
REG_SPU1_WP65	REG_SPU1_AP62	SPU1	System Protection Unit

9 Dynamic Power Management (DPM)

The dynamic power management (DPM) unit of the processor controls transitions between different power-saving modes.

DPM Features

The DPM allows programs to control the power mode of the processor as follows.

- Permits operation of multiple, external wake-up sources

DPM Functional Description

The DPM can be programmed to transition between power modes.

CM41X_M4 DPM Register List

A set of registers govern DPM operations. For more information on DPM functionality, see the DPM register descriptions.

Table 9-1: CM41X_M4 DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_PER_DIS0	Peripherals Disable Register 0
DPM_REVID	Revision ID
DPM_STAT	Status Register
DPM_WAKE_EN	Wakeup Enable Register
DPM_WAKE_POL	Wakeup Polarity Register
DPM_WAKE_STAT	Wakeup Status Register

DPM Definitions

To make the best use of the DPM, it is useful to understand the following terms.

CGU

Acronym for the clock generation unit (CGU), which is comprised of the PLL and PCU

DPM

Acronym for the dynamic power management (DPM) controller.

Full-on mode

The normal operating mode in which all clock domains are derived from the PLL.

PCU

Acronym for the PLL control unit (PCU).

PLL

Acronym for the phase-locked loop (PLL).

RCU

Acronym for the reset control unit (RCU).

DPM Operating Modes

The DPM includes several operating modes. The modes are:

- Reset
- Full-on

Reset State

Reset is the initial state of the processor and is the result of a hardware or software triggered event. The DPM itself does not trigger entering reset. The external SYS_HWRST pin or the RCU triggers entering reset. The DPM responds to reset by transitioning to its default state.

From Reset, the DPM always transitions to PLL Bypassed state.

Full-on Mode

In full-on mode, the processor can reach its maximum clock rate and power dissipation can be at its highest.

Deep Sleep Mode

To enter deep sleep mode, the processor sets the DPM_CTL.DEEPSLEEP bit. All processor cores are in idle state. The program software must guarantee that system transfers, including DMA, are stopped before each processor core goes into idle state and the processor enters deep sleep mode. In this state, power dissipation on the VDD_INT0 power domain is reduced by gating all the core and system clocks and disabling the PLL.

The enabled hardware wake-up signals or a hardware reset signal make the processor exit deep sleep mode. The DPM_WAKE_EN.WS[n] bits and DPM_WAKE_POL.WS[n] bits work together to determine which hardware wake-up signals are enabled and the polarity of the signals. Wake-up signal assertion is latched only when the signal is enabled. The enabled wake-up signal assertion occurring first is recorded in the DPM_WAKE_STAT register.

When a wake-up occurs, the DPM does the following:

- Signals a DPM event interrupt to the ARM Cortex-M4 processor.
- Transitions to Full-on mode
- Enables all clock domains that are not disabled in the CGU_SCBF_DIS register

The DPM event interrupt stays active until the program clears any bits that are set in the DPM_WAKE_STAT register. The DPM event interrupt is the first indication that the processor has exited Deep Sleep.

One option for waking up the core is to enable the CGU event interrupt, which asserts after the PLL locks.

Another option is to use the DPM event interrupt to make a core exit idle and to enable the corresponding core clock buffer.

DPM Event Control

The DPM event is triggered when an enabled wake-up is asserted. The DPM generates bus errors when a misaligned access to a register occurs. It also generates errors when an attempt is made to access unused DPM address space or a write-protected register.

DPM Events

The DPM event interrupt is triggered when any bit in the DPM_STAT register is set, indicating that an enabled wake-up was asserted. The DPM event interrupt stays active until the user clears any bits that are set in the DPM_STAT register.

DPM Errors

The DPM generates a bus error when a read or write transaction is attempted to an unused address within the DPM address range. It also generates a bus error when a misaligned access is made to a DPM register. In addition to the bus error, the DPM sets the DPM_STAT.ADDRERR bit.

If a write to a write-protected DPM register is attempted, the DPM generates a bus error. In addition, the DPM sets the DPM_STAT.LWERR bit.

DPM Programming Model

Configuring Deep Sleep Mode

The mode for deep sleep gates all core and system clocks to save power.

The processor enters deep sleep mode from any state in which software can run. Reading the DPM_STAT.CURMODE field reveals the current power mode. Clocks do not stop immediately after entry to deep sleep mode is requested, but no further action is required to guarantee that the mode transition occurs.

The processor cores must be idle before the clocks are shut down.

1. Enable the DPM event interrupt to wake up the desired core, directing exit from idle after exit from deep sleep mode.
2. Set the polarity of wake-up sources as needed with the DPM_WAKE_POL.WS[n] bits.
3. Enable the wake-up sources as needed with the DPM_WAKE_EN.WS[n] bits.
4. Set the DPM_CTL.DEEPSLEEP bit.
5. Clear all pending core transactions, DMA transactions, and interrupts. If applicable, use a system synchronization instruction.
6. Place all processor cores in idle state.

The processor is now in deep sleep mode. To wake the processor, assert any of the enabled wake-up sources.

CM41X_M4 DPM Register Descriptions

Dynamic Power Management (DPM) contains the following registers.

Table 9-2: CM41X_M4 DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_PER_DIS0	Peripherals Disable Register 0
DPM_REVID	Revision ID
DPM_STAT	Status Register
DPM_WAKE_EN	Wakeup Enable Register
DPM_WAKE_POL	Wakeup Polarity Register
DPM_WAKE_STAT	Wakeup Status Register

Control Register

The DPM_CTL register controls sleep modes selections and PLL operations of the DPM. A write protect feature permits locking out changes to this register.

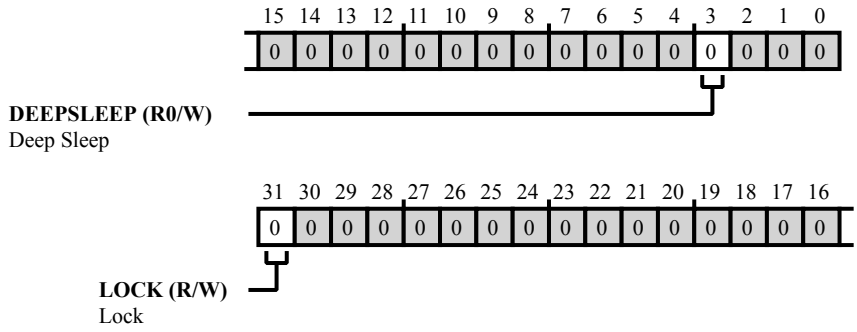


Figure 9-1: DPM_CTL Register Diagram

Table 9-3: DPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the DPM_CTL.LOCK bit is set, the DPM_CTL register is read only (locked).	
		0	Unlock
		1	Lock
3 (R0/W)	DEEPSLEEP	Deep Sleep. The DPM_CTL.DEEPSLEEP bit puts the processor into deep sleep mode. The processor stays in this mode until a wakeup event occurs. For more information about DPM modes, see the functional description.	
		0	No Action
		1	Deep Sleep

Peripherals Disable Register 0

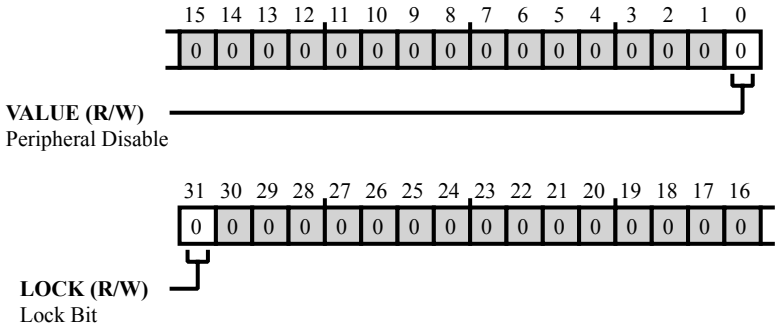


Figure 9-2: DPM_PER_DIS0 Register Diagram

Table 9-4: DPM_PER_DIS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Bit.
0 (R/W)	VALUE	Peripheral Disable.

Revision ID

The DPM_REVID register provides the revision of the DPM module.

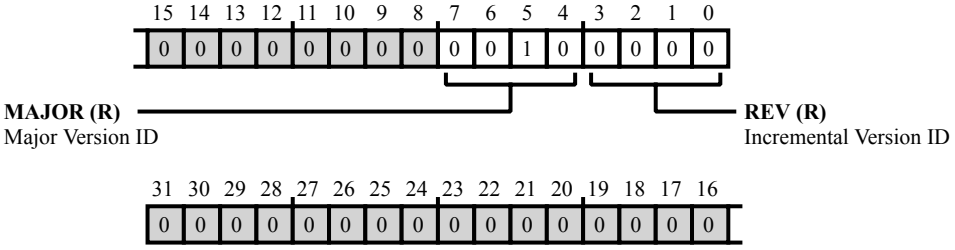


Figure 9-3: DPM_REVID Register Diagram

Table 9-5: DPM_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

Status Register

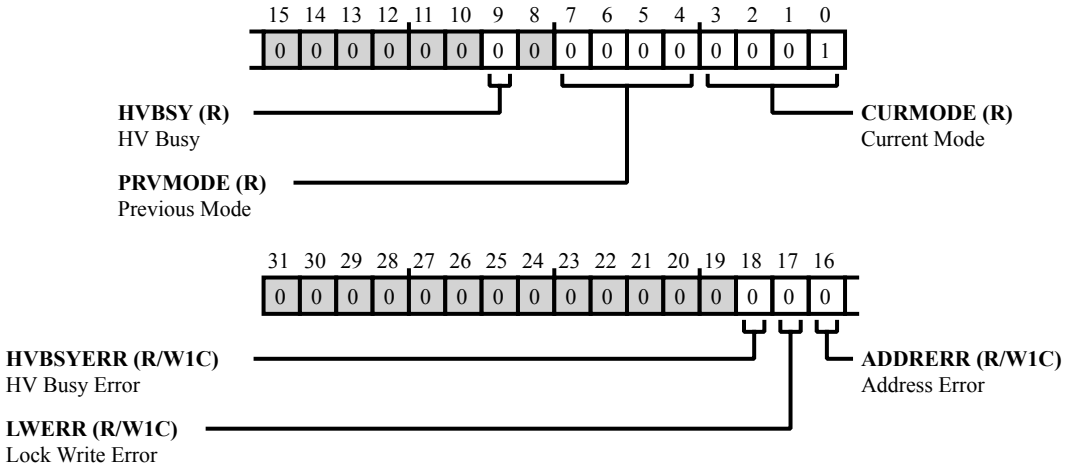


Figure 9-4: DPM_STAT Register Diagram

Table 9-6: DPM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W1C)	HVBSYERR	HV Busy Error. Reading registers during restore of DPM-LV from DPM-HV.	
		0	Inactive
		1	Active
17 (R/W1C)	LWERR	Lock Write Error.	
		0	Inactive
		1	Active
16 (R/W1C)	ADDRERR	Address Error.	
		0	Inactive
		1	Active
9 (R/NW)	HVBSY	HV Busy.	
		0	Not Busy (ready)
		1	Busy
7:4 (R/NW)	PRVMODE	Previous Mode.	
		0	Reset
		1	Full-On
		2	Reserved

Table 9-6: DPM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	Reserved
		4	Deep Sleep
		5-15	Reserved
3:0 (R/NW)	CURMODE	Current Mode.	
		0	Reserved
		1	Full-On
		2	Reserved
		3	Reserved
		4-15	Reserved

Wakeup Enable Register

The DPM_WAKE_EN register enables the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wake-up source 1, and so on. This register includes a write protection lock. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

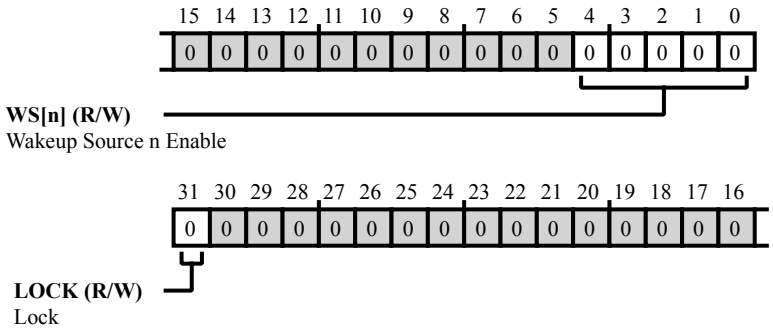


Figure 9-5: DPM_WAKE_EN Register Diagram

Table 9-7: DPM_WAKE_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the DPM_WAKE_EN.LOCK bit is set, the DPM_WAKE_EN register is read only (locked).	
		0	Unlock
		1	Lock
4:0 (R/W)	WS[n]	Wakeup Source n Enable. The DPM_WAKE_EN.WS[n] bits enable wakeup sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	

Wakeup Polarity Register

The DPM_WAKE_POL register select polarity (active high or low) of the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. This register includes a write protection lock. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

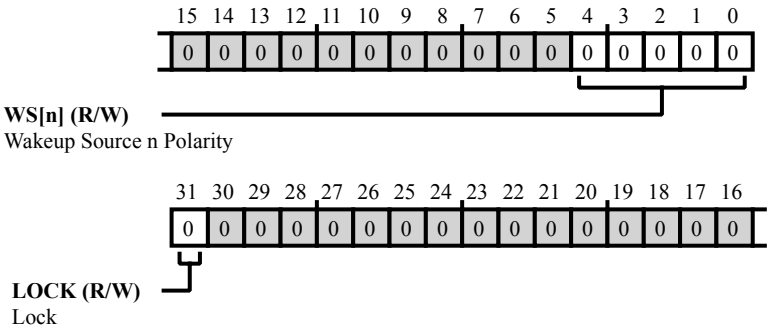


Figure 9-6: DPM_WAKE_POL Register Diagram

Table 9-8: DPM_WAKE_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the DPM_WAKE_POL.LOCK bit is set, the DPM_WAKE_POL register is read only (locked).	
		0	Unlock
		1	Lock
4:0 (R/W)	WS[n]	Wakeup Source n Polarity. The DPM_WAKE_POL.WS[n] bits select polarity (active high or low) of wakeup sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	
		0	Low Active Wakeup
		31	High Active Wakeup

Wakeup Status Register

The DPM_WAKE_STAT register indicates the enabled and active status of the wakeup event sources for exiting deep sleep mode. The number of wakeup sources varies with the processor design, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.

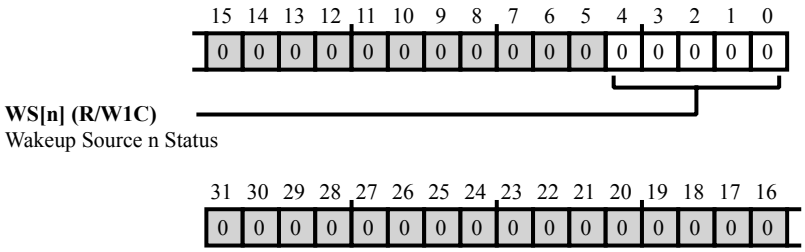


Figure 9-7: DPM_WAKE_STAT Register Diagram

Table 9-9: DPM_WAKE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4:0 (R/W1C)	WS[n]	Wakeup Source n Status. The DPM_WAKE_STAT.WS[n] bits indicate the enabled and active status of wake-up sources for exiting deep sleep mode, with bit 0 corresponding to wakeup source 0, bit 1 corresponding to wakeup source 1, and so on. For information about wakeup source assignments, see the DPM Wakeup Sources topic.	
		0	No Status
		31	Enabled and Active

10 System Event Controller (SEC)

System event management is the responsibility of the system event controller (SEC). Both M4P and M0P support a tightly integrated controller called the Nested Vectored Interrupt Controller (NVIC). SECs also feature a System Fault Interface (SFI) to perform efficient fault management.

All of the peripheral interrupts are routed using a single SEC interrupt to the desired core. The SEC allows programmability of the peripheral interrupt's priority, supporting up to 256 priority levels that are arbitrated within the SEC itself. The SEC also allows these interrupts to be grouped and masked by priority level and provides the flexibility to choose which core(s) the interrupt is routed to.

The SEC also supports self-nesting of interrupts, which is required when sharing a single interrupt request to an individual core, as this allows for a higher-priority peripheral interrupt to be passed to the core while it is currently servicing a lower-priority peripheral interrupt. For more information, refer to "Self-Nesting Mode for System Event Controller Interrupt (SECI)" in the *SHARC+ Core Programming Reference*.

For more information about the ARM NVIC, visit the ARM Information Center.

SEC Features

The following list describes the system event controller features.

Cortex M4 NVIC

The Cortex M4 NVIC supports:

- A total of 164 interrupts.
- A programmable priority level, with 4 bits of implementation in Interrupt Priority Registers.
- Level and pulse detection of interrupt signals.
- Dynamic re prioritizing of interrupts.
- Grouping of priority values into group priority and sub priority fields.
- Interrupt tail-chaining
- An external Non maskable Interrupt (NMI).

- Fault action configuration, timeout, external indication, and system reset
- Fault from an external event through the GPIO

Cortex M0 NVIC

The Cortex M0 NVIC supports:

- A total of 32 user interrupts.
- A programmable priority level, with 2 bits of implementation in Interrupt Priority Registers.
- Level and pulse detection of interrupt signals.
- Interrupt tail-chaining.
- An external NMI.

SEC Functional Description

The following sections provide a functional description of the SEC.

CM41X_M4 SEC Register List

The System Event Controller (SEC) manages the system fault sources, including control features such as enable/disable, priority, and active/pending source status. For more information on SEC functionality, see the SEC register descriptions.

Table 10-1: CM41X_M4 SEC Register List

Name	Description
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_FCTL	Fault Control Register
SEC_FDLY	Fault Delay Register
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FEND	Fault End Register
SEC_FSID	Fault Source ID Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FSTAT	Fault Status Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register

Table 10-1: CM41X_M4 SEC Register List (Continued)

Name	Description
SEC_SCTL[n]	Source Control Register n
SEC_SSTAT[n]	Source Status Register n

CM41X_M4 SEC Interrupt List

Table 10-2: CM41X_M4 SEC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
32	SEC1_ERR	SEC1 Error	Level	
33	SEC0_ERR	SEC0 Error	Level	

CM41X_M4 Interrupt List

Table 10-3: CM41X_M4 Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
-15	RESET	SCS0 RESET		
-14	NonMaskableInt	SCS0 NonMaskable Interrupt		
-13	HardFault	SCS0 Hard Fault		
-12	MemoryManagement	SCS0 Memory Management		
-11	BusFault	SCS0 Bus Fault	Edge	
-10	UsageFault	SCS0 Usage Fault		
-5	SVCall	SCS0 Service Call		
-4	DebugMonitor	SCS0 Debug Monitor		
-2	PendSV	SCS0 Pending Service		
-1	SysTick	SCS0 System Time Tick		
0	VMU0_VDDINT_EVT	VMU0 Internal voltage management fault	Level	
1	VMU0_VDDEXT_EVT	VMU0 External voltage management fault	Level	
2	Reserved	Reserved	Reserved	Reserved
3	CGU0_OSCW_INT	CGU0 Oscillator Watchdog Interrupt	Level	
4	OCU0_ERR	OCU0 Freq. out-of-range, or dead clock, or counter overflow detected	Level	
5	M4P_BCODE_ERR	M4P Boot Code CRC Error	Level	

Table 10-3: CM41X_M4 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
6	CTI0_EVT0	CTI0 Cross Trigger Interface Event 0	Level	
7	CTI0_EVT1	CTI0 Cross Trigger Interface Event 1	Level	
8	CTI0_EVT2	CTI0 Cross Trigger Interface Event 2	Level	
9	M4P_LOCKUP	M4P Lockup Error (Fault only; not an interrupt)	Level	
10	M4P_BUS_FAULT	M4P Bus Fault	Edge	
11	M0P_LOCKUP	M0P m0 supervisor lockup error	Level	
12	SPU1_TIMEOUT	SPU1 Timeout Interrupt	Level	
13	C1_BUS_TIMEOUT	SYSBLK1 Bus Timeout	Level	
14	M4P_CORE_SRAM_EERR	M4P SRAM Core ECC Error	Level	
15	M4P_DMA_SRAM_EERR	M4P SRAM DMA ECC Error	Level	
16	FLC0_CORE_FLASH_EERR	FLC0 Flash Core ECC Error	Level	
17	FLC0_DMA_FLASH_EERR	FLC0 Flash DMA ECC Error	Level	
18	FLC0_FLASH_EVT	FLC0 Flash Event	Level	
19	M0P_SRAM_EERR	M0P supervisor sram ecc error	Level	
20	MBOX0_PORT1_EERR	MBOX0 Port 1 ECC error	Level	
21	MBOX0_PORT0_EERR	MBOX0 Port 0 ECC error	Level	
22	FFTB0_SPERR	FFTB0 Parity error in the limit buffer (LBUFF), or the window buffer (WNDBUFF). If this interrupt request is asserted, ignore the result of this spectrum capture sequence and rewrite both LBUFF and WNDBUFF.	Level	
23	FFTB0_DPERR	FFTB0 Data parity error	Level	
24	HAE0_PERR	HAE0 Parity Error	Level	
25	CSTF0_PERR	CSTF0 Parity Error Detected	Level	
26	Reserved	Reserved	Reserved	Reserved
27	SPU0_TIMEOUT	SPU0 Timeout Interrupt	Level	
28	C0_BUS_TIMEOUT	SYSBLK0 Bus Timeout	Level	
29	SYSBLK0_POSTWR_ERR	SYSBLK0 Posted Write Error	Level	
30	CGU0_EVT	CGU0 Event	Edge	
31	DPM0_EVT	DPM0 Event	Level	
32	SEC1_ERR	SEC1 Error	Level	

Table 10-3: CM41X_M4 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
33	SEC0_ERR	SEC0 Error	Level	
34	WDOG1_EXP	WDOG1 Expiration	Level	
35	WDOG0_EXP	WDOG0 Expiration	Level	
36	TAPC_KEYFAIL	TAPC User Key Fail Interrupt	Edge	
37	AFE_NOTOK	AFE is functioning normally	Edge	
38	AFE_LIMIT	AFE FOCP Limit Detection	Level	
39	PWM0_TRIP	PWM0 Trip	Level	
40	PWM1_TRIP	PWM1 Trip	Level	
41	PWM2_TRIP	PWM2 Trip	Level	
42	SMPU1_ERR	SMPU1 Error Interrupt	Level	
43	SMPU2_ERR	SMPU2 Error Interrupt	Level	
44	SMPU0_ERR	SMPU0 Error Interrupt	Level	
45	SPU1_INT	SPU1 Interrupt	Level	
46	SPU0_INT	SPU0 Interrupt	Level	
47	SWU2_EVT	SWU2 Event	None	
48	SWU3_EVT	SWU3 Event	None	
49	SWU4_EVT	SWU4 Event	None	
50	SWU5_EVT	SWU5 Event	None	
51	SWU6_EVT	SWU6 Event	None	
52	SWU0_EVT	SWU0 Event	None	
53	SWU1_EVT	SWU1 Event	None	
54	CPTMR0_CPT0_ERR	CPTMR0 CPT 0 Error Interrupt	Level	
55	CPTMR0_CPT1_ERR	CPTMR0 CPT 1 Error Interrupt	Level	
56	CPTMR0_CPT2_ERR	CPTMR0 CPT 2 Error Interrupt	Level	
57	CRC0_ERR	CRC0 Error	Level	
58	SPI1_ERR	SPI1 Error	Level	
59	SPI0_ERR	SPI0 Error	Level	
60	FFTB0_LIMERR	FFTB0 One or more points in the power spectrum output exceeded the predefined limit in the limit buffer.	Level	
61	MATH0_MSTAT	MATH0 MATH unit Accumulate Interrupt Status	Level	

Table 10-3: CM41X_M4 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
62	ADCC1_ERR	ADCC1 ADC Error	Level	
63	DACC0_ERR	DACC0 DAC Error	Level	
64	ADCC0_ERR	ADCC0 ADC Error	Level	
65	DMA4_ERR	DMA4 Error interrupt	Level	
66	DMA5_ERR	DMA5 Error interrupt	Level	
67	DMA6_ERR	DMA6 Error interrupt	Level	
68	DMA7_ERR	DMA7 Error interrupt	Level	
69	DMA8_ERR	DMA8 Error interrupt	Level	
70	DMA9_ERR	DMA9 Error interrupt	Level	
71	DMA10_ERR	DMA10 Error interrupt	Level	
72	DMA11_ERR	DMA11 Error interrupt	Level	
73	DMA12_ERR	DMA12 Error interrupt	Level	
74	DMA13_ERR	DMA13 Error interrupt	Level	
75	DMA0_ERR	DMA0 Error interrupt	Level	
76	DMA1_ERR	DMA1 Error interrupt	Level	
77	DMA2_ERR	DMA2 Error interrupt	Level	
78	DMA3_ERR	DMA3 Error interrupt	Level	
79	PWM0_SYNC	PWM0 PWMTMR Grouped	Edge	
80	PWM1_SYNC	PWM1 PWMTMR Grouped	Edge	
81	PWM2_SYNC	PWM2 PWMTMR Grouped	Edge	
82	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level	
83	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level	
84	PINT3_BLOCK	PINT3 Pin Interrupt Block	Level	
85	PINT4_BLOCK	PINT4 Pin Interrupt Block	Level	
86	PINT5_BLOCK	PINT5 Pin Interrupt Block	Level	
87	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level	
88	TIMER1_TMR0	TIMER1 Timer 0	Level	
89	TIMER1_TMR1	TIMER1 Timer 1	Level	
90	TIMER1_TMR2	TIMER1 Timer 2	Level	
91	TIMER1_TMR3	TIMER1 Timer 3	Level	
92	TIMER1_TMR4	TIMER1 Timer 4	Level	

Table 10-3: CM41X_M4 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
93	TIMER1_TMR5	TIMER1 Timer 5	Level	
94	TIMER1_TMR6	TIMER1 Timer 6	Level	
95	TIMER1_TMR7	TIMER1 Timer 7	Level	
96	TIMER0_TMR0	TIMER0 Timer 0	Level	
97	TIMER0_TMR1	TIMER0 Timer 1	Level	
98	TIMER0_TMR2	TIMER0 Timer 2	Level	
99	TIMER0_TMR3	TIMER0 Timer 3	Level	
100	TIMER0_TMR4	TIMER0 Timer 4	Level	
101	TIMER0_TMR5	TIMER0 Timer 5	Level	
102	TIMER0_TMR6	TIMER0 Timer 6	Level	
103	TIMER0_TMR7	TIMER0 Timer 7	Level	
104	TIMER1_STAT	TIMER1 Status	Level	
105	TIMER0_STAT	TIMER0 Status	Level	
106	CPTMR0_CPT0_MEAS	CPTMR0 CPT 0 Measure Interrupt	Level	
107	CPTMR0_CPT1_MEAS	CPTMR0 CPT 1 Measure Interrupt	Level	
108	CPTMR0_CPT2_MEAS	CPTMR0 CPT 2 Measure Interrupt	Level	
109	DMA4_INT	DMA4 Done interrupt	Level	4
110	DMA5_INT	DMA5 Done interrupt	Level	5
111	DMA6_INT	DMA6 Done interrupt	Level	6
112	DMA7_INT	DMA7 Done interrupt	Level	7
113	DMA8_INT	DMA8 Done interrupt	Level	8
114	DMA9_INT	DMA9 Done interrupt	Level	9
115	DMA10_INT	DMA10 Done interrupt	Level	10
116	DMA11_INT	DMA11 Done interrupt	Level	11
117	DMA12_INT	DMA12 Done interrupt	Level	12
118	DMA13_INT	DMA13 Done interrupt	Level	13
119	DMA0_INT	DMA0 Done interrupt	Level	0
120	DMA1_INT	DMA1 Done interrupt	Level	1
121	DMA2_INT	DMA2 Done interrupt	Level	2
122	DMA3_INT	DMA3 Done interrupt	Level	3
123	TRU1_INT0	TRU1 Interrupt request 0	Edge	

Table 10-3: CM41X_M4 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
124	TRU1_INT1	TRU1 Interrupt request 1	Edge	
125	TRU1_INT2	TRU1 Interrupt request 2	Edge	
126	TRU1_INT3	TRU1 Interrupt request 3	Edge	
127	TRU0_INT0	TRU0 Interrupt request 0	Edge	
128	FFTB0_DONE	FFTB0 Non-continuous FFT averaging has completed. In addition, the interrupt request is asserted after a channel frame is processed (FFT, PSD, Averaging, PSD comparison) in multichannel mode.	Level	
129	ADCC1_TMR0_EVT	ADCC1 Timer 0 Event Complete	Level	
130	ADCC1_TMR1_EVT	ADCC1 Timer 1 Event Complete	Level	
131	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level	
132	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level	
133	DACC0_DAC0	DACC0 DAC Interrupt 0	Level	
134	DACC0_DAC1	DACC0 DAC Interrupt 1	Level	
135	SPI1_STAT	SPI1 Status	Level	
136	SPI0_STAT	SPI0 Status	Level	
137	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
138	SPORT0_B_STAT	SPORT0 Channel B Status	Level	
139	UART1_STAT	UART1 Status	Level	
140	UART2_STAT	UART2 Status	Level	
141	UART3_STAT	UART3 Status	Level	
142	UART4_STAT	UART4 Status	Level	
143	UART0_STAT	UART0 Status	Level	
144	HAE0_STAT	HAE0 Status	Level	
145	CRC0_DCNTEXP	CRC0 Data count expiration	Level	
146	CAN1_TX	CAN1 Transmit	Level	
147	CAN1_RX	CAN1 Receive	Level	
148	CAN1_STAT	CAN1 Status	Level	
149	CAN0_TX	CAN0 Transmit	Level	
150	CAN0_RX	CAN0 Receive	Level	
151	CAN0_STAT	CAN0 Status	Level	

Table 10-3: CM41X_M4 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
152	CNT0_STAT	CNT0 Status	Level	
153	SINC0_STAT	SINC0 Status	Level	
154	TWI0_DATA	TWI0 Data Interrupt	Level	
155	MBOX0_PORT1_MSG	MBOX0 Port 1 software interrupt	Level	
156	MBOX0_PORT0_MSG	MBOX0 Port 0 software interrupt	Level	
157	LBA0_SYS_OUT0	LBA0 Registered LBA outputs available to system	Edge	
158	M4P_SOFT_INT0	M4P Software Interrupt 0	Edge	
159	M4P_SOFT_INT1	M4P Software Interrupt 1	Edge	
160	M4P_SOFT_INT2	M4P Software Interrupt 2	Edge	
161	M4P_SOFT_INT3	M4P Software Interrupt 3	Edge	
162	M4P_SOFT_INT4	M4P Software Interrupt 4	Edge	
163	M4P_SOFT_INT5	M4P Software Interrupt 5	Edge	

CM41X_M0 Interrupt List

Table 10-4: CM41X_M0 Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
-15	RESET	SCS1 RESET		
-14	NonMaskableInt	SCS1 Non Maskable Interrupt		
-13	HardFault	SCS1 Hard Fault		
-5	SVCall	SCS1 Service Call		
-2	PendSV	SCS1 Pending Service		
-1	SysTick	SCS1 System Time Tick		
0	CGU0_OSCW_INT	CGU0 Oscillator Watchdog Interrupt	Level	
0	OCU0_ERR	OCU0 Freq. out-of-range, or dead clock, or counter overflow detected	Level	
0	VMU0_VDDEXT_EVT	VMU0 External voltage management fault	Level	
0	VMU0_VDDINT_EVT	VMU0 Internal voltage management fault	Level	
2	M0P_SRAM_EERR	M0P supervisor sram ecc error	Level	

Table 10-4: CM41X_M0 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
3	SPU0_TIMEOUT	SPU0 Timeout Interrupt	Level	
3	C0_BUS_TIMEOUT	SYSBLK0 Bus Timeout	Level	
3	SYSBLK0_POSTWR_ERR	SYSBLK0 Posted Write Error	Level	
4	MBOX0_PORT0_EERR	MBOX0 Port 0 ECC error	Level	
5	M4P_BUS_FAULT	Bus Fault		
5	M4P_CORE_SRAM_EERR	SRAM Core ECC Error		
5	M4P_LOCKUP	Lockup Error (Fault only; not an interrupt)		
5	FLC0_CORE_FLASH_EERR	Flash Core ECC Error		
5	MBOX0_PORT1_EERR	MBOX0 Port 1 ECC error	Level	
5	SPU1_TIMEOUT	SPU1 Timeout Interrupt	Level	
5	C1_BUS_TIMEOUT	SYSBLK1 Bus Timeout	Level	
6	FFTB0_DPERR	FFTB0 Data parity error	Level	
6	FFTB0_SPERR	FFTB0 Parity error in the limit buffer (LBUFF), or the window buffer (WNDBUFF). If this interrupt request is asserted, ignore the result of this spectrum capture sequence and rewrite both LBUFF and WNDBUFF.	Level	
6	HAE0_PERR	HAE0 Parity Error	Level	
7	CGU0_EVT	CGU0 Event	Edge	
7	DPM0_EVT	DPM0 Event	Level	
8	SEC0_ERR	SEC0 Error	Level	
8	SMPU0_ERR	SMPU0 Error Interrupt	Level	
8	SPU0_INT	SPU0 Interrupt	Level	
9	WDOG0_EXP	WDOG0 Expiration	Level	
10	AFE_LIMIT	AFE FOCP Limit Detection	Level	
10	AFE_NOTOK	AFE is functioning normally	Edge	
11	SWU0_EVT	SWU0 Event	None	
11	SWU1_EVT	SWU1 Event	None	
12	DMA0_ERR	DMA0 Error interrupt	Level	
12	DMA1_ERR	DMA1 Error interrupt	Level	
12	DMA2_ERR	DMA2 Error interrupt	Level	

Table 10-4: CM41X_M0 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
12	DMA3_ERR	DMA3 Error interrupt	Level	
12	DMA8_ERR	DMA8 Error interrupt	Level	
12	DMA9_ERR	DMA9 Error interrupt	Level	
12	DMA10_ERR	DMA10 Error interrupt	Level	
13	ADCC0_ERR	ADCC0 ADC Error	Level	
14	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level	
15	TIMER0_STAT	TIMER0 Status	Level	
15	TIMER0_TMR0	TIMER0 Timer 0	Level	
15	TIMER0_TMR1	TIMER0 Timer 1	Level	
15	TIMER0_TMR2	TIMER0 Timer 2	Level	
15	TIMER0_TMR3	TIMER0 Timer 3	Level	
15	TIMER0_TMR4	TIMER0 Timer 4	Level	
15	TIMER0_TMR5	TIMER0 Timer 5	Level	
15	TIMER0_TMR6	TIMER0 Timer 6	Level	
15	TIMER0_TMR7	TIMER0 Timer 7	Level	
16	CPTMR0_CPT0_MEAS	CPTMR0 CPT 0 Measure Interrupt	Level	
16	CPTMR0_CPT1_MEAS	CPTMR0 CPT 1 Measure Interrupt	Level	
16	CPTMR0_CPT2_MEAS	CPTMR0 CPT 2 Measure Interrupt	Level	
17	DMA0_INT	DMA0 Done interrupt	Level	0
17	DMA2_INT	DMA2 Done interrupt	Level	2
18	DMA1_INT	DMA1 Done interrupt	Level	1
18	DMA3_INT	DMA3 Done interrupt	Level	3
19	DMA8_INT	DMA8 Done interrupt	Level	8
19	DMA9_INT	DMA9 Done interrupt	Level	9
19	DMA10_INT	DMA10 Done interrupt	Level	10
20	TRU0_INT1	TRU0 Interrupt request 1	Edge	
21	TRU1_INT4	TRU1 Interrupt request 4	Edge	
22	FFTB0_DONE	FFTB0 Non-continuous FFT averaging has completed. In addition, the interrupt request is asserted after a channel frame is processed (FFT, PSD, Averaging, PSD comparison) in multichannel mode.	Level	

Table 10-4: CM41X_M0 Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
22	FFTB0_LIMERR	FFTB0 One or more points in the power spectrum output exceeded the predefined limit in the limit buffer.	Level	
23	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level	
24	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level	
25	SPI0_ERR	SPI0 Error	Level	
25	SPI0_STAT	SPI0 Status	Level	
26	UART0_STAT	UART0 Status	Level	
27	HAE0_STAT	HAE0 Status	Level	
28	CAN0_RX	CAN0 Receive	Level	
28	CAN0_STAT	CAN0 Status	Level	
28	CAN0_TX	CAN0 Transmit	Level	
29	TWI0_DATA	TWI0 Data Interrupt	Level	
30	MBOX0_PORT0_MSG	MBOX0 Port 0 software interrupt	Level	

CM41X_M4 SEC Trigger List

Table 10-5: CM41X_M4 SEC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
1	SEC0_FAULT	SEC0 Fault	Edge
2	SEC1_FAULT	SEC1 Fault	Edge

Table 10-6: CM41X_M4 SEC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

SEC Definitions

The event controller uses the following definitions.

System Events

System source indications including interrupts and faults.

System Source

Point of origin of system event.

SID (Identification, unique)

Source numeric identifier for each system source connected to the SEC.

SSI

SEC source interface, system event source control, and status subblock of the SEC.

SFI

SEC Fault Interface, fault management subblock of the SEC.

NVIC

Nested Vectored Interrupt Controller

SEC Block Diagram

The ***SEC Block Diagram*** shows the event management architecture.

As shown in the figure, the SEC has two blocks for event management. NVIC deals with the interrupts from various system sources while the SFI monitors and manages any fault event triggered from various fault input sources. System interrupt sources are routed to the SFI through SEC source interface (SSI).

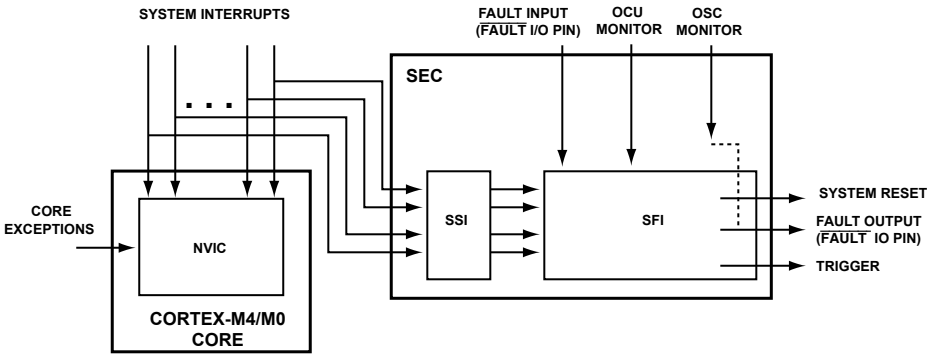


Figure 10-1: SEC Block Diagram

NOTE: The NVIC is an independent unit inside the SEC and is closely tied to the ARM Cortex core. Therefore, its programming is the same as described in the ARM Cortex standard documentation. In contrast, the SFI is mostly integrated to the SEC such that it uses all SEC-specific registers, typically for fault management. For dealing with system interrupts, there is virtually no need for accessing any of the SEC registers.

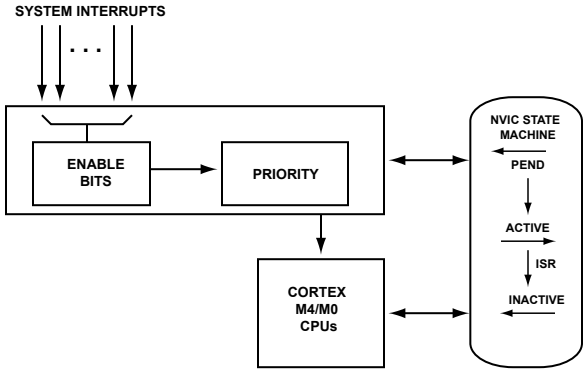


Figure 10-2: NVIC Block Diagram

NOTE: The Cortex-M4 and Cortex-M0 have separate NVIC Controllers which are not connected to each other. The sub systems include the following SEC blocks: Cortex-M4 sub-system = SEC1, Cortex-M0 sub-system = SEC0

A total of 32 interrupts are supported by the NVIC in the ARM Cortex-M0 core. Since more peripherals require interrupt support, a shared mechanism is used for most of the peripheral interrupts. For example, the SWU0_EVT and SWU0_EVT interrupts are both mapped to interrupt ID 11. When the NVIC registers an interrupt, the program has to verify the actual source (between the shared interrupts) by reading the `SYSBLK_SISTAT0` register (Shared Interrupt Status register).

SEC Fault Interface (SFI)

The SFI manages fault events and associated actions. The fault management support provided in the SEC helps satisfy the safety requirements of various applications. The SSI provides the highest priority pending source that is enabled as a fault. The SFI captures this value and enables a countdown, and once the countdown expires, takes the prescribed action.

Fault actions which can be configured, as shown in ***SFI Block Diagram***, include

- Trigger Output
- System Reset
- Fault Output
  - Computer Operating Properly (COP) mode
  - Fault Mode

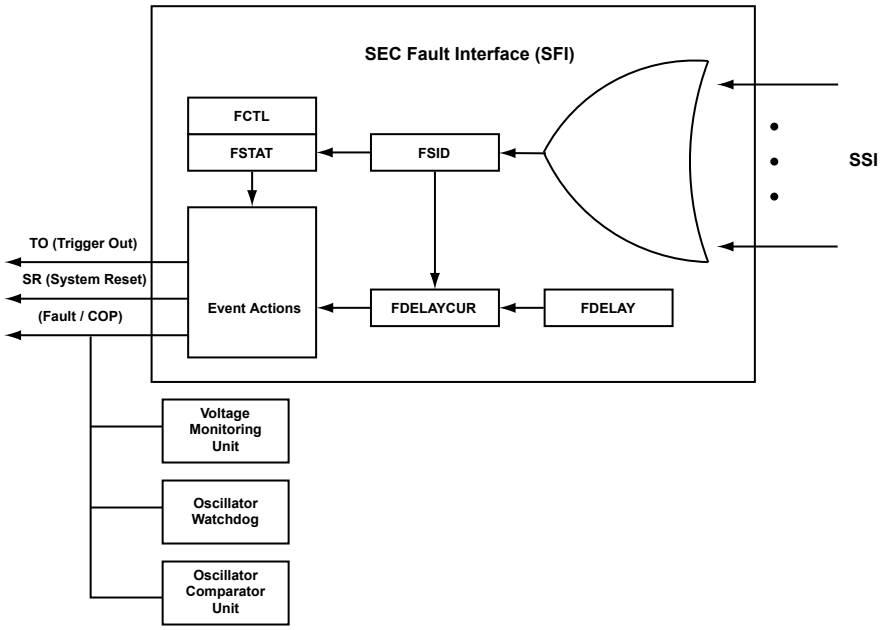


Figure 10-3: SFI Block Diagram

Fault Management

System sources can be enabled as fault sources in the SEC_SCTL[n] register. When a source enabled as a fault moves to pending, it is forwarded to the SFI as a fault indication. The pending bit (SEC_FSTAT.PND) indicates a source has signaled a fault assertion but it has not yet triggered the event actions (if delay is enabled). The SEC fault interface sets the SEC_FSTAT.PND bit when the fault source ID register (SEC_FSID) is updated on assertion of a fault source input. The system source pending triggers a fault pending and after a programmable delay the fault moves to active. Event actions then execute if appropriate action is not taken by the core. The SEC_FSTAT.ACT bit indicates that the SEC has received a fault source input, the delay has expired, and the fault actions are enabled.

The SEC_FSTAT.NPND bit indicates if one or more sources have signaled a fault assertion, but the input has not yet triggered the fault pending detection in the SEC fault interface. The SEC sets the SEC_FSTAT.NPND bit when the fault interface detects assertion of any enabled fault source input, while either the SEC_FSTAT.PND or SEC_FSTAT.ACT bits are set. The SEC clears the SEC_FSTAT.NPND bit when there are no fault sources waiting.

A fault indication from an external device can also be detected on sampling the fault signals. When a fault is detected the SEC_FSTAT.ACT and SEC_FSID.FEXT bits are set. The assertion of either signal results in a fault input detection.

The SEC_FEND register receives a fault end indication from the core. The core writes the SID of the fault to the SEC_FEND register. If the SID matches the value in the SEC_FSID register, the SEC_FSTAT.PND and SEC_FSTAT.ACT bits are cleared.

SEC Architectural Concepts

The following sections describe SEC architectural features.

System Interrupt Acknowledge

A system interrupt acknowledge occurs when the core provides an indication that it has acquired the SID of the interrupt last issued by the SEC.

The SEC core interface option allows generation by asserting the input acknowledge signal (the connected core generates the signal).

Nested Vectored Interrupt Controller (NVIC)

The ARM Cortex-M4 processor closely integrates a configurable Nested Vectored Interrupt Controller (NVIC). The NVIC includes a non-maskable interrupt (NMI) that can provide up to 16 interrupt pre-emptive priority levels. The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing the interrupt latency. This performance is achieved through stacking hardware registers and suspending load-multiple and store-multiple operations. Additionally, tail-chain optimization significantly reduces the overhead when switching from one ISR to another. The following are descriptions of the interrupt types used in the system.

Cortex system exceptions

The exceptions triggered from within the ARM Cortex-M4 processor core have negative interrupt IDs and are identified in the interrupt list with an `M4_SCS0_` prefix. The Reset, Hard Fault, and NMI exceptions have fixed negative priority values, and these have higher priority than any other exception.

External interrupts

There are a total of 164 external interrupts in the processor. Almost all of these interrupts are triggered from several peripheral interrupt sources. A maximum of 16 levels of pre-emptive priority are possible. Interrupts can be enabled or disabled individually through the interrupt set or clear register and the priority level (priority + subpriority) can be defined by programming the interrupt priority registers.

Reset interrupt

When reset signal is de-asserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in thread mode. Refer to the Reset Control Unit chapter for more details on reset implementation and usage in the processor. Refer to the Boot ROM chapter for information about how the ROM handles the reset event, before jumping to application.

Non-Maskable Interrupt (NMI)

NMI can be asserted with the following sources:

- Through the non-maskable interrupt pin, $\overline{\texttt{SYS\_NMI}}$.
- Through trigger outputs from TRU unit.

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. On system reset, the vector table is fixed at address 0x00000000, which is inside the Boot ROM, for the processor. The address offset is aligned to the vector table size, extended to the next larger power of 2.

Priority Grouping

The NVIC is configured with 4 bits of priority. The lower bits of the register are always 0; ***PRI_N[7:4]*** sets the priority, and ***PRI_N[3:0]*** is 4'b0000.

Table 10-7: PRIGROUP Implementation

PRIGROUP	Binary Point	Group Priority		Subpriority	
		Bits	Levels	Bits	Levels
011	4.0	4	16	0	0
100	3.1	3	8	1	2
101	2.2	2	4	2	4
110	1.3	1	2	3	8
111	0.4	0	0	4	16

Interrupt Request Processing Using the SEC and NVIC

Most peripherals on the ADSP-CM41x can issue interrupt requests. This system has a great deal of flexibility in how to deal with these events to allow for user safety requirements. This section provides basic guidelines on the ability of the system to respond to interrupt requests. Different applications may use interrupt processing in different ways.

The main system contains SEC blocks that receive interrupt requests. This block is configurable and may issue a system fault or one of a small number of triggers for a particular interrupt request. The system also has an ARM Cortex-M4 and an ARM Cortex-M0 processor core. These cores also receive interrupt requests and process them with a NVIC. The programming model expects the processors to respond and handle standard interrupt requests from various peripherals that are performing system data processing. How a given system handles various system errors is a programming or system design decision. Some errors (like FFT parity) may simply require dropping a frame of data and continue processing. Other errors (like bus-timeout) may indicate a system with a significant unexpected failure and might warrant a full processor reset or a safety trip operation.

NOTE: Some major system faults indicate the system clocking or supplies are faulty. These faults have side mechanisms to allow them to issue a system fault without any flop-based logic.

A subset of all peripheral interrupt requests are inputs to the ARM Cortex-M0 NVIC unit and the ARM Cortex-M0 SEC0 subsystem block. In most cases the higher priority requests are given lower numbers. The NVIC unit has control registers that allow for enabling/disabling individual interrupts. The ARM Cortex-M0 NVIC has 2 bits of configurable priority. The ARM Cortex-M0 has 32 user interrupts. An additional 16 interrupts at the lower priority are reserved for the ARM Cortex-M0 core. These default IRQ entries are usually numbered from -15 to -1. There are more than 32 peripheral interrupt requests for the 32 allocated user interrupts. Some interrupts have been shared at the same priority levels (register details below), for a shared interrupt level the appropriate `SYSBLK_SISTAT0` register has the source(s) of an interrupt request to the ARM Cortex-M0. Due to the ability for the `SYSBLK_SISTAT0` register to change while in a vector routine, sources should be processed and cleared one at a time in each level until empty. One “software” interrupt is allocated for the ARM Cortex-M0 core. The SEC0 block

has logic for each individual interrupt request that allow the request to trigger the SEC0 block output system fault or one of its trigger outputs. No sharing is required.

System Fault Interface (SFI) and the NVIC

The system fault interface (SFI) operates independently from the NVIC. All system sources that can generate interrupts can also be routed to generate faults through the SEC source control registers. A fault can also be generated from the SYS_FAULT pin or from the oscillator watchdog. For more information on a fault generated from the oscillator watchdog, see the clock generation unit (CGU) chapter.

Fault Input Options

A number of sources can generate a fault:

- Through the SYS_FAULT pin (external source)
- The oscillator watchdog
- System interrupt sources
- The Voltage Monitoring Unit and the Oscillator Comparator Unit

Externally Generated through the SYS_FAULT Pin

When the SFI detects a falling edge at the SYS_FAULT pin, the SFI can sense this condition as an external fault and take necessary action. The SFI must be configured with the SEC_FCTL.FIEN bit to sample the pin for a fault. One usage case of this feature is to control the processor from another system or to permit another processor to communicate a fault occurrence.

NOTE: The ADSP-CM41x has a single SYS_FAULT pin that can be managed from either SEC modules (for M4 and M0). The SEC units should both be operated in FAULT mode to assert SYS_FAULT low when a fault is detected.

From the Oscillator Watchdog

Either the Oscillator Watchdog or the OCU may indicate a bad clock. If there is an oscillator watchdog fault input, the SEC could not be functional, because the SEC needs a clock for its operation. The direct path to the SYS_FAULT pin is provided in that case. The oscillator watchdog and its fault generation are explained in the Oscillator Watchdog section of the CGU chapter.

Faults from Voltage the Monitoring Unit

The VMU can assert the SYS_FAULT pin on faulty voltage conditions on internal and external power supplies. Please consult the VMU chapter for more details.

Faults from the Oscillator Comparator Unit

The OCU can assert the SYS_FAULT pin on faulty system clock conditions. Please consult the Oscillator Comparator Unit (OCU) chapter for more details.

Faults from System Interrupt Sources

System interrupt requests from various peripherals and other sources can be routed to the SFI through the SSI. This routing is enabled by setting the source signal enable (`SEC_SCTL[n].SEN`) and fault enable (`SEC_SCTL[n].FEN`) bits. When the SSI detects the interrupt assertion and if the fault is enabled for that interrupt, the fault is passed to the SFI for further fault management actions.

Because the NVIC is an independent block outside of the SSI and SFI, the NVIC does not know how the SFI manages the fault. Managing interrupts is dedicated to NVIC and managing faults is dedicated to SFI. So, configure the NVIC appropriately to service the interrupt using an interrupt service routine (ISR).

After the interrupt request is asserted, the signal is sent to both the NVIC and the SFI. Because the source for interrupt and fault generation is the same, a program can manage the fault inside the NVIC handler. Fault events are triggered when the delay registers count down to zero. In some cases, a program can perform a disable and reenable of the `SEC_SCTL[n].FEN` bit inside the handler to halt any fault event that is not desired when interrupt is running. For more information, see the examples in Programming Examples.

Managing Faults

A fault can be monitored and managed using the following methods.

Trigger Output

The fault interface does not have an interrupt line registered for dedicated management of fault actions. A program must assign a TRU slave (for example, NMI) to the SEC fault TRU master in order to manage the fault. This approach is not typically required when:

- a system interrupt is routed to the SFI through the SSI, and
- the NVIC interrupt service routine of that particular interrupt manages the fault event.

System Reset

It is possible to issue a system reset optionally. It is also possible to delay the assertion, such that the application can perform some critical housekeeping operations before the reset is generated.

Indication on the $\overline{\texttt{SYS\_FAULT}}$ Pin

SFI can drive the $\overline{\texttt{SYS\_FAULT}}$ pin to indicate fault to the external world. In a development system, this signal can be connected to an LED. In COP toggle mode, the SFI continuously sends out a series of pulses, and it stops when a fault is asserted.

SEC Error

The processor includes an SEC error (`SEC_GSTAT.ERR`) as a source input to the SEC to allow for handling the error as an interrupt or fault.

NVIC

Implementing a system interrupt service model using the NVIC requires, at a minimum:

- Proper configuration of a system interrupt source (for example, a peripheral or DMA)
- A core interrupt, fault or event service model
- Proper configuration of the NVIC

Programming Concepts

The following list provides the basic programming concepts necessary for configuring the NVIC and SEC.

- Configuring a System Interrupt with NVIC
- Configuring FMU as Fault Pin
- Managing Faults Inside a Triggered ISR
- Configuring and Managing Faults (that are also interrupts)

Programming Examples

This section provides example programming tasks that are typical for SEC usage.

Configuring the WDOG Expiry Event to Issue a System Reset

Use the following procedure to configure the WDOG timer to issue a system reset.

1. Configure the `SEC_GCTL` register to enable the SEC.

   ADDITIONAL INFORMATION:

   ```
   *pREG_SEC0_GCTL = BITM_SEC_GCTL_EN;
   ```

2. Configure the `SEC_FCTL` register to choose the Fault response mode. In the following code example, the system reset is issued.

   ADDITIONAL INFORMATION:

   ```
   *pREG_RCU0_CTL |= BITM_RCU_CTL_SRSTREQEN;
   *pREG_SEC0_FCTL |= BITM_SEC_FCTL_SREN;
   ```

3. Configure the `SEC_SCTL[n].SEN` and `SEC_SCTL[n].FEN` bits in the Source Control 34 and 35 (n=34 and 35) registers to determine how the fault source is handled. To configure the WDOG as the fault source, program the register. The program can configure any interrupt as the fault source by programming the corresponding register.

   ADDITIONAL INFORMATION:

   ```
   *pREG_SEC0_SCTL34 = BITM_SEC_SCTL_FEN|BITM_SEC_SCTL_SEN;
   *pREG_SEC0_SCTL35 = BITM_SEC_SCTL_FEN|BITM_SEC_SCTL_SEN;
   ```

ADDITIONAL INFORMATION: The SEC ID corresponding to WDOG0 is 3, as indicated in .

4. Write to the enable bit.

   ADDITIONAL INFORMATION:

```
*pREG_SEC0_FCTL |= BITM_SEC_FCTL_EN;
```

Configuring a System Interrupt with NVIC

The NVIC supports configuring system interrupts.

1. Set the NVIC priority and subpriority levels for the interrupt.
2. Enable the system interrupt at peripheral source.
3. Enable the interrupt with the NVIC.
4. Inside the interrupt service routine (ISR), the NVIC pushes and pops the C program ABI registers.
5. The program must clear the source for the interrupt inside the ISR.

Configuring FMU as Fault Pin

The NVIC supports configuring the FMU as a fault pin.

1. Enable the `SEC_GCTL.RESET` bit.
2. Enable the `SEC_GCTL.EN` bit.
3. Program the `SEC_FCTL.FIEN` bit.
4. Program the `SEC_FCTL` register with the `SEC_FDLY` delay time from the fault pending to the fault active.
5. Program the `SEC_FCTL` register with `SEC_FCTL.SREN` or `SEC_FCTL.TOEN`, depending on the requirements. If using the trigger mode, route the fault trigger to an interrupt handler such as an NMI interrupt service routine (ISR).
6. Enable the fault unit by setting the `SEC_FCTL.EN` bit.
7. If using the trigger mode, the fault is handled inside the interrupt handler of the ISR to which the fault was routed.

Managing Faults Inside a Triggered ISR

The SEC supports fault management within an interrupt service routine (ISR).

1. Check whether the `SEC_FSTAT.ACT` bit is set.
2. Check whether the `SEC_FSID.FEXT` bit is set.
   a. If set, clear the external fault pin source by writing the `SEC_FEND.FEXT` bit.

b. If not set, clear the system interrupt source by writing the SEC_FSID.SID bit.

Configuring and Managing Faults (that are also interrupts)

The SEC permits simultaneously registering an interrupt with NVIC and configuring the interrupt as fault (configuring and managing a fault that is also an interrupt).

1. Enable the SEC_GCTL.RESET bit.
2. Enable the SEC_GCTL.EN bit.
3. Program the SEC_SCTL[n].SEN and SEC_SCTL[n].FEN bits for the SSI.
4. Program the SEC_FCTL register with options: fault delay, COP toggle mode, system reset, and others.
5. Enable the fault unit by setting the SEC_FCTL.EN bit.

   ADDITIONAL INFORMATION: The NVIC handles interrupts. Inside the interrupt service routine (ISR), the software must first clear the system interrupt source. If the software does not handle the fault soon enough, the fault events are generated. So, prioritize interrupt handling first (finish it first).
6. To halt a fault event, perform the following:
   a. Clear the SEC_SCTL[n].FEN and SEC_SCTL[n].SEN bits.
   b. Write the SEC_SSTAT[n].PND bits.
   c. Handle the fault as described in triggered ISR case.
   d. Reenable the SEC_SCTL[n].FEN and SEC_SCTL[n].SEN bits.
   e. Return from the ISR.

SEC Programming Restrictions

Setting the SEC_FCTL.EN bit while the SEC_FSTAT.ACT bit is high can result in unpredictable behavior. To avoid this issue, set the SEC_FCTL.EN bit while the SEC_FSTAT.ACT bit is low. The SEC_FSTAT.ACT bit is only set when the SEC_FCTL.EN bit is high. Therefore, the problem can only occur if the SEC_FCTL.EN bit transitions from 1 to 0 and then to 1 again.

Writing to SEC_FEND to end a fault with both the SEC_FCTL.FOEN bit and the SEC_FCTL.FIEN bit set can result in erroneous external fault detection. If this operation (ending a fault) and configuration (fault input and fault output enabled) are required by the application, clear the SEC_FCTL.FOEN bit prior to writing to SEC_FEND. The recommended sequence for ending a fault with the SEC_FCTL.FIEN or SEC_FCTL.FOEN==1 is as follows:

1. Clear the SEC_FCTL.FOEN bit.
2. Write to the SEC_FEND register.
3. Set the SEC_FCTL.FOEN bit.

CM41X_M4 SEC Register Descriptions

System Event Controller (SEC) contains the following registers.

Table 10-8: CM41X_M4 SEC Register List

Name	Description
SEC_FCOPP	Fault COP Period Register
SEC_FCOPP_CUR	Fault COP Period Current Register
SEC_FCTL	Fault Control Register
SEC_FDLY	Fault Delay Register
SEC_FDLY_CUR	Fault Delay Current Register
SEC_FEND	Fault End Register
SEC_FSID	Fault Source ID Register
SEC_FSRDLY	Fault System Reset Delay Register
SEC_FSRDLY_CUR	Fault System Reset Delay Current Register
SEC_FSTAT	Fault Status Register
SEC_GCTL	Global Control Register
SEC_GSTAT	Global Status Register
SEC_RAISE	Global Raise Register
SEC_SCTL[n]	Source Control Register n
SEC_SSTAT[n]	Source Status Register n

Fault COP Period Register

The SEC fault COP period register (SEC_FCOPP) contains the width value (count in (SEC) clock cycles) for the high and low phase of the computer operating properly (COP) toggled output on the COP pin. Note that the actual high/low phase value is the SEC_FCOPP.COUNT programmed value plus 1.

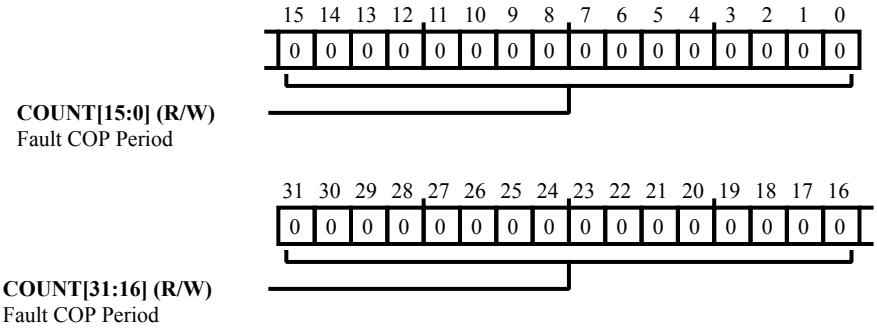


Figure 10-4: SEC_FCOPP Register Diagram

Table 10-9: SEC_FCOPP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault COP Period. The SEC_FCOPP.COUNT bit field is the width value for the high and low phase of the computer operating properly (COP) toggled output on the COP pin.

Fault COP Period Current Register

The SEC fault COP period current register (SEC_FCOPP_CUR) contains the active count (in (SEC) clock periods) for the current phase (high or low) of the computer operating properly (COP) toggled output on the COP pin. The SEC loads the SEC_FCOPP_CUR register from the SEC_FCOPP register when the SEC_FCOPP_CUR.COUNT field is cleared and the SEC is in COP mode (SEC_FCTL.CMS bit =1). The SEC decrements the SEC_FCOPP_CUR count each (SEC) clock cycle while SEC_FCTL.CMS is set and the SEC_FSTAT.ACT bit is not set.

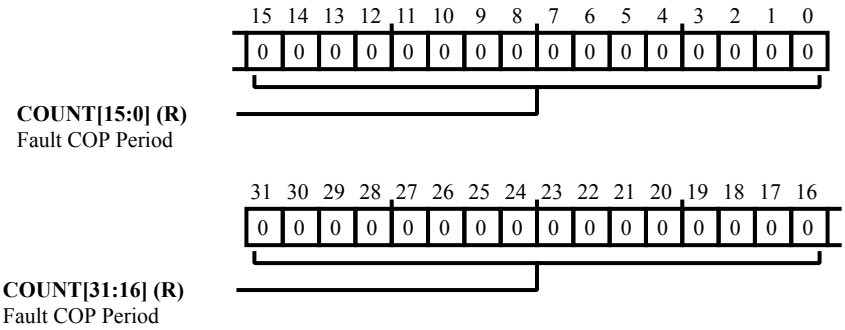


Figure 10-5: SEC_FCOPP_CUR Register Diagram

Table 10-10: SEC_FCOPP_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault COP Period. The SEC_FCOPP_CUR.COUNT bit field is the active count for the current phase (high or low) of the computer operating properly (COP) toggled output on the COP pin.

Fault Control Register

The SEC fault control register (SEC_FCTL) contains fault control bits for all SEC channels. This register controls the operation of the System Fault Management Interface (SFI).

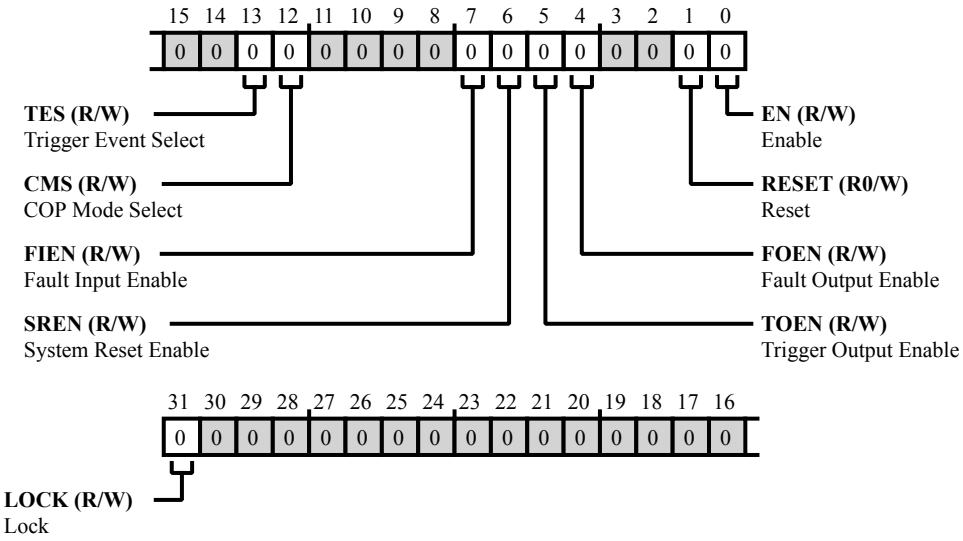


Figure 10-6: SEC_FCTL Register Diagram

Table 10-11: SEC_FCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the SEC_FCTL.LOCK bit is enabled, the SEC_FCTL register is read only.	
		0	UnLock
		1	Lock
13 (R/W)	TES	Trigger Event Select. The SEC_FCTL.TES bit selects the event that directs the SEC to assert trigger output. In fault pending mode, the SEC asserts trigger output when a fault is pending. In fault active mode, the SEC asserts trigger output when a fault is active.	
		0	Fault Active Mode
		1	Fault Pending Mode

Table 10-11: SEC_FCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	CMS	COP Mode Select. The SEC_FCTL.CMS selects the SEC mode for handling fault input. In COP mode, the SEC toggles the COP pin to indicate that no fault is active and ceases toggling the pin to indicate that a fault is active. In fault mode, the SEC deasserts the fault pin (=0) and fault_b pin (=1) when no fault is active and asserts the fault pin (=1) and fault_b pin (=0) when a fault is active. Not all processors feature both the fault and fault_b pins. Refer to the product data sheet for details.	
		0	Fault Mode
		1	COP Mode
7 (R/W)	FIEN	Fault Input Enable. The SEC_FCTL.FIEN bit enables the SEC to sample fault input. If SEC_FCTL.FIEN is set (=1), a fault indication from an external device sets the SEC_FSTAT.ACT bit and SEC_FSID.FEXT bit.	
		0	Disable
		1	Enable
6 (R/W)	SREN	System Reset Enable. The SEC_FCTL.SREN bit enables the SEC to issue a system reset request when a fault becomes active.	
		0	Disable
		1	Enable
5 (R/W)	TOEN	Trigger Output Enable. The SEC_FCTL.TOEN bit enables the SEC to produce trigger output when a fault becomes active.	
		0	Disable
		1	Enable
4 (R/W)	FOEN	Fault Output Enable. The SEC_FCTL.FOEN bit enables the SEC to indicate fault status, according to the SEC_FCTL.CMS bit configuration.	
		0	Disable
		1	Enable
1 (R0/W)	RESET	Reset. Setting the SEC_FCTL.RESET bit resets ALL SEC registers to their default values.	
		0	No Action
		1	Reset

Table 10-11: SEC_FCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The `SEC_FCTL.EN` bit controls the operational state of the SEC. Clearing the `SEC_FCTL.EN` bit halts the execution of the SEC without resetting status registers. Setting the `SEC_FCTL.EN` bit enables the SEC to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

Fault Delay Register

The SEC fault delay register (SEC_FDLY) contains the number (SEC_FDLY.COUNT field) of (SEC) clock periods to delay from fault pending to fault active, when actions are enabled.

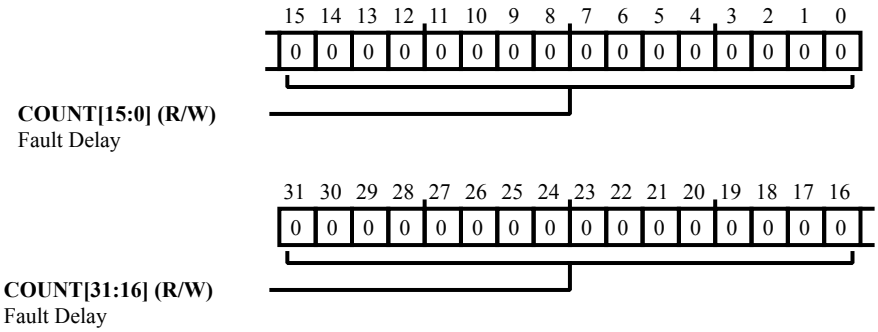


Figure 10-7: SEC_FDLY Register Diagram

Table 10-12: SEC_FDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault Delay. The SEC_FDLY.COUNT bit field is the number of (SEC) clock periods to delay from fault pending to fault active, when actions are enabled.

Fault Delay Current Register

The SEC fault delay current register (SEC_FDLY_CUR) contains the active count (SEC_FDLY_CUR.COUNT field) in (SEC) clock periods for the delay from fault pending to fault active, when actions are enabled. The count is loaded from the SEC_FDLY register when a fault becomes pending (SEC_FSTAT.PND bit is set). The SEC decrements the value in SEC_FDLY_CUR each (SEC) clock cycle while the SEC_FSTAT.PND bit is set.

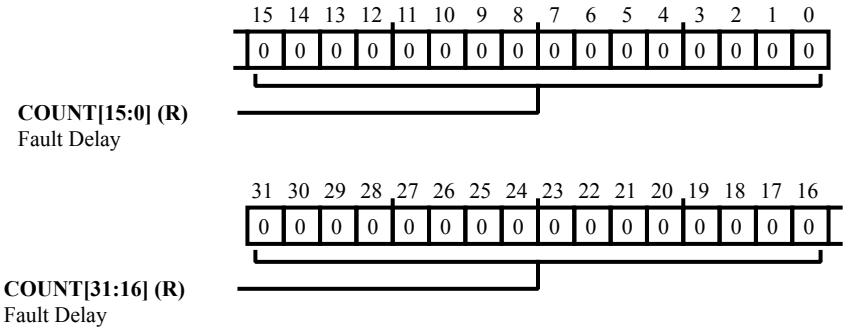


Figure 10-8: SEC_FDLY_CUR Register Diagram

Table 10-13: SEC_FDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault Delay. The SEC_FDLY_CUR.COUNT bit field is the active count in (SEC) clock periods for the delay from fault pending to fault active, when actions are enabled.

Fault End Register

The SEC fault end register (SEC_FEND) contains fault source ID and internal/external fields. This register receives fault end indication from a core.

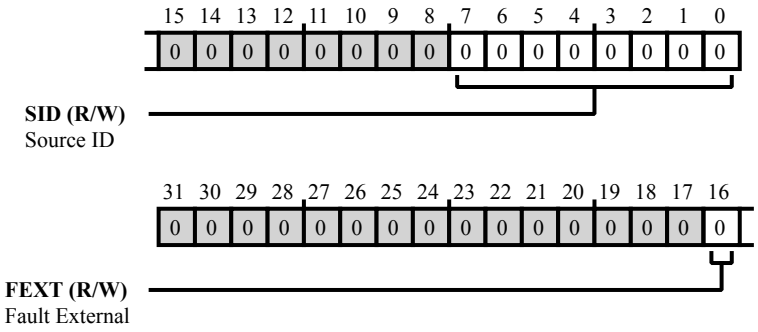


Figure 10-9: SEC_FEND Register Diagram

Table 10-14: SEC_FEND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	FEXT	Fault External. Setting the SEC_FEND.FEXT bit, when the SEC_FEND.SID field is cleared, clears an active fault from an external source.	
		0	Fault Internal
		1	Fault External
7:0 (R/W)	SID	Source ID. The SEC_FEND.SID identifies a fault to be ended as indicated to the SEC by the core. The core loads the SEC_FEND.SID field value. If the SEC_FEND.SID value matches the SEC_FSID.SID value, the SEC_FSTAT.PND bit and SEC_FSTAT.ACT bit are cleared.	

Fault Source ID Register

The SEC fault source ID register (SEC_FSID) contains a fault source ID and internal/external fields.

NOTE:These bits are not reset by system reset so that a fault that automatically triggers a system reset to avoid a fault may be analyzed after the reset occurs.

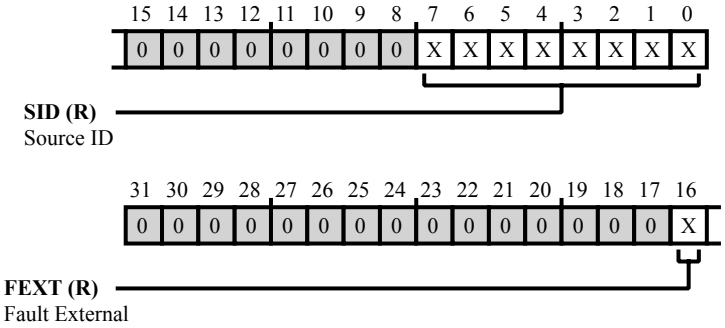


Figure 10-10: SEC_FSID Register Diagram

Table 10-15: SEC_FSID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	FEXT	Fault External. The SEC_FSID.FEXT bit indicates that the last active fault was asserted by an external device. The SEC sets the SEC_FSID.FEXT bit when the SEC_FSTAT.ACT bit is set by the fault input pins. The SEC_FSID.FEXT bit is cleared when the SEC_FSTAT.ACT bit is set by an internal fault or when the external fault is ended. When the SEC_FSID.FEXT bit is set, the SEC_FSID.SID is cleared.	
		0	Fault Internal
		1	Fault External
7:0 (R/NW)	SID	Source ID. The SEC_FSID.SID identifies the fault assertion detected by the SEC fault interface. The SEC loads the SEC_FSID.SID field value when a system fault indication is asserted. The SEC fault interface does not change the SEC_FSID.SID value until the fault is no longer pending or active, as indicated by the SEC_FSTAT.PND bit and SEC_FSTAT.ACT bit being cleared in the SEC_FSTAT register.	

Fault System Reset Delay Register

The SEC fault system reset delay register (SEC_FSRDLY) contains the number (SEC_FSRDLY.COUNT field) of (SEC) clock periods for the delay from a fault becoming active to system reset request assertion, if enabled.

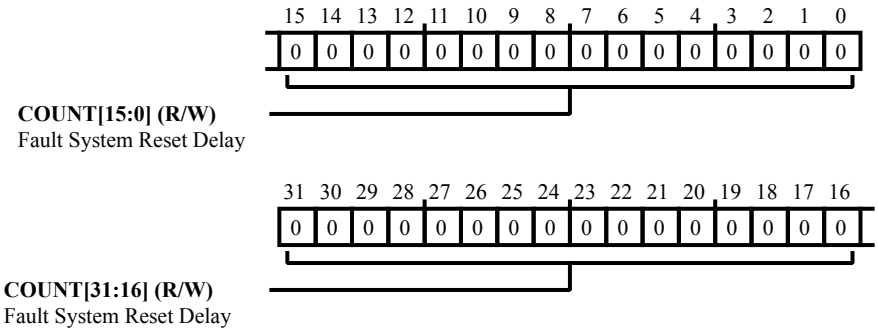


Figure 10-11: SEC_FSRDLY Register Diagram

Table 10-16: SEC_FSRDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	COUNT	Fault System Reset Delay. The SEC_FSRDLY.COUNT bit field is the number of (SEC) clock periods for the delay from a fault becoming active to system reset request assertion.

Fault System Reset Delay Current Register

The SEC fault system reset delay current register (SEC_FSRDLY_CUR) contains the active count (SEC_FSRDLY_CUR.COUNT field) in (SEC) clock periods for the delay from fault active to system reset assertion, if enabled. The count is loaded from the SEC_FSRDLY register when a fault becomes active (SEC_FSTAT.ACT bit is set). The SEC decrements the value in SEC_FSRDLY_CUR each (SEC) clock cycle while the SEC_FSTAT.ACT bit is set.

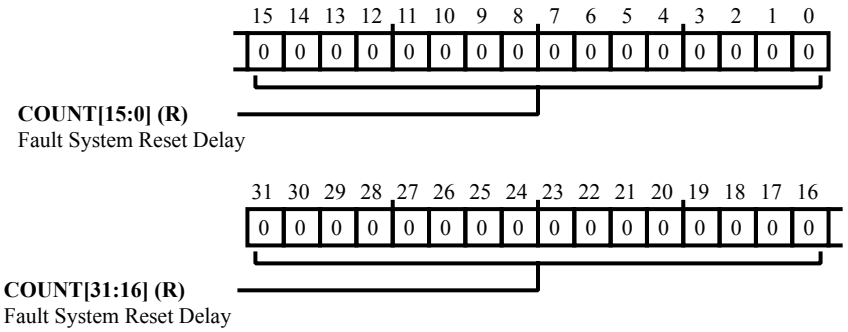


Figure 10-12: SEC_FSRDLY_CUR Register Diagram

Table 10-17: SEC_FSRDLY_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	COUNT	Fault System Reset Delay. The SEC_FSRDLY_CUR.COUNT bit field is the active count in (SEC) clock periods for the delay from fault active to system reset assertion.

Fault Status Register

The SEC fault status register (SEC_FSTAT) indicates the operational status of the SFI.

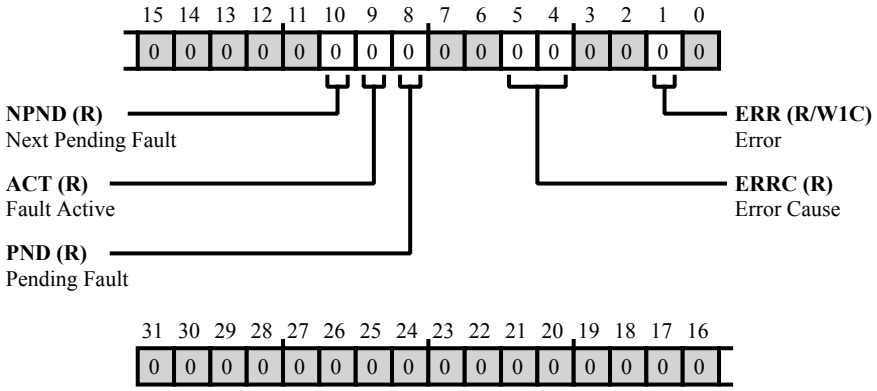


Figure 10-13: SEC_FSTAT Register Diagram

Table 10-18: SEC_FSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	NPND	Next Pending Fault. The SEC_FSTAT.NPND bit indicates that one or more sources have signaled fault assertion, but the input has not yet triggered the fault pending detection in the SEC fault interface. The SEC sets the SEC_FSTAT.NPND bit when the fault interface detects assertion of any enabled fault source input, while either the SEC_FSTAT.PND or SEC_FSTAT.ACT bits are set. The SEC clears the SEC_FSTAT.NPND bit when there are no fault sources waiting.	
		0	Not Pending
		1	Pending
9 (R/NW)	ACT	Fault Active. The SEC_FSTAT.ACT bit indicates that the SEC has received a fault source input, the current fault delay count (in the SEC_FDLY_CUR register) has expired, and the fault actions are enabled. The SEC also sets the SEC_FSTAT.ACT bit on fault input detection if the SEC_FCTL.FIEN bit is set. The SEC_FSTAT.ACT bit is cleared by writing the ID value of the asserted fault from SEC_FSID register to the SEC_FEND register.	
		0	No Fault
		1	Active Fault

Table 10-18: SEC_FSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	PND	Pending Fault. The SEC_FSTAT.PND bit indicates a fault source has signaled a fault assertion to the SEC, but the SEC has not yet triggered the event actions due to the delay selected with the SEC_FDLY register. The SEC fault interface sets the SEC_FSTAT.PND bit when the SEC_FSID is updated on assertion of a fault source input. The SEC_FSTAT.PND bit is only set when the SEC_FSTAT.ACT bit is cleared. The SEC updates the SEC_FSID register with the SID value when the SEC_FSTAT.PND bit is set. The SEC_FSTAT.PND bit is cleared *either* by the SEC fault interface when the current delay count in the SEC_FDLY_CUR register expires *or* by writing the SEC_FSID.SID field value (which indicates the ID of the asserted fault) to the SEC_FEND register.	
		0	Not Pending
		1	Pending
5:4 (R/NW)	ERRC	Error Cause. When the SEC_FSTAT.ERR bit is asserted, the SEC updates SEC_FSTAT.ERRC field to convey the interrupt source error type. When the error type is source overflow, the status indicates that a source signal assertion occurred or an SEC raise operation was attempted while pending was already set. The source overflow is detected when the source is set for edge only.	
		0	Source Overflow Error
		1	Reserved
		2	End Error
		3	Reserved
1 (R/W1C)	ERR	Error. The SEC_FSTAT.ERR bit indicates an SEC fault interface error. When SEC_FSTAT.ERR is set, the SEC updates the SEC_FSTAT.ERRC field to indicate the corresponding error cause. When multiple errors occur, the SEC_FSTAT register captures the status for the first error and does not capture subsequent errors until the status is cleared.	
		0	No Error
		1	Error Occurred

Global Control Register

The SEC global control register (SEC_GCTL) provides register locking, reset, and enable for the SEC module.

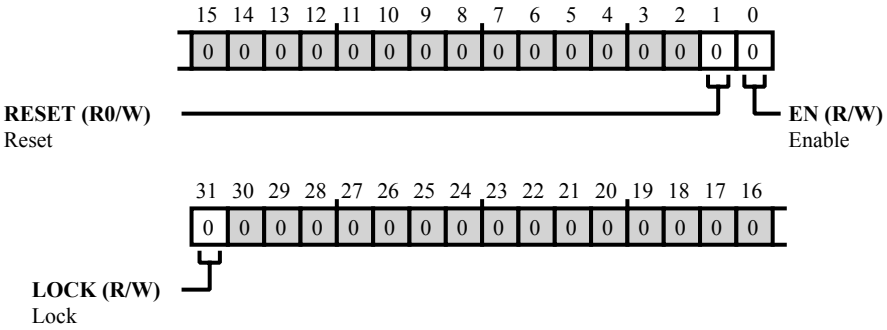


Figure 10-14: SEC_GCTL Register Diagram

Table 10-19: SEC_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the SEC_GCTL.LOCK bit is enabled, the SEC_GCTL register is read only.	
		0	Unlock
		1	Lock
1 (R0/W)	RESET	Reset. The SEC_GCTL.RESET bit is write-1-action and triggers a soft reset to all SEC registers.	
		0	No Action
		1	Reset
0 (R/W)	EN	Enable. The SEC_GCTL.EN bit is read/write and must be set for the SEC to begin/resume SEC operation with the current configuration and status. Clearing the SEC_GCTL.EN bit halts the execution of the SFI. All SSIs remain active, along with all error detection, without resetting status registers.	
		0	Disable
		1	Enable

Global Status Register

The SEC global status register (SEC_GSTAT) contains global status bits for the SEC.

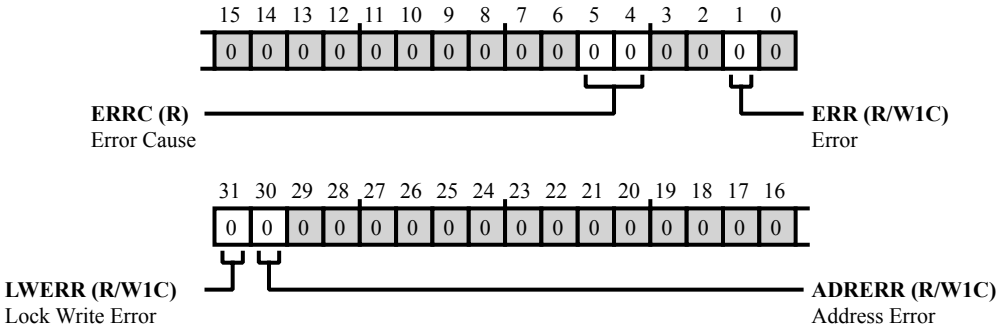


Figure 10-15: SEC_GSTAT Register Diagram

Table 10-20: SEC_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W1C)	LWERR	Lock Write Error. The SEC_GSTAT.LWERR bit indicates (when set) there was an attempted write to an SEC register while the SEC_GCTL.LOCK bit was set and while the global lock bit was enabled (SPU_CTL.GLCK bit =1). This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
30 (R/W1C)	ADRERR	Address Error. The SEC_GSTAT.ADRERR bit indicates that the SEC generated and address error. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
5:4 (R/NW)	ERRC	Error Cause. When the SEC updates the SEC_GSTAT.ERR bit, the SEC updates the SEC_GSTAT.ERRC bits to indicate the error type. Note that for SSI errors, the error status indicates an error is active for any SSI input. This error is an OR of all the interrupt source errors.	
		0	SFI Error
		1	Reserved
		2	SSI Error
		3	Reserved

Table 10-20: SEC_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	ERR	Error. The SEC_GSTAT.ERR bit indicates an error has occurred in the SEC. When the SEC asserts this bit (=1), the SEC updates the SEC_GSTAT.ERRC field to indicate the corresponding error cause. Even if multiple errors occur, only the first error is captured on assertion of this bit. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred

Global Raise Register

The SEC global raise register (SEC_RAISE) contains a source ID event set-to-pending field (SEC_RAISE.SID). When a source ID value is written to this field, the SEC raises the source's event status to pending.

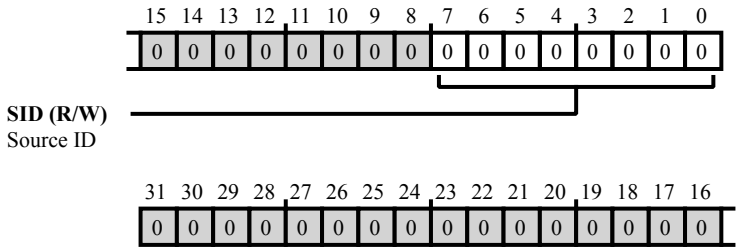


Figure 10-16: SEC_RAISE Register Diagram

Table 10-21: SEC_RAISE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID. The SEC_RAISE.SID bit field is the source ID of event that is set to pending status.

Source Control Register n

The SEC source control register (SEC_SCTL[n]) contains control bits to configure the SEC event sources. This register controls the configuration of the corresponding SEC event source.

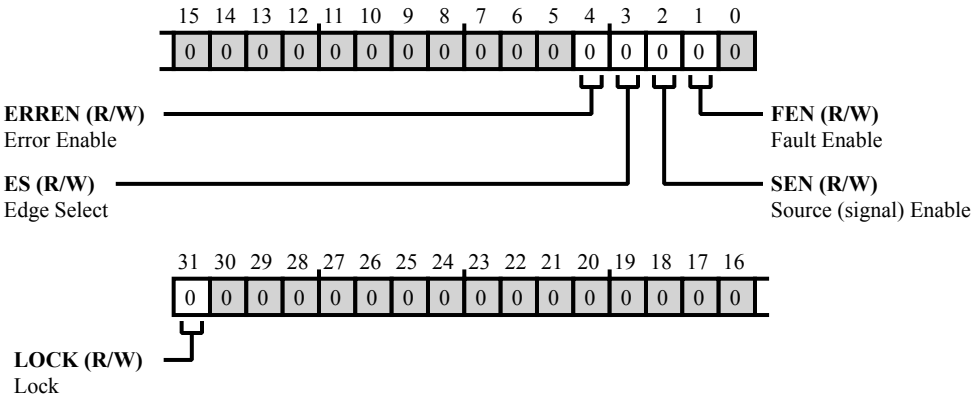


Figure 10-17: SEC_SCTL[n] Register Diagram

Table 10-22: SEC_SCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the SEC_SCTL[n].LOCK bit is enabled, the SEC_SCTL[n] register is read only.	
		0	Unlock
		1	Lock
4 (R/W)	ERREN	Error Enable. The SEC_SCTL[n].ERREN bit permits the SEC_SSTAT[n].ERR status bit to be set on error detection. If SEC_SCTL[n].ERREN is cleared, no errors are detected.	
		0	Disable
		1	Enable
3 (R/W)	ES	Edge Select. The SEC_SCTL[n].ES bit selects the operational and sensitivity mode of the SEC source interface input.	
		0	Level Sensitive
		1	Edge Sensitive

Table 10-22: SEC_SCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	SEN	Source (signal) Enable. The `SEC_SCTL[n].SEN` bit controls whether the system event source input signal may affect the pending status of the source. Clearing the `SEC_SCTL[n].SEN` bit disables the source input signal from affecting the pending status. Setting `SEC_SCTL[n].SEN` enables the source input signal to affect the pending status.	
		0	Disable
		1	Enable
1 (R/W)	FEN	Fault Enable. The `SEC_SCTL[n].FEN` bit controls whether the SEC may forward an interrupt request to the SEC fault interface as a fault source. This bit does not affect the ability of an interrupt source to set an interrupt as pending. The `SEC_SCTL[n].FEN` bit only affects whether the pending request may be forwarded to the SEC fault interface.	
		0	Disable
		1	Enable

Source Status Register n

The SEC event source status register (SEC_SSTAT[n]) contains bits indicating the status of the corresponding event source n. An event source may be: pending, active, active and pending, or neither pending nor active.

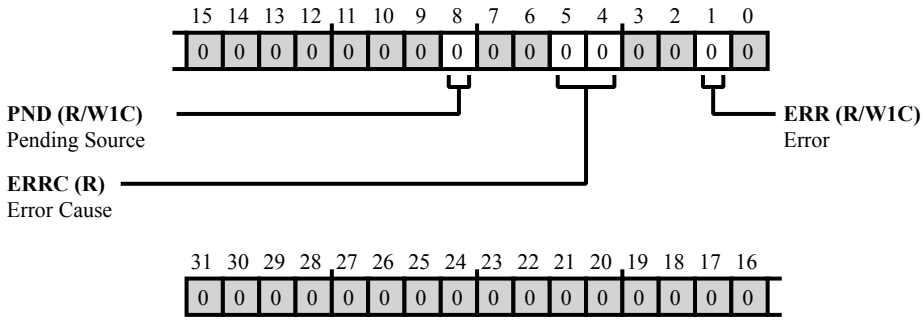


Figure 10-18: SEC_SSTAT[n] Register Diagram

Table 10-23: SEC_SSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	PND	Pending Source. The SEC_SSTAT[n].PND bit indicates the source has signaled an event request, but the event request has not been (or is not currently being) serviced. A SEC_SSTAT[n].PND bit is set by the SEC on detection of an assertion of the corresponding system source input. A SEC_SSTAT[n].PND bit is cleared by a W1C operation.	
		0	Not Pending
		1	Pending
5:4 (R/NW)	ERRC	Error Cause. When the SEC_SSTAT[n].ERR bit is asserted, the SEC updates SEC_SSTAT[n].ERRC field to convey the interrupt source error type. When the error type is source overflow, the status indicates that a source signal assertion occurred or an SEC raise operation was attempted while pending was already set. The source overflow is detected when the source is set for edge only. .	
		0	Source Overflow Error
		1	Reserved
		2	End Error
		3	Reserved

Table 10-23: SEC_SSTAT[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	ERR	Error. The SEC_SSTAT[n].ERR bit indicates an error for a specific system interrupt source. When the SEC_SSTAT[n].ERR bit is set, the SEC updates the SEC_SSTAT[n].ERRC field to the value of the corresponding error cause. Even if multiple errors occur, only the first error is captured on assertion of the SEC_SSTAT[n].ERR bit.	
		0	No Error
		1	Error Occurred

11 Trigger Routing Unit (TRU)

The TRU provides system-level sequence control without core intervention. The TRU maps trigger masters (generators of triggers) to trigger slaves (receivers of triggers). Slave endpoints can be configured to respond to triggers in various ways. Multiple TRUs may be provided in a multiprocessor system to create a trigger network. Common applications enabled by the TRU include:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes
- Software triggering
- Synchronization of concurrent activities

TRU Features

The TRU supports the following features:

- Automatically triggering the start of a DMA sequence after a sequence from another DMA channel completes. Once a DMA channel completes data transfer, it can act as a Trigger Master and signal an internal trigger pulse to the programmed Trigger Slave which can also be another DMA channel. The Slave Trigger connected to the DMA channel kicks off the DMA transfer automatically. None of this requires core intervention once the initialization is done.
- Software triggers. The best use of triggers is to minimize core intervention. It is also possible to initiate a trigger pulse to a Trigger Slave, in the software.
- Synchronization of concurrent activities. A single Trigger Master can initiate a trigger pulse to multiple Trigger Slaves so that several system level activities can be synchronized on an internally or externally generated event.
- Configuration protection through register-level lock bits and global lock indication

TRU Functional Description

The following sections provide a description of the TRU.

CM41X_M4 TRU Register List

The Trigger Routing Unit (TRU) provides simple sequence control of distributed modules without the penalties associated with core intervention (for example, interrupt overhead). The TRU receives trigger inputs from all master trigger inputs (MTI) and the TRU master trigger register (TRU_MTR). Based on these inputs, the TRU logic generates trigger outputs that initiate slave operations in the processor core and peripherals. A set of registers governs TRU operations. For more information on TRU functionality, see the TRU register descriptions.

Table 11-1: CM41X_M4 TRU Register List

Name	Description
TRU_ERRADDR	Error Address Register
TRU_GCTL	Global Control Register
TRU_MTR	Master Trigger Register
TRU_SSR[n]	Slave Select Register
TRU_STAT	Status Information Register

CM41X_M0 TRU Interrupt List

Table 11-2: CM41X_M0 TRU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
20	TRU0_INT1	TRU0 Interrupt request 1	Edge	
21	TRU1_INT4	TRU1 Interrupt request 4	Edge	

CM41X_M4 TRU Interrupt List

Table 11-3: CM41X_M4 TRU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
123	TRU1_INT0	TRU1 Interrupt request 0	Edge	
124	TRU1_INT1	TRU1 Interrupt request 1	Edge	
125	TRU1_INT2	TRU1 Interrupt request 2	Edge	
126	TRU1_INT3	TRU1 Interrupt request 3	Edge	
127	TRU0_INT0	TRU0 Interrupt request 0	Edge	

CM41X_M0 TRU Trigger List

Table 11-4: CM41X_M0 TRU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
27	TRU1_GTP0	TRU1 Generic Trigger Outputs 0	Edge
28	TRU1_GTP1	TRU1 Generic Trigger Outputs 1	Edge
29	TRU1_GTP2	TRU1 Generic Trigger Outputs 2	Edge
30	TRU1_GTP3	TRU1 Generic Trigger Outputs 3	Edge
31	TRU1_GTP4	TRU1 Generic Trigger Outputs 4	Edge
32	TRU1_GTP5	TRU1 Generic Trigger Outputs 5	Edge
33	TRU1_GTP6	TRU1 Generic Trigger Outputs 6	Edge
34	TRU1_GTP7	TRU1 Generic Trigger Outputs 7	Edge
35	TRU1_GTP8	TRU1 Generic Trigger Outputs 8	Edge
36	TRU1_GTP9	TRU1 Generic Trigger Outputs 9	Edge
37	TRU1_GTP10	TRU1 Generic Trigger Outputs 10	Edge
38	TRU1_GTP11	TRU1 Generic Trigger Outputs 11	Edge
39	TRU1_GTP12	TRU1 Generic Trigger Outputs 12	Edge
40	TRU1_GTP13	TRU1 Generic Trigger Outputs 13	Edge
41	TRU1_GTP14	TRU1 Generic Trigger Outputs 14	Edge
42	TRU1_GTP15	TRU1 Generic Trigger Outputs 15	Edge
43	TRU1_GTP16	TRU1 Generic Trigger Outputs 16	Edge
44	TRU1_GTP17	TRU1 Generic Trigger Outputs 17	Edge
45	TRU1_GTP18	TRU1 Generic Trigger Outputs 18	Edge
46	TRU1_GTP19	TRU1 Generic Trigger Outputs 19	Edge
47	TRU1_GTP20	TRU1 Generic Trigger Outputs 20	Edge
48	TRU1_GTP21	TRU1 Generic Trigger Outputs 21	Edge
49	TRU1_GTP22	TRU1 Generic Trigger Outputs 22	Edge
50	TRU1_GTP23	TRU1 Generic Trigger Outputs 23	Edge
51	TRU1_GTP24	TRU1 Generic Trigger Outputs 24	Edge
52	TRU1_GTP25	TRU1 Generic Trigger Outputs 25	Edge
53	TRU1_GTP26	TRU1 Generic Trigger Outputs 26	Edge
54	TRU1_GTP27	TRU1 Generic Trigger Outputs 27	Edge
55	TRU1_GTP28	TRU1 Generic Trigger Outputs 28	Edge
56	TRU1_GTP29	TRU1 Generic Trigger Outputs 29	Edge

Table 11-4: CM41X_M0 TRU Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
57	TRU1_GTP30	TRU1 Generic Trigger Outputs 30	Edge
58	TRU1_GTP31	TRU1 Generic Trigger Outputs 31	Edge

Table 11-5: CM41X_M0 TRU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
41	TRU0_INT0	TRU0 Interrupt request 0	Pulse
42	TRU0_INT1	TRU0 Interrupt request 1	Pulse
43	TRU0_GTP0	TRU0 Generic Trigger Ports 0	Pulse
44	TRU0_GTP1	TRU0 Generic Trigger Ports 1	Pulse
45	TRU0_GTP2	TRU0 Generic Trigger Ports 2	Pulse
46	TRU0_GTP3	TRU0 Generic Trigger Ports 3	Pulse

CM41X_M0 Trigger List

Table 11-6: CM41X_M0 Trigger List Masters

Trigger ID	Name	Description	Sensitivity
1	SEC0_FAULT	SEC0 Fault	Edge
2	SEC1_FAULT	SEC1 Fault	Edge
3	CGU0_EVT	CGU0 Event	Edge
4	TIMER0_TMR0_MST	TIMER0 TMR 0 Trigger Master	Edge
5	TIMER0_TMR1_MST	TIMER0 TMR 1 Trigger Master	Edge
6	TIMER0_TMR2_MST	TIMER0 TMR 2 Trigger Master	Edge
7	TIMER0_TMR3_MST	TIMER0 TMR 3 Trigger Master	Edge
8	TIMER0_TMR4_MST	TIMER0 TMR 4 Trigger Master	Edge
9	TIMER0_TMR5_MST	TIMER0 TMR 5 Trigger Master	Edge
10	TIMER0_TMR6_MST	TIMER0 TMR 6 Trigger Master	Edge
11	TIMER0_TMR7_MST	TIMER0 TMR 7 Trigger Master	Edge
12	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level
13	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level
14	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level
15	DMA0_MST	DMA0 Trigger Master	Edge
16	DMA1_MST	DMA1 Trigger Master	Edge

Table 11-6: CM41X_M0 Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
17	DMA2_MST	DMA2 Trigger Master	Edge
18	DMA3_MST	DMA3 Trigger Master	Edge
19	CAN0_STAT	CAN0 Status	Level
23	SWU0_EVT	SWU0 Event	None
24	SWU1_EVT	SWU1 Event	None
59	AFE_LIMIT	AFE FOCP Limit Detection	Level
60	AFE_NOTOK	AFE is functioning normally	Edge

Table 11-7: CM41X_M0 Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
0	RCU0_SYSRST0	RCU0 System Reset 0	Pulse
1	TIMER0_TMR0_SLV0	TIMER0 TMR 0 Trigger Slave 0	Pulse
2	TIMER0_TMR0_SLV1	TIMER0 TMR 0 Trigger Slave 1	Pulse
3	TIMER0_TMR1_SLV0	TIMER0 TMR 1 Trigger Slave 0	Pulse
4	TIMER0_TMR1_SLV1	TIMER0 TMR 1 Trigger Slave 1	Pulse
5	TIMER0_TMR2_SLV0	TIMER0 TMR 2 Trigger Slave 0	Pulse
6	TIMER0_TMR2_SLV1	TIMER0 TMR 2 Trigger Slave 1	Pulse
7	TIMER0_TMR3_SLV0	TIMER0 TMR 3 Trigger Slave 0	Pulse
8	TIMER0_TMR3_SLV1	TIMER0 TMR 3 Trigger Slave 1	Pulse
9	TIMER0_TMR4_SLV0	TIMER0 TMR 4 Trigger Slave 0	Pulse
10	TIMER0_TMR4_SLV1	TIMER0 TMR 4 Trigger Slave 1	Pulse
11	TIMER0_TMR5_SLV0	TIMER0 TMR 5 Trigger Slave 0	Pulse
12	TIMER0_TMR5_SLV1	TIMER0 TMR 5 Trigger Slave 1	Pulse
13	TIMER0_TMR6_SLV0	TIMER0 TMR 6 Trigger Slave 0	Pulse
14	TIMER0_TMR6_SLV1	TIMER0 TMR 6 Trigger Slave 1	Pulse
15	TIMER0_TMR7_SLV0	TIMER0 TMR 7 Trigger Slave 0	Pulse
16	TIMER0_TMR7_SLV1	TIMER0 TMR 7 Trigger Slave 1	Pulse
17	PORTA_TOGGLE	PORTA Port Toggle Trigger	Pulse
18	M0P_NMI_SLV0	M0P NMI Trigger Slave 0	Pulse
20	ADCC0_SLV0	ADCC0 Trigger Slave 0	Pulse
21	ADCC0_SLV1	ADCC0 Trigger Slave 1	Pulse

Table 11-7: CM41X_M0 Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
22	ADCC0_SLV2	ADCC0 Trigger Slave 2	Pulse
23	ADCC0_SLV3	ADCC0 Trigger Slave 3	Pulse
24	ADCC0_SLV4	ADCC0 Trigger Slave 4	Pulse
25	ADCC0_SLV5	ADCC0 Trigger Slave 5	Pulse
26	DMA0_SLV0	DMA0 Trigger Slave 0	Pulse
27	DMA1_SLV0	DMA1 Trigger Slave 0	Pulse
28	DMA2_SLV0	DMA2 Trigger Slave 0	Pulse
29	DMA3_SLV0	DMA3 Trigger Slave 0	Pulse
30	SWU0_EN	SWU0 Enable	Pulse
31	SWU1_EN	SWU1 Enable	Pulse

CM41X_M4 TRU Trigger List

Table 11-8: CM41X_M4 TRU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
74	TRU0_GTP0	TRU0 Generic Trigger Ports	Edge
75	TRU0_GTP1	TRU0 Generic Trigger Ports	Edge
76	TRU0_GTP2	TRU0 Generic Trigger Ports	Edge
77	TRU0_GTP3	TRU0 Generic Trigger Ports	Edge

Table 11-9: CM41X_M4 TRU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
98	TRU1_INT0	TRU1 Interrupt request 0	Pulse
99	TRU1_INT1	TRU1 Interrupt request 1	Pulse
100	TRU1_INT2	TRU1 Interrupt request 2	Pulse
101	TRU1_INT3	TRU1 Interrupt request 3	Pulse
102	TRU1_INT4	TRU1 Interrupt request 4	Pulse
103	TRU1_GTP0	TRU1 Generic Trigger Outputs 0	Pulse
104	TRU1_GTP1	TRU1 Generic Trigger Outputs 1	Pulse
105	TRU1_GTP2	TRU1 Generic Trigger Outputs 2	Pulse
106	TRU1_GTP3	TRU1 Generic Trigger Outputs 3	Pulse
107	TRU1_GTP4	TRU1 Generic Trigger Outputs 4	Pulse

Table 11-9: CM41X_M4 TRU Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
108	TRU1_GTP5	TRU1 Generic Trigger Outputs 5	Pulse
109	TRU1_GTP6	TRU1 Generic Trigger Outputs 6	Pulse
110	TRU1_GTP7	TRU1 Generic Trigger Outputs 7	Pulse
111	TRU1_GTP8	TRU1 Generic Trigger Outputs 8	Pulse
112	TRU1_GTP9	TRU1 Generic Trigger Outputs 9	Pulse
113	TRU1_GTP10	TRU1 Generic Trigger Outputs 10	Pulse
114	TRU1_GTP11	TRU1 Generic Trigger Outputs 11	Pulse
115	TRU1_GTP12	TRU1 Generic Trigger Outputs 12	Pulse
116	TRU1_GTP13	TRU1 Generic Trigger Outputs 13	Pulse
117	TRU1_GTP14	TRU1 Generic Trigger Outputs 14	Pulse
118	TRU1_GTP15	TRU1 Generic Trigger Outputs 15	Pulse
119	TRU1_GTP16	TRU1 Generic Trigger Outputs 16	Pulse
120	TRU1_GTP17	TRU1 Generic Trigger Outputs 17	Pulse
121	TRU1_GTP18	TRU1 Generic Trigger Outputs 18	Pulse
122	TRU1_GTP19	TRU1 Generic Trigger Outputs 19	Pulse
123	TRU1_GTP20	TRU1 Generic Trigger Outputs 20	Pulse
124	TRU1_GTP21	TRU1 Generic Trigger Outputs 21	Pulse
125	TRU1_GTP22	TRU1 Generic Trigger Outputs 22	Pulse
126	TRU1_GTP23	TRU1 Generic Trigger Outputs 23	Pulse
127	TRU1_GTP24	TRU1 Generic Trigger Outputs 24	Pulse
128	TRU1_GTP25	TRU1 Generic Trigger Outputs 25	Pulse
129	TRU1_GTP26	TRU1 Generic Trigger Outputs 26	Pulse
130	TRU1_GTP27	TRU1 Generic Trigger Outputs 27	Pulse
131	TRU1_GTP28	TRU1 Generic Trigger Outputs 28	Pulse
132	TRU1_GTP29	TRU1 Generic Trigger Outputs 29	Pulse
133	TRU1_GTP30	TRU1 Generic Trigger Outputs 30	Pulse
134	TRU1_GTP31	TRU1 Generic Trigger Outputs 31	Pulse

TRU Definitions

The following definitions are helpful when using the TRU module.

Trigger Master

A trigger master is any system module that provides trigger event indication to the TRU. Trigger master modules define trigger events and conditions for assertion.

Trigger Master ID

Trigger masters are assigned a unique numeric ID according to their physical connection to the TRU. Trigger master ID 0 is reserved and defined as null.

Trigger Slave

A trigger slave is any system module that receives a trigger event indication from the TRU. Trigger slave modules define a trigger event response.

TRU Block Diagram

Trigger masters and the Master Trigger Register (MTR) generate trigger assertions. Each trigger slave has a dedicated Slave Select Register (SSR) that specifies the unique trigger master from which it receives the trigger indication.

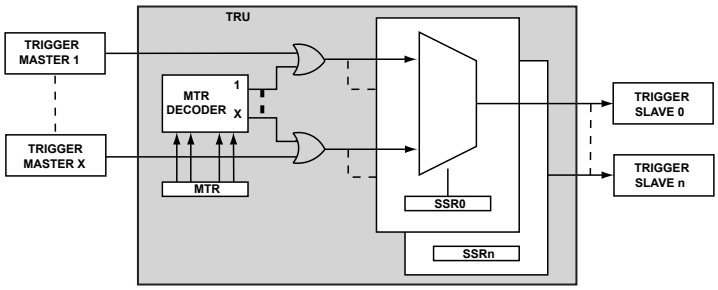


Figure 11-1: TRU Block Diagram

Inter Core Trigger Communication

In the ADSP-CM41xF, there is a mechanism to communicate between the TRUs of each of core. The TRU0 is assigned for trigger masters and trigger slaves associated with the Cortex-M0 platform and TRU1 is assigned for trigger masters and trigger slaves associated with the Cortex-M4 platform. The Generic Trigger Outputs (GTP) are used to generate internal triggers between TRU0 and TRU1. The TRU1 in the Cortex-M4 has 32 GTP trigger slaves, and four GTP trigger masters. The TRU0 in the Cortex-M0 has four GTP trigger slaves, and 32 GTP trigger slaves.

If a trigger is received at a GTP slave trigger input TRUn, it is automatically forwarded to the corresponding GTP master trigger at the other TRUn which can then generate a trigger to any slave. The ***Trigger Communication*** figure shows how the connections are accomplished.

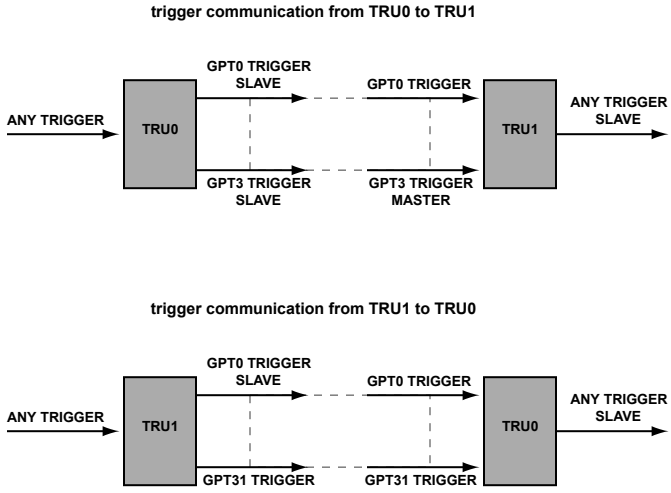


Figure 11-2: Trigger Communication

TRU Architectural Concepts

The TRU supports a simple trigger-in/trigger-out model for modules that comply with the triggering functional model. The TRU is the controller of the trigger system. Trigger outputs from trigger masters are mapped to trigger inputs of trigger slaves through a set of programmable registers (`TRU_SSR[n]`).

System modules are trigger master only, trigger slave only, or trigger master and trigger slave.

All of the trigger input and output signals are connected to a trigger routing unit (TRU) which manages the connections of triggers between modules.

In multi-processor systems, multiple TRU units are provided. These TRUs are networked together. Generic Trigger Ports (GTPs) are provided to forward trigger events from one TRU unit to another, forming a pathway from trigger masters to trigger slaves wherever they might lie in the system.

TRU Programming Model

Implementing sequence control using the TRU requires, at a minimum, proper configuration of a trigger slave, a trigger master, and the TRU module itself. The only requirement for the configuration procedure is that the trigger master is configured and enabled as the last step.

Complete the following other steps:

- Configure the trigger slave for response to triggers.
- Configure the TRU to map the trigger master to the trigger slave through the `TRU_SSR[n]` registers.
- Configure the trigger master to generate trigger assertions.
- Alternatively, use software triggering for trigger assertion. Writing the trigger master ID to the MTR register generates software triggers.

Programming Concepts

The following concepts aid in programming the TRU.

- ***Trigger Sequence Configuration.*** A simple sequence consists of one trigger master and one trigger slave. More complex trigger sequences consist of several trigger slaves functioning as trigger slave and trigger master. Additionally, trigger sequences can loopback to the original master forming a perpetual sequence.
- ***Software Triggering.*** Writing a trigger master ID to the MTR generates a trigger within the TRU from the trigger master ID specified.
- ***Synchronization.*** The TRU can be used to coarsely synchronize events by mapping multiple trigger slaves to the same trigger master or by generating multiple trigger master assertions simultaneously through the MTR.
- ***Configuration Protection.*** The `TRU_SSR[n].LOCK` bit and the `TRU_GCTL.LOCK` bit enable register level write-protection when the global lock is asserted in the SPU.

Programming Examples

The following examples shows the steps to create a single trigger and a Timer period expiry event automatically toggling a GPIO.

Configuring a Simple Trigger Sequence

The following example shows the steps to create a simple trigger.

1. Write to the `TRU_GCTL` register to enable the TRU.
2. Write to the `TRU_SSR[n]` register of a specific trigger slave to assign it to a specific trigger master.
3. Enable the trigger slave to wait for and accept a trigger.
4. Enable the trigger master to generate a trigger.

Toggle a GPIO on Timer Expiry Event

This example shows a case where a Timer period expiry event automatically toggles a GPIO. This is achieved by programming the Slave trigger register for GPIO with the Timer as Master.

1. Enable a specific pin in the PORT F to toggle up on trigger (`PORT_TRIG_TGL`)
2. Enable timer to generate trigger up (`TIMER_TRG_MSK` register)
3. Program the Slave trigger 23 (`TRU_SSR[n]` register) (this toggles the PORTs) with the Timer as the Master.
4. Enable the TRU (`TRU_GCTL.EN` bit).

```
*pREG_PORTF_TRIG_TGL = 0x00000010;
*pREG_TIMER1_TRG_MSK &= ~BITM_TIMER_TRG_MSK_TMR03;
*pREG_TRU1_SSR23 = TRGM_TIMER1_TMR3_MST;
*pREG_TRU1_GCTL = 0x1;
```

TRU Event Control

The TRU is a major part of event control solutions. It is the center of the trigger functional model and can extend to support the interrupt and fault management models as well.

TRU Status and Error Signals

The TRU does not have dedicated status and error output signals other than the MMR interface. Slave errors are reported to the master over the standard peripheral bus protocol.

Trigger Latencies

The *ADSP-CM41x Trigger Latencies* table

Table 11-10: ADSP-CM41x Trigger Latencies

Block	Parameter / Macro	From (Event source or block-level TRU Slave input port)	To (Event Result or block-level TRU Master output port)	Formula	Unit (clock cycles)	Comments
TRU0	ADI_TD_TRU	any TRU0 input (TRU master), for example TRGM_xxxx	any TRU0 output (TRU slave), for example TRGS_xxxx	4	SCLK0	Intrinsic TRU latency, M0-side TRU
TRU1	ADI_TD_TRU	any TRU1 input (TRU master), for example TRGM_xxxx	any TRU1 output (TRU slave), for example TRGS_xxxx	4	SYSCLK	Intrinsic TRU latency, M4-side TRU
TTU0	ADI_TD_TTU	TTU0 input (TRU slave): `TTU_TRIG_IN[n]`TRGS_TTU0_TRIG_INn	TTU0 output (TRU master): `TTU_TRIG_OUT[m]`TRGM_TTU0_TRIG_OUTn	2	SYSCLK	Intrinsic TTU delay
ADCC0	ADI_TD_ADCC_SYNC (ACLK)	ADCC0 sync trigger in: `ADCC_SLV0`TRGS_ADCC1_SLV0	ADCC0 AFE event start (first AFE_CSb)	3 + 2 × (ACLK)	SCLK0	ADCC0 latency, from trigger in to start of AFE event at EVTM=0, given AFE clkock divisor (TCAy.CKDIV) = aclk
ADCC1	ADI_TD_ADCC_SYNC (ACLK)	ADCC0 sync trigger in: TRGS_ADCC1_SLV1	ADCC1 AFE event start (first AFE_CSb)	3 + 2 × (ACLK)	SYSCLK	ADCC1 latency, from trigger in to start of AFE event at EVTM=0, given AFE clkock divisor (TCAy.CKDIV) = aclk
PWMn	ADI_TD_PWM_SYNC_IN	PWMn sync trigger in:	PWMn SYNC pin output pulse	3	SYSCLK	PWMn start latency from trigger to start of PWM waveforms, referenced to

Table 11-10: ADSP-CM41x Trigger Latencies (Continued)

Block	Parameter / Macro	From (Event source or block-level TRU Slave input port)	To (Event Result or block-level TRU Master output port)	Formula	Unit (clock cycles)	Comments
		TRGS_PWMn_SYNC				PWM_SYNC leading edge
PWMn	ADI_TD_PWM_SYNC_OUT	PWMn SYNC pin output pulse	PWMn SYNC trigger out: TRGM_SYS_PWMn_SYNC_IN	0	SYSCLK	PWM internal SYNC generator latency
PWMn	ADI_TD_PWM_TRIP	PWMn trip trigger in: TRGS_SYS_PWMn_TRIP_SLVx	PWMn outputs in tripped state	0	SYSCLK	PWMn trip latency to PWM waveforms in safe (tripped) states (asynchronous after TRU)
TMR0	ADI_TD_TMR_IN	Timer-in waveform	TMR0 trigger out: TRGM_TIMER0_TMRx_MST	TBD	SCLK0	Timer0 input latency, M0 side
TMR1	ADI_TD_TMR_IN	Timer-in waveform	TMR1 trigger out: TRGM_TIMER1_TMRx_MST	TBD	SYSCLK	Timer1 input latency, M4 side
TMR0	ADI_TD_TMR_OUT	TMR0 trigger in: TRGS_TIMER0_TMRx_SLVy	Timer-out waveform	TBD	SCLK0	Timer0 output latency, M0 side
TMR1	ADI_TD_TMR_OUT	TMR1 trigger in: TRGS_TIMER1_TMRx_SLVy	Timer-out waveform	TBD	SYSCLK	Timer1 output latency, M4 side
PORTA	ADI_TD_PORT_TOGGLE	GPIO toggle trigger in: TRGS_PORTA_TOGGLE	GPIO PORTA pin toggle	1	SCLK0	GPIO Toggle latency, M0 side
PORTB-F	ADI_TD_PORT_TOGGLE	GPIO toggle trigger in: TRGS_PORTx_TOGGLE	GPIO PORTB-F pin toggle	1	SYSCLK	GPIO Toggle latency, M4 side
PORTA-F	ADI_TD_PORT_FRC_SAFE	Functional safety trigger in: TRGS_SYS_FRC_PIN_SAFE_SLV0	GPIO PORTA-F pin in safe state	0	SYSCLK	GPIO Pin Safe State latency (asynchronous after TRU)
SINC0	ADI_TD_SINC	SINC0 sync trigger in: TRGS_SINC0_SYNC0	SINC0 MCLK leading edge	TBD	SYSCLK	SINC unit trigger-to-MCLK latency

Table 11-10: ADSP-CM41x Trigger Latencies (Continued)

Block	Parameter / Macro	From (Event source or block-level TRU Slave input port)	To (Event Result or block-level TRU Master output port)	Formula	Unit (clock cycles)	Comments
CPTMR	ADI_TD_CPTMR	Capture Timer trigger in: TRGS_CPTMR0_CPTx_SLVy	(Capture timer event -what event?)	TBD	SYSCLK	Capture Timer latency
DDE0-3	ADI_TD_DDE	DMA Engine trigger in: TRGS_DMA{0-3}_SLV0	(DMA transaction start?)	TBD	SCLK0	DDE latency, M0 side
DDE4-13	ADI_TD_DDE	DMA Engine trigger in: TRGS_DMA{4-13}_SLV0	(DMA transaction start?)	TBD	SYSCLK	DDE latency, M4 side

CM41X_M4 TRU Register Descriptions

Trigger Routing Unit (TRU) contains the following registers.

Table 11-11: CM41X_M4 TRU Register List

Name	Description
TRU_ERRADDR	Error Address Register
TRU_GCTL	Global Control Register
TRU_MTR	Master Trigger Register
TRU_SSR[n]	Slave Select Register
TRU_STAT	Status Information Register

Error Address Register

The TRU error address register (TRU_ERRADDR) holds the address from the memory-mapped register access generating an access error of TRU registers.

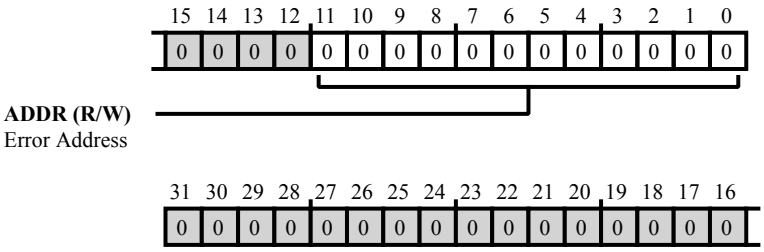


Figure 11-3: TRU_ERRADDR Register Diagram

Table 11-12: TRU_ERRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	ADDR	Error Address. The TRU_ERRADDR.ADDR holds the address from the memory-mapped register access generating an access error of TRU registers. These errors occur on access to the TRU_SSR[n] or TRU_MTR registers when these registers are locked or on access to an invalid address. See the TRU_SSR[n] and TRU_MTR register descriptions for more information about locking. The TRU_ERRADDR register holds the address of the first error to occur. In the event of multiple errors occurring, the TRU_ERRADDR register contains the address of the first error. To re-enable the TRU_ERRADDR register for update, both status bits (TRU_STAT.LWERR and TRU_STAT.ADDRERR) in the TRU_STAT register must be cleared.

Global Control Register

The TRU global control register (TRU_GCTL) provides register locking, TRU reset, and TRU enable.

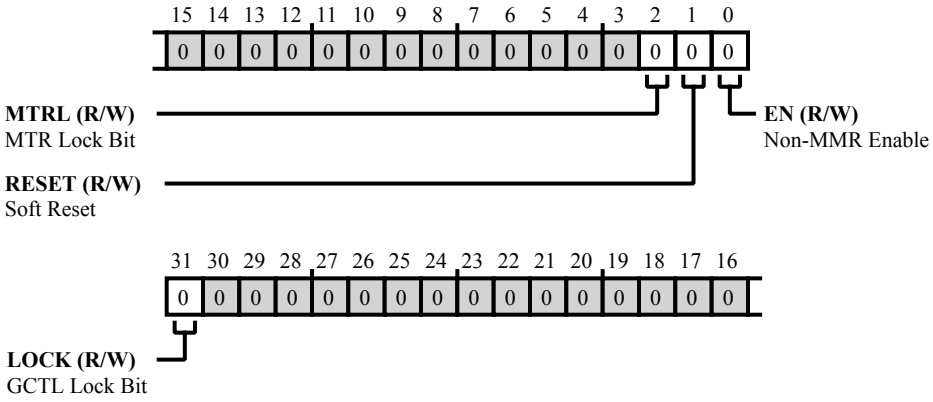


Figure 11-4: TRU_GCTL Register Diagram

Table 11-13: TRU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	GCTL Lock Bit. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the TRU_GCTL.LOCK bit is enabled, the TRU_GCTL register is read only.	
		0	Read write
		1	Read only
2 (R/W)	MTRL	MTR Lock Bit. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the TRU_GCTL.MTRL bit is enabled, the TRU_MTR register is read only.	
		0	Read write
		1	Read only
1 (R/W)	RESET	Soft Reset. The TRU_GCTL.RESET bit is write-1-action and triggers a soft reset to all TRU registers.	
		0	No action
		1	Soft reset

Table 11-13: TRU_GCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Non-MMR Enable. The `TRU_GCTL.EN` bit is read/write and must be set for the TRU to propagate trigger events. All TRU register read/write operations continue to operate independent of the `TRU_GCTL.EN` bit.	
		0	No trigger events
		1	Propagate trigger events

Master Trigger Register

The TRU master trigger register (TRU_MTR) permits trigger generation through software by writing a trigger master ID value to one of the four fields in the TRU_MTR register. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the TRU_GCTL.LOCK bit is set, the TRU_MTR register is read only. Note this register is primarily used for debug to trigger a TRU output

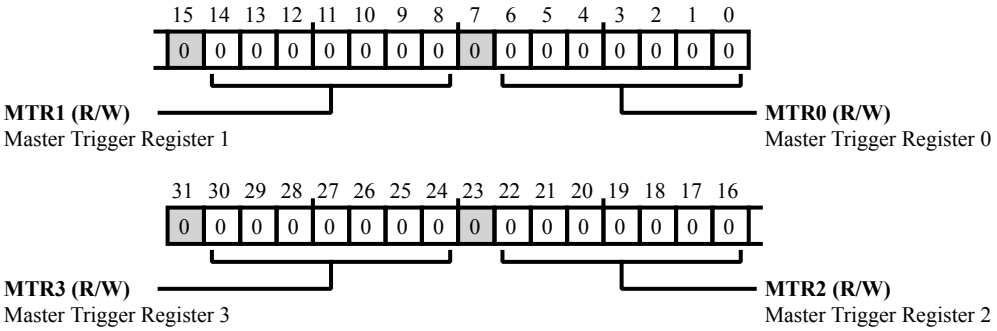


Figure 11-5: TRU_MTR Register Diagram

Table 11-14: TRU_MTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30:24 (R/W)	MTR3	Master Trigger Register 3. The TRU_MTR.MTR3 bit field is the trigger master ID value for master 3.	
		0	No master specified
		1-97	Range of valid masters
22:16 (R/W)	MTR2	Master Trigger Register 2. The TRU_MTR.MTR2 bit field is the trigger master ID value for master 2.	
		0	No master specified
		1-97	Range of valid masters
14:8 (R/W)	MTR1	Master Trigger Register 1. The TRU_MTR.MTR1 bit field is the trigger master ID value for master 1.	
		0	No master specified
		1-97	Range of valid masters
6:0 (R/W)	MTR0	Master Trigger Register 0. The TRU_MTR.MTR0 bit field is the trigger master ID value for master 0.	
		0	No master specified
		1-97	Range of valid masters

Slave Select Register

The TRU slave select registers (TRU_SSR[n]) each provide slave selection and register locking.

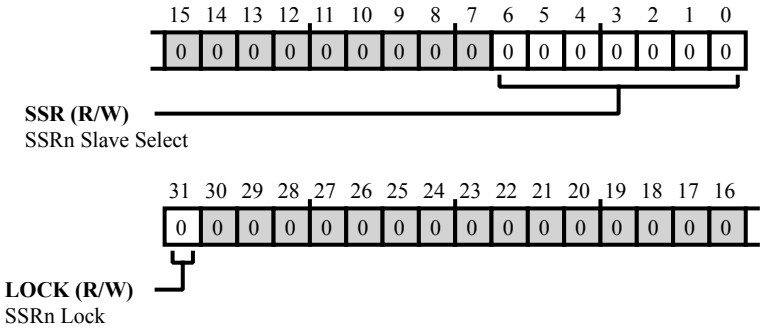


Figure 11-6: TRU_SSR[n] Register Diagram

Table 11-15: TRU_SSR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	SSRn Lock. If the global lock is enabled (SPU_CTL.GLCK bit =1) and the TRU_SSR[n].LOCK bit is enabled, the TRU_SSR[n] register is read only.	
		0	Unlock register
		1	Lock register
6:0 (R/W)	SSR	SSRn Slave Select. The TRU_SSR[n] register selects the trigger master ID to which the trigger slave responds. For example, when a TRU_SSR[n] register is set to respond to trigger master ID n, a trigger that is generated by trigger master ID n results in a trigger out to the slave.	
		0	No master specified
		1-97	Range of valid masters

Status Information Register

The TRU status register (TRU_STAT) contains the status of TRU_MTR and TRU_SSR[n] register writes and status of bus read/write errors.

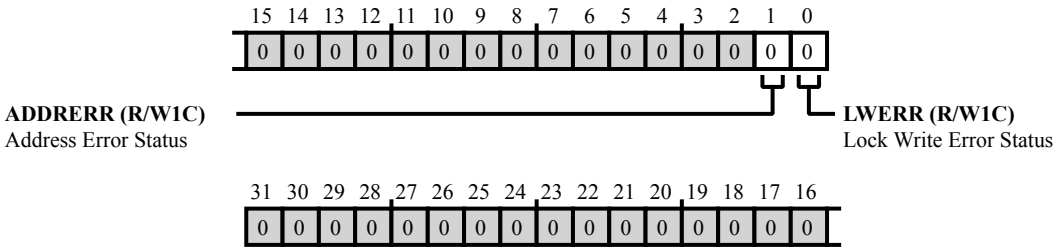


Figure 11-7: TRU_STAT Register Diagram

Table 11-16: TRU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	ADDRERR	Address Error Status. The TRU_STAT.ADDRERR bit is set when an invalid address is provided for an MMR access while the TRU is selected. Writing a one to this bit clears the error indication. The TRU_ERRADDR register also is updated when an address error occurs during an MMR access while the TRU is selected.	
		0	No error
		1	Error occurred
0 (R/W1C)	LWERR	Lock Write Error Status. If TRU_STAT.LWERR is set, a lock write error has occurred. Writing a one to this bit clears the error indication.	
		0	No error
		1	Error occurred

12 Trigger Timing Unit (TTU)

The Trigger Timing Unit (TTU) provides a simple way to generate event trigger signals with precise timing relationships. Triggers can then be routed to or from the TTU using the Trigger Routing Unit (TRU). The TTU is designed in a scalable, modular fashion. The available output trigger generators can be organized into independent trigger delay groups, each sensitive to its own input trigger.

Trigger delay groups in the TTU work in one of two functional modes. In single-shot mode, the TTU group can delay each input trigger pulse to make a constellation of one or more precisely-delayed output trigger pulses. In periodic mode, the TTU group generates a sequence of output triggers with precise timing relative to one another, using an internal periodic counter. An input trigger pulse starts the counter.

TTU Features

The TTU has the following features.

- Up to 16 trigger master outputs with independent time offsets (delays).
- Up to 8 timing counters with 24-bit SYSCLK resolution, initiated by independent trigger slave inputs.
- Counters can operate in single-shot or periodic mode.
- Output triggers are freely assignable to any timing counter source.
- In periodic mode, output triggers support both positive (lagging) and negative (leading) delay settings relative to the timing counter's zero time.
- Operates entirely in the SYSCLK clock domain.

TTU Functional Description

The TTU is used to connect system devices in precisely timed relationships, whether single-shot mode (one shot per trigger in) or periodic mode (repeating after started by trigger in). The trigger routing unit fabric routes trigger pulses from trigger masters (M1, M2, and so on.) to trigger slaves (S1, S2, and so on).

CM41X_M4 TTU Register List

A set of registers governs TTU operations. For more information on TTU functionality, see the TTU register descriptions.

Table 12-1: CM41X_M4 TTU Register List

Name	Description
TTU_CNT[n]	Counter Current Value
TTU_CTL	TTU Control Register
TTU_DGRP[m]	Trigger Output Counter Assignment
TTU_DLY[m]	Trigger Output Delay Register
TTU_PER[n]	Counter Period Register
TTU_REVID	Revision ID Register
TTU_STAT	Counter Status Register
TTU_STOP	Counter Stop Request Register
TTU_CHK	Counter Check Register

CM41X_M4 TTU Trigger List

Table 12-2: CM41X_M4 TTU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
88	TTU0_TRIG_OUT0	TTU0 Trigger output for trigger delay 0	Edge
89	TTU0_TRIG_OUT1	TTU0 Trigger output for trigger delay 1	Edge
90	TTU0_TRIG_OUT2	TTU0 Trigger output for trigger delay 2	Edge
91	TTU0_TRIG_OUT3	TTU0 Trigger output for trigger delay 3	Edge
92	TTU0_TRIG_OUT4	TTU0 Trigger output for trigger delay 4	Edge
93	TTU0_TRIG_OUT5	TTU0 Trigger output for trigger delay 5	Edge
94	TTU0_TRIG_OUT6	TTU0 Trigger output for trigger delay 6	Edge
95	TTU0_TRIG_OUT7	TTU0 Trigger output for trigger delay 7	Edge

Table 12-3: CM41X_M4 TTU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
92	TTU0_TRIG_IN0	TTU0 Trigger input to delay counter 0	Pulse
93	TTU0_TRIG_IN1	TTU0 Trigger input to delay counter 1	Pulse
94	TTU0_TRIG_IN2	TTU0 Trigger input to delay counter 2	Pulse
95	TTU0_TRIG_IN3	TTU0 Trigger input to delay counter 3	Pulse

TTU Definitions

To make the best use of the TTU, it is useful to understand the following terms.

Counter

A TRU trigger slave input starts the counter. Once started, it runs either periodically or until all associated trigger delay outputs are finished. The period of a TTU counter *n* is programmed with the corresponding `TTU_PER[n]` register, and its current count value is available in the `TTU_CNT[n]` register.

Delay Unit

A comparator that is associated with one of the TTU counters. It generates a delayed TRU trigger slave output when the count value of the counter matches the programmed delay of the unit. The delay of a TTU delay unit *m* is programmed with the corresponding `TTU_DLY[m]` register.

Delay Group

Consists of one TTU counter and the set of all TTU delay units associated with it. The assignment of delay units to counters is made by programming the `TTU_DGRP[m]` registers.

Trigger Routing Unit

A system infrastructure fabric which routes event pulses (triggers) from masters to slaves according to programmable routing settings.

Trigger Pulse

An assertion of a trigger signal. Trigger pulse signals are synchronous to the system clock domain (SYSCLK) and are active high. The first SYSCLK cycle in which the trigger signal is asserted indicates the occurrence of the trigger event. The width of the trigger pulse is not significant. The response of the TRU is the same in the two cases: when the signal is asserted (high) for multiple successive cycles and when the signal is asserted for a single cycle.

The trigger pulses generated by the TTU trigger master outputs `TTU_TRIG_OUT[m]` are normally one clock cycle in length. A `TTU_TRIG_OUT[m]` pulse can be extended if the DEBUG_HALT signal is asserted (for example, an emulation breakpoint pauses the SOC) to pause the associated counter `TTU_CNT[n]` in the count value that matches the delay of the output channel.

Trigger Master

The source of a trigger pulse routed by the TRU fabric. This source can be a hardware event such as a timer output, the completion of a DMA transfer, or a software event initiated by writing an MMR register in the TRU.

Trigger Slave

The destination of a trigger pulse routed by the TRU fabric. This destination is a system peripheral or other resource that takes a specific action when the trigger pulse is received. This action can be a hardware event, such as toggling a GPIO, starting a PWM or timer, or initiating an ADC sampling event, or it can be a software event (an interrupt).

TTU Block Diagram

The ***TTU Block Diagram*** shows an example of the trigger unit. Trigger source M1 (for example, a GPIO input) is routed through the TRU fabric to TRU slave S1, which connects to TTU counter 0. This counter controls a group of TTU delays which drive trigger masters M10 and M11, each with a different delay. These triggers are routed through the TRU to slaves S10, S11, and S12, which connect to the ADC Controller (ADCC) and to two PWM units, 0 and 1.

Similarly, the second TTU group, delay group 1, routes a trigger from trigger source M2 to trigger slaves S13, S14 and S15. But, in this case, the TTU counter is programmed for periodic operation. This configuration shows that once started by trigger master M2 (which could be a software MMR write), the same pattern of output triggers is precisely generated at the selected periodic rate.

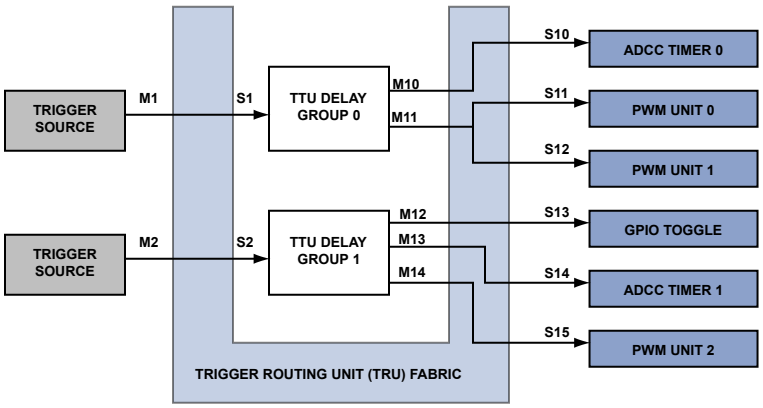


Figure 12-1: TTU Trigger System Diagram

Module Level Block Diagram

The ***Module Level Block Diagram*** shows the relationship of the TTU counters, COUNTERn, assigned to the trigger delay units, TRIGm_DELAY. The global `TTU_CTL` register is used to enable any or all of the TTU counters. The `TTU_DGRP[m]` registers are used to assign TRIGm_DELAY units to specific TTU counters (where every TRIGm_DELAY is associated with only one COUNTERn).

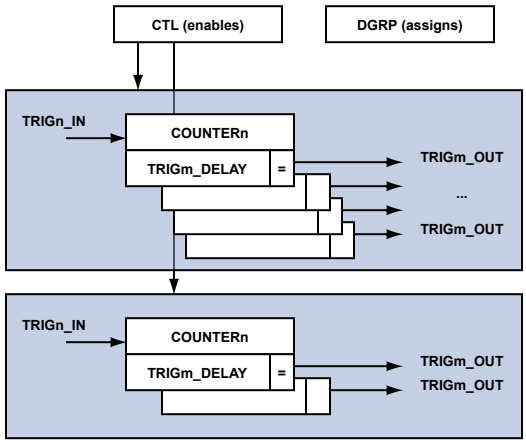


Figure 12-2: Module Level Block Diagram

TTU Architectural Concepts

The following section provides information about the debug interface.

Debug Interface

The TTU receives signaling from the debug system of the SoC. This interface consists of the DEBUG_HALT signal, which indicates to the TTU that the SoC debug system has entered a halt state (for example, a breakpoint).

Using the `TTU_CTL.EMURUN[n]` bits, programs can individually control the function of each counter during the debug halt state. If the `TTU_CTL.EMURUN[n]` bit =1, then counter *n* runs during debug halt. If the `TTU_CTL.EMURUN[n]` bit =0, then counter *n* pauses and does not increment during debug halt.

TTU Operating Modes

The TTU counters can operate in single-shot mode or in periodic mode.

In the following sections, the TTU is configured such that:

- Input trigger slaves n are connected to trigger counters n
- Output trigger masters m are connected to trigger delay units m
- Using the assignment registers `TTU_DGRP[m]`, the program partitions the available trigger delay units into trigger delay groups, each controlled by one of the counters.

Trigger inputs are effectively edge-sensitive: the trigger event is defined on the first *PCLK* cycle in which a `TTU_TRIG_IN[n]` signal is asserted high. Any subsequent asserted cycles have no effect. A `TTU_TRIG_IN[n]` signal must return to a deasserted (low) state for at least one cycle before another input event is recognized, as shown below.

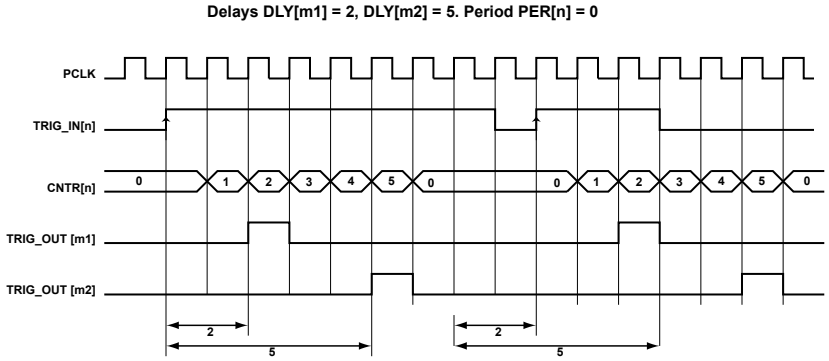


Figure 12-3: TRIG_IN Signal Timing

Single-Shot Mode

Single-shot mode is selected for a counter group (configured using the `TTU_CNT[n]` registers), by setting the `TTU_PER[n]` register to zero. In this mode, every pulse on the input trigger, `TTU_TRIG_IN[n]`, causes the `TTU_CNT[n]` register to begin counting when the counter is enabled by setting the `TTU_CTL.EMURUN[n]` bit =1. The counter takes the value 1 in the next cycle after the arriving trigger.

Each trigger delay unit in the delay group then observes this counter. During the cycle where the `TTU_CNT[n]` register value equals the selected delay in the `TTU_DLY[m]` register, the specific unit asserts its trigger output pulse, `TTU_TRIG_OUT[m]`, for one system clock cycle.

The counter then observes each of the delay units assigned to the delay group. The counter continues to count until each of the delay units has successfully generated a trigger (for example, when `TTU_DLY[m]` = `TTU_CNT[n]` has occurred for all delay units m in the counter group n). On the next cycle after the last output trigger is generated, the counter resets to zero and stops.

The active state of the counter group is available in the `TTU_STAT.RUN[n]` bit. The group becomes active upon receiving a `TTU_TRIG_IN[n]` pulse, and becomes inactive after the last `TTU_TRIG_OUT[m]` pulse has been generated.

The timing for single shot-mode is shown in the ***Single-Shot Mode Timing*** figure.

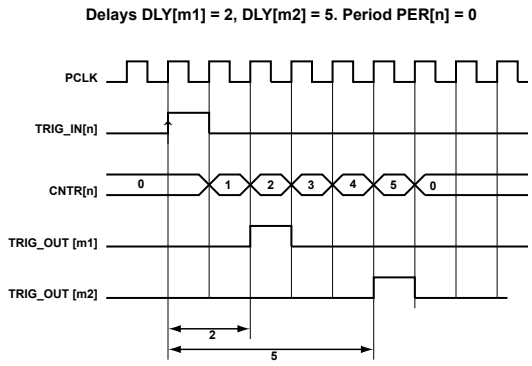


Figure 12-4: Single-Shot Mode Timing

Re triggering in Single-Shot Mode

If a trigger arrives while the counter is counting, the counter restarts and begins counting again at 1 on the next cycle. Any pending output triggers from the previous input trigger event are aborted (although an output trigger scheduled in the same cycle as the arrival of the new input trigger is generated on-time). Starting with the next PCLK cycle after the new trigger arrives, the delay group generates a complete sequence of delayed output triggers as programmed.

In the following timing diagram, two trigger-in pulses produce a first and second group of trigger-out pulses. The first group is truncated by a new trigger that arrives before the trigger out sequence is completed. The second group completes all of its sequence of programmed trigger-out pulses.

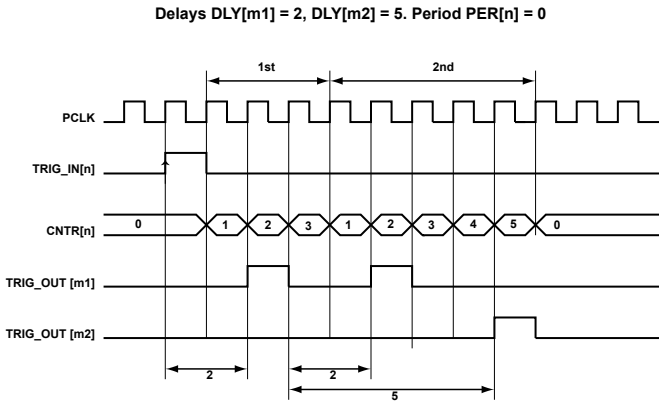


Figure 12-5: Re triggering in Single Shot Mode Timing

Restrictions on DLY in Single-Shot mode

In single-shot mode, the trigger delay values must be positive (`TTU_DLY[m]` ≥ 1). The TTU does not support a delay of zero from trigger-in to trigger-out. (If this function is desired, program the TRU to route the source of the input trigger directly to the destination of the output trigger without going through the TTU.)

The `TTU_DLY[m]` registers are double-buffered. If a new value is written to the `TTU_DLY[m]` register while the trigger group is operating, that new value is applied after the arrival of the next `TTU_TRIG_IN[n]` pulse.

Periodic Mode

Periodic mode is selected for a counter group, `TTU_CNT[n]`, by setting the `TTU_PER[n]` register to any nonzero value. (Period values are always zero or positive, but never negative.)

In periodic mode, once the counter is enabled by setting the `TTU_CTL.CNTREN[n]` bit to 1, the counter group waits for a trigger. When the trigger arrives, the counter becomes active, and begins counting up from 1 (on the next cycle after the trigger). The counter increments until the value in the `TTU_PER[n]` register =1, and then wraps back to zero. This process continues in a periodic way until the counter is stopped or disabled.

The minimum value for the period register `TTU_PER[n]` is 2. The maximum value is 2^{24} 1.

Each trigger delay unit in the delay group then observes this counter. During the cycle where the counter register value (TTU_CNT[n]) equals the selected delay in the TTU_DLY[m] register, the unit asserts its trigger output pulse TTU_TRIG_OUT[m] for one system clock.

In periodic mode, delay values can take on any value, including zero and negative values. Positive bit values (contained in the TTU_DLY[m].VALUE register) generate a *lagging* or delayed trigger pulse, which occur the specified number of cycles after the *zero cycle* of the counter for the delay group. Negative TTU_DLY[m].VALUE bit values generate a *leading* or advanced trigger pulse, which occur the specified number of cycles *before* the zero cycle. Zero values of TTU_DLY[m].VALUE result in a trigger pulse coincident with the zero cycle.

These relationships hold even if the period register TTU_PER[n] of the delay group is changed while active. The timing for this mode is shown in the figure below.

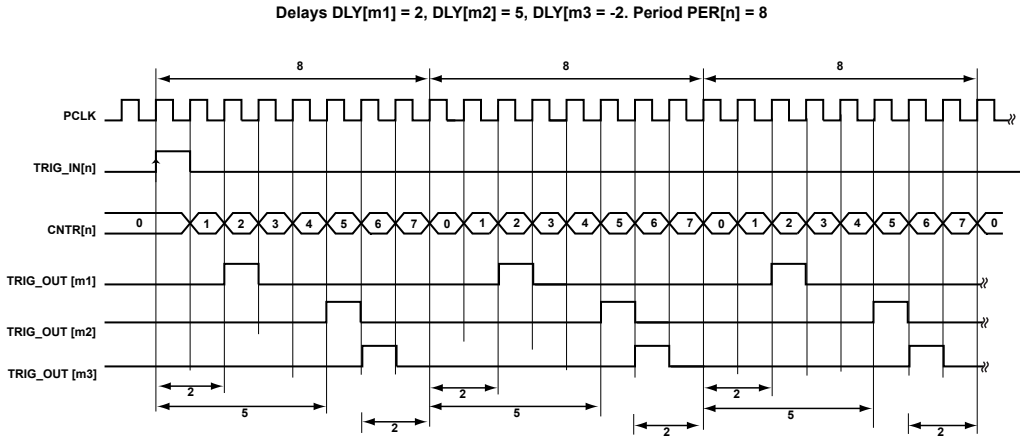


Figure 12-6: Periodic Mode Timing

Changing Period Dynamically

In the following timing diagram example, the period register, TTU_PER[n], is changed while the delay group is operating. Note how the leading (negative) delay on TTU_TRIG_OUT3 signal automatically responds to the change of period, arriving 2 PCLKs before the end of the period in both cases.

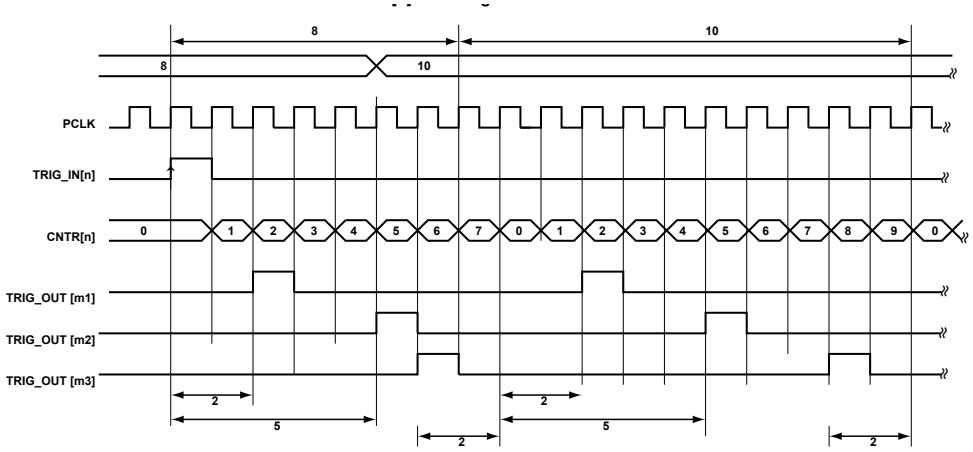


Figure 12-7: Changing Period Dynamically Timing

Re triggering

If a trigger arrives while the counter is counting, the counter restarts and begins counting again at 1 on the next cycle. Any output triggers that are still pending in the current counter period are aborted (although an output trigger scheduled in the same cycle as the arrival of the new input trigger is generated on-time). Starting with the next PCLK cycle after the new trigger arrives, the delay group generates a complete sequence of delayed output triggers, as programmed.

The TTU cannot predict leading (zero or negative) delays for new `TTU_TRIG_IN[n]` pulses which arrive while the counter is active and which change the phase of the counter. (A `TTU_TRIG_IN[n]` pulse which arrives in the same PCLK cycle as the zero cycle of the counter has no effect.)

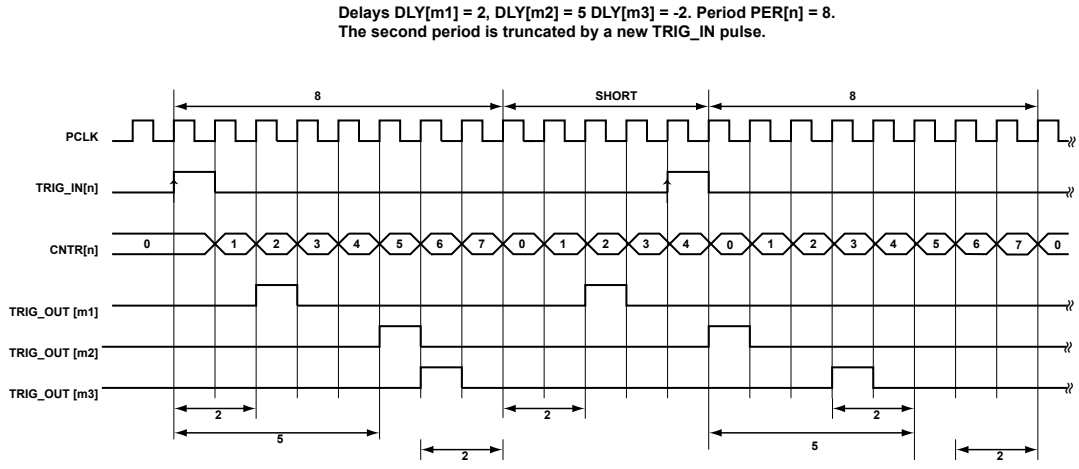


Figure 12-8: Re triggering in Periodic Mode Timing

If re triggering functionality is necessary to maintain phase synchronization between a periodic trigger delay group and some external system, observe the following guidelines:

- Determine the constraints on the system so that the maximum phase error on any new TTU_TRIG_IN[n] pulse can be known: $PE_{min} \leq$ phase error $\leq PE_{max}$. (Phase error is defined as the difference in time of the arrival of a trigger and the expected time of the group's zero cycle. In the example, the second trigger arrived four cycles early.)
- If the trigger can arrive early, for example, $PE_{min} \leq 0$, then ensure that:
  - All leading (negative) delays are less than or equal to PE_{min}, and
  - All lagging (positive) delays are less than or equal to (PE_{min} + TTU_PER[n]), so that they are generated before an early trigger can preempt them
- If the trigger can arrive late, for example, $PE_{max} \geq 0$, then ensure that:
  - All lagging (positive) delays are greater than PE_{max}, so that they are not generated twice, and
  - All leading (negative) delays are greater than (PE_{max} TTU_PER[n]), for the same reason

Stopping Periodic Trigger Delay Groups

A trigger delay group that is operating in periodic mode can be stopped gracefully or abruptly. To stop a periodic delay group gracefully, write the TTU_STOP.STOPREQ[n] bit to 1 for that delay group counter. This operation requests that the counter completes the current counter period, then stops. All the remaining trigger outputs for the current cycle are generated on-time. Reading the TTU_STOP.STOPREQ[n] bit provides the status of this operation. The bit reads 1 until the stop request has been satisfied and the counter has become inactive. In this state, when the counter is inactive but still enabled, another TTU_TRIG_IN[n] pulse can restart it. If this state is not desired, the trigger source or TRU routing must be disabled before issuing the TTU_STOP.STOPREQ[n].

The TTU_STOP.STOPREQ[n] feature only works for groups configured for periodic operation. (Groups configured for single-shot operation stop gracefully on their own at the end of each trigger sequence.)

To stop a delay group abruptly, write its TTU_CTL.CNTREN[n] bit =0. The counter stops immediately and no further trigger outputs are generated.

The active state of the counter group is available in the TTU_STAT.RUN[n] bit. The group becomes active when it receives a TTU_TRIG_IN[n] pulse. The group becomes inactive after it is stopped (at the same time as TTU_STOP.STOPREQ[n] clears to 0) or when it is disabled (by writing TTU_CTL.CNTREN[n] =0).

Restrictions on DLY in Periodic Mode

The TTU_DLY[m] delay register can be set to any value between –(TTU_PER[n] 1) and +(TTU_PER[n]). The trigger delay unit does not generate any trigger pulses when the TTU_DLY[m] register is set to a value outside this range.

TTU_DLY[m].VALUE ≤ 0 cannot be used to generate an output pulse coincident with or before the initial trigger TTU_TRIG_IN[n] that starts the counter operation, or a re trigger that changes the phase of the delay counter. (A trigger which arrives while the group is operating.)

TTU Programming Model

The basic programming model for the TTU involves enabling and stopping a trigger delay group. In some cases, a trigger delay group must be disabled abruptly. The following sections describe all of these cases.

Programming Concepts

When programming the TTU module, adhere to the following basic concepts.

Changing a Trigger Delay Group Period or Delay During Operation

While a trigger delay group is operating, the period of the counter and the delays of the individual generated triggers can be changed by writing new values to the registers. The `TTU_PER[n]` and `TTU_DLY[m]` registers are double-buffered, so the new values take effect on the next trigger-in, or (in periodic mode) on the next cycle where the counter `TTU_CNT[n]` =0.

Illegal Changes of Trigger Delay Group Mode

The TTU does not support changing timer groups between periodic and single-shot modes while the group is running. In other words, do not change the `TTU_PER[n]` register from a non-zero value to a zero value (or conversely) while the timer is enabled. Disable the timer first (by writing `TTU_CTL.CNTREN[n]` to 0) before changing the timer mode.

Illegal Changes to Group Assignments

After a trigger delay group is enabled, do not change the DGRP assignments for the group. Do not disable or reassign trigger delay units assigned to the group. Do not add new trigger delay units to the group. Instead, disable the group before changing the delay unit assignment registers.

TTU Programming Examples

The following sections provide the steps and code for the Programming Concepts described above.

Enabling a Trigger Delay Group

Use the following steps to enable a trigger delay group.

1. Set the period register (`TTU_PER[n]`) of the group counter. Non-zero values select a periodic operation. A zero value selects a single-shot operation.
2. Assign one or more trigger delay units to the group by writing `TTU_DGRP[m].VALUE` bit field =n for each delay unit m. Also, set the trigger delay unit enable bit `TTU_DGRP[m].TRIGEN` =1.
3. For each delay unit, assign the delay value in the `TTU_DLY[m].VALUE` bit field.
4. Configure the TRU to route the desired trigger source to the TTU input slave `TTU_TRIG_IN[n]`, and to route all the generated trigger output masters `TTU_TRIG_OUT[m]` to the desired trigger destinations.

5. Enable the delay group by setting TTU_CTL.CNTREN[n] bit field =1.

   The delay group now responds to input triggers.

   After the group is enabled, the TTU_DGRP[m] assignments for this group must not be changed. Trigger delay units assigned to the group must not be disabled or reassigned, and new trigger delay units must not be added to the group.

```
ADI_TTU_Typedef *pttu = pADI_TTU0;
// set up pointer to TTU

// Set up Trigger Delay Group 1 in periodic mode
pttu->PER[1] = 100;
// set TTU Counter 1 period to 100 sysclks

// Assign delay units 2 and 3 to the group, and enable them
pttu->DGRP[2] = ((1 << BITP_TTU_DGRP_VALUE)
//assign to group 1
| BITM_TTU_DGRP_TRIGEN );
pttu->DGRP[3] = ((1 << BITP_TTU_DGRP_VALUE)
//assign to group 1
| BITM_TTU_DGRP_TRIGEN );

// Assign the output trigger delays
pttu->DLY[2] = 15;
// output at t0 +15 sysclks
pttu->DLY[3] = 27;
// output at t0 +27 sysclks

// ... configure the TRU ...
ADI_TRU_CFG0_TypeDef *ptru1 = pADI_TRU1;
ptru1->GCTL = (BITM_TRU_GCTL_EN);
// enable trigger propagation
// route some trigger to TTU Trigger group 1;
ptru1->SSR[TRGS_TTU0_TRIG_IN1] = TRGM_<someSource>;
// route the TTU trigger outputs 2 and 3 to some destinations
ptru1->SSR[TRGS_<someDest1>] = TRGS_TTU_TRIG_OUT2;
ptru1->SSR[TRGS_<someDest2>] = TRGS_TTU_TRIG_OUT3;

// Enable the TTU trigger delay group 1
pttu->CTL |= ( 1<< BITP_TTU_CTL_CNTREN1 );
```

Gracefully Stopping a Periodic Trigger Delay Group

The following steps gracefully stop a trigger delay group configured for periodic operation.

1. Optionally, disable the source of the TTU_TRIG_IN[n] signal to the trigger delay group counter, to prevent accidentally restarting the counter after it is stopped. For example, this operation can be done by disabling the TRU routing to this trigger slave.
2. Request a stop to the delay group by setting the TTU_STOP.STOPREQ[n] bit =1.

   The delay group continues the until the current counter period completes, generating all scheduled output triggers (both leading and lagging), and then it stops.
3. If desired, poll for the completion of the stop request by testing the TTU_STOP.STOPREQ[n] bit. When this bit =0, the trigger group has stopped.

```
// Disable the TRU trigger source or routing to group 1.
ptru1->SSR[TRGS_TTU0_TRIG_IN1] = 0;
// Request to stop TTU trigger delay group 1
pttu->STOP = ( 1 << BITP_TTU_STOP_STOPREQ1 );
// poll for delay group becoming inactive, if desired
while( pttu->STOP & BITM_TTU_STOP_STOPREQ1 ) { }
```

Abruptly Disabling a Trigger Delay Group

Use the following step to abruptly disable a trigger delay group.

1. Disable the delay group by setting TTU_CTL.CNTREN[n] =0.

   The delay group immediately stops processing input and output triggers.

```
// Disable the TTU trigger delay group 1
pttu->CTL &= ~( 1 << BITP_TTU_CTL_CNTREN1 );
// or simply
pttu->CTL = 0;
```

Additional Programming Examples

The following examples...

- **Changing a Trigger Delay Group Period or Delay While Operating**

```
// enable group 1
pttu->CTL |= ( 1<< BITP_TTU_CTL_CNTREN1 );
// ...
// Change delay parameters
pttu->DLY[2] = 20;
pttu->PER[1] = 110;
```

- **Illegal Changes to Group Assignments**

```
//assign delay 3 to group 1
pttu->DGRP[3] = ((1 << BITP_TTU_DGRP_VALUE)
```

```
| BITM_TTU_DGRP_TRIGEN );
// enable group 1
pttu->CTL |= ( 1<< BITP_TTU_CTL_CNTREN1 );
// ERROR: do not change delay 3's assignment while its group is on
pttu->DGRP[3] = ((2 << BITP_TTU_DGRP_VALUE)
//assign to group 2?
| BITM_TTU_DGRP_TRIGEN );
// -- ERROR!
// ERROR: do not assign new delay 4 to a group that is already on
pttu->DGRP[4] = ((1 << BITP_TTU_DGRP_VALUE)
//assign to group 1? | BITM_TTU_DGRP_TRIGEN );
// -- ERROR!
```

CM41X_M4 TTU Register Descriptions

Trigger Timing Unit (TTU) contains the following registers.

Table 12-4: CM41X_M4 TTU Register List

Name	Description
TTU_CNT[n]	Counter Current Value
TTU_CTL	TTU Control Register
TTU_DGRP[m]	Trigger Output Counter Assignment
TTU_DLY[m]	Trigger Output Delay Register
TTU_PER[n]	Counter Period Register
TTU_REVID	Revision ID Register
TTU_STAT	Counter Status Register
TTU_STOP	Counter Stop Request Register
TTU_CHK	Counter Check Register

Counter Current Value

The `TTU_CNT[n]` register displays the current counter value.

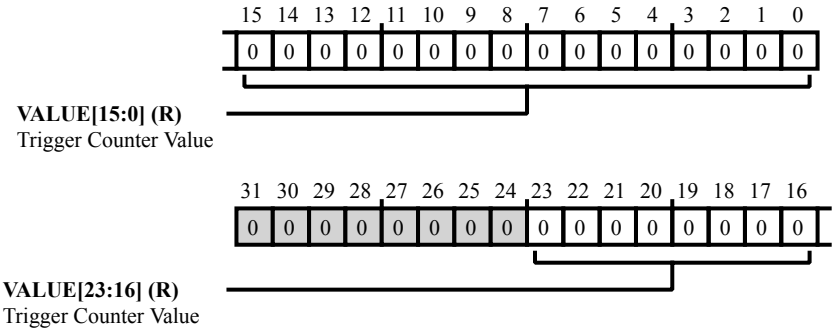


Figure 12-9: TTU_CNT[n] Register Diagram

Table 12-5: TTU_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/NW)	VALUE	Trigger Counter Value. The `TTU_CNT[n].VALUE` bit field displays the current value of the TTU counter n.

TTU Control Register

The TTU_CTL register contains global control bits for enabling the TTU features. This register contains individual enable and mode bits for each TTU counter.

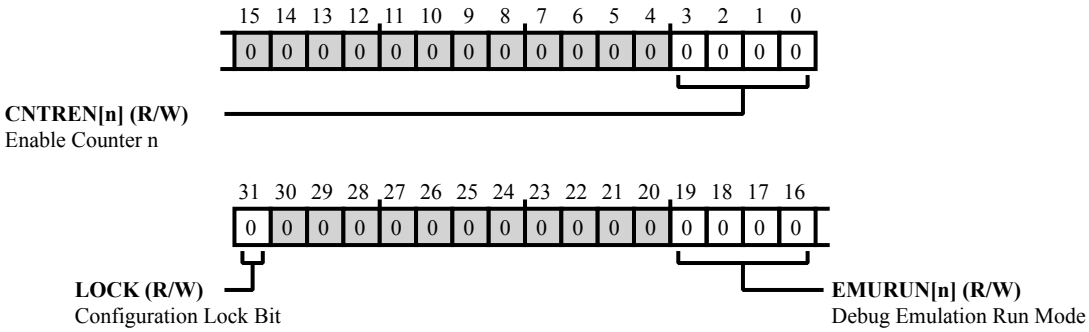


Figure 12-10: TTU_CTL Register Diagram

Table 12-6: TTU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Configuration Lock Bit. If the global lock (SPU_CTL.GLCK bit=1) is set, and the TTU_CTL.LOCK bit is set, the TTU registers are write-protected, except for the LOCK bit itself.	
		0	Unlock register
		1	Lock Register
19:16 (R/W)	EMURUN[n]	Debug Emulation Run Mode. The TTU_CTL.EMURUN[n] bit controls whether each counter runs during the debug emulation halt.	
		0	Stop counter during emulation
		1	Run counter during emulation
3:0 (R/W)	CNTREN[n]	Enable Counter n. Each read or write TTU_CTL.CNTREN[n] bit enables the one of the TTU counters. A write of 1 enables the group, a write of 0 disables the group.	
		0	Disable
		1	Enable

Trigger Output Counter Assignment

The TTU_DGRP[m] register contains bits that associate the trigger output with the counter n and enable trigger generation.

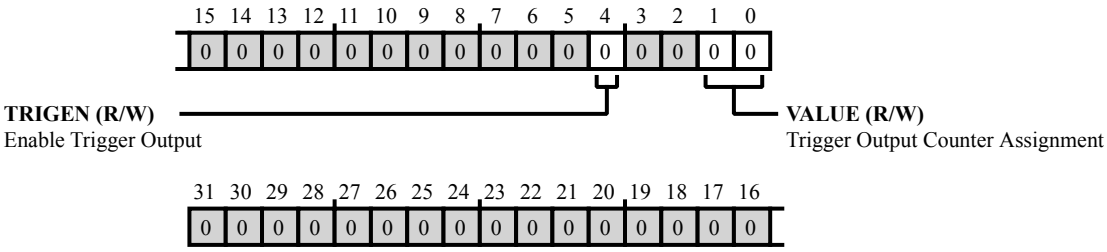


Figure 12-11: TTU_DGRP[m] Register Diagram

Table 12-7: TTU_DGRP[m] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	TRIGEN	Enable Trigger Output. The TTU_DGRP[m].TRIGEN bit, when set (=1), enables the generation of trigger m at the delay match. When the TTU_DGRP[m].TRIGEN bit =0, the output trigger m is deasserted.
1:0 (R/W)	VALUE	Trigger Output Counter Assignment. When TTU_DGRP[m] = TTU_DGRP[m].VALUE n, the TTU associates the trigger output m with the counter n (and its trigger input n).

Trigger Output Delay Register

The TTU_DLY[m] register specifies the output delay of output trigger m relative to the input trigger of the counter.

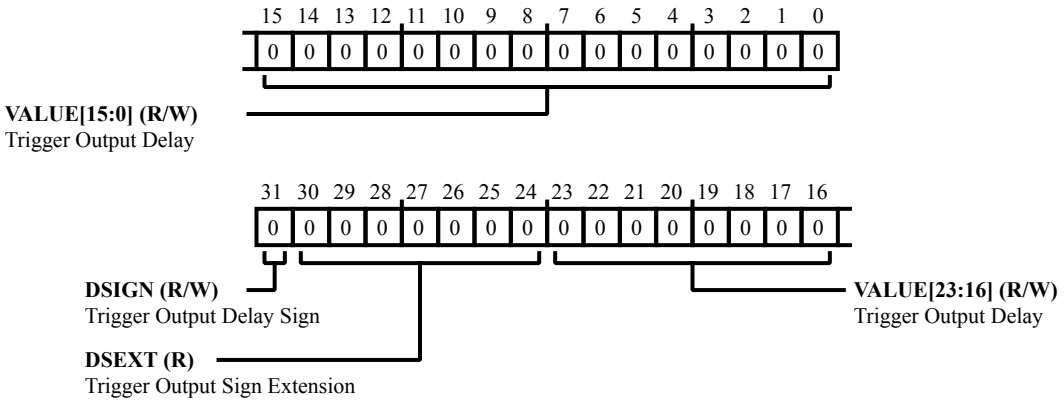


Figure 12-12: TTU_DLY[m] Register Diagram

Table 12-8: TTU_DLY[m] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	DSIGN	Trigger Output Delay Sign. The TTU_DLY[m].DSIGN bit indicates the sign of the trigger output delay m relative to the counter start point. If the counter n associated with trigger output m (by TTU_DGRP[m]) is configured for a single-shot operation (TTU_PER[n].VALUEn==0), then negative TTU_DLY[m] values are not allowed, and TTU_DLY[m].DSIGNm must be 0. (If TTU_DLY[m].DSIGNm=1, then the trigger output m does not produce trigger output pulses) If the counter n is periodic (TTU_PER[n].VALUEn !=0), then both positive and negative trigger delays (TTU_DLY[m].DSIGNm, DELAYm) are supported, and can take on any value in the range -(TTU_PER[n]-1) ... 0 ... +(TTU_PER[n]-1) inclusive. Negative trigger delays (-D) cause an output pulse to be generated when the value of the counter is (TTU_PER[n]-D); positive delays (+D) cause pulses to be generated when the value of the counter is +D.
30:24 (R/NW)	DSEXT	Trigger Output Sign Extension. The TTU_DLY[m].DSEXT bit field contains the read-only sign extension of the TTU_DLY[m].DSIGNm field. It extends to 32-bit width for convenient reading as a 32-bit signed integer. The bits in this field each are equal to TTU_DLY[m].DSIGNm.
23:0 (R/W)	VALUE	Trigger Output Delay. The TTU_DLY[m].VALUE bit specifies the trigger output delay for output trigger m relative to the start event of the corresponding counter.

Counter Period Register

The TTU_PER[n] register displays the counter period.

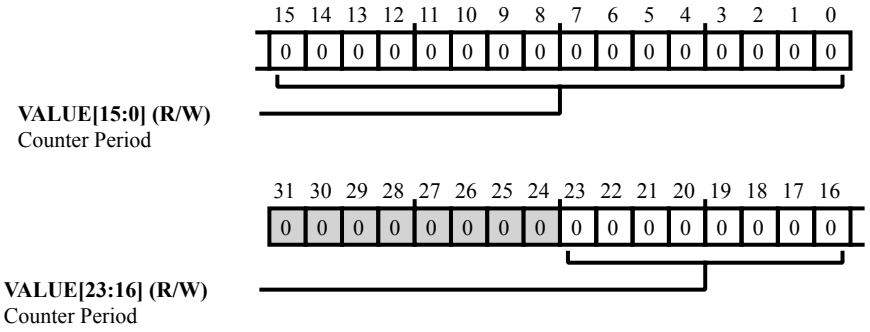


Figure 12-13: TTU_PER[n] Register Diagram

Table 12-9: TTU_PER[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Counter Period. The TTU_PER[n].VALUE bit indicates the period of the TTU counter n. If COUNT=0, then the counter functions as single-shot delay (the default). If COUNT !=0, then the counter operates in a periodic repeating sequence with the specified period in SYSCLKs.

Revision ID Register

The TTU_REVID register provides the TTU revision ID.

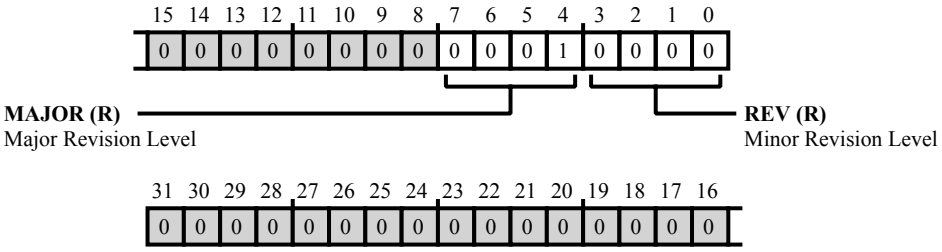


Figure 12-14: TTU_REVID Register Diagram

Table 12-10: TTU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Revision Level. The TTU_REVID.MAJOR bit indicates the TTU major ID.
3:0 (R/NW)	REV	Minor Revision Level. The TTU_REVID.REV bit indicates the TTU revision ID.

Counter Status Register

The TTU_STAT register indicates the status of the counter.

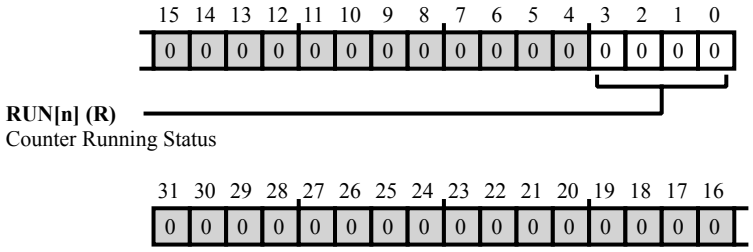


Figure 12-15: TTU_STAT Register Diagram

Table 12-11: TTU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/NW)	RUN[n]	Counter Running Status. Bit RUN[n] reads as 1 if the counter[n] is active.

Counter Stop Request Register

The TTU_STOP register indicates a pending stop request.

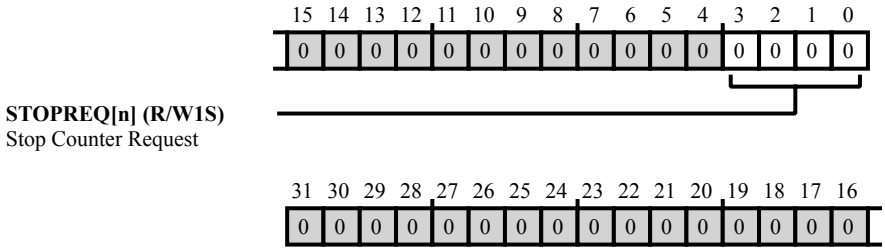


Figure 12-16: TTU_STOP Register Diagram

Table 12-12: TTU_STOP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W1S)	STOPREQ[n]	Stop Counter Request. For all TTU_STOP.STOPREQ[n] bits, write 1 to request the stopping of the periodic counter[n] after the completion of the current period. The register reads as 1 if a stop request is pending. Otherwise, the register reads as 0.

Counter Check Register

The TTU_CHK register contains a value used to speed up checking of long counts.

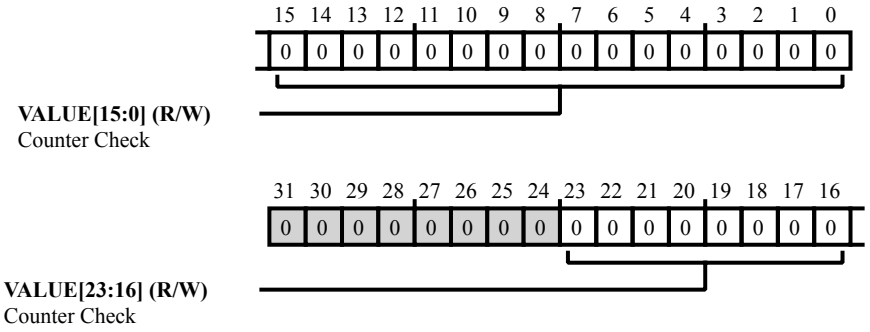


Figure 12-17: TTU_CHK Register Diagram

Table 12-13: TTU_CHK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Counter Check. In counter check mode, TTU_CHK.VALUE is used to speed up checking of long counts. On the cycle after this is written, the TTU_CHK.VALUE is substituted for the current count value of every active counter. However, if a trigger_in arrives, the trigger has priority.

13 Static Memory Controller (SMC)

The static memory controller is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. It provides a glueless interface to various external memories and peripheral devices, including:

- SRAM
- ROM
- EPROM
- FPGA/ASIC devices

The SMC acts as an SCB slave. The processor SCB interconnect fabric arbitrates accesses to the SMC. On the chip boundary, the SMC connects to an address bus, a data bus, and signal pins for memory control (such as read, write, output enable, and memory select lines).

SMC Features

SMC features include:

- 16-bit I/O width
- Provides flexible timing control through extended timing parameters

SMC Definitions

The timing registers contain bits to program the setup time, hold time, and access time for read and write access to each bank separately. The SMC allows for different setup, hold, or access times for reads and writes. The SMC_B0TIM – SMC_B3TIM registers control the timing characteristics of the asynchronous memory interface using the following parameter definitions. Each of these parameters can be programmed in terms of SCLK clock cycles.

Read setup time

The time between the beginning of a memory cycle ($\overline{\text{SMC\_AMS0}}$ signal low) and the read-enable assertion ($\overline{\text{SMC\_ARE}}$ signal low).

Read hold time

The time between read-enable deassertion ($\overline{\text{SMC\_ARE}}$ signal high) and the end of the memory cycle ($\overline{\text{SMC\_AMS0}}$ signal high).

Read access

The time between read-enable assertion ($\overline{\text{SMC\_ARE}}$ signal low) and deassertion ($\overline{\text{SMC\_ARE}}$ signal high).

Write setup time

The time between the beginning of a memory cycle ($\overline{\text{SMC\_AMS0}}$ signal low) and the write-enable assertion ($\overline{\text{SMC\_AWE}}$ signal low).

Write hold time

The time between write-enable deassertion ($\overline{\text{SMC\_AWE}}$ signal high) and the end of the memory cycle ($\overline{\text{SMC\_AMS0}}$ signal high).

Write access

The time between write-enable assertion ($\overline{\text{SMC\_AWE}}$ signal low) and deassertion ($\overline{\text{SMC\_AWE}}$ signal high).

The SMC provides another register for defining more timing characteristics of control signals by programming the extended SMC_B0TIM – SMC_B3TIM timing registers. These registers contain bits to program following timing characteristics.

Pre-setup time

The number of cycles the $\overline{\text{SMC\_AMS0}}$ signal is asserted before the $\overline{\text{SMC\_AOE}}$ signal is asserted.

Pre-access time

The number of cycles inserted after the $\overline{\text{SMC\_AOE}}$ signal is deasserted and before the $\overline{\text{SMC\_ARE}}$ signal is asserted for the next access.

Memory idle time

The number of bus idle cycles between the $\overline{\text{SMC\_AMS0}}$ deasserting edge and next asserting edge.

Memory transition time

The number of bus idle cycles extending the idle time cycles. These idle cycles occur in the case where a subsequent access has a different data direction or the access is to a different bank.

Bus contention

State of the bus in which more than one device on the bus attempts to place values on the bus at the same time. For more information, see Avoiding Bus Contention.

ARDY signal

The SMC uses the SMC_ARDY signal to insert wait states for slower asynchronous memories. There is no upper limit to how many wait states the SMC_ARDY signal can enter. As long as it is held, the processor waits for the access to the asynchronous memory. Once asserted, the processor accesses the memory according to the timing diagrams. For more information, see ARDY Input Control.

SMC Functional Description

The SMC contains memory-mapped registers that control the access characteristics for each asynchronous memory bank. Different banks can be programmed in various modes and independently-controlled using the functional and cycle time bit settings for each bank.

Independent bank control

The SMC provides separate sets of registers, SMC_B0CTL through SMC_B3CTL (control), SMC_B0TIM through SMC_B3TIM (timing) and SMC_B0ETIM through SMC_B3ETIM (extended timing) to control the mode and timing characteristic of each bank independently. The control registers contain bits for enabling the bank and bits for selecting mode of operation.

Bank select control signal control

The control registers also contain bits to control the type of bank select control signal. External FIFO devices often do not have a separate chip select pin. As a result, for a read, the FIFOs output enable ($\overline{\text{SMC\_AOE}}$) pin must be connected to the OR of the $\overline{\text{SMC\_AMS0}}$ and the $\overline{\text{SMC\_ARE}}$. Similarly, the write case requires an OR between $\overline{\text{SMC\_AMS0}}$ and $\overline{\text{SMC\_AWE}}$. The SMC provides this function so that an external OR gate is not required. The appropriate AMS function can be selected for each memory bank region using the SMC_B0CTL.SELCTRL bits.

CM41X_M4 SMC Register List

The Static Memory Controller SMC is a protocol converter and data transfer interface between the internal processor bus and the external L3 memory. The SMC acts as a bus slave, and accesses to SMC are arbitrated by the module's system crossbar. On the chip boundary, the SMC is connected to an external memory address bus, a 16-bit data bus and memory control signal pins (read, write, select). This memory interface can support a sizable external memory connected to one or more banks, each bank being controlled by a chip select signal. A set of registers governs SMC operations. For more information on SMC functionality, see the SMC register descriptions. For the memory map, see the product data sheet.

Table 13-1: CM41X_M4 SMC Register List

Name	Description
SMC_B0CTL	Bank 0 Control Register
SMC_B0ETIM	Bank 0 Extended Timing Register
SMC_B0TIM	Bank 0 Timing Register
SMC_B1CTL	Bank 1 Control Register
SMC_B1ETIM	Bank 1 Extended Timing Register
SMC_B1TIM	Bank 1 Timing Register
SMC_B2CTL	Bank 2 Control Register
SMC_B2ETIM	Bank 2 Extended Timing Register
SMC_B2TIM	Bank 2 Timing Register
SMC_B3CTL	Bank 3 Control Register
SMC_B3ETIM	Bank 3 Extended Timing Register
SMC_B3TIM	Bank 3 Timing Register

SMC Architectural Concepts

The SMC can support connection to multiple different external banks, with each bank controlled by the SMC_AMS[n] chip select signal. Check the processor data sheet for details on the bank address ranges and configurations.

NOTE: The processor data sheet shows the address range allocated to each bank. It is not necessary to populate all of an enabled memory bank.

The processor does not directly support 8-bit accesses to the external memories. So, the SMC address lines start from SMC_A01; there is no SMC_A0 pin.

The SMC does indirectly support 8-bit accesses through the additional byte enable signals SMC_ABE0 and SMC_ABE1. Some 16-bit memory systems allow the processor to perform 8-bit reads and writes, which are selected through the SMC_ABE0 and SMC_ABE1 signals.

The byte enable pins are both three-stated during all asynchronous reads and are driven low during 16-bit asynchronous writes. When an asynchronous write is made to the upper byte of a 16-bit memory, SMC_ABE1 =0 and SMC_ABE0 =1. When an asynchronous write is made to the lower byte of a 16-bit memory, SMC_ABE1 =1 and SMC_ABE0 =0.

NOTE: Some SRAM devices expect byte-enable signals to be driven low during read accesses rather than being three-stated, which can be achieved using external pull-down resistors. For applications requiring alternate functions on the byte-enable pins during run time where pull-down resistors are not an option, the same functionality can be achieved using the external logic shown as follows:

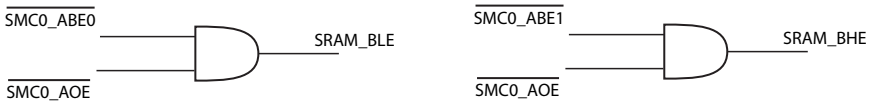


Figure 13-1: External Logic

Avoiding Bus Contention

Bus contention occurs when one device is getting off the bus and another is getting on. If the first device is slow to three-state and the second device is quick to drive, the devices contend. Bus contention causes excessive power dissipation and can lead to device failure.

There are two cases where contention can occur.

- When a read followed by a write to the same memory space occurs, there is a potential for bus contention. The data bus drivers used for the write can potentially contend with the drivers used by the memory device being read.
- When there are back-to-back reads from two different memory spaces, the two memory devices addressed by the two reads can contend at the transition between the two read operations.

To avoid contention, program the turnaround time appropriately in the extended time registers (SMC_B0ETIM – SMC_B3ETIM). The programming is done by setting the number of clock cycles between these types of accesses on a bank-by-bank basis.

The idle time bit (SMC_B0ETIM.IT) applies to similar back-to-back access types on the same bank. The transition time bit (SMC_B0ETIM.TT) applies to the SMC_B0ETIM.IT bit. For actual turnaround situations, idle time and transition time function in an accumulated fashion. The sequence of access types and times are:

- A write followed by write to same bank – SMC_B0ETIM.IT
- A read followed by read to same bank – SMC_B0ETIM.IT
- A write followed by read to same bank – SMC_B0ETIM.IT + SMC_B0ETIM.TT
- A read followed by write to same bank – SMC_B0ETIM.IT + SMC_B0ETIM.TT
- Any access to a given bank followed by any access to a different bank – SMC_B0ETIM.IT + SMC_B0ETIM.TT

The reset value of turnaround transition time is two cycles. Program the SMC_B0ETIM.TT bit to a value either greater than or equal to two cycles, depending on memory AC-timing specifications. It is important to be aware that the SMC_B0ETIM.TT bit is programmed to 0 *only* when:

- There are *either* only read accesses *or* only write accesses possible to external memory devices for the current device configuration or processor operation situation.

ARDY Input Control

Each bank can be programmed to sample the SMC_ARDY input after the read or write access timer has counted down. It can also be programmed to ignore this input signal. If enabled and disabled at the sample window, the SMC module uses SMC_ARDY to extend the access time, as required.

The processor treats SMC_ARDY as an asynchronous input. The input must reach the desired value (either asserted or deasserted) more than two SCLK cycles before the completion of access time (scheduled rising edge of SMC_AWE or SMC_ARE). This timing determines whether the SMC extends the access with the assertion of SMC_ARDY. After the processor samples SMC_ARDY high, the total delay between SMC_ARDY going high at the pads and SMC_ARE being deasserted at the pads is a maximum of five SCLK cycles. (The memory device asserts SMC_ARDY.)

Asynchronous SRAM writes are also possible with the SMC_ARDY signal enabled. In asynchronous SRAM writes, the write access is extended beyond the programmed write access cycles depending on the SMC_ARDY signal state. Once SMC_ARDY is sampled asserted, the SMC_AWE signal is deasserted after two *CLKOUT* cycles and the write access ends.

The polarity of SMC_ARDY is programmable on a per-bank basis. Since SMC_ARDY is not sampled until an access is in-progress to a bank in which the SMC_ARDY enable is asserted, it is not driven by default. When using flash memory, connect the WAIT input to SMC_ARDY.

To avoid stalls in the case of erroneous SMC_ARDY behavior, set the SMC_B0CTL.RDYABTEN bit to enable the SMC_ARDY abort counter. When the abort counter is enabled, it starts counting down as soon as the programmed read/write access cycles expire. If the SMC interface does not sample the SMC_ARDY signal as asserted within 64 cycles, a counter-timeout occurs. This timeout ensures that the processor does not hang if SMC_ARDY is not sampled correctly.

SMC Operating Modes

The SMC supports the following operating modes:

- Asynchronous Flash Mode
- Asynchronous Page Mode

Asynchronous Flash Mode

When the access selected mode is asynchronous flash (SMC_B0CTL.MODE =01), external bank accesses operate the same as in standard asynchronous mode, except for the pin configuration. Use this mode when accessing burst devices in non-read array modes.

Asynchronous Page Mode

When asynchronous page mode access is selected (SMC_B0CTL.MODE =10), asynchronous page reads are enabled. The SMC module supports page sizes of 4, 8 and 16 words. When performing a page mode read, the first access in the page proceeds according to the read access time configured in SMC_B0TIM register. This access opens the page

and the subsequent reads in that page have a period equal to the page wait states programmed in the SMC_B0ETIM register. Besides the start of the setup phase, the read address is incremented at the start of every page cycle.

The SMC module supports page mode access only for back-to-back access DMA reads. It treats write accesses in asynchronous page mode as simple asynchronous flash write accesses.

SMC Event Control

SMC event control consists of recording the status of SMC errors. Accesses to reserved locations and writes to read-only registers result in bus errors. The SMC translates bus errors into internal SCB crossbar errors which get translated into interrupts. To report errors occurring in the slave memory devices (for both this memory interface and the MMR interface), the core combines the SCB crossbar response signals. This combination generates a combined error signal indication which is routed to the fault management unit.

SMC Programmable Timing Characteristics

This section describes the programmable timing characteristics for the SMC. Timing relationships depend on the programming of the SMC bank registers, whether initiation is from the core or from DMA. The relationships also depend on the sequence of transactions (read followed by read, read followed by write, and others).

NOTE: All memory control, address, and data signals are driven out of the chip based on the falling edge of the *CLKOUT* signal. The *CLKOUT* signal is SCLK on the chip pins (pad delayed).

Asynchronous SRAM Reads and Writes

The ***Basic Asynchronous SRAM Write Followed by Read*** figure shows a basic single write and read operation to an external device with the SMC programmed in asynchronous SRAM mode.

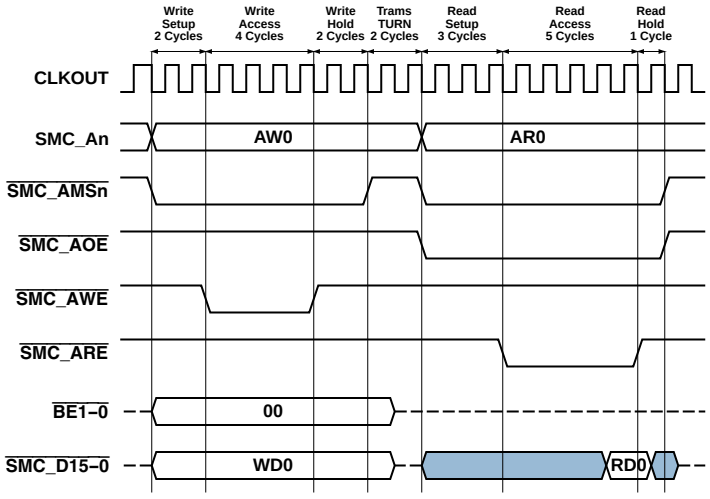


Figure 13-2: Basic Asynchronous SRAM Write Followed by Read

For the current bank, the programmed time cycles are:

- Write setup time = 2 cycles
- Write access time = 4 cycles
- Write hold time is = 2 cycles
- Read setup time = 3 cycles
- Read access time = 5 cycles
- Read hold time = 1 cycle
- Turnaround transition time = 2 cycles
- Idle transition time = 0 cycles

The asynchronous SRAM bus cycles proceed as follows.

1. At the start of the write setup period, the chip select signal ($\overline{\texttt{SMC\_AMS[n]}}$) for the target bank is asserted. The write data (WD0), address (AW0), and byte enables become valid.
2. At the end of the setup phase and at the start of the write access period, the write enable ($\overline{\texttt{SMC\_AWE}}$) is asserted.
3. At the end of the programmed write access, the $\overline{\texttt{SMC\_AWE}}$ signal is deasserted. The target device is assumed to have captured the write data before $\overline{\texttt{SMC\_AWE}}$ is deasserted.
4. At the end of the write hold period, the $\overline{\texttt{SMC\_AWE}}$ signal is deasserted because the pending access is a read access, and the turnaround transition time cycles start. The write data and byte enables become invalid within 1 cycle of the $\overline{\texttt{SMC\_AMS0}}$ signal deasserting.
5. At the end of turnaround transition time, the read setup period starts with the assertion of the $\overline{\texttt{SMC\_AMS0}}$ and $\overline{\texttt{SMC\_AOE}}$ signals and a new read address (AR0) is presented on the address bus.
6. At the start of the read access period, the read enable signal, $\overline{\texttt{SMC\_ARE}}$ is asserted.
7. At the end of the read access period, the $\overline{\texttt{SMC\_ARE}}$ signal is deasserted and the read hold period starts. Read data is latched along with $\overline{\texttt{SMC\_ARE}}$ deasserting.
8. At the end of the read hold period, the SMC pulls the $\overline{\texttt{SMC\_AMS[n]}}$ signal high and appends turnaround transition cycles unless there is a pending read request to the same bank.

Asynchronous SRAM Reads with IDLE Transition Cycles Inserted

The ***Asynchronous SRAM Read with IDLE Transition*** figure shows two consecutive asynchronous SRAM modes reads to the same bank separated by programmed IDLE transition time cycles.

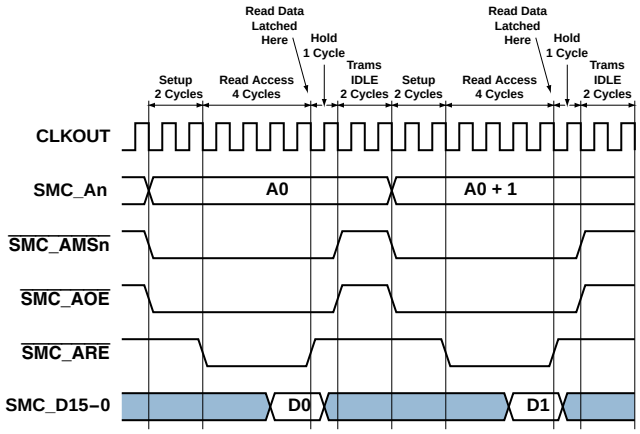


Figure 13-3: Asynchronous SRAM Read with IDLE Transition

Programmed cycle times are:

- SMC_B0TIM.RST = 2 cycles
- SMC_B0TIM.RAT = 4 cycles
- SMC_B0TIM.RHT = 1 cycle
- IDLE transition time = 2 cycles

At the start of the IDLE transition cycle, the SMC deasserts the $\overline{\text{SMC\_AMS[n]}}$ and $\overline{\text{SMC\_AOE}}$ signals. The setup period of the second read starts at the end of the IDLE transition cycle with the assertion of the $\overline{\text{SMC\_AMS[n]}}$ and $\overline{\text{SMC\_AOE}}$ signals and a new address on the address bus.

High-Speed Asynchronous SRAM Read Burst

The ***Fast Asynchronous SRAM Reads, Burst of Four Word*** figure shows a high-speed asynchronous SRAM read bus cycle. This read bus cycle is typical for SRAM devices with small access times connecting through SCB read bursts, especially for boot purposes.

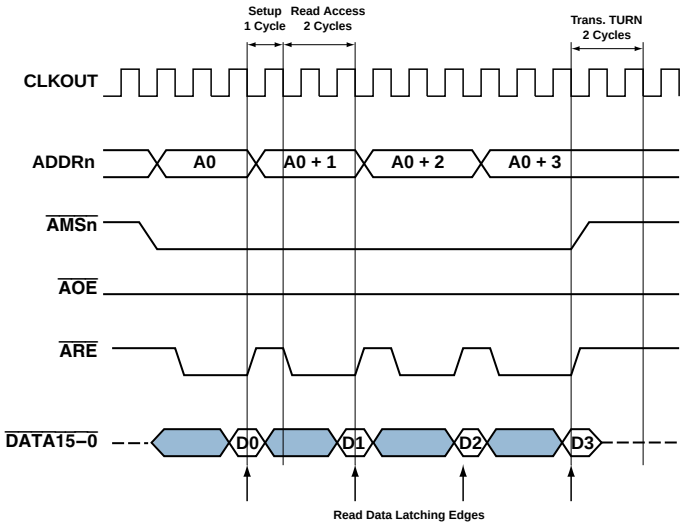


Figure 13-4: Fast Asynchronous SRAM Reads, Burst of Four Word

In this case, the target SMC bank has been programmed with:

- read setup time = 1 cycle
- read access time = 2 cycles
- read hold time = 0
- SMC_B0ETIM.PREAT = 0
- SMC_B0ETIM.PREST = 0
- IDLE transition time = 0

The SMC_AMS[n] signal is asserted at the start of the setup cycle of the first read out of the burst. Since the hold time and the idle transition time have been programmed to 0, the SMC_AMS[n] signal does not deassert until the entire set of reads concludes. Only the SMC_ARE signal deasserts periodically for 1 cycle for the setup period. The read address changes to the next address at the start of each individual setup cycle. Read data words are latched at the end of each individual read access period.

High-Speed Asynchronous SRAM Writes

High-speed asynchronous SRAM writes are similar to the high-speed read accesses. The ***Fast Asynchronous SRAM Writes*** figure shows the bus protocol for a write burst of 4 words. Here, the write setup time is 1 cycle and the write access time has been programmed to 2 cycles. Write hold time, pre-access time, pre-setup time, and idle transition time are programmed to 0.

The chip select signal SMC_AMS[n] asserts at the start of the entire write burst and deasserts only at the end of the last individual write access period. Write address, byte enables and write data for each individual write access are presented onto the bus at the start of each individual write setup cycle. The SMC_AWE signal asserts for the write access period and deasserts during the setup period for each individual data write.

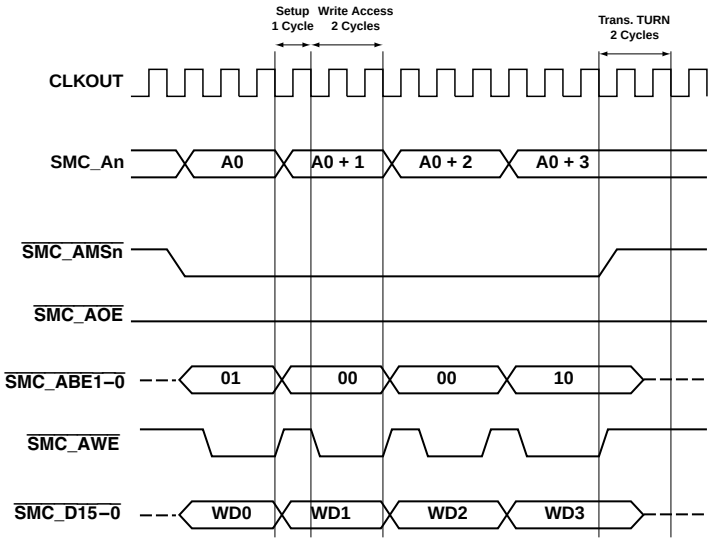


Figure 13-5: Fast Asynchronous SRAM Writes

Asynchronous SRAM Reads with ARDY

The ***Asynchronous SRAM Read with ARDY*** figure shows an extended asynchronous SRAM read bus cycle with `SMC_ARDY` enabled.

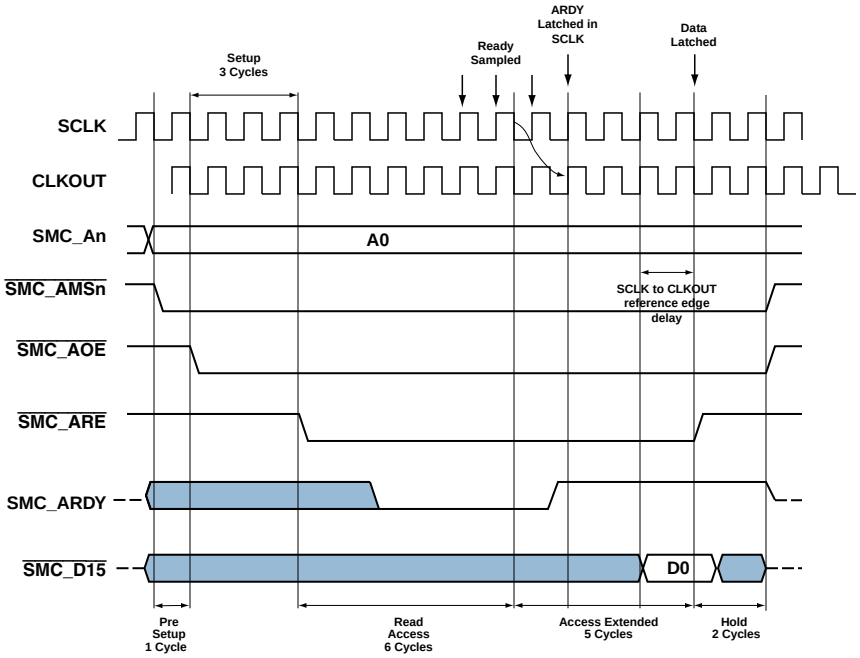


Figure 13-6: Asynchronous SRAM Read with ARDY

NOTE: SCLK in the ***Asynchronous SRAM Read with ARDY*** figure is SCLK.

The programmed SMC bank control parameters are:

- Pre-setup time = 1 cycle
- Read setup time = 3 cycles
- Read access time = 6 cycles
- Read hold time = 2 cycles
- SMC_B0CTL.RDYPOL =1 (memory is ready when SMC_ARDY =1)

The bus cycles proceed as follows:

- At the start of the pre-setup phase, $\overline{\text{SMC\_AMS[n]}}$ asserts, and read address SMC_A01 is presented on the address bus.
- At the start of the setup period, $\overline{\text{SMC\_AOE}}$ is asserted.
- At the start of the read access, $\overline{\text{SMC\_ARE}}$ is asserted.
- The CLKOUT signal is SCLK which is driven out of the pads. The CLKOUT signal falling edge can be delayed from the internal SCLK falling edge. See the data sheet for the specification related to this delay. All output signals out of the pads, for example $\overline{\text{SMC\_ARE}}$ and $\overline{\text{SMC\_AOE}}$, are driven with regard to the falling edge of CLKOUT.
- The SMC starts sampling the SMC_ARDY signal on every rising edge of internal SCLK two cycles before the programmed number of read access cycles expires. The read access is extended ($\overline{\text{SMC\_ARE}}$ is kept asserted) until the SMC samples SMC_ARDY high.
- Once the SMC samples the SMC_ARDY signal (asserted by memory device) high in SCLK, the read signal is pulled off internally in the SCLK domain. The total delay between the SMC_ARDY signal going high at the pads and the deassertion of the $\overline{\text{SMC\_ARE}}$ signal at the pads can be a maximum of five SCLK cycles.
- Read data is latched at the falling edge of CLKOUT on the same edge where the SMC deasserts $\overline{\text{SMC\_ARE}}$.
- Hold bus cycles start after the SMC deasserts the $\overline{\text{SMC\_ARE}}$ signal.
- At the end of the hold period, the $\overline{\text{SMC\_AMS[n]}}$ and $\overline{\text{SMC\_AOE}}$ signals deassert and the SMC goes into the transition state.

Asynchronous Flash Reads

The ***Asynchronous Flash Read with Pre-Setup and Pre-Access Cycles*** figure illustrates a single asynchronous flash mode read bus cycle.

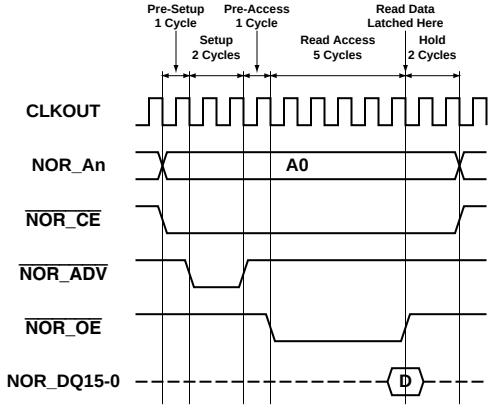


Figure 13-7: Asynchronous Flash Read with Pre-Setup and Pre-Access Cycles

In this case, the target SMC bank has been programmed with:

- Pre-setup time = 1 cycle
- Read setup time = 2 cycles
- Pre-access time = 1 cycle
- Read access time = 5 cycles
- Read hold time = 2 cycles

The read bus cycle is almost identical to the asynchronous SRAM read bus cycle. The only difference is the behavior of the SMC_AOE signal which the SMC uses as the flash address valid SMC_NORDV signal. The flash address valid SMC_NORDV signal asserts at the start of the setup cycle and deasserts at the end of the setup cycle.

The pre-access cycle inserts a one-cycle gap between the deassertion of the flash address valid SMC_NORDV signal and the assertion of the flash read strobe NOR_OE at the start of read access. The SMC also uses asynchronous flash reads with SMC_ARDY enabled for flash devices which use SMC_NORWT in asynchronous mode. In this case, the read bus cycle operation is identical to the asynchronous SRAM with SMC_ARDY enabled except for the SMC_AOE/SMC_NORDV signal behavior.

The ***32-bit Asynchronous Flash Read*** figure shows a 32-bit read access to a flash device in asynchronous mode which is split into two 16-bit external memory accesses. For this bank, read setup and read hold are programmed as two cycles whereas the read access time is five cycles. The flash device chip select signal (NOR_CE) remains asserted for the entire duration of both read accesses. NOR_CE is deasserted at the end of the hold period of the second read access. The SMC asserts the SMC_NORDV signal during the setup phase of both read accesses. Read data is latched at the end of the read access period.

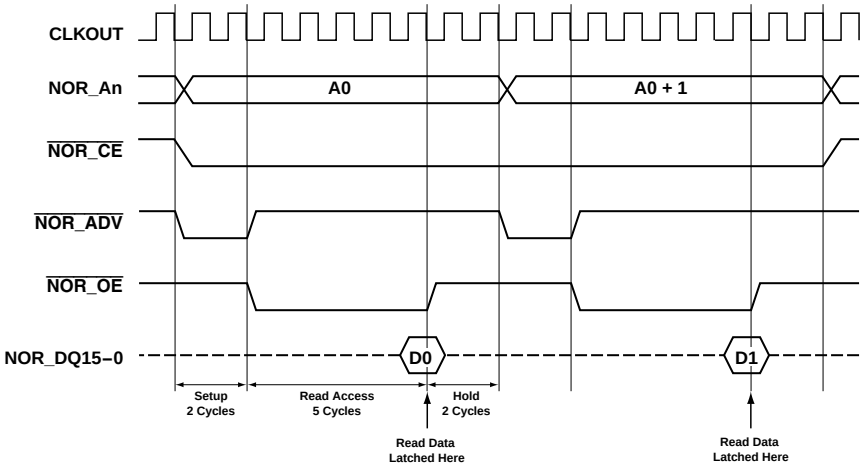


Figure 13-8: 32-bit Asynchronous Flash Read

Asynchronous Flash Writes

The ***Asynchronous Flash Write Operation*** figure shows a single asynchronous flash write bus cycle.

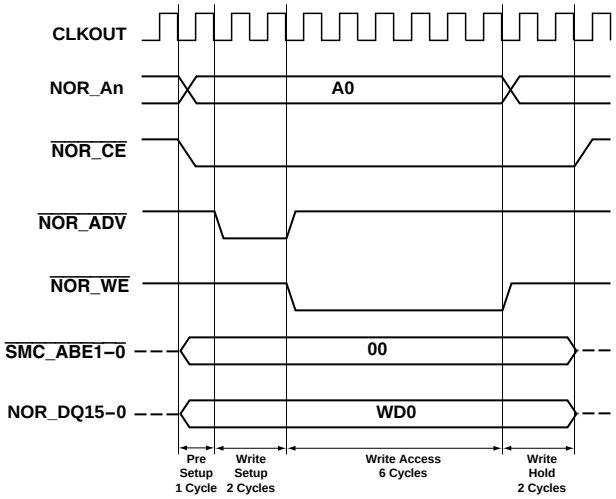


Figure 13-9: Asynchronous Flash Write Operation

For this example, the SMC has been programmed with:

- Pre-setup time = 1 cycle
- Write setup time = 2 cycles
- Write access time = 6 cycles
- Write hold time = 2 cycles
- Pre-access time = 0

The asynchronous flash write bus cycle is again almost identical to the asynchronous SRAM write. The SMC_AWE pin is connected to flash write enable signal (NOR_WE). However, in asynchronous flash writes the SMC uses the

SMC_AOE signal as the address valid signal (SMC_NORDV). The SMC_AOE signal asserts during the setup period, unlike in asynchronous SRAM writes where the SMC_AOE signal never asserts.

Asynchronous Flash Page Mode Reads

The ***Asynchronous Page Mode Read Bus Cycle*** figure shows an asynchronous page mode bus read cycle for a burst of five reads. The reads are split into four reads followed by a single read.

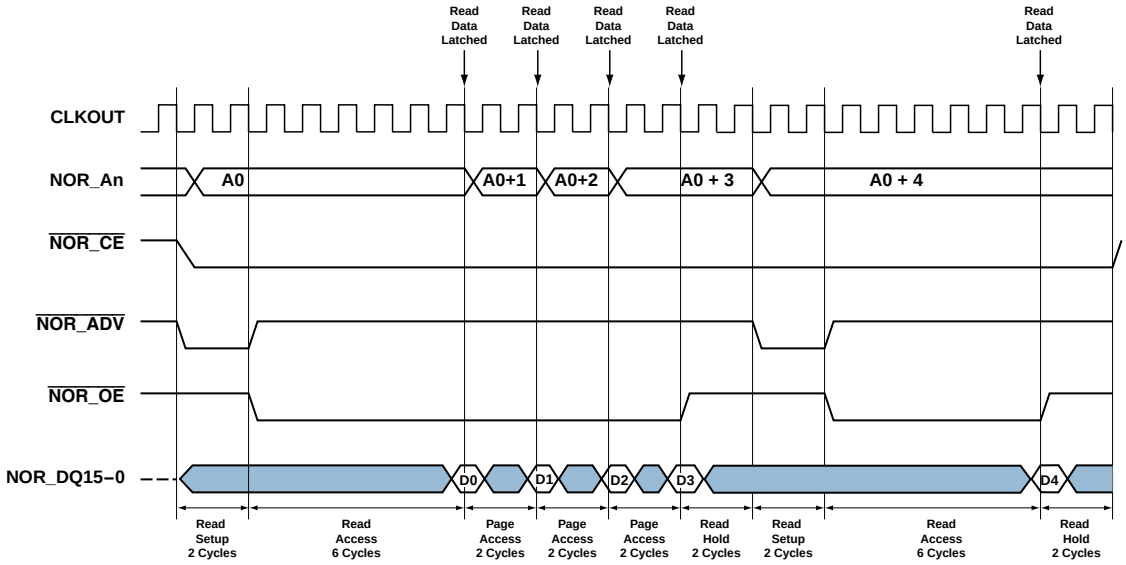


Figure 13-10: Asynchronous Page Mode Read Bus Cycle

The programmed bank parameters are:

- Read setup time = 2 cycles
- Read access time = 6 cycles
- Page wait = 2 cycles
- Hold time =2 cycles

The maximum number of read bursts in a total page access depends on the bank SMC_B0CTL.PGSZ bits (00 =4 words, 01 =8 words, 1x =16 words). The first read access is extended for three more page-read cycles whose period is equal to the page wait states. Besides the start of the setup phase, the read address is incremented at the start of every page cycle. Read data is latched with the falling edge of CLKOUT the end of the read access period, and also at the end of the page cycles.

Asynchronous FIFO Reads and Writes

The ***Asynchronous FIFO Read Bus Cycles*** figure shows the read bus cycles for an asynchronous FIFO device. The SMC bank is programmed in asynchronous SRAM mode, with SMC_B0CTL.SELCTRL =01 (SMC_AMS[n] is OR'ed with SMC_ARE).

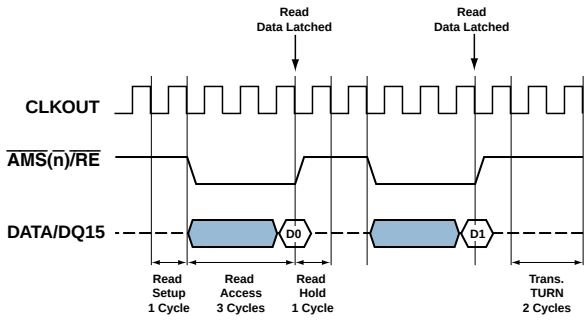


Figure 13-11: Asynchronous FIFO Read Bus Cycles

Other settings are:

- Read setup time = 1 cycle
- Read access time = 3 cycles
- Read hold time = 1 cycle
- Idle transition time = 0 cycles
- Turnaround transition time = 2 cycles

The SMC_AMS[n] signal connects to the read enable (RE) of the FIFO device, and the data bus connects to the output data bus (DQ) of the FIFO. The SMC_AMS[n] signal or the FIFO read strobe assert only during the read access. Read data is latched at the falling edge of CLKOUT at the end of the read access, when the SMC deasserts SMC_AMS[n].

The ***Asynchronous FIFO Write Bus Cycles*** figure illustrates the write bus cycles for an asynchronous FIFO device. The SMC bank is programmed in asynchronous SRAM mode, with SMC_B0CTL.SELCTRL =10 (SMC_AMS[n] is OR'ed with SMC_AWE). Other settings are:

- Write setup time = 1 cycle
- Write access time = 3 cycles
- Write hold time = 1 cycle
- Idle transition time = 0
- Turnaround transition time = 2 cycles

The SMC_AMS[n] signal connects to the write enable (WE) of the FIFO device. The data bus connects to the input data bus (DIN) of the FIFO. The SMC_AMS[n] signal or the FIFO write strobe asserts only during the write access. However, the SMC asserts write data at the start of the setup cycle and removes it at the end of the hold period for each individual write access.

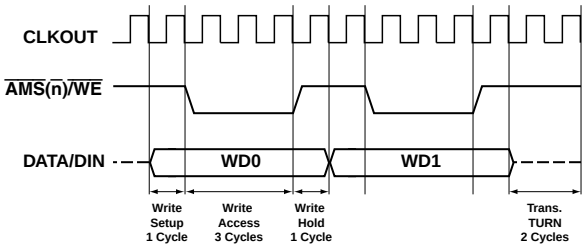


Figure 13-12: Asynchronous FIFO Write Bus Cycles

SMC Programming Model

The following general guidelines are used for configuring and enabling the SMC interface. Failure to follow these guidelines can lead to erroneous behavior.

- In asynchronous page mode, always program SMC_B0CTL.RDYEN to 0.
- Enable the ARDY abort counter (the SMC_B0CTL.RDYABTEN bit =1) whenever the SMC_ARDY signal is enabled (the SMC_B0CTL.RDYEN is set to 1). This step ensures that the interface does not hang due to erroneous SMC_ARDY signal behavior or erroneous sampling of the SMC_ARDY signal.
- Do not program read access time (SMC_B0TIM.RAT), write access time (SMC_B0TIM.WAT), read setup time (SMC_B0TIM.RST), and write setup time (SMC_B0TIM.WST) to 0.
- Never program page mode wait-states (SMC_B0ETIM.PGWS) to 0 or 1.
- Program the page size bits (SMC_B0CTL.PGSZ) to match the configurations of the flash device connected to the SMC interface.
- Select the SMC_B0CTL.RDYPOL bit to be the complement of the WAIT polarity that is configured in the flash device.
- In asynchronous SRAM and asynchronous flash modes with SMC_ARDY enabled, and where SMC_B0TIM.RHT, SMC_B0TIM.WHT, SMC_B0TIM.RAT, and SMC_B0TIM.WAT are the read and write hold and access times and SMC_B0ETIM.IT and SMC_B0ETIM.TT are the idle and transition times, ensure that the following conditions are true:
  - SMC_B0TIM.RHT + SMC_B0ETIM.IT + SMC_B0ETIM.TT≥ 2
  - SMC_B0TIM.WHT + SMC_B0ETIM.IT + SMC_B0ETIM.TT≥ 2
  - SMC_B0TIM.RAT≥ 5
  - SMC_B0TIM.WAT≥ 5

CM41X_M4 SMC Register Descriptions

Static Memory Controller (SMC) contains the following registers.

Table 13-2: CM41X_M4 SMC Register List

Name	Description
SMC_B0CTL	Bank 0 Control Register
SMC_B0ETIM	Bank 0 Extended Timing Register
SMC_B0TIM	Bank 0 Timing Register
SMC_B1CTL	Bank 1 Control Register
SMC_B1ETIM	Bank 1 Extended Timing Register
SMC_B1TIM	Bank 1 Timing Register
SMC_B2CTL	Bank 2 Control Register
SMC_B2ETIM	Bank 2 Extended Timing Register
SMC_B2TIM	Bank 2 Timing Register
SMC_B3CTL	Bank 3 Control Register
SMC_B3ETIM	Bank 3 Extended Timing Register
SMC_B3TIM	Bank 3 Timing Register

Bank 0 Control Register

The SMC_B0CTL register enables bank 0 accesses and configures the memory access features for this bank.

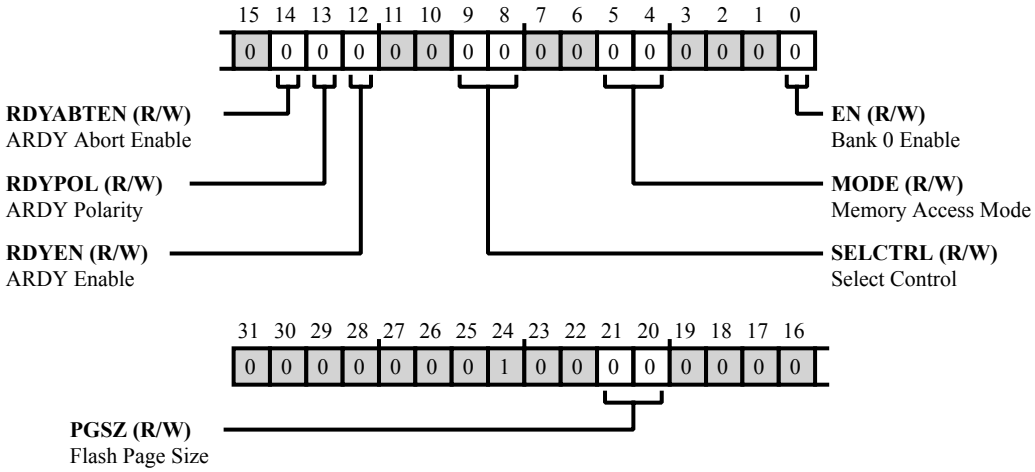


Figure 13-13: SMC_B0CTL Register Diagram

Table 13-3: SMC_B0CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B0CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B0CTL.MODE > 1). Note that the SMC_B0CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B0CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B0CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 13-3: SMC_B0CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B0CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B0CTL.RDYEN =1). After SMC_B0TIM.RAT or SMC_B0TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B0CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B0CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B0CTL.RDYEN bit enables SMC_ARDY pin operation for bank 0 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B0CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMS[n]}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS0 only
		1	AMS0 ORed with ARE
		2	AMS0 ORed with AOE
		3	AMS0 ORed with AWE

Table 13-3: SMC_B0CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B0CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 0 Enable. The SMC_B0CTL.EN bit enables accesses to the memory in bank 0. When this bit is disabled, accesses to bank 0 return an error response.	
		0	Disable access
		1	Enable access

Bank 0 Extended Timing Register

The SMC_B0ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B0TIM register.

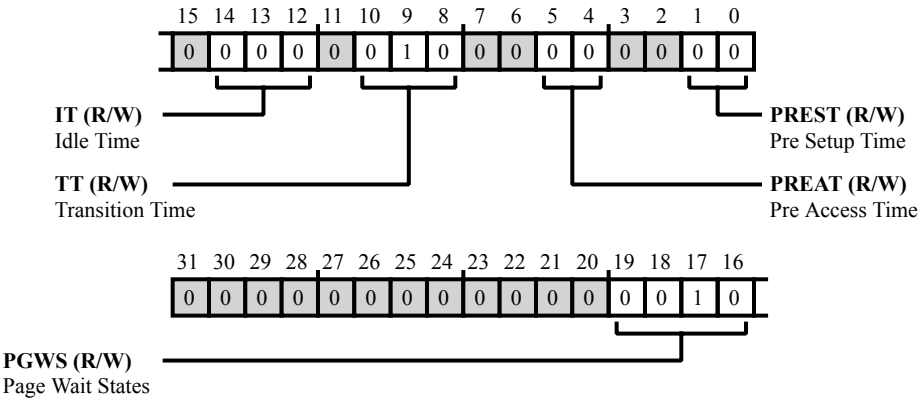


Figure 13-14: SMC_B0ETIM Register Diagram

Table 13-4: SMC_B0ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B0ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B0CTL.MODE =2). The wait time is from 2 to 15 SCLK cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B0ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the SMC_AMS[n] pin and asserting the SMC_AMS[n] pin for the next access. Note that the SMC_B0ETIM.IT period may be extended using the SMC_B0ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles

Table 13-4: SMC_B0ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The SMC_B0ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B0ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B0ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AOE/ADV pin before asserting the SMC_ARE/SMC_AWE pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B0ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMS[n] pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 0 Timing Register

The SMC_B0TIM register configures bank 0 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

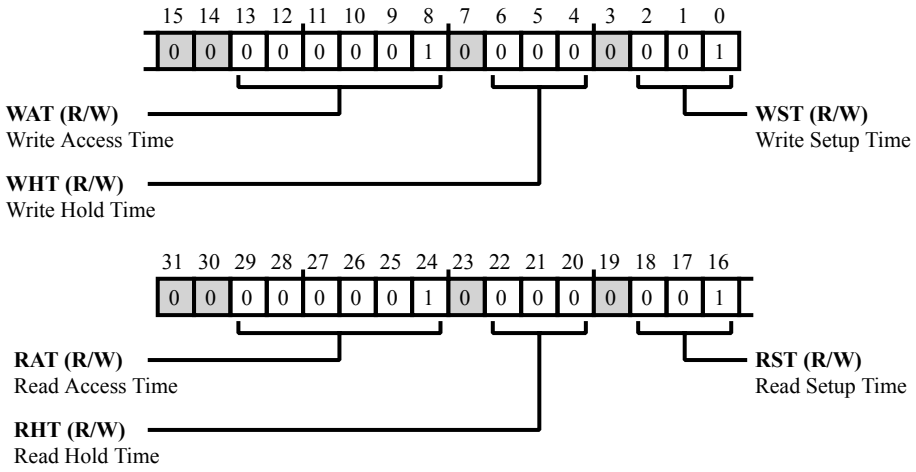


Figure 13-15: SMC_B0TIM Register Diagram

Table 13-5: SMC_B0TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B0TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B0TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Table 13-5: SMC_B0TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The SMC_B0TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B0TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B0TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AWE pin before de-asserting the SMC_AOE pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B0TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_AWE pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 1 Control Register

The SMC_B1CTL register enables bank 1 accesses and configures the memory access features for this bank.

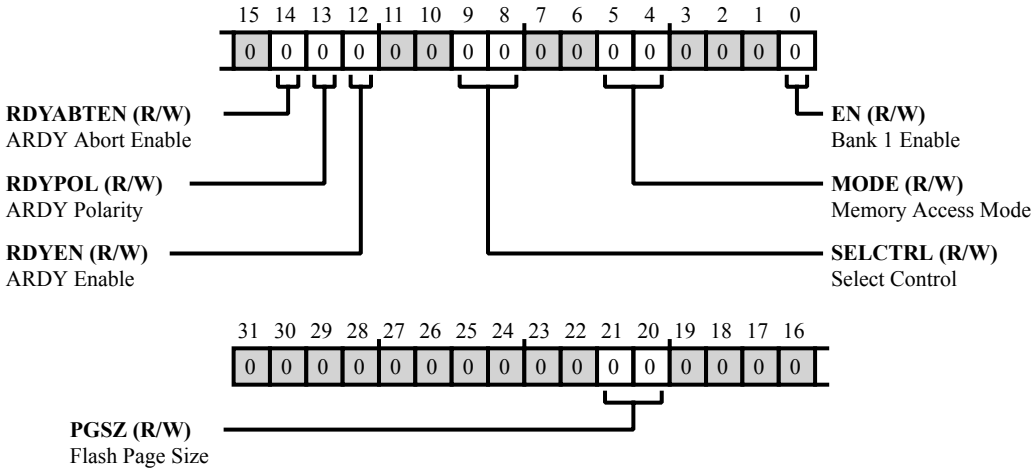


Figure 13-16: SMC_B1CTL Register Diagram

Table 13-6: SMC_B1CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B1CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B1CTL.MODE > 1). Note that the SMC_B1CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B1CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B1CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 13-6: SMC_B1CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B1CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B1CTL.RDYEN =1). After SMC_B1TIM.RAT or SMC_B1TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B1CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B1CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B1CTL.RDYEN bit enables SMC_ARDY pin operation for bank 1 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B1CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMS[n]}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS1 only
		1	AMS1 ORed with ARE
		2	AMS1 ORed with AOE
		3	AMS1 ORed with AWE

Table 13-6: SMC_B1CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The `SMC_B1CTL.MODE` bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 1 Enable. The `SMC_B1CTL.EN` bit enables accesses to the memory in bank 1. When this bit is disabled, accesses to bank 1 return an error response.	
		0	Disable access
		1	Enable access

Bank 1 Extended Timing Register

The SMC_B1ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B1TIM register.

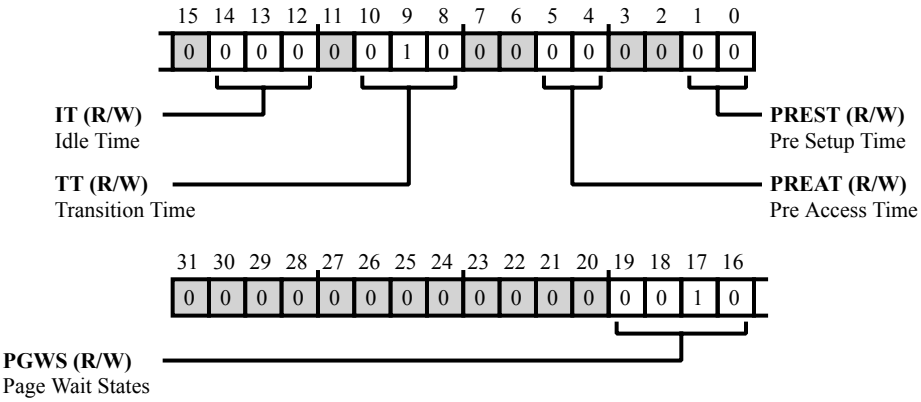


Figure 13-17: SMC_B1ETIM Register Diagram

Table 13-7: SMC_B1ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B1ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B1CTL.MODE =2). The wait time is from 2 to 15 SCLK cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B1ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the SMC_AMS[n] pin and asserting the SMC_AMS[n] pin for the next access. Note that the SMC_B1ETIM.IT period may be extended using the SMC_B1ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles

Table 13-7: SMC_B1ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The SMC_B1ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B1ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B1ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AOE/ADV pin before asserting the SMC_ARE/SMC_AWE pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B1ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMS[n] pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 1 Timing Register

The SMC_B1TIM register configures bank 1 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

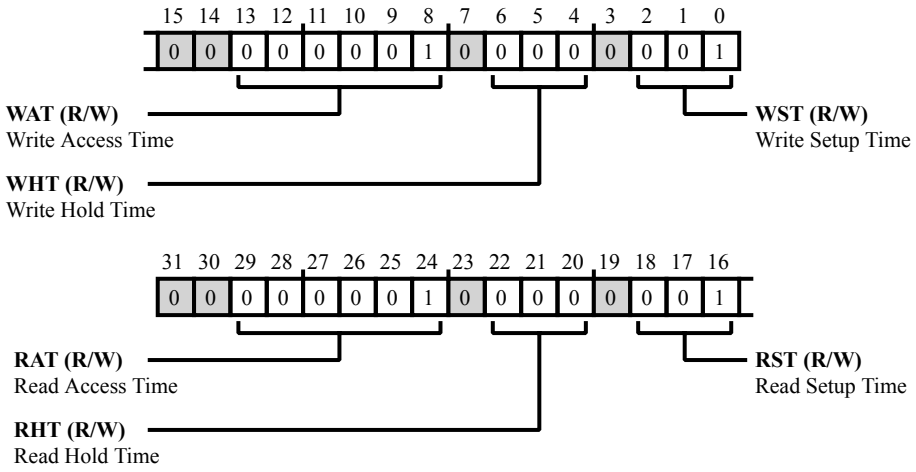


Figure 13-18: SMC_B1TIM Register Diagram

Table 13-8: SMC_B1TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B1TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B1TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Table 13-8: SMC_B1TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The SMC_B1TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B1TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B1TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AWE pin before de-asserting the SMC_AOE pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B1TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_AWE pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 2 Control Register

The SMC_B2CTL register enables bank 2 accesses and configures the memory access features for this bank.

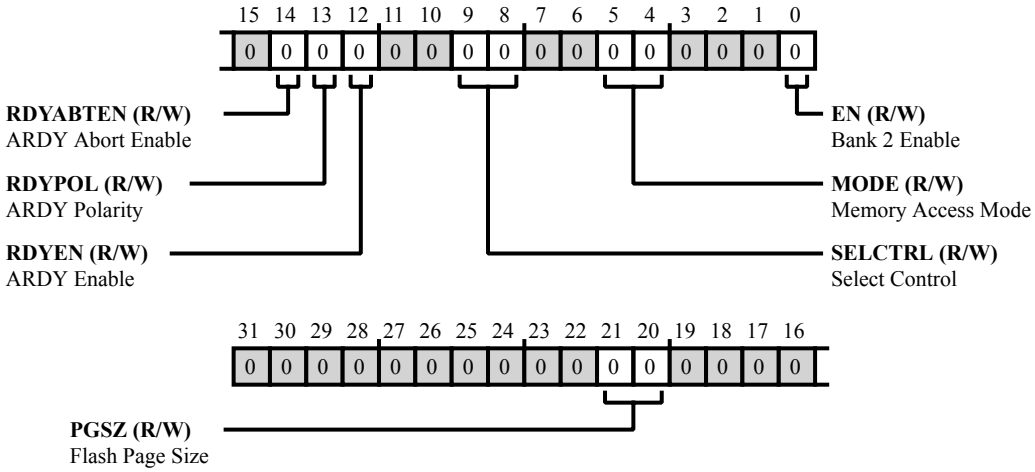


Figure 13-19: SMC_B2CTL Register Diagram

Table 13-9: SMC_B2CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B2CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B2CTL.MODE > 1). Note that the SMC_B2CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B2CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B2CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 13-9: SMC_B2CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B2CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B2CTL.RDYEN =1). After SMC_B2TIM.RAT or SMC_B2TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B2CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B2CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B2CTL.RDYEN bit enables SMC_ARDY pin operation for bank 2 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B2CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMS[n]}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS2 only
		1	AMS2 ORed with ARE
		2	AMS2 ORed with AOE
		3	AMS2 ORed with AWE

Table 13-9: SMC_B2CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B2CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 2 Enable. The SMC_B2CTL.EN bit enables accesses to the memory in bank 2. When this bit is disabled, accesses to bank 2 return an error response.	
		0	Disable access
		1	Enable access

Bank 2 Extended Timing Register

The SMC_B2ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B2TIM register.

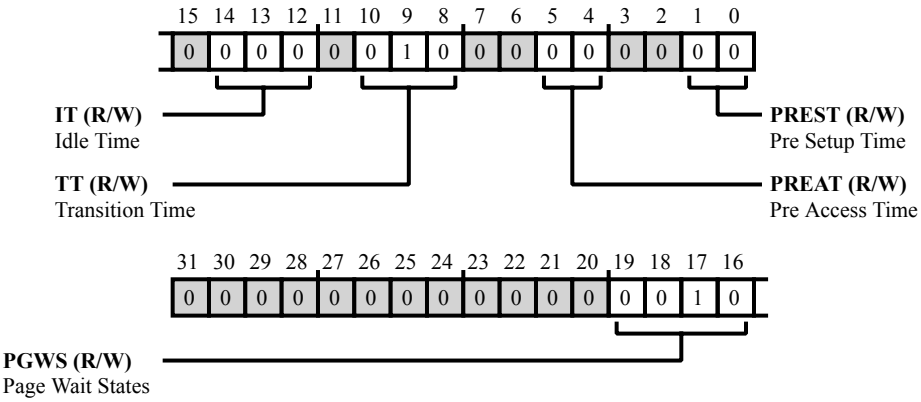


Figure 13-20: SMC_B2ETIM Register Diagram

Table 13-10: SMC_B2ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B2ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B2CTL.MODE =2). The wait time is from 2 to 15 SCLK cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B2ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the SMC_AMS[n] pin and asserting the SMC_AMS[n] pin for the next access. Note that the SMC_B2ETIM.IT period may be extended using the SMC_B2ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles

Table 13-10: SMC_B2ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The SMC_B2ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B2ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B2ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AOE/ADV pin before asserting the SMC_ARE/SMC_AWE pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B2ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMS[n] pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 2 Timing Register

The SMC_B2TIM register configures bank 2 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

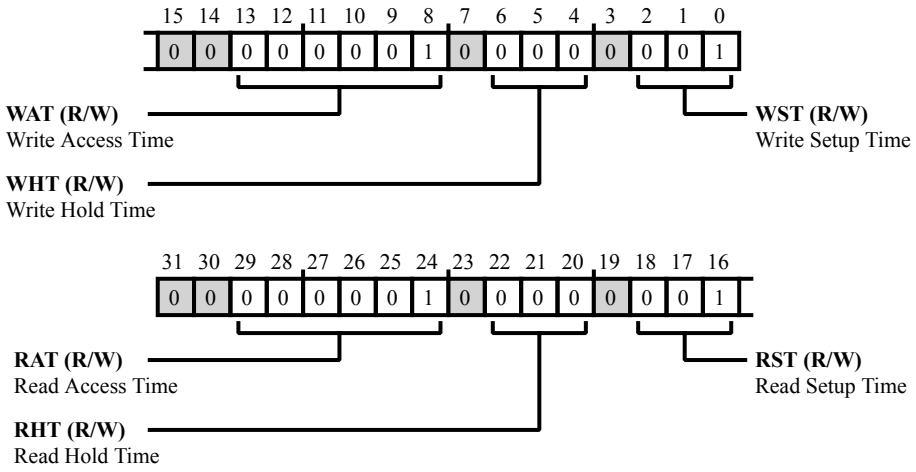


Figure 13-21: SMC_B2TIM Register Diagram

Table 13-11: SMC_B2TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B2TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B2TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Table 13-11: SMC_B2TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The SMC_B2TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B2TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B2TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AWE pin before de-asserting the SMC_AOE pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B2TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_AWE pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Bank 3 Control Register

The SMC_B3CTL register enables bank 3 accesses and configures the memory access features for this bank.

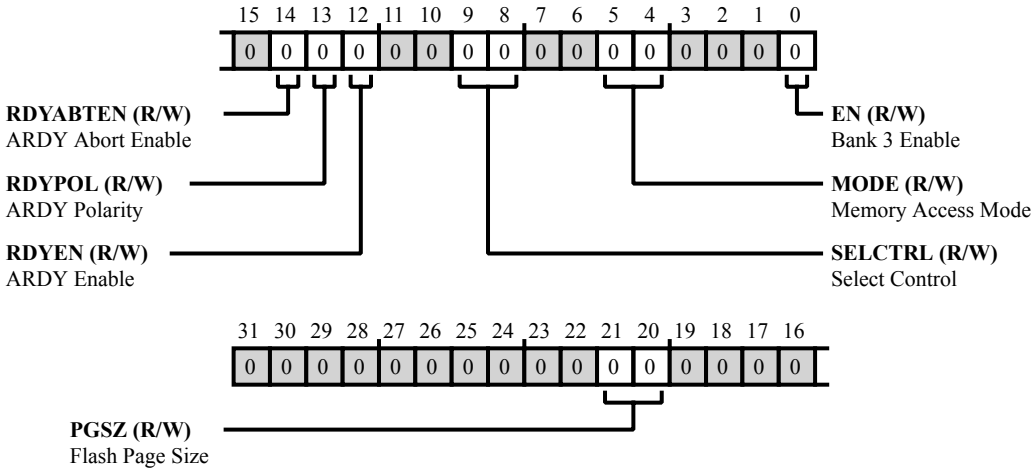


Figure 13-22: SMC_B3CTL Register Diagram

Table 13-12: SMC_B3CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	PGSZ	Flash Page Size. The SMC_B3CTL.PGSZ bits select the flash page size, if page flash or sync burst flash protocol has been enabled (SMC_B3CTL.MODE > 1). Note that the SMC_B3CTL.PGSZ bits must be set to match the flash protocol of the external flash memory device in the system. The typical SMC_B3CTL.PGSZ selection for external devices supporting async flash or async flash page protocols is 4 or 8 words. The typical SMC_B3CTL.PGSZ selection for external devices supporting sync burst flash protocol is 16 words.	
		0	4 words
		1	8 words
		2	16 words
		3	16 words

Table 13-12: SMC_B3CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	RDYABTEN	ARDY Abort Enable. The SMC_B3CTL.RDYABTEN bit enables the abort counter for the SMC_ARDY pin, if enabled (SMC_B3CTL.RDYEN =1). After SMC_B3TIM.RAT or SMC_B3TIM.WAT cycles, the SMC starts sampling the SMC_ARDY pin and starts the abort down counter (if enabled). The abort count is 64 cycles of SCLK. If the SMC detects that SMC_ARDY remains de-asserted when the counter expires, the SMC aborts the access and returns an error response back on the system bus.	
		0	Disable abort counter
		1	Enable abort counter
13 (R/W)	RDYPOL	ARDY Polarity. The SMC_B3CTL.RDYPOL bit selects the polarity (active high or low) for the SMC_ARDY pin, if enabled (SMC_B3CTL.RDYEN =1). When the SMC samples the SMC_ARDY pin in the selective active state, the transaction completes.	
		0	Low active ARDY
		1	High active ARDY
12 (R/W)	RDYEN	ARDY Enable. The SMC_B3CTL.RDYEN bit enables SMC_ARDY pin operation for bank 3 accesses. When enabled, the SMC uses SMC_ARDY (after the access time countdown) to determine completion of access to this memory bank. When disabled, the SMC ignores SMC_ARDY for accesses to this memory bank.	
		0	Disable ARDY
		1	Enable ARDY
9:8 (R/W)	SELCTRL	Select Control. The SMC_B3CTL.SELCTRL bits select the handling of the $\overline{\text{SMC\_AMS[n]}}$, $\overline{\text{SMC\_ARE}}$, $\overline{\text{SMC\_AOE}}$, and $\overline{\text{SMC\_AWE}}$ pins for memory access control.	
		0	AMS3 only
		1	AMS3 ORed with ARE
		2	AMS3 ORed with AOE
		3	AMS3 ORed with AWE

Table 13-12: SMC_B3CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	MODE	Memory Access Mode. The SMC_B3CTL.MODE bits select the protocol the SMC uses for static memory read/write access. Note that the write protocol for async flash, async flash page, and sync burst flash are all similar; only the read protocols differ for these modes.	
		0	Async SRAM protocol
		1	Async flash protocol
		2	Async flash page protocol
		3	Reserved
0 (R/W)	EN	Bank 3 Enable. The SMC_B3CTL.EN bit enables accesses to the memory in bank 3. When this bit is disabled, accesses to bank 3 return an error response.	
		0	Disable access
		1	Enable access

Bank 3 Extended Timing Register

The SMC_B3ETIM register configures extensions to access times and idle times, augmenting the setup, hold, and access times configured with the SMC_B3TIM register.

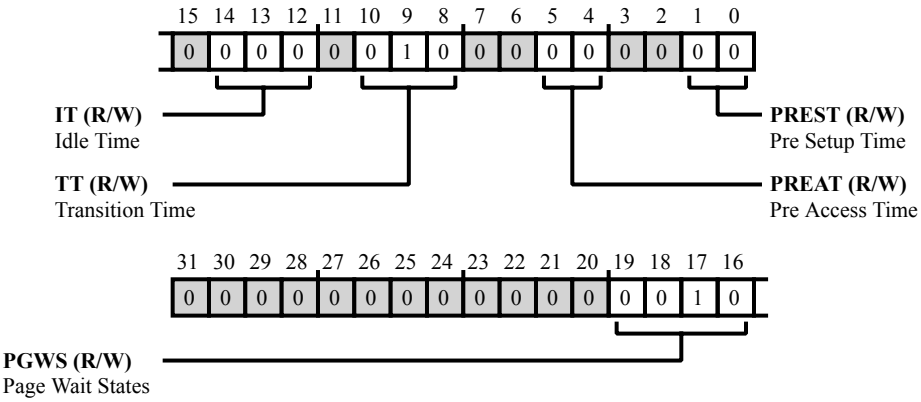


Figure 13-23: SMC_B3ETIM Register Diagram

Table 13-13: SMC_B3ETIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16 (R/W)	PGWS	Page Wait States. The SMC_B3ETIM.PGWS bits select a page access extension time (in SCLK cycles) that the SMC waits during read accesses when configured for flash page protocol (SMC_B3CTL.MODE =2). The wait time is from 2 to 15 SCLK cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B3ETIM.IT bits select a bus idle time (in SCLK cycles) that the SMC waits between de-asserting the SMC_AMS[n] pin and asserting the SMC_AMS[n] pin for the next access. Note that the SMC_B3ETIM.IT period may be extended using the SMC_B3ETIM.TT selection. The idle time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		7	7 SCLK clock cycles

Table 13-13: SMC_B3ETIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/W)	TT	Transition Time. The SMC_B3ETIM.TT bits select a bus idle time (in SCLK cycles) that the SMC extends the SMC_B3ETIM.IT to allow for the subsequent access either using a different transfer direction or accessing a different bank. The transition time is from 1 to 7 SCLK cycles.	
		0	No bank transition
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B3ETIM.PREAT bits select the pre-access time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AOE/ADV pin before asserting the SMC_ARE/SMC_AWE pin for the current access. The pre-access time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B3ETIM.PREST bits select the pre-setup time (in SCLK cycles) that the SMC asserts the SMC_AMS[n] pin before asserting the SMC_AOE/ADV pin for an access. The pre-setup time is from 0 to 3 SCLK cycles.	
		0	0 SCLK clock cycles
		3	3 SCLK clock cycles

Bank 3 Timing Register

The SMC_B3TIM register configures bank 3 read and write access, setup, and hold timing for this bank. Note that read and write timing configurations are independent and may differ.

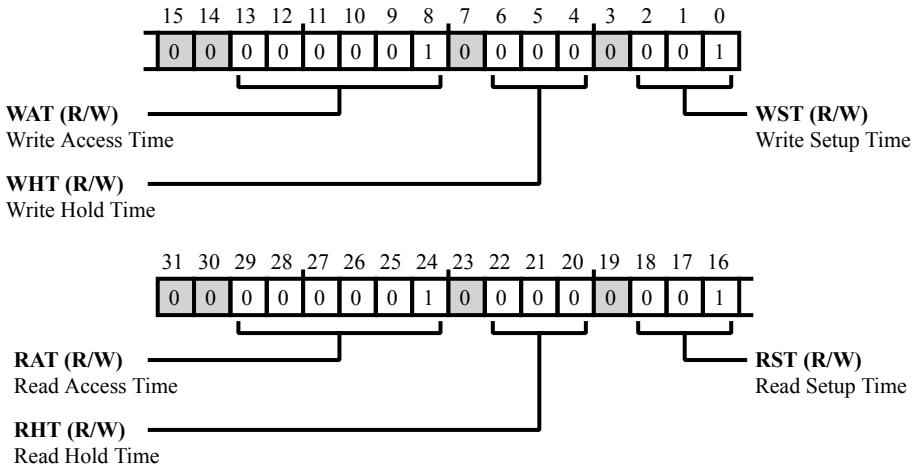


Figure 13-24: SMC_B3TIM Register Diagram

Table 13-14: SMC_B3TIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	RAT	Read Access Time. The SMC_B3TIM.RAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B3TIM.RHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_ARE pin before asserting the SMC_AOE pin for the next access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

Table 13-14: SMC_B3TIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RST	Read Setup Time. The SMC_B3TIM.RST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_ARE pin for an access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B3TIM.WAT bits select the access time (in SCLK cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK cycles.	
		0	Not supported
		1	1 SCLK clock cycle
		63	63 SCLK clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B3TIM.WHT bits select the hold time (in SCLK cycles) that the SMC waits after de-asserting the SMC_AWE pin before de-asserting the SMC_AOE pin for the current access. The hold time is from 0 to 7 SCLK cycles.	
		0	0 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B3TIM.WST bits select the setup time (in SCLK cycles) that the SMC asserts the SMC_AOE pin before asserting the SMC_AWE pin for a write access. The setup time is from 1 to 8 SCLK cycles.	
		0	8 SCLK clock cycles
		1	1 SCLK clock cycle
		7	7 SCLK clock cycles

14 Flash Controller (FLC)

The processor features ECC-protected embedded parallel flash memory. It consists of up to 1024 KB of main memory and up to 8 KB of information memory. The main memory is used to store program code and data, while the information block can be used to store specialized information, such as security keys.

FLC Features

The FLC module has the following features.

- 128-bit wide flash
- SEC-DED type ECC protection
- Read/write protected Information block
- Pre-fetcher, branch detector and small cache
- Flash event interrupts and triggers
- Sleep mode

FLC Functional Description

The following sections provide information on the function of the FLC.

CM41X_M4 FLC Register List

The FLC module control the flash memory. This memory space includes up to 1M byte of flash memory which holds the user program and constant data. The initial vector table and reset boot vector are located at the base of flash memory.

Table 14-1: CM41X_M4 FLC Register List

Name	Description
FLC_ADDR	Command Address Register
FLC_CMD	Command Register

Table 14-1: CM41X_M4 FLC Register List (Continued)

Name	Description
FLC_CMD_KEY	Command Key Register
FLC_CTL	Control Register
FLC_DATA0	Command Data Register 0
FLC_DATA1	Command Data Register 1
FLC_ECC	Command ECC Register
FLC_FILL_CNT	Pre-fetcher Fill Count Register
FLC_IMSK	Interrupt Enable Register
FLC_MISS_CNT	Pre-fetcher Miss Count Register
FLC_PRFCTL	Prefetch Control Register
FLC_REF_CNT	Pre-fetcher Reference Count Register
FLC_REVID	Revision ID Register
FLC_SBERR_THR	ECC Single Bit Error Threshold Register
FLC_STAT	Status Register
FLC_TIM_PWR	Power Timing Register

FLC Block Diagram

The dedicated Flash controller connected to each flash block controls the read, write, erase and power management operation of flash. The ARM M4 processor issues the commands by writing to the memory-mapped registers of the flash controller. Executing these commands results in performing flash operations such as page erase, mass erase, 64-bit write (or program), sleep, powerdown and wakeup.

The flash controller allows the processor core to directly execute the code from flash with performance optimized by a flash prefetcher. As shown in the ***Flash Controller Block Diagram,*** the ARM M4 processor can access the embedded flash memory through its ICODE and DCODE interfaces, which are tightly coupled to the flash controller's core ports. Flash memory can be read by other system modules using the flash controller's DMA interface.

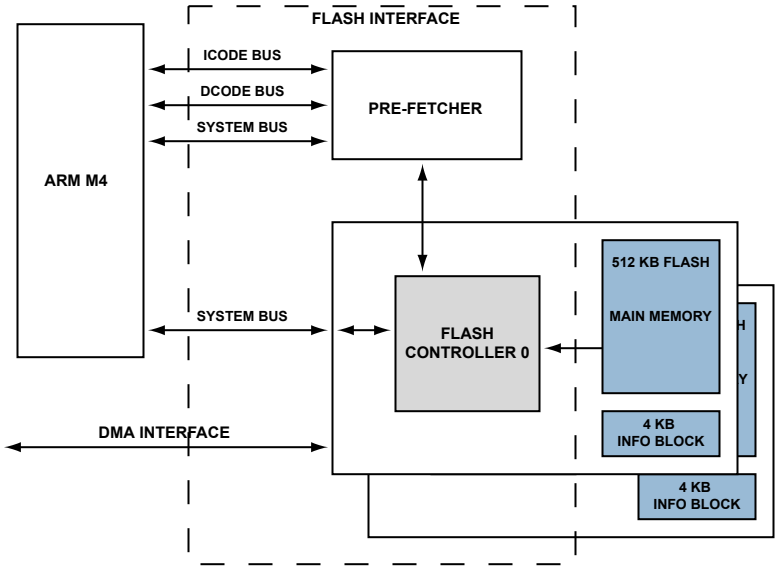


Figure 14-1: Flash Controller Block Diagram

FLC Architectural Concepts

The following sections provide information on the architecture of the Flash controller module.

Flash Memory Organization

Flash memory is partitioned into two equal blocks (shown in the ***Flash Memory Organization*** figure), each containing 512 KB of main memory and 4 KB of information memory. Each flash block is independently controlled by a dedicated flash controller, which supports reading the flash contents, erasure and programming, power management functions, and management of ECC error protection.

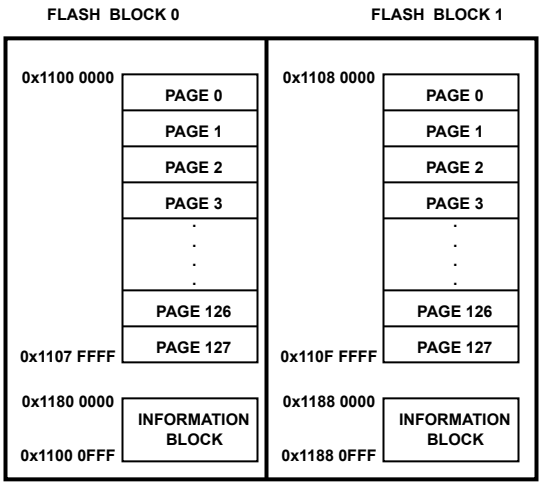


Figure 14-2: Flash Memory Organization

The ***Flash Page Architecture*** figure shows the details of flash organization. The main memory of each flash block is organized into 128 programmable pages of 4K-bytes each. A 4K byte page in the memory occupies 8 rows of 32

columns, where each column is of 128-bits. Each column is divided into two 64-bit data long words, each with an associated 8-bit Error Correcting Code (ECC) capable of SEC-DED (single-bit error correction, Dual-bit error detection).

The smallest independently programmable unit of flash memory is an aligned 64-bit long word. The state of unprogrammed (blank) flash memory is all 1s including the ECC bits, which is defined as a legal ECC state (not an ECC error.) Flash memory bits can be programmed from 1 to 0, but require an Erase or Mass Erase operation to return from 0 to 1. Flash memory may be erased in one of three ways: in aligned 4K-byte Pages, in a Mass Erase of an entire 512KB main memory without the Information memory, or in a Mass Erase which includes the Information memory. If either information memory block is erased, a reformatting operation is required before the next chip reset (see Information Memory below.)

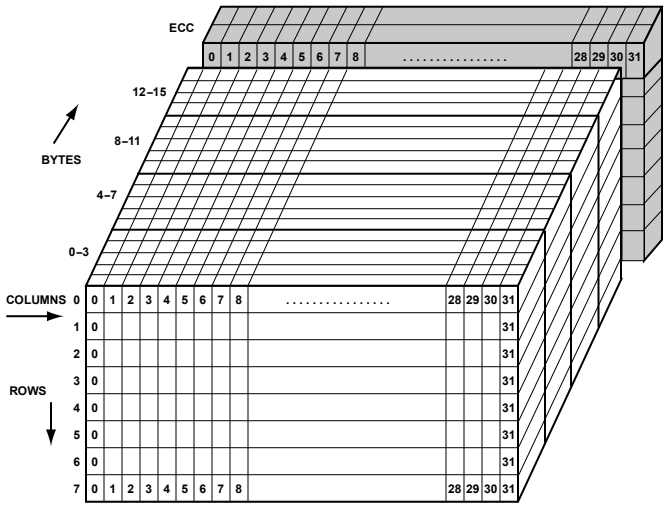


Figure 14-3: Flash Page Architecture

Flash Prefetcher

At the extremely high core clock rates supported by the flash memory subsystem, the core and DMA may operate at a speed far in excess of the underlying physical flash memory cycle time. The flash prefetcher is provided to sustain high processor performance under these conditions. It uses proprietary access prediction methods to anticipate the core's future accesses, so that effectively zero-wait-state accesses are nearly always achieved. Both code and data accesses are accelerated by the prefetcher.

These prediction methods are not inherently history-dependent, unlike traditional cache memory methods, so the same high performance is achieved each time the code is executed: first time, every time.

Information Memory

Each flash block is associated with a separate 4 KB information block. Access controls are provided to manage access to information blocks, including reading, writing and erasure.

The `FLC_CTL.IRFWEN` (information block write enable) bit controls write or erase command access to the information block. If this bit =0, then neither write, program, erase, or mass erase commands are allowed to the information block. If this bit =1, then these operations in the information block are allowed.

The FLC_CTL.IRFREN (information block read enable) bit controls read access to the information block. If this bit =0, neither bus reads nor register-command reads to the information block are allowed. In this case bus reads return a bus error and register command reads return 0. If this bit =1, then read accesses to the information block are allowed.

The flash controller provides mass erase operations, which can erase the main 512 KB blocks either with or without the associated information block. Given these capabilities, the information blocks may be used to store protected information about the system, such as serial numbers or system calibration or configuration data.

The information memory is also used by certain processor features, which use dedicated, reserved locations in the information memory. These features include:

- The user's Secure Debug Key (128 bits), which if non-blank requires a matching key to be provided by any connected JTAG/SWD emulator/debugger in order to attach to the product;
- The user's Secure Debug Key ID (128 bits), which can be used for any purpose by the user, and can be read by an attached JTAG/SWD debugger prior to unlocking the part (for example to provide a serial number for identifying individual parts if individualized Secure Debug Keys are desired);
- Pre-boot system initialization features, including RunControl switches, lists of MMR address/value pairs for hardware configuration, and a user-defined pre-boot initialization function. These can be used, for example, to rapidly configure a peripheral like a Logic Block Array or a GPIO port.

Flash Block 0	Address Offset			
Address Base	**0xC**	**0x8**	**0x4**	**0x0**
0x1180_0FF0	SDBGKEY3	SDBGKEY2	SDBGKEY_CON_3	SDBGKEY_CON_2
0x1180_0FE0	SDBGKEY1	SDBGKEY0	SDBGKEY_CON_1	SDBGKEY_CON_0
0x1180_0FD0	Reserved	Reserved	RCMSG	RCMSG_KEY
0x1180_0FC0	Reserved	RFUNC_CSIZ	RFUNC_PTR	RFUNC_KEY
0x1180_0FB0	RINIT_CNT	RINIT_CSIZ	RINIT_PTR	RINIT_KEY
0x1180_0FA0	SDBGKEYID3	SDBGKEYID2	SDBGKEYID1	SDBGKEYID0

Flash Block 1	Address Offset			
Address Base	**0xC**	**0x8**	**0x4**	**0x0**
0x1188_0FF0	SDBGKEY3	SDBGKEY2	SDBGKEY_CON_3	SDBGKEY_CON_2
0x1188_0FE0	SDBGKEY1	SDBGKEY0	SDBGKEY_CON_1	SDBGKEY_CON_0

At system initialization time, the flash controller hardware accesses these keys and sets the Locked or Unlocked security states for the product, which control access to the part by the JTAG/SWD debug port and the UART boot loader. Refer to the Security chapter for more details about these fields.

ECC Protection

The entire flash region is ECC protected by SEC-DED (single-error correcting and double-error detecting) codes. The ECC checksum can detect and correct single bit errors in flash data and it can detect 2-bit errors and trigger error interrupts to convey this condition to core.

Each 64-bits of flash data is associated with 1-byte of ECC. Therefore, SEC-DED ECC operations are performed on 64-bits (8-bytes) of flash data. See the Figure 14-3 Flash Page Architecture figure for more details.

Flash can be programmed for 64-bits data at-a-time. While programming, 1-byte ECC is calculated internally and is programmed along with data bits.

Note that flash data bits are writeable only from 1 to 0, and that the ECC pattern corresponding to a 64-bit data word is complex. Cumulative programming of the same location is not supported (for example changing different bits from 1 to 0 at different times), since the resulting ECC codes for each incremental pattern are likely to require changing ECC bits in an illegal direction (0 back to 1). A given location can only be re-programmed after an erase of the page in which it resides (or by a mass erase of that flash memory block.)

Each flash access provides 128-bits of flash data. Single or multi-bit errors can occur on either half of a 128-bit flash read. Errors on both halves are reported irrespective of which 32-bit word was accessed by DMA or core reads.

Non-Volatile Write Control

The non-volatile write control unit provides low-level timing control for the flash memory, and implements flash self-protection mechanisms. To use this feature, set the `PADS_NVWR_RSTCTL.EN` bit (=1). At power-on, the unit is disabled.

The unit is typically enabled automatically by the boot ROM code, before control transfers to the user application. The status of the write control unit can be queried by reading the `PADS_NVWR_RSTCTL.EN` bit. This bit must read 1 before any programming or erase operations are performed.

Flash Performance Modes

The flash operates in the ARM M4 processor's core clock domain, CCLK. The flash has four performance modes called A, B, C, and E, which depend on the CCLK frequency. These modes control prefetcher operation and the Flash wait state count for each physical flash access. Program the `FLC_PRFCTL.PMODE` field appropriate to the Cortex-M4 core clock frequency.

The flash memory can only be accessed when the CCLK frequency does not exceed the upper limit for the selected flash performance mode (including both core and DMA accesses). It is recommended (but not required) that the performance mode be set so that the CCLK frequency is in the range of the selected performance mode, in order to achieve optimum CPI (cycles per instruction) performance. Care must be taken whenever changing CCLK frequencies to comply with these requirements throughout the frequency transition.

For example, when operating at a CCLK of 150 MHz, performance mode B is recommended in order to provide the best CPI effective core performance, while modes C and E would be allowed but at less optimal performance, and mode A would not be allowed. On the other hand, performance mode E is permitted at any CCLK frequency, although it is the best choice only at speeds above 225 MHz.

Flash Operations

The Flash controller controls all the flash operations, such as code fetch, data read, DMA read, page erase, mass erase, program, entering or exiting flash low power mode and other operations.

The Flash controller operates in two modes: normal mode and command mode. In normal mode, flash controller serves the flash read requests from core for direct code execution or data read operations. It also serves flash data read operations from the system controller's DMA interface for system peripherals. In this mode, the flash is directly accessed by its memory-map address.

For flash program or erase operations, the flash controller must be in command mode. This mode is entered by setting (=1) the FLC_CTL.CMDM bit, followed by writing a specific set of keys to the FLC_CMD_KEY register. The FLC_STAT.CMDM bit reflects flash controller mode. When in command mode, the flash controller disables all traffic from the flash prefetch unit and DMA port. If memory-mapped flash accesses are attempted, the controller returns ERROR on the respective bus. The command mode can be exited to enter into normal mode by clearing (=0) the FLC_CTL.CMDM bit.

The non-volatile write control unit is enabled using the non-volatile write reset control register (PADS_NVWR_RSTCTL). (See Non-Volatile Write Control). This unit is normally enabled by the boot ROM before the user application starts.

Programming and erase operations must be carried out by code resident outside the flash memory being operated on, for example in SRAM or in the opposite flash block. This is because normal read/fetch access to a flash memory is disabled during these operations.

The page erase, mass erase and program operations are timed by an internal 1 MHz clock, irrespective of the core clock (CCLK) or system clock (SYSCLK) settings. These timings do not require any user intervention when CGU clock ratios are changed. This 1 MHz clock is supplied by the OSCWD unit. Disabling the clock causes flash programming and/or erase operations to fail.

A flash controller can perform the operations described in the following sections when in command mode. The ARM M4 processor can issue these commands by writing to the FLC_CMD register.

Page Erase (PER)

When the flash controller is in command mode, it can accept the page erase command. The FLC_ADDR register specifies the page address of flash to be erased. If an information memory page needs to be erased, then the applicable write access permission is needed, which is controlled by the FLC_CTL.IRFWEN bit.

The erase operation is started by issuing the page erase (PER) command. When the controller accepts this command, the FLC_STAT.PERP bit is set (=1) to indicate that the page erase operation is in progress. Once this operation is complete, the FLC_STAT.PERP bit is cleared (=0) and the FLC_STAT.CMDEX bit is set. The optional controller event interrupt is generated when the FLC_IMSK.PER bit is set.

The page erase operation erases all 4608 bytes within a page including the 4096 memory bytes and the 512 bytes that contain the ECC. All the bits are set to 1 after the erase operation, which can be changed to 0 using the program. One page from each flash block can be erased at the same time, as the two flash blocks are independently controlled by their dedicated flash controller.

The erased state (all 1s) is a legal ECC state.

Mass Erase (MER)

When the flash controller is in command mode, it can accept the mass erase command. The command erases all 128 pages of main memory, with or without information memory. If the information memory page also needs to be erased, then the info-block write access permission, controlled by the `FLC_CTL.IRFWEN` bit, is required.

The mass erase operation can be started by issuing the mass erase (MER) command. When the flash controller accepts this command, the `FLC_STAT.MERP` bit is set (=1) to indicate that the mass erase operation is in progress. Once this operation is complete, the `FLC_STAT.MERP` bit is cleared (=0) and the `FLC_STAT.CMDEX` bit is set. The optional controller event interrupt is generated when the `FLC_IMSK.MER` bit is set.

Both flash blocks can be erased at the same time, as the two flash blocks are independently controlled by their dedicated flash controller.

Flash Program

A program cycle consists of three operations; program mode entry, program locations in the row, and program mode exit.

- In program mode entry, a row within a particular flash page is activated.
- In program mode, any aligned 64-bit long word within the activated row can be programmed, along with its ECC.
- In program exit mode, the activated row is deactivated.

Entering the Program Mode (SPGMM)

When the flash controller is in command mode, it can accept the command to enter into program mode. The `FLC_ADDR` register should point to the required row address of flash. If the row address is within information block, write access is required and is controlled by the `FLC_CTL.IRFWEN` bit.

The program mode can be entered by issuing the SPGMM (start program mode) command. When the flash controller accepts this command, the `FLC_STAT.PGMMP` bit is set to indicate that the ‘set program mode’ operation is in progress. Once this operation is complete, the `FLC_STAT.PGMMP` bit is cleared and the `FLC_STAT.PGMM` bit is set to indicate that the flash controller is in program mode. At the end of execution, the Flash controller event interrupt is generated if the `FLC_IMSK.PGMMC` bit is set.

Program Command (PGMC)

When the flash controller successfully executes the program mode entry command, it sets itself in program mode for that particular row. The flash row contains 32-columns and each column contains 128-bits, of which any of the 64-bit halves can be programmed at one time. All the bits within row (32 x 128-bits) can be programmed when a row is activated. It is not recommended to reprogram the same 64-bits multiple times, as the ECC corresponding to that data may become invalid with multiple writes.

The program command (PGMC) takes the data in the FLC_DATA0 register and the FLC_DATA1 register and programs it to a flash location specified by contents of the FLC_ADDR register, along with the internally calculated 8-bit ECC. The FLC_ADDR registers must specify an address aligned to an 8-byte boundary. The FLC_STAT.PGMCP bit is set to indicate that the program operation is in progress. After completion, the FLC_STAT.PGMCP bit is cleared and the FLC_STAT.CMDEX bit is set. The optional controller event interrupt is generated when the FLC_IMSK.PGM bit is set.

To program flash locations in a different row, execute the end program mode (EPGMM) command to exit the program mode, followed by a start program mode (SPGMM) command with the FLC_ADDR register set to the new row address.

Exiting the Program Mode (EPGMM)

After successfully programming the row contents, to finish the program cycle properly, the end program mode (EPGMM) command must be issued at the end. If the flash controller accepts this command, the FLC_STAT.PGMMP bit is set to indicate that the program mode change operation is pending. After this process completes, the FLC_STAT.PGMMP bit is cleared and the FLC_STAT.CMDEX bit set.

Do not leave a flash block in programming mode after completing a programming operation. See the data sheet for timing details.

Flash Read

When in command mode, the flash controller disables all the traffic from the flash prefetch unit and DMA port. If memory mapped accesses to flash are attempted, it returns an ERROR on the respective bus.

To read the flash contents in command mode, use the register-based read (RDC) command. When the flash controller receives the RDC command, the FLC_STAT.RDCP bit is set to indicate that a read operation is in progress. The FLC_ADDR register should specify which flash location needs to be read. At the end of execution, the FLC_STAT.RDCP bit is cleared and 64-bit flash read data is stored into the FLC_DATA0 (lower word) and the FLC_DATA1 (upper word) registers. An ECC check is performed on the read data and the FLC_STAT.EERRMF bit is set when multi-bit ECC errors occur.

NOTE: This read is through the command mode, it is unrelated to array reads by processor ICODE/DCODE buses during normal operation.

Flash Sleep Mode and Wake from Sleep (SLPC and WKSL)

The flash supports a sleep mode, which can be used as a low power mode. It is entered by issuing the sleep (SLPC) command. When the flash controller receives this command it completes all the outstanding requests and sets the FLC_STAT.SLPCP bit (sleep command pending) to indicate that a sleep mode command is being executed. After successful execution, the FLC_STAT.SLPCP bit is cleared and the FLC_STAT.SLP bit is set to indicate the flash is in sleep mode.

When in sleep mode, the only allowed flash commands are wakeup from sleep (WKSL) and ABORT. If other commands are attempted, the FLC_STAT.CFSLP error bit is set indicating that a command was issued in sleep mode and optionally a Flash Error interrupt is triggered if the FLC_IMSK.IFL bit is set.

Sleep mode can be exited by issuing wakeup from sleep (WKSL) command. At the end of execution, the FLC_STAT.WKUPP bit is cleared (=0) and a flash event interrupt is generated if the FLC_IMSK.WKUP bit is set.

In sleep mode, if the FLC_CTL.WKAE bit =0, the flash controller ignores the data requests from the pre-fetcher and the DMA engines and triggers a bus error. When the FLC_CTL.WKAE bit =1, it enables flash sleep wakeup on flash access. Therefore, any read access can cause the flash to wakeup from sleep mode.

The wakeup time from sleep mode to standby mode is determined by the FLC_TIM_PWR.WKS bit field. This field is defined in terms of the core clock (CCLK) and should be programmed as per the wakeup time specified in the data sheet.

Abort Operation (ABORT)

The abort command is used if a flash operation must be terminated before its normal completion. This is initiated when the flash is in command mode and the FLC_CMD.FLCMDS bit field is written with the abort command value. This clears the FLC_STAT.PGMM bit and other command-pending bits, and sets the FLC_STAT.ABTP (abort command pending) status bit. The abort completes after a short internally-timed delay. At the completion of the abort sequence, the FLC_STAT.ABTP bit is cleared. The optional controller event interrupt is generated when the FLC_IMSK.ABORT interrupt enable bit is set.

See FLC Operating Requirements below for more information about the abort mechanism.

FLC Operating Requirements

Certain flash memory operations require significant time to complete. Operations which erase or program the memory may take on the order of microseconds or milliseconds. (Refer to the data sheet for details.) In recommended usage, programs may employ interrupts or status bits to synchronize the software program with the duration of the internally-timed hardware operation.

For these flash operations to complete successfully, the device must remain supplied with power and clock within device specifications and remain operating normally throughout the entire operation sequence (for example, not subjected to a hard reset from the SYS_HWRST pin).

For cases where a flash operation must be terminated abruptly, the flash controller supports an abort mechanism. This mechanism can be initiated in two ways: (1) by the abort command, or (2) by a loss-of-power event detected

by a Voltage Monitoring Unit (VMU). The abort mechanism does require a short time to complete, during which internal flash protection mechanisms are bringing the flash into a safe state. The time required for the abort mechanism is much shorter than the long erase operations (a few microseconds rather than many milliseconds. See the product specific data sheet for details).

Like the program and erase mechanisms, the abort mechanism is internally self timed, and further, it is insensitive to externally or internally initiated resets. All the abort needs to complete is for the power supplies to remain within specification for the length of time specified in the datasheet. Note that if a short hard-reset pulse is applied after starting a program, erase, or abort flash operation, the abort mechanism may still be in process when the chip emerges from reset. New programming operations after reset should always test the state of the `PADS_NVWR_RSTCTL.EN` bit before proceeding.

If a flash operation is not allowed to complete normally, its effect on the addressed flash memory is UNSPECIFIED. The selected locations may not be completely erased or programmed. If a program or erase flash operation is started but is terminated by a complete abort sequence, the remainder of the flash memory is unaffected.

CAUTION: If a program or erase operation is not completed normally and is not terminated by a complete abort, for example before power loss, the effect on the entire flash memory block is UNSPECIFIED and device damage may result. For this reason, the VMU and other protective mechanisms are provided. With careful system and power supply design, the safety and abort mechanisms function together to protect the flash memory and its contents.

ADI recommends testing the Flash Security Enable state on each device prior to programming secure IP into the ADSP-CM41x Flash memory. Flash Security Enable is a one-time setting that is programmed to 1 in every device by ADI prior to shipment. The Flash Security Enable state can be checked by reading the `SYSBLK_LROM_STAT.FSEREG`. This can be accomplished using either of two methods:

- From the debugger: Read the `SYSBLK_LROM_STAT.FSEREG` directly.
- From the UART loader: Load a program into SRAM that reads from the `SYSBLK_LROM_STAT.FSEREG`, and reports the state to the master device.

If `SYSBLK_LROM_STAT.FSEREG` = 1, programming Secure IP into the flash memory can proceed.

If `SYSBLK_LROM_STAT.FSEREG` = 0, the flash programming operation must be halted, as the device cannot be secured. Contact ADI for support.

FLC Event Control

The following sections provide information about the event and ECC interrupts associated with the Flash Controller.

Interrupts

The Flash controller provides three interrupt channels corresponding to different events and error conditions.

Flash Event Interrupt

The flash controller generates a Flash event interrupt on completion of various flash operations. These operations include interrupts on:

- completion of Page Erase operation
- completion of Mass Erase operation
- completion of Program operation
- completion of Program mode entry operation
- completion of Sleep Wakeup command
- completion of Abort command
- Improper flash command
    - command was issued while flash was busy
    - command was issued while flash was in sleep or powered down mode
    - command was issued when the controller was not in command mode
    - command violated information block privileges
    - command violated program mode guidelines

The interrupts from both flash controllers are combined on single flash Event interrupt channel. It is required to check the status registers of both flash controllers to determine the source of interrupt.

Flash ECC Interrupt

Flash memory is protected by a SEC_DEC type ECC. The ECC checksum is performed on read accesses. Single-bit errors are corrected. Two-bit error are detected and reported through the ECC interrupt.

The processor provides two separate error interrupts for multi-bit ECC errors when data is read in core mode (Flash core ECC error interrupt) and when data is read by the DMA engine (Flash DMA ECC error interrupt). Interrupts from both flash controllers are combined on their respective interrupt channel.

The Flash DMA ECC error interrupt can be enabled by setting the `FLC_IMSK.MBERRD` bit. The Flash core ECC error interrupt can be enabled by setting the `FLC_IMSK.MBERRC` and `FLC_IMSK.MBERRD` bits.

CM41X_M4 FLC Register Descriptions

Flash Controller (FLC) contains the following registers.

Table 14-2: CM41X_M4 FLC Register List

Name	Description
FLC_ADDR	Command Address Register
FLC_CMD	Command Register
FLC_CMD_KEY	Command Key Register
FLC_CTL	Control Register
FLC_DATA0	Command Data Register 0
FLC_DATA1	Command Data Register 1
FLC_ECC	Command ECC Register
FLC_FILL_CNT	Pre-fetcher Fill Count Register
FLC_IMSK	Interrupt Enable Register
FLC_MISS_CNT	Pre-fetcher Miss Count Register
FLC_PRFCTL	Prefetch Control Register
FLC_REF_CNT	Pre-fetcher Reference Count Register
FLC_REVID	Revision ID Register
FLC_SBERR_THR	ECC Single Bit Error Threshold Register
FLC_STAT	Status Register
FLC_TIM_PWR	Power Timing Register

Command Address Register

The FLC_ADDR register holds the address needed by the commands executed by flash controller in the command mode. This is treated as a byte address. In normal mode, this register is updated on any multi-bit ECC error with the address of the error location. In case of a simultaneous multi-bit errors, it is updated with address of one of the DCODE, ICODE, DMA or pre-fetcher transactions listed in decreasing order of priority. If the controller is busy, a write to this register is ignored to avoid corrupting data for the command being executed.

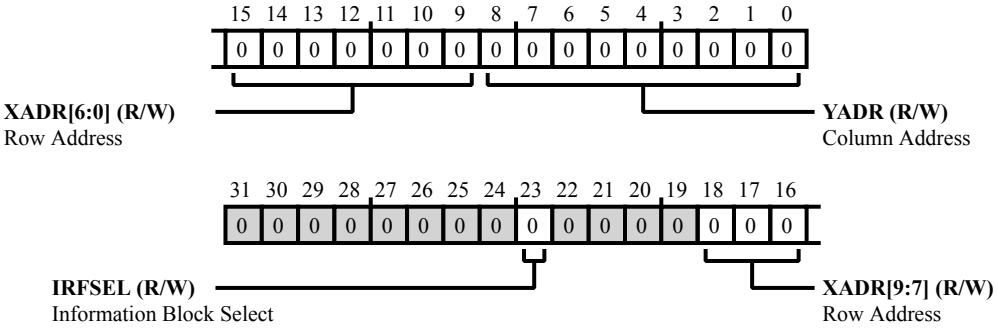


Figure 14-4: FLC_ADDR Register Diagram

Table 14-3: FLC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23 (R/W)	IRFSEL	Information Block Select. The FLC_ADDR.IRFSEL bit selects the information block for register-based commands.	
		0	All commands (program, erase, mass erase, read) apply to the main array.
		1	The commands (read, program, erase) apply to the info block. The command mass erase applies to the entire memory both main and info blocks.
18:9 (R/W)	XADR	Row Address. The FLC_ADDR.XADR bit field holds the row address for the flash controller commands.	
8:0 (R/W)	YADR	Column Address. The FLC_ADDR.YADR bit field holds the column address for the flash controller commands.	

Command Register

Various flash controller commands are executed by writing the command code to the FLC_CMD register.

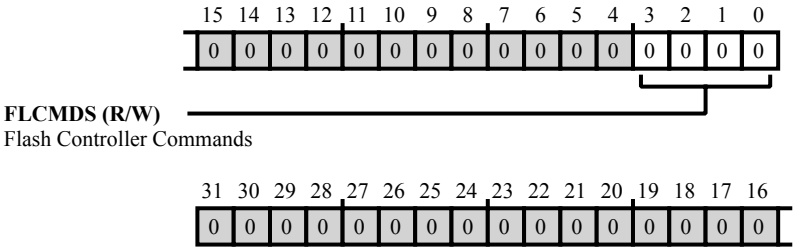


Figure 14-5: FLC_CMD Register Diagram

Table 14-4: FLC_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	FLCMDS	Flash Controller Commands. The FLC_CMD.FLCMDS bit field executes the various flash commands.	
		1	Page Erase Command (PER). Erases a page specified by the address in FLC_ADDR. If FLC_ADDR.IRFSEL and FLC_CTL.IRFWEN bits are set, a page in information block is erased instead of one in the data block. FLC_STAT.PERP bit is set to indicate a page erase operation is in progress. At the end of execution, FLC_STAT.PERP bit is cleared and interrupt is generated if FLC_IMSK.PER bit is set.
		2	Mass Erase command (MER). Erases the entire flash. If FLC_ADDR.IRFSEL and FLC_CTL.IRFWEN bits in FLC_CTL are set, information block is erased as well. FLC_STAT.MERP bit is set to indicate a mass erase operation is in progress. At the end of execution, FLC_STAT.MERP bit is cleared and interrupt is generated if FLC_IMSK.MER bit is set.
		3	Program command (PGMC). Programs data in FLC_DATA0 and FLC_DATA1 to a location specified by contents of FLC_ADDR. 8 ECC bits originating from internal ECC generation logic are programmed as well. FLC_STAT.PGMCP bit is set to indicate that the program operation is in progress. At the end of execution, FLC_STAT.PGMCP bit is cleared and interrupt is generated if FLC_IMSK.PGM bit is set.
		4	Start Program Mode (SPGMM). Sets the flash controller in program mode for a particular row.

Table 14-4: FLC_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
			FLC_STAT.PGMMP bit is set to indicate that the set program mode operation is in progress. After the operation is complete, FLC_STAT.PGMM bit is set to indicate that the flash controller is in program mode; FLC_STAT.PGMMP is cleared as well. At the end of execution, FLC_STAT.PGMMP bit is cleared and interrupt is generated if FLC_IMSK.PGMMC bit is set. Once in the program mode, any of the 64 double-words in the row can be programmed. (A row contains 512-bytes of data) To program locations in a different row, EPGMM command needs to be executed to exit the program mode followed by SPGMM command with ADR register set to the new row address.
		5	End Program Mode (EPGMM). Takes the flash controller out of the program mode. FLC_STAT.PGMMP bit is set to indicate that the exit program mode operation is in progress. After the operation is complete, FLC_STAT.PGMM bit is cleared to indicate that the flash controller exited the program mode; FLC_STAT.PGMMP is cleared as well. At the end of execution, FLC_STAT.PGMMP bit is cleared and interrupt is generated if FLC_IMSK.PGMMC bit is set.
		6	Read Command (RDC). Reads flash data into FLC_DATA0, FLC_DATA1 register from a location specified by the contents of FLC_ADDR. ECC check is performed on the read data and FLC_STAT.EERRMF bit in FLC_STAT is set in case of multi-bit ECC error. FLC_STAT.RDCP bit is set to indicate that a read operation is in progress. At the end of execution, FLC_STAT.RDCP bit is cleared.
		7	Sleep command (SLPC). Asserts the Sleep mode enable signal of flash memory to put it in sleep mode. FLC_STAT.SLP bit is set to indicate that the flash is in sleep mode.
		8	Wakeup from sleep (WKSL). Wakes the flash from sleep mode. After wake up delay defined in FLC_TIM_PWR register, FLC_STAT.SLP bit is cleared. FLC_STAT.WKUPP bit stays high while this command is being executed. At the end of execution, FLC_STAT.WKUPP bit is cleared and interrupt is generated if FLC_IMSK.WKUP bit is set.
		9	Reserved

Table 14-4: FLC_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		10	Reserved
		11	Reserved
		15	Abort the current command (ABORT). Aborts the current command being executed. This also discharges high voltage buildup if any. This clears `FLC_STAT.PGMM` bit along with other command pending bits. Results may be unpredictable. `FLC_STAT.ABTP` bit is set to indicate an abort operation in progress. At the end of execution, `FLC_STAT.ABTP` bit is cleared and interrupt is generated if `FLC_IMSK.ABORT` bit is set.

Command Key Register

The FLC_CMD_KEY register holds a key which is used to provide protection against accidental execution of all flash controller commands like page erase, mass erase, read, program, and other commands. A read of this register returns 0. A procedure to enter the command mode using this register outlined below.

- The FLC_CTL.CMDM bit is set.
- A 16-bit key (CMD_KEY1 == 0xF378) is written to this register.
- A 16-bit key (CMD_KEY2 == 0x9D6A) is written to this register.

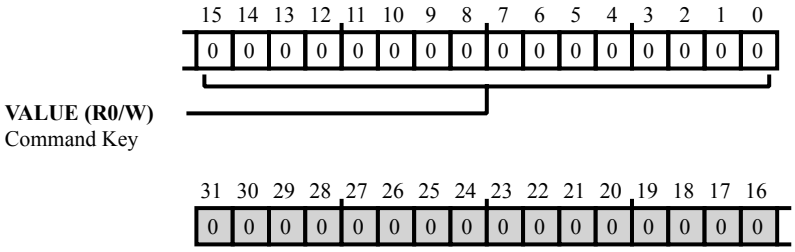


Figure 14-6: FLC_CMD_KEY Register Diagram

Table 14-5: FLC_CMD_KEY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Command Key. The FLC_CMD_KEY.VALUE bit field holds a key which is used to provide protection against accidental execution of all flash controller commands like page erase, mass erase, read, program, and other commands.

Control Register

The FLC_CTL register contains settings for the flash controller.

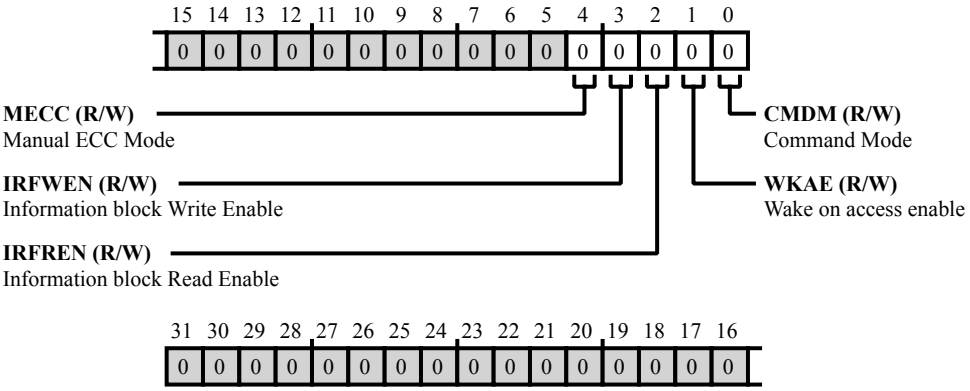


Figure 14-7: FLC_CTL Register Diagram

Table 14-6: FLC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	MECC	Manual ECC Mode.
3 (R/W)	IRFWEN	Information block Write Enable. The FLC_CTL.IRFWEN bit controls write/erase command access to the information block. If this bit = 0, then neither write, program, erase, or mass erase commands of the information block are allowed. (A mass erase operation erases only the main array). If the FLC_CTL.IRFWEN bit = 1, then these operations in the info block are allowed.
2 (R/W)	IRFREN	Information block Read Enable. The FLC_CTL.IRFREN bit controls read access to the information block. If this bit = 0, neither bus reads nor register-command reads to the info block are allowed. Bus reads return a bus error, register-command reads return 0. If this bit = 1, read accesses to the info block are allowed.
1 (R/W)	WKAE	Wake on access enable. If the FLC_CTL.WKAE bit = 1, access causes wakeup from sleep mode and stalls until ready; if 0 or in powered down state, access causes bus error. Wakeup can also be caused by power management unit and WKSL and WKPD commands.

Table 14-6: FLC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	CMDM	Command Mode. The `FLC_CTL.CMDM` bit should be set during program and erase operations. This bit, when set, disables all the traffic from flash pre-fetch unit and DMA port; accesses return ERROR on the respective bus. Asserting this signal automatically invalidates all hopper locations. The controller enters the command mode after setting this bit and writing 2 keys to the `FLC_CMD_KEY` register. This bit can not be cleared during program mode (`FLC_STAT.PGMM` is set) or controller is busy executing a command. A forceful way to exit the command mode is issuing ABORT command and then clear this bit, which may cause unpredictable results and is not recommended.

Command Data Register 0

The FLC_DATA0 register holds the least significant part of the data used by flash controller commands. If the controller is busy, a write to this register is ignored to avoid corrupting data for the command being executed.

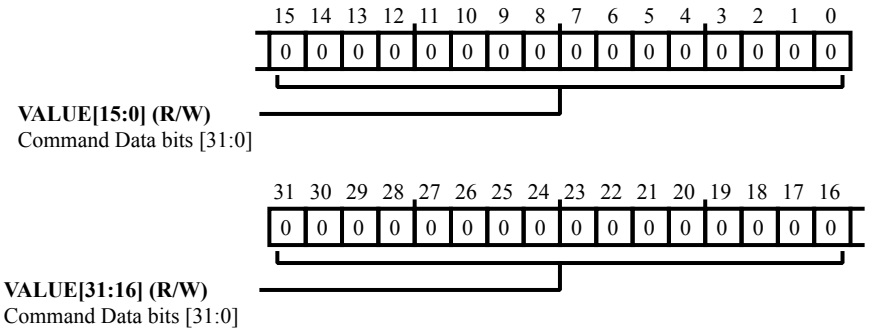


Figure 14-8: FLC_DATA0 Register Diagram

Table 14-7: FLC_DATA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Data bits [31:0]. The FLC_DATA0.VALUE bit field holds the least significant part of the data used by flash controller commands.

Command Data Register 1

The FLC_DATA1 register holds the most significant part of the data used by flash controller commands. If the controller is busy, a write to this register is ignored to avoid corrupting data for the command being executed.

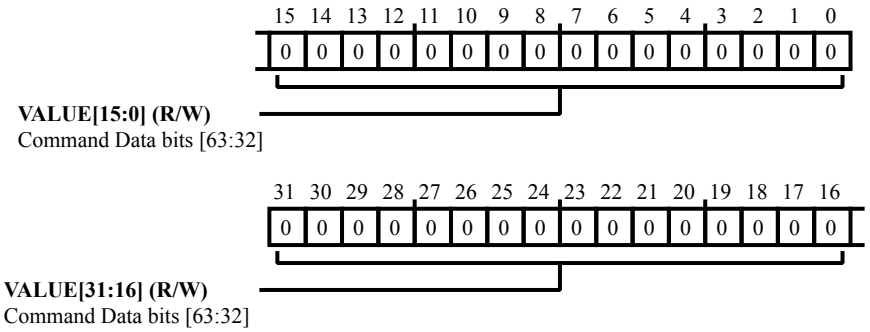


Figure 14-9: FLC_DATA1 Register Diagram

Table 14-8: FLC_DATA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Data bits [63:32]. The FLC_DATA1.VALUE bit field holds most significant part of the data used by flash controller commands.

Command ECC Register

Bits in the FLC_ECC register represent ECC bits for the data in FLC_DATA1 and FLC_DATA0. RDC command always updates this register from flash read data. If the controller is busy, a write to this register is ignored to avoid corrupting data for the command being executed.

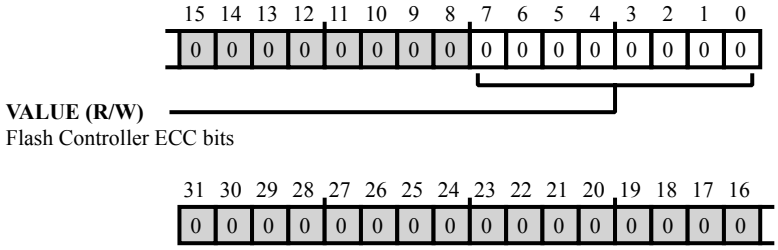


Figure 14-10: FLC_ECC Register Diagram

Table 14-9: FLC_ECC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Flash Controller ECC bits. The FLC_ECC.VALUE bit field represent ECC bits for the data in FLC_DATA1 and FLC_DATA0 registers.

Pre-fetcher Fill Count Register

The FLC_FILL_CNT register indicates the number of accesses made to the flash. If this counter overflows, FLC_FILL_CNT.OVF bit is set. This counter and the overflow bit are cleared when the FLC_PRFCTL.PEREN bit is set while it was in the cleared state.

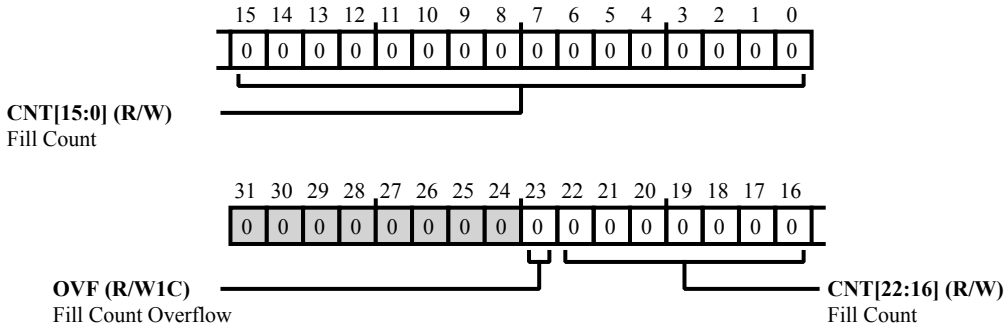


Figure 14-11: FLC_FILL_CNT Register Diagram

Table 14-10: FLC_FILL_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W1C)	OVF	Fill Count Overflow. The FLC_FILL_CNT.OVF bit is set if the counter overflows.
22:0 (R/W)	CNT	Fill Count. The FLC_FILL_CNT.CNT bit field indicates the number of accesses made to the flash.

Interrupt Enable Register

The FLC_IMSK register contains the interrupt enable bits.

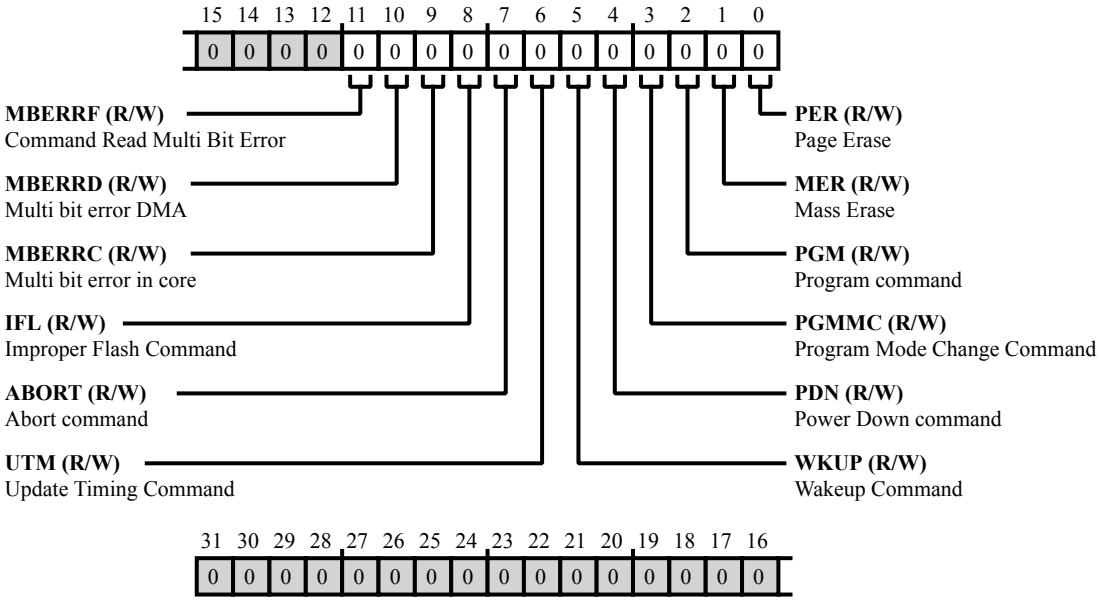


Figure 14-12: FLC_IMSK Register Diagram

Table 14-11: FLC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	MBERRF	Command Read Multi Bit Error. The FLC_IMSK.MBERRF bit enables an interrupt on a command read multi bit error.
10 (R/W)	MBERRD	Multi bit error DMA. The FLC_IMSK.MBERRD bit enables an interrupt on a DMA multi bit error.
9 (R/W)	MBERRC	Multi bit error in core. The FLC_IMSK.MBERRC bit enables an interrupt on a core multi bit error.
8 (R/W)	IFL	Improper Flash Command. The FLC_IMSK.IFL bit enables an interrupt on an improper flash command. These include: • A command was issued while flash was busy. • A command was issued while flash was in sleep or powered down mode. • A command was issued when the controller was not in command mode. • A command violated information block privileges. • A command violated program mode guidelines

Table 14-11: FLC_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	ABORT	Abort command. The `FLC_IMSK.ABORT` bit enables an interrupt on completion of the Abort command.
6 (R/W)	UTM	Update Timing Command. The `FLC_IMSK.UTM` bit enables an interrupt on completion of the Update Timing command.
5 (R/W)	WKUP	Wakeup Command. The `FLC_IMSK.WKUP` bit enables an interrupt on completion of the Wakeup command.
4 (R/W)	PDN	Power Down command. The `FLC_IMSK.PDN` bit enables an interrupt on completion of the Power down command.
3 (R/W)	PGMMC	Program Mode Change Command. The `FLC_IMSK.PGMMC` bit enables an interrupt on completion of the Program Mode Change command.
2 (R/W)	PGM	Program command. The `FLC_IMSK.PGM` bit enables an interrupt on completion of the Program command.
1 (R/W)	MER	Mass Erase. The `FLC_IMSK.MER` bit enables an interrupt on completion of the Mass Erase command.
0 (R/W)	PER	Page Erase. The `FLC_IMSK.PER` bit enables an interrupt on completion of the Page Erase command.

Pre-fetcher Miss Count Register

The FLC_MISS_CNT register holds the number pre-fetch misses after the FLC_PRFCTL.PEREN bit is set. If this counter overflows, the FLC_FILL_CNT.OVF bit is set. This counter and the overflow bit are cleared when the FLC_PRFCTL.PEREN bit is set while it was in the cleared state.

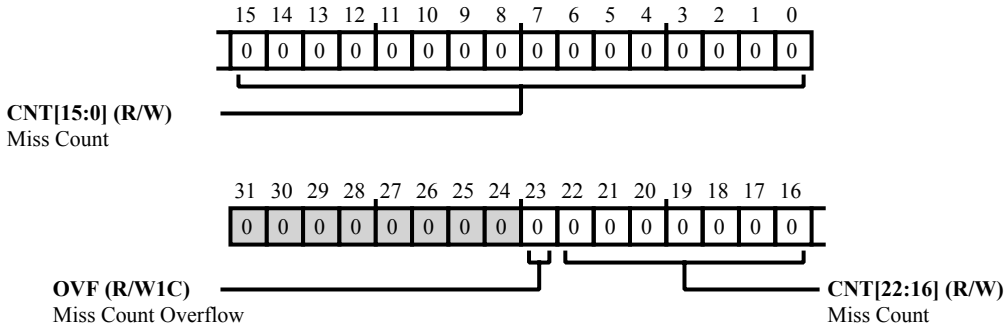


Figure 14-13: FLC_MISS_CNT Register Diagram

Table 14-12: FLC_MISS_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W1C)	OVF	Miss Count Overflow. The FLC_MISS_CNT.OVF bit field indicates a miss count overflow.
22:0 (R/W)	CNT	Miss Count. The FLC_MISS_CNT.CNT bit field indicates the number pre-fetch misses after the FLC_PRFCTL.PEREN bit is set.

Prefetch Control Register

The FLC_PRFCTL register controls pre-fetcher behavior.

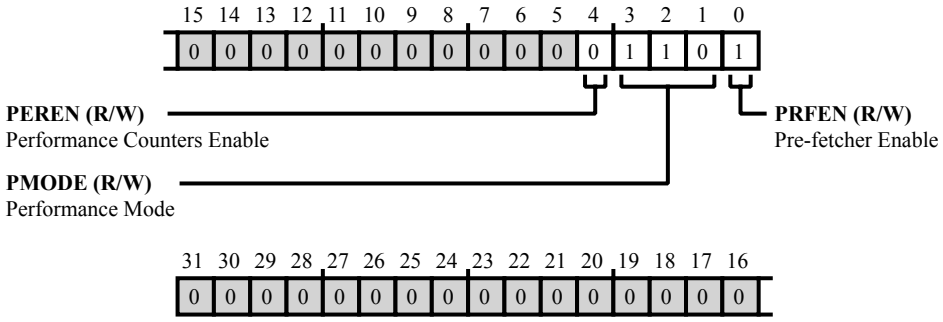


Figure 14-14: FLC_PRFCTL Register Diagram

Table 14-13: FLC_PRFCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	PEREN	Performance Counters Enable. The FLC_PRFCTL.PEREN bit enables the FLC_FILL_CNT, FLC_MISS_CNT and FLC_REF_CNT counters. Setting this bit in the cleared state clears these counters and the associated overflow bits	
3:1 (R/W)	PMODE	Performance Mode. The FLC_PRFCTL.PMODE bit field controls the Mode Frequency range.	
		0	Performance Mode A is <= 75 MHz
		2	Performance Mode B is 76 to 150 MHz
		4	Performance Mode C is 151 to 225 MHz
		6	Performance Mode E is 226 to 240 MHz
0 (R/W)	PRFEN	Pre-fetcher Enable. When the FLC_PRFCTL.PRFEN bit is set the Pre-fetcher fetches ahead and performs branch detection. When cleared, pre-fetching is disabled, although the internal cache is used to feed ARM ICODE and DCODE accesses. Clearing this bit invalidates all cache locations except the one used for DMA, which is cleared when the FLC_CTL.CMDM bits in FLC_CTL change. This bit should be cleared before changing FLC_PRFCTL.PMODE bits.	

Pre-fetcher Reference Count Register

The FLC_REF_CNT register holds the number of hoppers filled and used after the FLC_PRFCTL.PEREN bit is set. If this counter overflows, the FLC_FILL_CNT.OVF bit is set. This counter and the overflow bit are cleared when the FLC_PRFCTL.PEREN bit is set while it was in the cleared state.

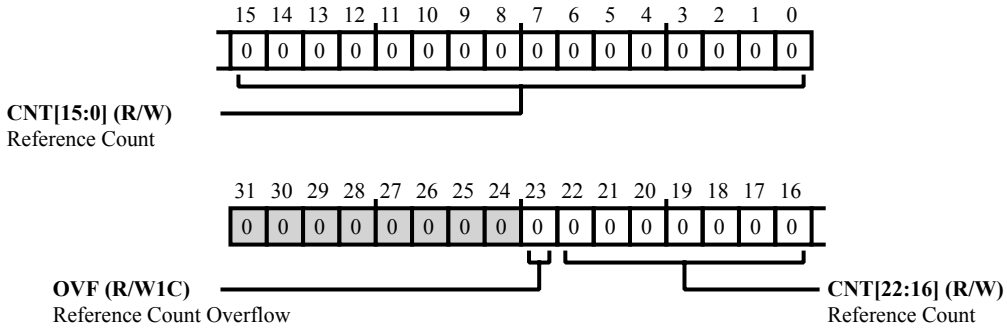


Figure 14-15: FLC_REF_CNT Register Diagram

Table 14-14: FLC_REF_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W1C)	OVF	Reference Count Overflow. The FLC_REF_CNT.OVF bit field indicates a reference count overflow.
22:0 (R/W)	CNT	Reference Count. The FLC_REF_CNT.CNT bit field holds the number of hoppers filled and used after the FLC_PRFCTL.PEREN bit is set.

Revision ID Register

The FLC_REVID register contains the major and minor revision block IDs.

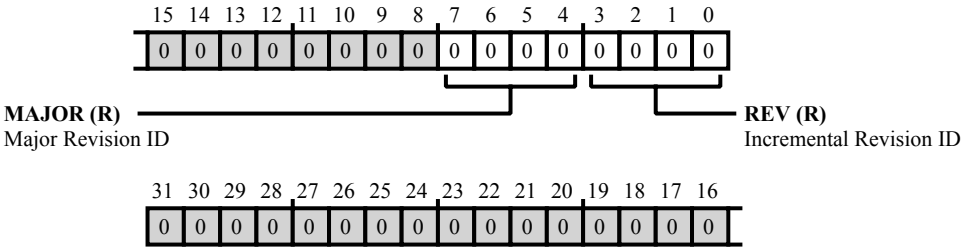


Figure 14-16: FLC_REVID Register Diagram

Table 14-15: FLC_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Revision ID.
3:0 (R/NW)	REV	Incremental Revision ID. Block revision ID

ECC Single Bit Error Threshold Register

The FLC_SBERR_THR register holds an ECC single-bit-error count threshold. When the number of single bit error corrections observed during all the flash reads exceeds the value in this register, FLC_STAT.EERRSRT bit is set. The FLC_STAT.EERRSRT bit can be cleared by writing 1 to it which also restarts the error counter. The counter is incremented each time a bit error is read from any word. In this case, reading the same word several times causes the threshold to exceed.

Note that the single bit errors are corrected by an in-built ECC. Even if the register value exceeds, the application runs correctly as ECC corrects all single bit errors.

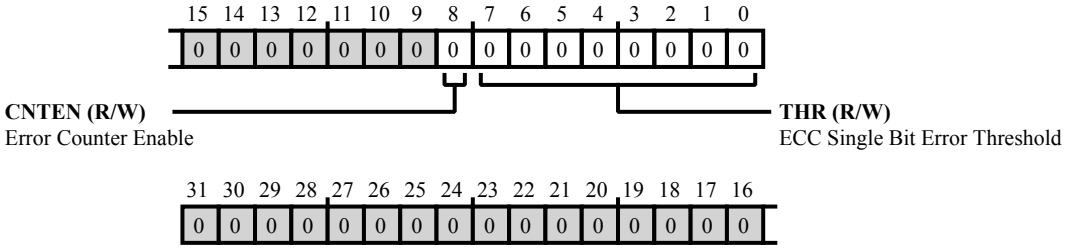


Figure 14-17: FLC_SBERR_THR Register Diagram

Table 14-16: FLC_SBERR_THR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W)	CNTEN	Error Counter Enable. The FLC_SBERR_THR.CNTEN bit enables the ECC single bit error counter.
7:0 (R/W)	THR	ECC Single Bit Error Threshold. The FLC_SBERR_THR.THR bit field holds an ECC single-bit-error count threshold.

Status Register

The FLC_STAT register holds status bits to convey information generated during the flash controller, pre-fetcher and DMA operations.

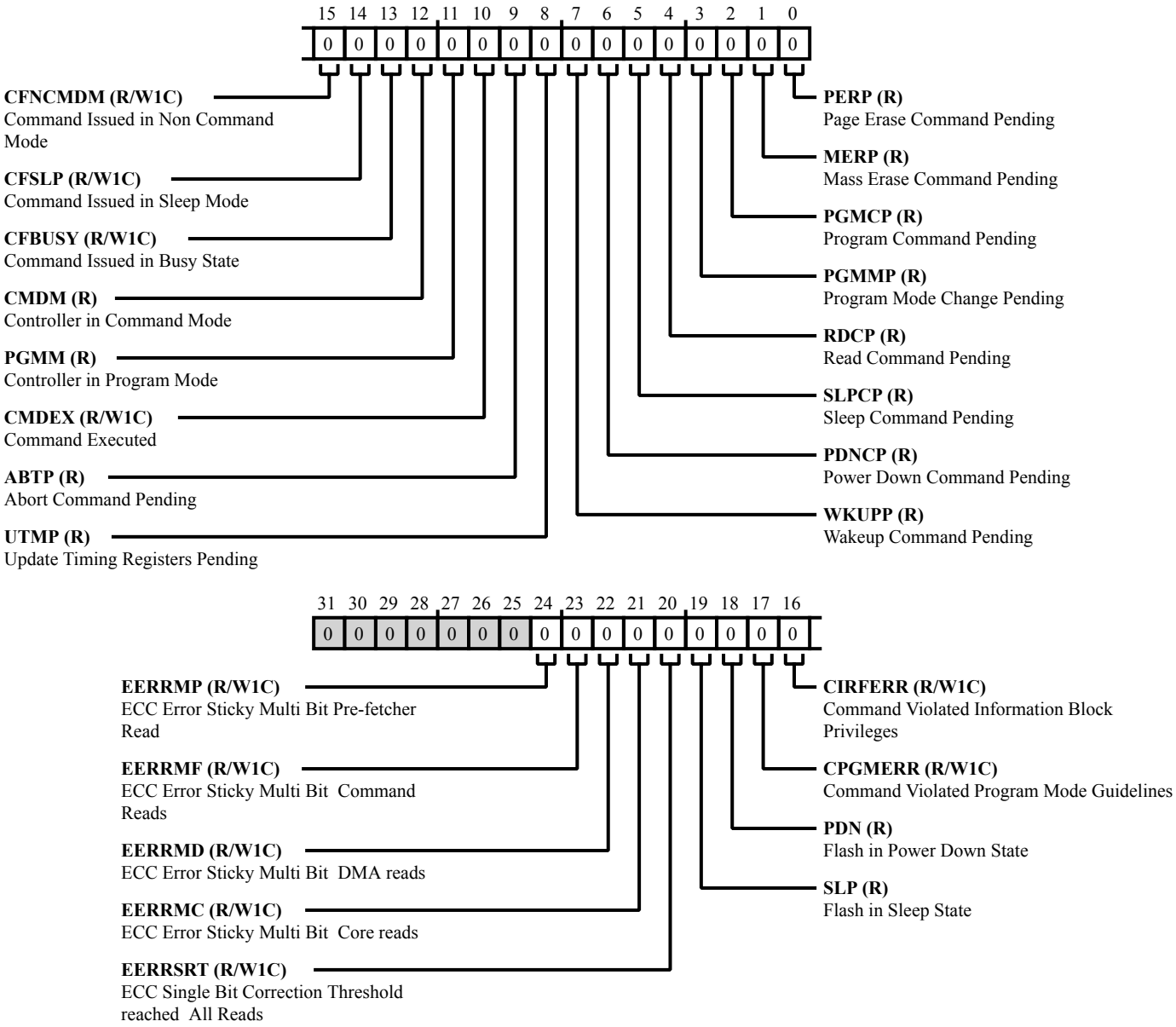


Figure 14-18: FLC_STAT Register Diagram

Table 14-17: FLC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W1C)	EERRMP	ECC Error Sticky Multi Bit Pre-fetcher Read. The FLC_STAT.EERRMP bit is set when a multi-bit ECC error occurred during a pre-fetcher read operation. The read data may get stored into a hopper (cache line) without ECC correction. If this data is read by core, an error response is sent on the bus. FLC_ADDR is updated with the error address. This bit is sticky and can be cleared by writing 1 to it. Clearing this bit also invalidates hoppers (cache lines).
23 (R/W1C)	EERRMF	ECC Error Sticky Multi Bit Command Reads. When set by hardware, the FLC_STAT.EERRMF bit indicates that a multi-bit error occurred during flash controller command reads. This bit is sticky and can be cleared by writing 1 to it. FLC_ADDR contains the error address of the last multi-bit error. An interrupt is generated if FLC_IMSK.MBERRF bit is set.
22 (R/W1C)	EERRMD	ECC Error Sticky Multi Bit DMA reads. When set by hardware, the FLC_STAT.EERRMD bit indicates that a multi-bit error occurred during DMA reads. FLC_ADDR contains the error address of the last multi-bit error. An interrupt is generated if FLC_IMSK.MBERRD bit is set. This bit is sticky and can be cleared by writing 1 to it. Clearing this bit also invalidates DMA cache line (hopper)
21 (R/W1C)	EERRMC	ECC Error Sticky Multi Bit Core reads. When set by hardware, the FLC_STAT.EERRMC bit indicates that a multi-bit error occurred during core reads. FLC_ADDR contains the error address of the last multi-bit error. An interrupt is generated if FLC_IMSK.MBERRC bit is set. This bit is sticky and can be cleared by writing 1 to it. Clearing this bit also invalidates pre-fetcher hoppers (cache lines).
20 (R/W1C)	EERRSRT	ECC Single Bit Correction Threshold reached All Reads. When set by hardware, the FLC_STAT.EERRSRT bit indicates that a single bit error threshold was reached. This bit is sticky and can be cleared by writing 1 to it which also restarts the error counter.
19 (R/NW)	SLP	Flash in Sleep State. When set by hardware, the FLC_STAT.SLP bit indicates that the flash is in sleep mode.
18 (R/NW)	PDN	Flash in Power Down State. When set by hardware, the FLC_STAT.PDN bit indicates that the flash is powered down.

Table 14-17: FLC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W1C)	CPGMERR	Command Violated Program Mode Guidelines. When set by hardware, the FLC_STAT.CPGMERR bit indicates that a command violated command mode guidelines. This bit is set when PGMC or EPGMM is issued when FLC_STAT.CMDM is LOW. It is also set when any command other than PGMC, EPGMM or ABORT is issued when FLC_STAT.CMDM is HIGH. It is sticky and can be cleared by writing 1 to it. An interrupt is generated if FLC_IMSK.IFL bit is set.
16 (R/W1C)	CIRFERR	Command Violated Information Block Privileges. When set by hardware, the FLC_STAT.CIRFERR bit indicates that a command violated information block privileges. This bit is sticky and can be cleared by writing 1 to it. An interrupt is generated if FLC_IMSK.IFL bit is set.
15 (R/W1C)	CFNCMDM	Command Issued in Non Command Mode. When set by hardware, the FLC_STAT.CFNCMDM bit indicates that a command was issued while flash was not in command mode. This bit is sticky and can be cleared by writing 1 to it. An interrupt is generated if FLC_IMSK.IFL bit is set.
14 (R/W1C)	CFSLP	Command Issued in Sleep Mode. When set by hardware, the FLC_STAT.CFSLP bit indicates that a command was issued while flash was in sleep or powered down mode. This bit is sticky and can be cleared by writing 1 to it. An interrupt is generated if FLC_IMSK.IFL bit is set. Commands allowed in sleep and powered down modes are ABORT, SLPC, WKSL, PDNC and WKPD.
13 (R/W1C)	CFBUSY	Command Issued in Busy State. When set by hardware, the FLC_STAT.CFBUSY bit indicates that a command was issued while flash was busy executing other command. This bit is sticky and can be cleared by writing 1 to it. An interrupt is generated if FLC_IMSK.IFL bit is set.
12 (R/NW)	CMDM	Controller in Command Mode. The FLC_STAT.CMDM bit indicates that the flash controller is in the command mode.
11 (R/NW)	PGMM	Controller in Program Mode. The FLC_STAT.PGMM bit indicates that the flash controller is in the program mode.
10 (R/W1C)	CMDEX	Command Executed. The FLC_STAT.CMDEX bit is set when a controller command is executed. An interrupt is generated if an interrupt enable bit corresponding to the executed command is set in FLC_IMSK. It is sticky and can be cleared by writing 1 to it.

Table 14-17: FLC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
9 (R/NW)	ABTP	Abort Command Pending. When set by hardware, the `FLC_STAT.ABTP` bit indicates that an abort command is being executed. A new command should not be issued until this bit is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.
8 (R/NW)	UTMP	Update Timing Registers Pending. When set by hardware, the `FLC_STAT.UTMP` bit indicates that timing registers are being updated from core clock to aux clock domain. A new command should not be issued until this bit is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.
7 (R/NW)	WKUPP	Wakeup Command Pending. When set by hardware, the `FLC_STAT.WKUPP` bit indicates that a wakeup sequence is being executed. A new command should not be issued until this bit is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.
6 (R/NW)	PDNCP	Power Down Command Pending. When set by hardware, the `FLC_STAT.PDNCP` bit indicates that a power down command is being executed. A new command should not be issued until this bit is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.
5 (R/NW)	SLPCP	Sleep Command Pending. When set by hardware, the `FLC_STAT.SLPCP` bit indicates that a sleep command is being executed. A new command should not be issued until this bit is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.
4 (R/NW)	RDCP	Read Command Pending. When set by hardware, the `FLC_STAT.RDCP` bit indicates that a read command is being executed. A new command should not be issued until this is cleared. This command takes cycles equal to flash wait states plus 1 to complete.
3 (R/NW)	PGMMP	Program Mode Change Pending. When set by hardware, the `FLC_STAT.PGMMP` bit indicates that a program start or a program end event is in progress, initiated by executing SPRGM or EPRGM commands. A new command should not be issued until this status bit is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.

Table 14-17: FLC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/NW)	PGMCP	Program Command Pending. When set by hardware, the `FLC_STAT.PGMCP` bit indicates that a program command is being executed. A new command should not be issued until this is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.
1 (R/NW)	MERP	Mass Erase Command Pending. When set by hardware, the `FLC_STAT.MERP` bit indicates that a mass erase operation is in progress. A new command should not be issued until this bit is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.
0 (R/NW)	PERP	Page Erase Command Pending. When set by hardware, the `FLC_STAT.PERP` bit indicates that a page erase operation is in progress. A new command should not be issued until this bit is cleared. An optional interrupt can be generated when this bit is being cleared by the hardware at the end of the operation.

Power Timing Register

The FLC_TIM_PWR register specifies number of core clock cycles required to meet certain power management timing specs.

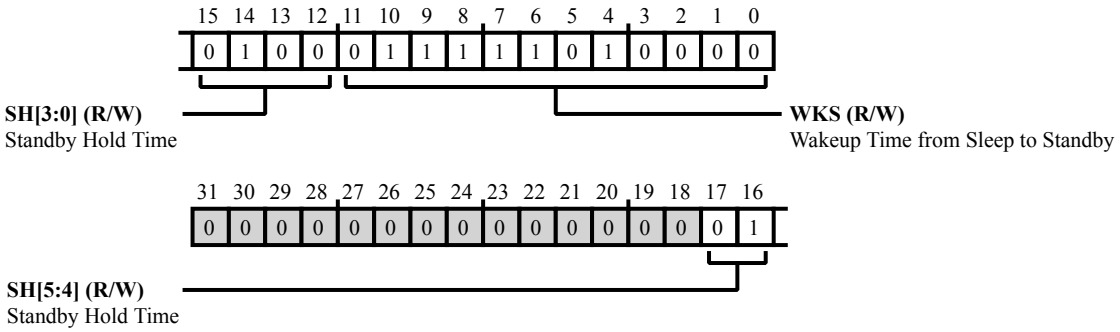


Figure 14-19: FLC_TIM_PWR Register Diagram

Table 14-18: FLC_TIM_PWR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17:12 (R/W)	SH	Standby Hold Time. The FLC_TIM_PWR.SH bit field specifies the standby hold time (100ns min). The same value used for PDM33 setup and hold time. Default represents min value at 200MHz.
11:0 (R/W)	WKS	Wakeup Time from Sleep to Standby. The FLC_TIM_PWR.WKS bit field specifies the wakeup time from sleep to standby (7s min). The same value used for power-down to standby wakeup time WKPD. Default represents min value at 200 MHz.

15 System Memory Protection Unit (SMPU)

The SMPU provides a flexible way of protecting memory regions against read or write access from any or all masters in the system. In addition, it can guard against memory access depending on security privileges of the system master.

SMPU Features

The system memory protection unit has the following features.

- After reset, the default state of the system is fully open. The SMPUs admit any access to memory spaces by any master.
- Each SMPU instance can be configured to monitor multiple regions. Each can be individually enabled.
- Each region can be configured with its own protection settings.
- Provides general read or write protection.
- Read and write transactions are restricted or allowed depending on the transaction ID.

On the ADSP-CM41x, an SPMU is provided for each ARM Cortex core's main memory and for external memory:

- SMPU0 protects the ARM Cortex-M0 System memory's system (DMA) port.
- SMPU1 protects the ARM Cortex-M4 Main SRAM memory's system (DMA) port.
- SMPU2 protects the external memory port managed by the SMC.

SMPU Functional Description

The following sections provide details on the function of the SMPU module. If the region security settings allow transactions to go through, the ID in the ID-based region protection settings can still filter the transactions.

For the memory that an SMPU protects, programs can configure region-based settings with the `SMPU_RCTL[n]` registers. (There can be multiple SMPUs in a system.) The `SMPU_RCTL[n]` registers define the ID-based protection for memory regions.

If the target address does not reside in any configured memory region, the transaction permission resorts back to the global configuration setting.

Protection Units

Each SMPU provides two protection units, A and B for ID-based matching in the region-based memory protection. This feature provides a degree of flexibility for the user to match against multiple IDs.

Instruction Fetches

When the core executes instructions from memory, this operation is also considered a memory transaction. If the SMPU is configured to protect a memory region from read accesses that contain instructions, the core cannot fetch and execute these instructions.

Speculative Reads

If speculative reads are enabled (`SMPU_CTL.RSDIS` =0), the SMPU forwards the read transaction directly to the memory before checking the protection setting corresponding to the addressed memory region. This functionality saves one clock cycle in the clock domain of the SMPU. The SMPU checks the protection setting while the read transaction occurs with the memory. If the protection setting dictates that the target memory address is blocked, the SMPU blocks the read to the master.

If speculative reads are disabled (`SMPU_CTL.RSDIS` =1), the SMPU checks the protection settings first and forwards the transaction to memory only if it passes the configured protection settings. This functionality incurs a one-cycle latency per read.

NOTE: Reads affect certain memory operations such as automatic clearing of the memory (that is, FIFOs). When the SMPU protects this type of memory, disable read speculation since the blocking can occur without the read transaction reaching the target memory.

CM41X_M4 SMPU Register List

The System Memory Protection Unit (SMPU) provides selective protection of the processor's memory resources. The SMPU includes a set of processor events that can be monitored during program execution. A set of registers governs SMPU operations. For more information on SMPU functionality, see the SMPU register descriptions.

Table 15-1: CM41X_M4 SMPU Register List

Name	Description
`SMPU_BADDR`	Bus Error Address Register
`SMPU_BDTLS`	Bus Error Details Register
`SMPU_CTL`	SMPU Control Register
`SMPU_IADDR`	Interrupt Address Register
`SMPU_IDTLS`	Interrupt Details Register
`SMPU_RADDR[n]`	Region n Address Register
`SMPU_RCTL[n]`	Region n Control Register

Table 15-1: CM41X_M4 SMPU Register List (Continued)

Name	Description
SMPU_REVID	SMPU Revision ID Register
SMPU_RIDA[n]	Region n ID A Register
SMPU_RIDB[n]	Region n ID B Register
SMPU_RIDMSKA[n]	Region n ID Mask A Register
SMPU_RIDMSKB[n]	Region n ID Mask B Register
SMPU_STAT	SMPU Status Register

CM41X_M0 SMPU Interrupt List

Table 15-2: CM41X_M0 SMPU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
8	SMPU0_ERR	SMPU0 Error Interrupt	Level	

Memory Writes

A write transaction to address n is prevented when the following is true.

Address n is in memory region m and memory region m is write-protected (SMPU_RCTL[n].WPROTEN =1) and ID is not a match. (See ID Comparison). The block occurs because the memory region is configured for write-protection and the ID comparison does not result in a match. If an ID comparison results in a match, the write transaction is allowed through.

Memory Reads

A read transaction from address n is prevented when the following is true:

- Address n is in memory region and memory region m is read-protected (SMPU_RCTL[n].RPROTEN =1) and
- ID is not a match

The block occurs because the memory region is configured for read-protection and the ID comparison does not result in a match. If the ID comparison results in a match, the read transaction is permitted. (See ID Comparison).

ID Comparison

ID comparison automatically occurs during region-based memory protection. ID matches allow the transaction to bypass the configured memory protection for that region. The following sections describe the calculation of a write ID match and read ID match.

Write Transaction

The state of the following values determines the ID value that is compared with the ID of an incoming write transaction:

- The SMPU_RCTL[n].WIDCINV bit
- The SMPU_RIDA[n].ID and SMPU_RIDB[n].ID bit fields
- The SMPU_RIDMSKA[n].MSK and SMPU_RIDMSKB[n].MSK bit fields

Write IDA match = ((ID of incoming write transaction AND SMPU_RIDMSKA[n].MSK) == (SMPU_RIDA[n].ID AND SMPU_RIDMSKA[n].MSK))

Write IDB match = ((ID of incoming write transaction AND SMPU_RIDMSKB[n].MSK) == (SMPU_RIDB[n].ID AND SMPU_RIDMSKB[n].MSK))

Write ID match = (Write IDA match OR Write IDB match) XOR SMPU_RCTL[n].WIDCINV bit

Read Transaction

The state of the following values determines the ID value that is compared with the ID of an incoming read transaction:

- The SMPU_RCTL[n].RIDCINV bit
- The SMPU_RIDA[n].ID and SMPU_RIDB[n].ID bit fields
- The SMPU_RIDMSKA[n].MSK and SMPU_RIDMSKB[n].MSK bit fields

Read IDA match = ((ID of incoming read transaction AND SMPU_RIDMSKA[n].MSK) == (SMPU_RIDA[n].ID AND SMPU_RIDMSKA[n].MSK))

Read IDB match = ((ID of incoming read transaction AND SMPU_RIDMSKB[n].MSK) == (SMPU_RIDB[n].ID AND SMPU_RIDMSKB[n].MSK))

Read ID match = (Read IDA match OR Read IDB match) XOR SMPU_RCTL[n].RIDCINV

In the two cases described above, the incoming transaction (either write or read) ID is AND'ed with the configured mask value in protection unit A. It is then compared to the value of the configured ID value which is also AND'ed with the configured mask value in protection unit A. The mask provides a method to allow a group of IDs to match. This process is also performed for protection unit B. The two outcomes (from A and B) are then OR'ed together.

Depending on the setting of the SMPU_RCTL[n].RIDCINV or the SMPU_RCTL[n].WIDCINV bits, the ID match comparison is inverted or not. The final result after applying the inversion, SMPU_RCTL[n].RIDCINV, or SMPU_RCTL[n].WIDCINV, determines whether the transaction bypasses the protection.

Usage

The masks, SMPU_RIDMSKA[n] and SMPU_RIDMSKB[n], are AND'ed with both the incoming transaction ID and the configured ID in SMPU_RIDA[n].ID and SMPU_RIDB[n].ID, respectively. By default the masks

are zero. If ID-based region protection is enabled by setting the SMPU_RCTL[n].WPROTEN or SMPU_RCTL[n].RPROTEN bit fields and the masks are not set, the ID comparison essentially compares zeros. The comparison allows all transactions to bypass (if the region-based security setting is also configured in a way to allow transactions to go through for the region). To have the ID-based region protection to function, the mask registers and ID registers must also be set.

System IDs

The ***System Master IDs*** table provides the IDs for the system masters. An x means that the bit can be a 0 or a 1. There are multiple IDs associated with that particular system master.

The table headings are defined in the list below. The A, B, and C characters have the meanings described in the table.

A	From DMA engine
B	4 bottom bits of 12-bit CONT_MST field
C	All 9 bits of SUPER_SLV field

- MASTER: CONT_MST is the M4 controller as memory access master and controls load/stores by the M4 core.
- MASTER: SUPER_MST is the M0 supervisor subsystem. This acts as the memory access master into the main system fabric and includes accesses by the M0 core as well as by DMA channels DMA0 through DMA3. The bits 'BBBB' indicate the master ID within the M0 subsystem, from the ***System Master IDs for SMPU0*** table.
- MASTER: SUPER_SLV is the M0 supervisor subsystem's slave port from the M4 controller subsystem fabric. The bits 'CCCCCCCCC' indicate the master ID within the M4 subsystem, from the ***System Master IDs for SMPU2 and SMPU1*** table.
- SLAVE: CONT_SLV is the M4 controller slave memory spaces which are M4 SRAM and flash.
- SLAVE: SUPER_SRAM is the M0 supervisor slave memory space. M0 SRAM is the only such space.

Table 15-3: System Master IDs for SMPU2 and SMPU1

MASTERS	SLAVES	
Master	**SMPU2 (SMC)**	**SMPU1 (CONT_SLV)**
DMA4: uart1_tx	9b00000A000	9b00000A000
DMA5: uart1_rx	9b00000A001	9b00000A001
DMA6: uart2_tx, spi1_tx	9b00000A010	9b00000A010
DMA7: uart2_rx, spi1_rx	9b00000A011	9b00000A011
DMA8: uart3_tx/hae_in0	9b00000A100	9b00000A100
DMA9: uart3_rx/hae_out	9b00000A101	9b00000A101

Table 15-3: System Master IDs for SMPU2 and SMPU1 (Continued)

MASTERS	SLAVES	
Master	**SMPU2 (SMC)**	**SMPU1 (CONT_SLV)**
DMA10: uart4_tx/sport0a/hae_in1	9b00000A110	9b00000A110
DMA11: uart4_rx/sport0b	9b00000A111	9b00000A111
DMA12: mdma0_rd	9b01000A000	9b01000A000
DMA13: mdma0_wr	9b01000A001	9b01000A001
SINC	9b10000A000	9b10000A000
AFE1: adcc1+dacc0	9b10000A001	9b10000A001
FFT	9b10000A010	9b10000A010
CONT_MST	9b110000011	--N/A--
SUPER_MST	9b11BBBB100	9b11BBBB100

Table 15-4: System Master IDs for SMPU0

Master	SMPU0 (SUPER_SRAM)
DMA0: spi0_tx	12b00000000A000
DMA1: spi0_rx	12b00000000A001
DMA2: uart0_tx	12b00000000A010
DMA3: uart0_rx	12b00000000A011
AFE0: adcc0	12b00000000A100
SUPER_SLV	12bCCCCCCCCC101
M0_CORE	N/A

Memory Region

Memory regions can start at address 0x00000000 or at any address that is a multiple of its size. The ***Supported Memory Region Size and Alignment*** table shows the memory region sizes that the processor supports and the alignment of the memory region. (X values are do-not-care).

SMPU0 and SMPU2 support a maximum of four regions. SMPU1 supports a maximum of eight regions.

Table 15-5: Supported Memory Region Size and Alignment

Size	SMPU_RCTLn.SIZE	Address	Possible Values for N
4KB	0b00000	0xXXXXX000	-
8KB	0b00001	0xXXXXN000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
16KB	0b00010	0xXXXXN000	0x0, 0x4, 0x8, 0xC

Table 15-5: Supported Memory Region Size and Alignment (Continued)

Size	SMPU_RCTLn.SIZE	Address	Possible Values for N
32KB	0b00011	0xXXXXN000	0x0, 0x8
64KB	0b00100	0xXXXX0000	-
128KB	0b00101	0xXXXN0000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
256KB	0b00110	0xXXXN0000	0x0, 0x4, 0x8, 0xC
512KB	0b00111	0xXXXN0000	0x0, 0x8
1MB	0b01000	0xXXX00000	-
2MB	0b01001	0xXXN00000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
4MB	0b01010	0xXXN00000	0x0, 0x4, 0x8, 0xC
8MB	0b01011	0xXXN00000	0x0, 0x8
16MB	0b01100	0xXX000000	-
32MB	0b01101	0xXN000000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
64MB	0b01110	0xXN000000	0x0, 0x4, 0x8, 0xC
128MB	0b01111	0xXN000000	0x0, 0x8
256MB	0b10000	0xX0000000	-
512MB	0b10001	0xN0000000	0x0, 0x2, 0x4, 0x8, 0xA, 0xC, 0xE
1GB	0b10010	0xN0000000	0x0, 0x4, 0x8, 0xC
2GB	0b10011	0xN0000000	0x0, 0x8
4GB	0b10100	0x00000000	-

For the case where the region size is selected as 4 GB, the region address must be at address 0x00000000.

NOTE: If a memory region address is not aligned to its size, the memory region start address protected by the SMPU is the configured address with the corresponding least significant bits masked. For example, if the size is configured for 16 KB (`SMPU_RCTL[n].SIZE` =0b00010), and the base address is configured for `SMPU_RADDR[n].BADDR`=0x00005018, the actual base address used by the SMPU is 0x00004000. When `SMPU_RADDR[n].BADDR` is read back, the program reads 0x00005000. This functionality is because only bits [11:0] are reserved as 0's. Programs must use care when setting the base address as it is not always the true base address.

SMPU Definitions

To make the best use of the SMPU, it is useful to understand the terms in this section.

Global Protection

Guarding of the entire memory space for the particular SMPU instantiation.

Region-Based Protection

Guarding individual segments of memory inside the memory space for the particular SMPU instantiation.

ID Match

A successful comparison of the ID associated with the incoming transaction and the ID and MASK configured in the SMPU.

SMPU Block Diagram

The ***SMPU Top-Level Block Diagram*** shows the SMPU block.

As seen in the diagram, the SMPU sits between the memory port (SCB master port) and the SCB fabric (SCB slave port). It acts as a gateway analyzing the transaction requests. It either rejects the transaction request or allows access based on the user-programmed configuration of the SMPU.

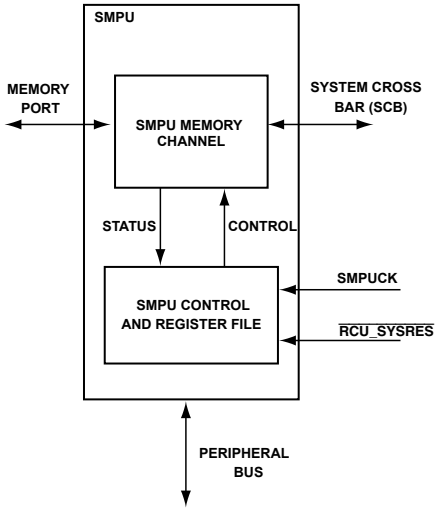


Figure 15-1: SMPU Top-Level Block Diagram

SMPU Architectural Concepts

The following sections provide brief descriptions of the architecture of the SMPU module.

Default Setting

At reset, the default state of the system is fully open. The SMPUs admit any access to memory spaces.

Latency

The SMPU adds latency to all the transactions to the memory except reads when read speculation is enabled (`SMPU_CTL.RSDIS` =0). In this case, read accesses are always forwarded to the memory and read responses are generated according to the SMPU settings. If read speculation is disabled (`SMPU_CTL.RSDIS` =1), reads are

blocked if they cause a security or protection violation. The SMPU generates the SCB read response that corresponds to a blocked transaction.

If read speculation is enabled, the SMPU adds 1 clock cycle latency to the read transaction. If read speculation is disabled, the SMPU adds 2 clock cycles latency to the read transaction.

SMPU Operating Modes

The SMPU does not have any strict modes of operation. However, it can be configured for region-based protection where a master with a particular ID can be blocked or allowed based on settings in the SMPU_RCTL[n] register.

Region-based protection is programmed with registers:

- SMPU_RCTL[n]
- SMPU_RADDR[n]
- SMPU_RIDA[n]
- SMPU_RIDMSKA[n]
- SMPU_RIDB[n]
- SMPU_RIDMSKB[n]

SMPU Interrupt Signals

There is one interrupt signal associated with the SMPU. If interrupts are enabled, the SMPU_STAT.IRQ bit is set. The SMPU_IRQ signal is asserted when the SMPU detects a memory access violation. The target address triggering the interrupt is found in the SMPU_IADDR register. The SMPU_IDTLS register provides further details about the cause of the interrupt.

Write errors are prioritized over read errors.

Protection violations (an ID-based violation) can trigger the SMPU interrupt, and can be enabled independently. The protection violation interrupt is enabled by setting the SMPU_CTL.PINTEN bit.

The SMPU interrupt is asserted for any of the following conditions:

If a second memory access violation occurs while the SMPU_STAT.IRQ bit is set, the SMPU_STAT.IOVR (interrupt overrun) bit is set. The SMPU_IADDR and the SMPU_IDTLS registers are not updated until the SMPU_STAT.IRQ bit is cleared. Any information on the subsequent interrupt is lost. Once the SMPU_STAT.IRQ bit and the SMPU_STAT.IOVR bit are cleared, any new memory access violations can trigger an interrupt and its details can be captured.

NOTE: When a blocked access occurs, the SMPU triggers an interrupt when interrupt generation is enabled. The SMPU can also be configured to generate a bus error that propagates back to the system master. The system master can also trigger an interrupt due to this bus error.

SMPU Status and Error Signals

If bus errors are enabled (SMPU_CTL.PBEDIS =0), the SMPU generates and returns a bus error to the master initiating the blocked access. This bit also sets the SMPU_STAT.BERR bit. The SMPU_BADDR and SMPU_BDTLS registers can be read to get the address and details of the transaction that caused the SMPU to generate the error.

Write errors are prioritized over read errors.

A bus error status is returned to the system master if:

- an ID-based violation happened and the SMPU_CTL.PBEDIS bit =0

If a second memory access violation occurs while the SMPU_STAT.BERR bit is set, the SMPU_STAT.BEOVR bit (bus error overrun) is set. The SMPU_BADDR and the SMPU_BDTLS registers are not updated until the SMPU_STAT.IRQ bit is cleared. The information about the transaction that caused the SMPU_STAT.BEOVR bit to be set is lost.

NOTE: If both the protection violation interrupt is not enabled (SMPU_CTL.PINTEN =0) and the protection bus error is disabled (SMPU_CTL.PBEDIS =1), the SMPU blocks invalid transactions. However, it does not provide any status or interrupt information indicating that a transaction is blocked.

SMPU Programming Example

The following example corresponds to using SMPU1 to protect M4 SRAM from MDMA write accesses. In the example the region configured for protection is 4KB at the start of M4 SRAM Bank A.

1. Enable SMPU1 interrupts by writing to the SMPU_CTL register.
2. Define the memory region to be protected by writing to the SMPU_RADDR[n] register.

```
*pREG_SMPU_RADDR0 = 0x20000000;
```

3. Program the SMPU_RIDA[n] and SMPU_RIDB[n] registers with the ID which is to be an exception to the rule.

```
*pREG_SMPU_RIDA0 = 0x0081; //corresponds to MDMA_WR
*pREG_SMPU_RIDB0 = 0x0081;
```

4. Program the SMPU_RIDMSKA[n] andSMPU_RIDMSKB[n] registers with the mask to be applied to the ID values programmed in the SMPU_RIDA[n] and SMPU_RIDB[n] registers respectively.

```
*pREG_SMPU_RIDMSKA0 = 0xFFFF;
*pREG_SMPU_RIDMSKB0 = 0xFFFF;
```

5. Program the Region Control register (SMPU_RCTL[n] register) with the required fields. Control fields like ID invert, write protection enable, read protection enable and memory region size can be configured in this register.

```
*pREG_SMPU_RCTL0 = ENUM_SMPU_RCTL_WPROTEN|ENUM_SMPU_RCTL_RPROTDIS|
BITM_SMPU_RCTL_WIDCINV|BITM_SMPU_RCTL_RIDCINV| ((0<<BITP_SMPU_RCTL_SIZE) &
BITM_SMPU_RCTL_SIZE)
```

6. Set the `SMPU_RCTL[n].EN` bit.

```
*pREG_SMPU_RCTL0  |= BITM_SMPU_RCTL_EN;
```

Any accesses violation to the protection enabled by this procedure will be blocked. If the corresponding interrupt is enabled, an interrupt is generated on protection violations.

CM41X_M4 SMPU Register Descriptions

The System Memory Protection Unit (SMPU) contains the following registers.

Table 15-6: CM41X_M4 SMPU Register List

Name	Description
SMPU_BADDR	Bus Error Address Register
SMPU_BDTLS	Bus Error Details Register
SMPU_CTL	SMPU Control Register
SMPU_IADDR	Interrupt Address Register
SMPU_IDTLS	Interrupt Details Register
SMPU_RADDR[n]	Region n Address Register
SMPU_RCTL[n]	Region n Control Register
SMPU_REVID	SMPU Revision ID Register
SMPU_RIDA[n]	Region n ID A Register
SMPU_RIDB[n]	Region n ID B Register
SMPU_RIDMSKA[n]	Region n ID Mask A Register
SMPU_RIDMSKB[n]	Region n ID Mask B Register
SMPU_STAT	SMPU Status Register

Bus Error Address Register

Programs read the SMPU_BADDR and the SMPU_BDTLS registers to determine the cause of a bus error. Write errors are prioritized over read errors.

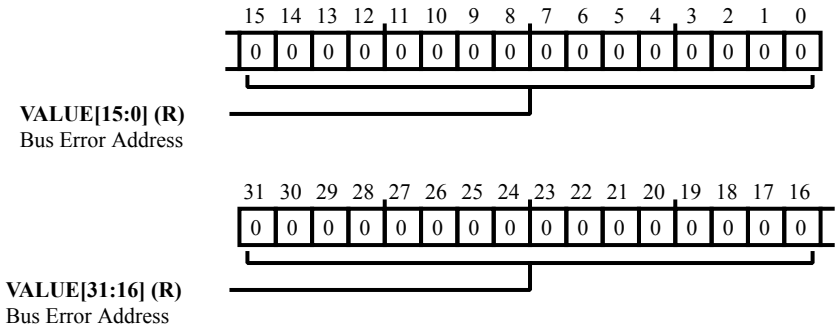


Figure 15-2: SMPU_BADDR Register Diagram

Table 15-7: SMPU_BADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Bus Error Address. The SMPU_BADDR.VALUE bit field contains the address of the bus error.

Bus Error Details Register

The SMPU_BDTLS register indicates the ID of the bus error transaction, whether the transaction that caused the last bus error was a read, a write, secure or non-secure.

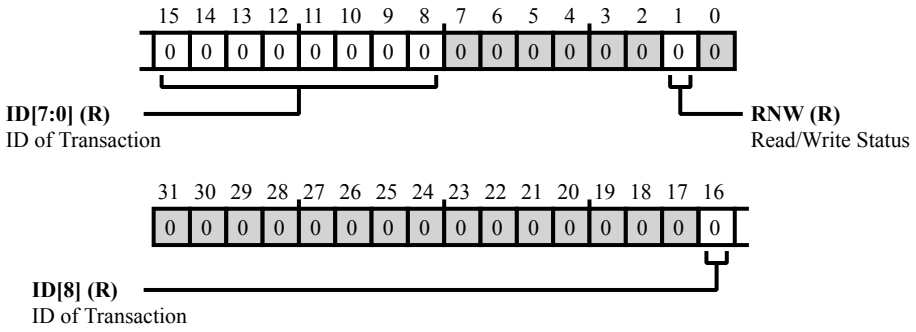


Figure 15-3: SMPU_BDTLS Register Diagram

Table 15-8: SMPU_BDTLS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:8 (R/NW)	ID	ID of Transaction. The SMPU_BDTLS.ID bit field provides the ID of the transaction that caused the bad address error.	
1 (R/NW)	RNW	Read/Write Status. The SMPU_BDTLS.RNW bit indicates whether the last transaction that caused the bad address error was a read or write.	
		0	Transaction that caused last bus error was a write
		1	Transaction that caused last bus error was a read

SMPU Control Register

The SMPU_CTL register provides access to the locking control, error interrupts and SMPU violations.

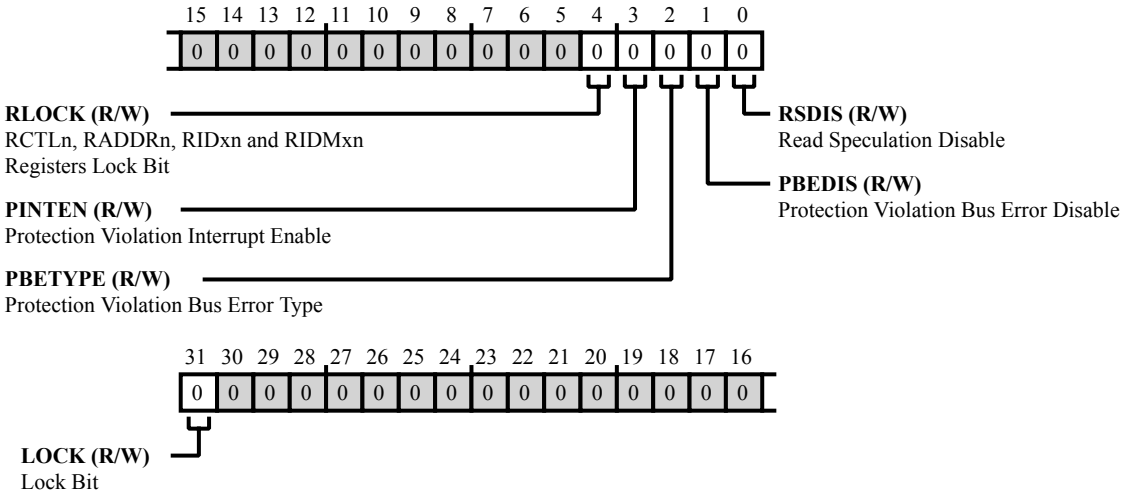


Figure 15-4: SMPU_CTL Register Diagram

Table 15-9: SMPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit. When the SMPU_CTL.LOCK bit is set and the global lock signal is asserted from the SPU, the SMPU_CTL register is write-protected. Write-protection is disabled only when the global lock signal becomes deasserted again.	
		0	CTL Global Lock Disable. The SMPU_CTL register is not write-protected.
		1	CTL Global Lock Enable. The SMPU_CTL register is write-protected.
4 (R/W)	RLOCK	RCTLn, RADDRn, RIDxn and RIDMxn Registers Lock Bit. When the SMPU_CTL.RLOCK bit is set, all the registers associated with region-based control (SMPU_RCTL[n], SMPU_RADDR[n], SMPU_RIDA[n], SMPU_RIDB[n], SMPU_RIDMSKA[n] and SMPU_RIDMSKB[n]) are write-protected when the global lock signal is active from the SPU. Write access is allowed again when the global lock signal is deasserted.	
		0	Region Registers Write-Protect Enable. All region registers are not write-protected.
		1	Region Registers Write-Protect Disable. All region registers are write-protected.

Table 15-9: SMPU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	PINTEN	Protection Violation Interrupt Enable. The SMPU_CTL.PINTEN bit controls whether or not an interrupt is generated when a protection violation occurs.	
		0	Protection Violation IRQ Disable. The protection violation interrupt is disabled.
		1	Protection Violation IRQ Enable. The protection violation interrupt is enabled.
2 (R/W)	PBETYPE	Protection Violation Bus Error Type. The SMPU_CTL.PBETYPE bit controls whether a protection violation produces a decode error or a slave error.	
		0	Decode Error Type. Decode error for transactions that violate the configured protection.
		1	Save Error Type. Slave Error for transactions which violate the configured protection
1 (R/W)	PBEDIS	Protection Violation Bus Error Disable. If set, the SMPU_CTL.PBEDIS bit blocks protection violations, but does not cause a bus error.	
		0	Bus Error Generation Enable. Transactions which violate the configured protection are blocked and cause a bus error.
		1	Bus Error Generation Disable. Transactions which violate the configured protection are blocked but do not cause a bus error.
0 (R/W)	RSDIS	Read Speculation Disable. The SMPU_CTL.RSDIS bit controls whether or not the read addresses are checked before being sent to the slave.	
		0	Read Speculation Enable. Read addresses are sent to the slave without checking.
		1	Read Speculation Disable. Read addresses are checked before being sent to the slave.

Interrupt Address Register

The SMPU_IADDR register indicates an attempt to make a read or write access to unimplemented addresses or accesses are non-aligned. The SMPU issues a bus error for this condition.

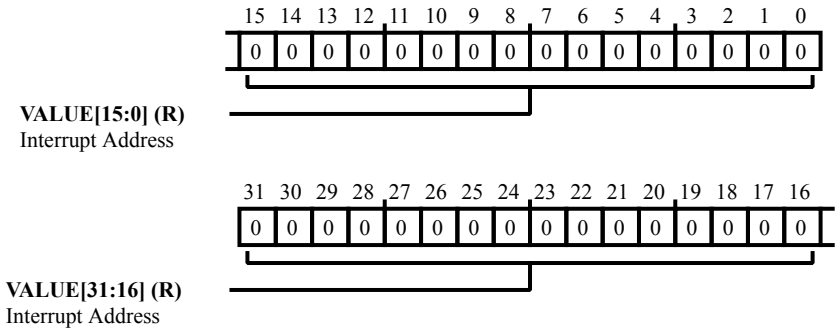


Figure 15-5: SMPU_IADDR Register Diagram

Table 15-10: SMPU_IADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Interrupt Address. The SMPU_IADDR.VALUE bit field is the address where an attempt to access an unimplemented address or a non-aligned access has occurred.

Interrupt Details Register

The SMPU_IDTLS register provides the ID of the last signaled interrupt, whether the interrupt was caused by a read or write, and whether the transaction that caused the last signaled interrupt was secure.

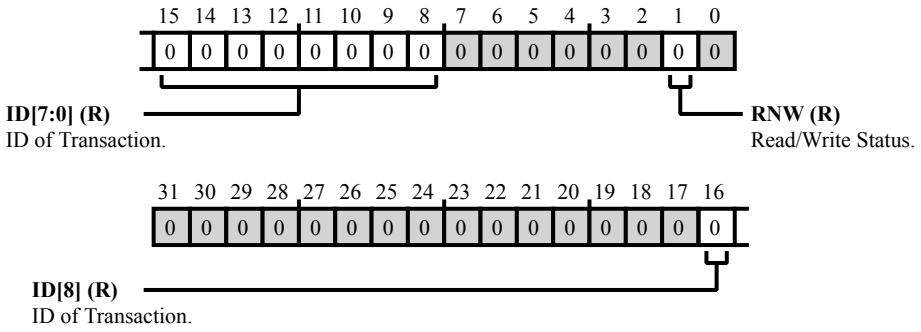


Figure 15-6: SMPU_IDTLS Register Diagram

Table 15-11: SMPU_IDTLS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:8 (R/NW)	ID	ID of Transaction.. The SMPU_IDTLS.ID bit field provides the ID of the transaction that caused the interrupt.	
1 (R/NW)	RNW	Read/Write Status.. The SMPU_IDTLS.RNW bit indicates whether the last transaction that caused the interrupt was a read or write.	
		0	Transaction that caused last signaled interrupt was a write
		1	Transaction that caused last signaled interrupt was a read

Region n Address Register

The SMPU_RADDR[n] register is used to define the base address for a memory region to be protected.

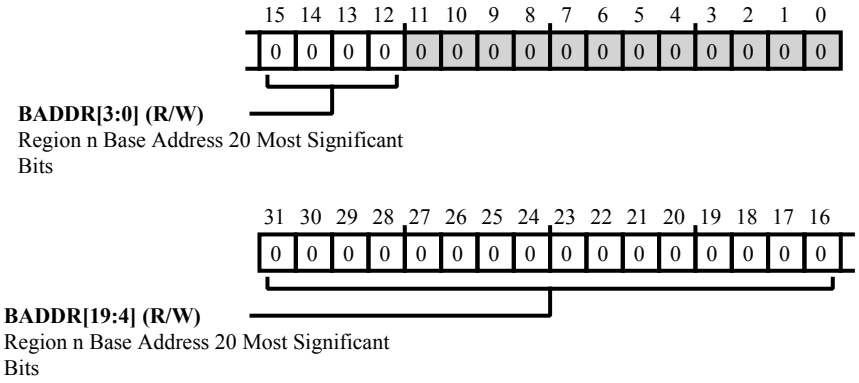


Figure 15-7: SMPU_RADDR[n] Register Diagram

Table 15-12: SMPU_RADDR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:12 (R/W)	BADDR	Region n Base Address 20 Most Significant Bits. The SMPU_RADDR[n].BADDR bit field defines the base address for a memory region to be protected.

Region n Control Register

The SMPU_RCTL[n] register is used to define the level of protection for a region of memory. The protection of a region is controlled and defined by this register and the SMPU_RADDR[n], SMPU_RIDA[n], SMPU_RIDB[n], SMPU_RIDMSKA[n], and SMPU_RIDMSKB[n] registers.

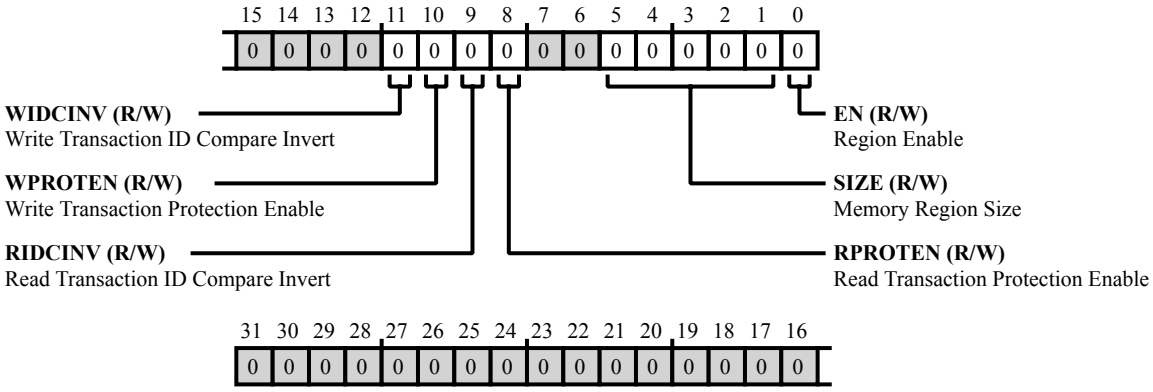


Figure 15-8: SMPU_RCTL[n] Register Diagram

Table 15-13: SMPU_RCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	WIDCINV	Write Transaction ID Compare Invert. The SMPU_RCTL[n].WIDCINV bit inverts the write ID match result.	
		0	Write transaction ID comparison result not inverted
		1	Write transaction ID comparison result inverted
10 (R/W)	WPROTEN	Write Transaction Protection Enable. The SMPU_RCTL[n].WPROTEN bit enables protection against ID-based write transactions for the memory region.	
		0	Write transaction ID-based protection disabled
		1	Write transaction ID-based protection enabled
9 (R/W)	RIDCINV	Read Transaction ID Compare Invert. When the SMPU_RCTL[n].RIDCINV bit is set, the read ID match result is inverted.	
		0	Read transaction ID comparison result not inverted
		1	Read transaction ID comparison result inverted

Table 15-13: SMPU_RCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	RPROTEN	Read Transaction Protection Enable. The `SMPU_RCTL[n].RPROTEN` bit enable bit to turn on protection against ID-based read transactions for the memory region.	
		0	Read transaction ID-based protection disabled
		1	Read transaction ID-based protection enabled
5:1 (R/W)	SIZE	Memory Region Size. The `SMPU_RCTL[n].SIZE` bit defines the size of the memory region to be protected.	
		0	4 KB
		1	8 KB
		2	16 KB
		3	32 KB
		4	64 KB
		5	128 KB
		6	256 KB
		7	512 KB
		8	1 MB
		9	2 MB
		10	4 MB
		11	8 MB
		12	16 MB
		13	32 MB
		14	64 MB
		15	128 MB
		16	256 MB
		17	512 MB
		18	1 GB
		19	2 GB
		20	4 GB
		21-31	Reserved

Table 15-13: SMPU_RCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Region Enable. The `SMPU_RCTL[n].EN` bit enables the protection of a region.	
		0	Disabled
		1	Enabled

SMPU Revision ID Register

The SMPU_REVID register provides the major and minor revision numbers of this module.

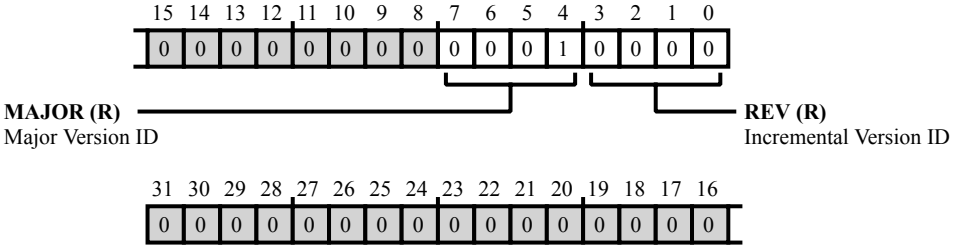


Figure 15-9: SMPU_REVID Register Diagram

Table 15-14: SMPU_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major Version ID.
3:0 (R/NW)	REV	Incremental Version ID.

Region n ID A Register

The SMPU_RIDA[n] register is used for ID comparison 'A'. This comparison is performed after a mask is applied to both the transaction ID (from either the read or write IDs) and the register value. An ID match means that the ID is the exception to the rule and the read or write is allowed even if the region is read or write-protected. For more detail, refer to the ID Comparison section.

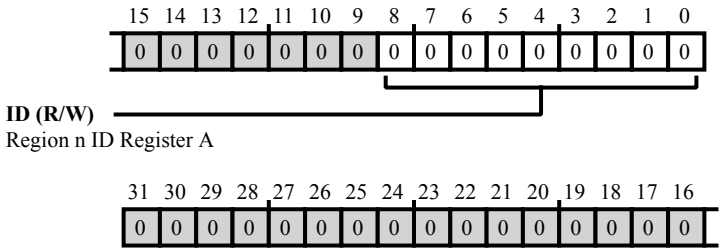


Figure 15-10: SMPU_RIDA[n] Register Diagram

Table 15-15: SMPU_RIDA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	ID	Region n ID Register A. The SMPU_RIDA[n].ID bit field, combined with the mask provides the means to bypass the configured memory protection for a region.

Region n ID B Register

The SMPU_RIDB[n] register is used for ID comparison 'B'. This comparison is performed after a mask is applied to both the transaction ID (from either the read or write IDs) and the register value. An ID match means that the ID is the exception to the rule and the read or write is allowed even if the region is read or write-protected. For more details, refer to the ID Comparison section.

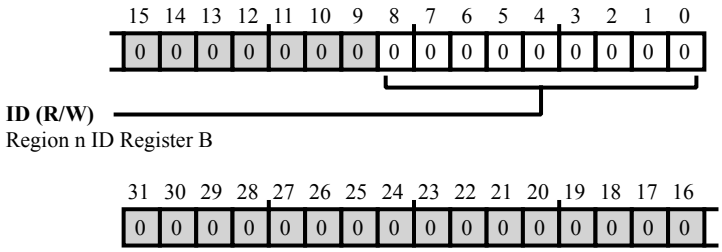


Figure 15-11: SMPU_RIDB[n] Register Diagram

Table 15-16: SMPU_RIDB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	ID	Region n ID Register B. The SMPU_RIDB[n].ID bit field, combined with the mask provides the means to bypass the configured memory protection for a region.

Region n ID Mask A Register

The SMPU_RIDMSKA[n] register is used for ID comparison 'A'. The mask allows or disallows certain IDs from affecting the final result of the ID match. For more details, refer to the ID Comparison section.

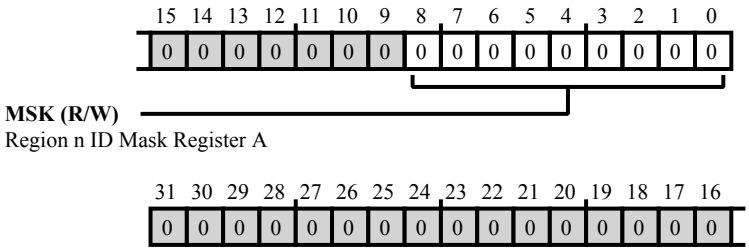


Figure 15-12: SMPU_RIDMSKA[n] Register Diagram

Table 15-17: SMPU_RIDMSKA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	MSK	Region n ID Mask Register A. The SMPU_RIDMSKA[n].MSK bit field, combined with the incoming transaction, provides the means to bypass the configured memory protection for a region.

Region n ID Mask B Register

The SMPU_RIDMSKB[n] register is used for ID comparison 'B'. The mask allows or disallows certain IDs from affecting the final result of the ID match. For more details, refer to the ID Comparison section.

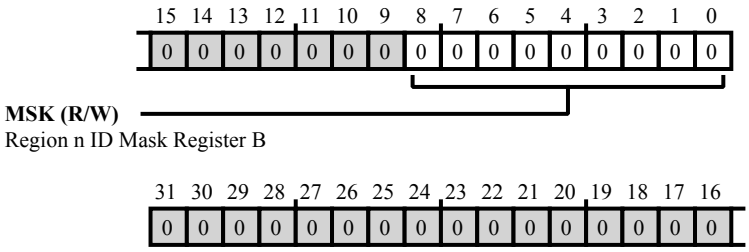


Figure 15-13: SMPU_RIDMSKB[n] Register Diagram

Table 15-18: SMPU_RIDMSKB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	MSK	Region n ID Mask Register B. The SMPU_RIDMSKB[n].MSK bit field, combined with the incoming transaction provides the means to bypass the configured memory protection for a region.

SMPU Status Register

The SMPU_STAT register provides the state of the SMPU and indicates various errors. All bits in this register are write 1 to clear.

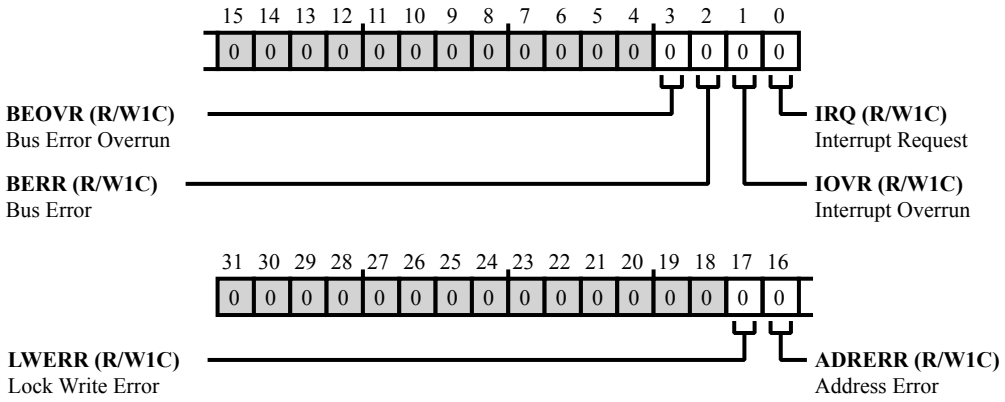


Figure 15-14: SMPU_STAT Register Diagram

Table 15-19: SMPU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	LWERR	Lock Write Error. The SMPU_STAT.LWERR bit is set when SMPU_CTL.LOCK bit =1, the global lock signal is asserted from the SPU and a read or write attempt was made to the SMPU_CTL MMR.	
		0	No Lock Write Error
		1	Lock Write Error
16 (R/W1C)	ADRERR	Address Error. The SMPU_STAT.ADRERR bit is set when the SMPU MMR is accessed as an un-aligned address, or when a read-only MMR is written to.	
		0	No Address Error
		1	Address Error
3 (R/W1C)	BEOVR	Bus Error Overrun. The SMPU_STAT.BEOVR bit indicates that another bus error had occurred. Any new information about the most recent violation which caused the bus error is not captured.	
		0	No Bus Error overrun
		1	Bus Error overrun has occurred

Table 15-19: SMPU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	BERR	Bus Error. This SMPU_STAT.BERR bit indicates if a bus error was generated.	
		0	No Bus Error since this bit has been cleared
		1	Bus Error has been generated
1 (R/W1C)	IOVR	Interrupt Overrun. The SMPU_STAT.IOVR bit indicates if another violation occurred while the previous violation interrupt was not finished being serviced. Information about the most recent violation is then not captured.	
		0	No Interrupt overrun
		1	Interrupt overrun has occurred
0 (R/W1C)	IRQ	Interrupt Request. The SMPU_STAT.IRQ bit provides an indication that an interrupt has been generated.	
		0	No Interrupt since this bit has been cleared
		1	Interrupt has been generated

16 Cyclic Redundancy Check (CRC)

The CRC peripheral performs the cyclic redundancy check (CRC) of the block of data that is presented to the peripheral. The peripheral provides a means to verify periodically the integrity of the system memory, the contents of memory-mapped registers (MMRs), or communication message objects. It is based on a CRC32 engine that computes the signature of 32-bit data presented to the peripheral.

The dedicated hardware compares the calculated signature of the operation to a pre-loaded expected signature. If the two signatures fail to match, the peripheral generates an error.

The source channel of the memory-to-memory DMA channels can provide data. The CRC optionally forwards data to memory through the destination DMA channel. Alternatively, the peripheral supports data presented by any qualified master of the CRC peripheral bus.

The CRC peripheral implements a reduced table-look-up algorithm to compute the signature of the data. The CRC uses a programmable 32-bit CRC polynomial to generate the look-up table (LUT) contents automatically.

More CRC peripheral modes allow for initializing large memory sections with a constant value, or for verifying that sections of memory are equal to a constant value.

CRC Features

The CRC peripheral supports a number of key features.

- Memory scan modes for memory verification
- Memory transfer modes for on-the-fly CRC calculations while transferring data from one memory to another
- A programmable 32-bit CRC polynomial with automatic LUT generation
- Data mirroring options

The CRC module also includes the following features.

- CRC checksum computation and comparison modes
- 32-bit programmable CRC polynomial with bit reverse option
- Automatic look-up table (LUT) generation
- Data mirroring options for endian and reflected polynomial cases

- Automatic clear and preset of results
- Fault and error interrupt reporting
- DMA and MMR based operation

Because the CRC module is closely tied to memory-to-memory DMA (MDMA) channel pairs, the use cases include the following features.

- Memory scan mode with CRC compute or compare
- Memory transfer mode with CRC compute or compare
- Memory fill operation with 32-bit data patterns
- Memory verify operation
- MMR write access to FIFO of destination DMA
- MMR read access to FIFO of source DMA
- Profiting from advanced DMA features, like descriptor mode and bandwidth control or monitor

CRC Functional Description

The CRC peripheral supports a number of modes of operation that allow for the initialization and verification of regions of memory. The peripheral supports efficient memory-fill and verification operations on regions of memory with or against a constant value. These modes of operation do not require the CRC engine to calculate a signature. Other modes of operation allow for the calculation of CRC signature and verification for a memory region. The modes allow for on-the-fly CRC calculation when performing memory-to-memory DMA transfers from one memory region to another.

To minimize the need for core accesses, the peripheral interfaces with one or more (depending on processor features) memory-to memory DMA (MDMA) channels. This connectivity permits flexible configuration, in which data can be written-to or read-from the peripheral using DMA transactions, core transactions, or a combination of both.

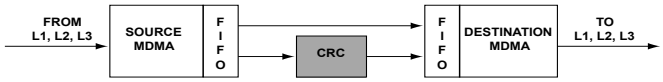


Figure 16-1: Memory Flow

CM41X_M4 CRC Register List

The Cyclic Redundancy Check (CRC) unit includes the data comparison, polynomial operation, and look up table generation features needed for CRC operation. The CRC provides CRC protection as specified by many functional safety requirements. This unit facilitates the system software's ability to periodically check the correctness of the code/data available in memory. A set of registers govern CRC operations. For more information on CRC functionality, see the CRC register descriptions.

Table 16-1: CM41X_M4 CRC Register List

Name	Description
CRC_COMP	Data Compare Register
CRC_CTL	Control Register
CRC_DCNT	Data Word Count Register
CRC_DCNTCAP	Data Count Capture Register
CRC_DCNTRLD	Data Word Count Reload Register
CRC_DFIFO	Data FIFO Register
CRC_FILLVAL	Fill Value Register
CRC_INEN	Interrupt Enable Register
CRC_INEN_CLR	Interrupt Enable Clear Register
CRC_INEN_SET	Interrupt Enable Set Register
CRC_POLY	Polynomial Register
CRC_RESULT_CUR	CRC Current Result Register
CRC_RESULT_FIN	CRC Final Result Register
CRC_STAT	Status Register

CM41X_M4 CRC Interrupt List

Table 16-2: CM41X_M4 CRC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
57	CRC0_ERR	CRC0 Error	Level	
145	CRC0_DCNTEXP	CRC0 Data count expiration	Level	

CRC Definitions

To make the best use of the CRC, it is useful to understand the following terms.

CRC

Acronym for Cyclic Redundancy Check. An error detection code that can detect changes within a block of data.

CRC Polynomial

The 32-bit polynomial used by the CRC engine to generate the look-up table required for the CRC implementation

LUT

Acronym for the Look-up Table. The look-up table is automatically generated from the supplied 32-bit CRC polynomial.

DMA

Acronym for Direct Memory Access. Used to describe a data transfer that takes place through a DMA channel allowing data distribution around a system without intervention from the core.

MDMA

Acronym for Memory-To-Memory DMA transfer that often requires the use of two DMA channels to transfer data from one memory region to another memory region. One DMA channel is configured as a source channel and the second as a destination channel.

CRC Block Diagram

The ***CRC Block Diagram*** shows the functional block diagram of the CRC. The following sections describe the blocks.

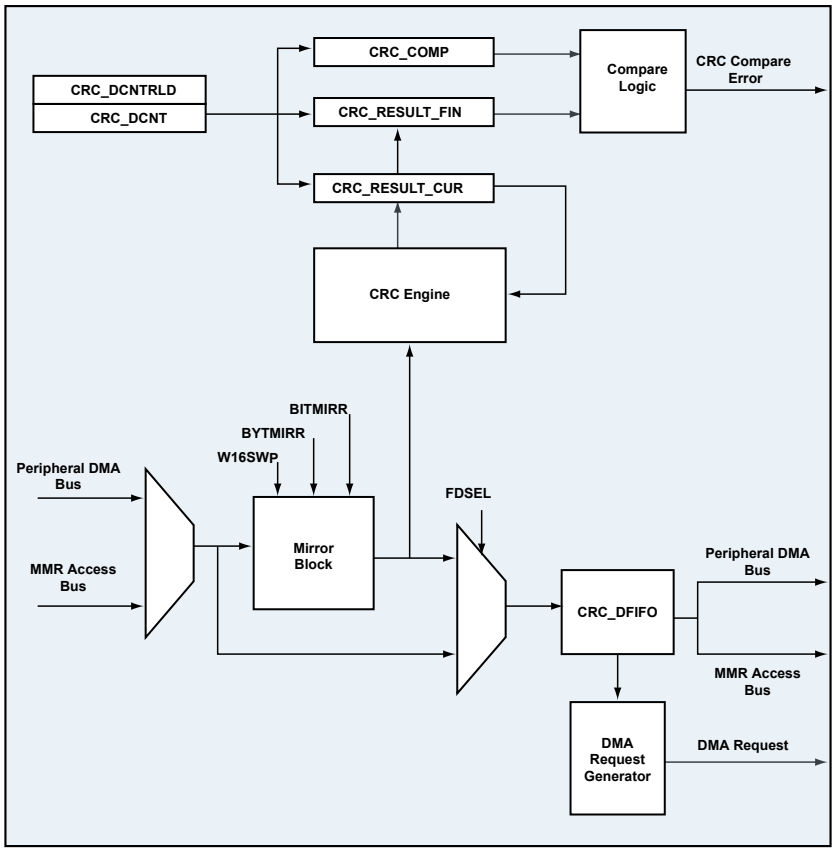


Figure 16-2: CRC Block Diagram

NOTE: The CRC Unit can be accessed by either the M0 or the M4 core. The CRC interrupts are not available to the M0.

Peripheral DMA Bus

The CRC peripheral provides both an incoming and outgoing datapath to the peripheral DMA bus. The MDMA source channel is interfaced to the incoming datapath providing data to the CRC peripheral. For memory transfer and data fill modes, the CRC uses the MDMA destination channel to either output the data from the CRC FIFO or use the data for the fill operation.

MMR Access Bus

The core uses the MMR access bus to access all the memory-mapped registers of the peripheral for configuration, status, and debug purposes. The core can also use the MMR access bus to feed data to the CRC peripheral or read data from the FIFO of the CRC peripheral. The CRC operation is an alternative to the DMA channel operation to read data from the FIFO.

Data received by MMR writes can transfer to destination DMA. Similarly, data received by source DMA can be output through the MMR interface. Optionally, intermediate results can be made available to the MMR interface.

Mirror Block

The mirror block individually controls bit-reversing of the polynomial, the computation results, and the expected result. Bit mirroring, byte mirroring, word swapping, and any combination of these operations can control endian and the reflection of processed data.

Data FIFO

The CRC data FIFO is a 32-bit-wide 4-entry FIFO. The FIFO is accessible to both the peripheral DMA bus and the MMR access bus. The FIFO status is accessible from the `CRC_STAT` register.

DMA Request Generator

The DMA request generator is responsible for granting incoming DMA requests from the source DMA channel and issuing outgoing DMA requests to the destination DMA channel.

CRC Engine

The CRC engine is a 32-bit CRC engine that implements the reduced table look-up scheme. The CRC engine provides support for a user-programmable 32-bit polynomial that the CRC uses to load the look-up table parameters required for the CRC calculation. The CRC engine is a single cycle implementation operating on 32 bits of data per cycle.

Compare Logic

The compare logic takes the final CRC signature and compares it to the expected CRC signature, generating a CRC compare error when the signatures do not match. A compare error can cause an error interrupt request to the Cortex-M4 system or to SEC1.

CRC Architectural Concepts

A 32-bit polynomial is required before calculation of the CRC signature can occur. The CRC uses the polynomial to generate the contents of an internal look-up table that the reduced table look-up implementation requires. The look-up table is automatically generated when the polynomial is written. It must be initialized prior to any operation that requires the use of the CRC engine.

The mirror block logic can manipulate the data presented to the CRC engine before the CRC uses the data in the calculation of the CRC signature. The data mirror operation is configurable to allow for bit reversing, byte reversing, and 16-bit word swapping operations on the incoming data. For memory transfer compute-and-compare operations, programs can configure the peripheral to output the data in the same form in which it was received. Or, the operation can output the mirrored data in the same manner that it is presented to the CRC engine.

While the CRC peripheral is in operation, the status of the FIFO is continually updated and reflected in the `CRC_STAT` register. The FIFO status is required for core-based accesses to the CRC peripheral. The status indicates when:

- The CRC peripheral can receive data
- Data is available for reading from the FIFO
- The result of the `CRC_RESULT_CUR` register has been updated

The status of the `CRC_RESULT_CUR` register indicates that the current CRC calculation has completed and the result is available.

Look-up Table

The look-up table consists of a set of sixteen 32-bit registers that hardware populates automatically when a write access takes place to the `CRC_POLY` register. 16 clock cycles are required to generate all 16 look-up table entries. The status of the process for generating the look-up table is reflected in `CRC_STAT.LUTDONE` allowing for software to poll on the completion of the event or for generation of an interrupt.

NOTE: Hardware must populate the look-up table before any operation using the CRC peripheral can take place, even if the operation does not use the CRC engine. The peripheral does not issue any data requests until the table generation process is complete. In addition, the `CRC_STAT.IBR` field, that indicates the input buffer status as required for core-based transfers, is only valid upon completion of the process for generating the look-up table.

Data Mirroring

The data mirror block can be configured to manipulate the incoming data before the data passes to the CRC engine and, optionally, to the FIFO. This configuration allows the peripheral to handle various forms of endianness and to function with reflected polynomials.

There are three configuration bits that control the data mirroring process: CRC_CTL.BITMIRR, CRC_CTL.BYTMIRR, and CRC_CTL.W16SWP. The ***Data Mirroring Options*** table details how these options affect the incoming data and the output generated by the mirror block.

Table 16-3: Data Mirroring Options

W16SWP	BYTMIRR	BITMIRR	Output Data
0	0	0	Dout[31:0] = Din[31:0]
0	0	1	Dout[31:0] = Din[24:31],Din[16:23],Din[8:15],Din[0:7]
0	1	0	Dout[31:0] = Din[7:0],Din[15:8],Din[23:16],Din[31:24]
0	1	1	Dout[31:0] = Din[0:7],Din[8:15],Din[16:23],Din[24:31]
1	0	0	Dout[31:0] = Din[15:0], D[31:16]
1	0	1	Dout[31:0] = Din[8:15],Din[0:7], Din[24:31],Din[16:23]
1	1	0	Dout[31:0] = Din[23:16],Din[31:24], Din[7:0],Din[15:8]
1	1	1	Dout[31:0] = Din[16:23],Din[24:31], Din[0:7],Din[8:15]

When the CRC is configured to operate in the memory transfer compute-and-compare mode, the bit-reversed output data can be written to the FIFO. This feature is controlled through the CRC_CTL.FDSEL field.

In addition to providing bit swapping and mirror options to the incoming data, the CRC peripheral also supports bit mirroring on the following registers.

- CRC_RESULT_CUR and CRC_RESULT_FIN, controlled through the CRC_CTL.RSLTMIRR field. When mirroring is enabled, the values to be written to these registers are fully bit-reversed before the write operation occurs.
- CRC_POLY, controlled through the CRC_CTL.POLYMIRR field. When mirroring is enabled, the 32-bit polynomial is fully bit-reversed before the write operation to the register occurs.
- CRC_COMP, controlled through the CRC_CTL.CMPMIRR field. When mirroring is enabled, the contents to be loaded to this register are fully bit-reversed before the write operation occurs.

FIFO Status and Data Requests

The CRC peripheral provides indication of the input and output buffer status through CRC_STAT.IBR and CRC_STAT.OBR respectively. For core-based operations, software must monitor these status fields prior to writing to or reading from the CRC FIFO. No write to the CRC FIFO can occur while CRC_STAT.IBR indicates that the buffer is not ready to accept data. Similarly, the CRC FIFO cannot be read until CRC_STAT.OBR indicates that data is available.

The memory scan modes of operation only require the monitoring of the input buffer status. The memory transfer, compute-and-compare mode uses both input and output buffer status. If the current result of the CRC computation is required, then software must verify that the current operation has completed and that the intermediate result is ready. The CRC_STAT.IRR indicates the status.

NOTE: The memory transfer fill mode of operation requires the use of a DMA channel. The CRC does not support core reads from the CRC FIFO for this mode of operation.

Memory transfer, compute-and-compare mode uses burst transactions to make the most efficient use of the available resources. In this mode, when the FIFO is initially empty and the peripheral is enabled, the CRC_STAT.IBR bit indicates that the CRC is ready to accept data. The peripheral generates data requests to the source DMA channel (if the CRC uses DMA). While the number of words remaining in the CRC_DCNT register is greater than the FIFO depth, the peripheral issues data requests or accepts incoming data in bursts. The peripheral continues until the CRC FIFO becomes full.

Once full, the CRC_STAT.IBR and CRC_STAT.OBR bits are updated, and then the CRC issues outgoing data requests. Only when the FIFO is empty can the peripheral accept further incoming data, and the CRC_STAT.IBR and CRC_STAT.OBR bits are updated once again.

Once CRC_DCNT is decremented such that the number of words waiting for processing is less than the number required to fill the FIFO, the burst mode of operation is disabled. Incoming data is accepted as long as the FIFO is not full. Outgoing data is available as long the FIFO is not empty. Therefore, there are no restrictions requiring the word count to be a multiple of the FIFO depth.

All other CRC modes of operation indicate that incoming data can be accepted as long as the FIFO is not full. Outgoing data is available as long as the FIFO is not empty.

The CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL bit configurations also influence how the CRC generates data requests and status bits. The following list describes the bits.

- The CRC_CTL.OBRSTALL bit can be configured such that the CRC peripheral stalls as soon as there is output data available in the FIFO. Use this mode of operation only in memory transfer, compute-and-compare mode. This mode results in the processing of one 32-bit word at a time. The peripheral does not request or accept incoming data until the current value being processed is read from the peripheral.
- The CRC_CTL.IRRSTALL bit can be configured so that the CRC peripheral stalls all further incoming data requests until the CRC_RESULT_CUR register is read after being updated. Use this mode of operation for CRC signature generation. It is not applicable to memory transfer data-fill mode or memory scan data-verify mode of operation.

CRC Operating Modes

The following sections describe the various operating modes of the CRC interface.

Data Transfer Modes

The CRC peripheral supports two main categories of operation involving data transfers:

- Memory scan mode
- Memory transfer mode

Memory scan modes are read-only operations that allow the contents of memory to be read into the peripheral and verified for correctness. There are two forms of memory scan mode:

- CRC compute-and-compare performs a CRC calculation on data presented to the peripheral and compares the CRC result with a pre-determined and pre-loaded result. An error is generated when the results differ.
- Data verify compares each 32-bit data word presented to the CRC peripheral to a pre-loaded 32-bit value and generates an error when the data differs.

Both of these modes of operation require, at the most, a single DMA channel to read the data from memory into the peripheral. No data is forwarded to the data output or destination DMA. The CRC can also use core-driven transfers for either of these modes of operation.

The memory transfer modes involve memory write or memory read-and-write operations allowing for memory to be initialized or transferred from one region of memory to another. There are two forms of memory transfer mode:

- CRC compute-and-compare performs a full data transfer from one memory region to another memory region. The CRC generates a signature on the data presented and compares it with a pre-determined and pre-loaded result. An error is generated when the results differ.
- Data fill initializes a region of memory with a pre-loaded 32-bit constant value.

The CRC compute-and-compare mode of operation requires both incoming and outgoing data channels. The operation occurs either using DMA channels, using core-driven write or read operations to and from the FIFO or using a combination of both. The data fill mode of operation requires only a memory write DMA destination channel—this mode does not support core driven operations.

Memory Scan Compute-and-Compare Mode

In this mode of operation, the CRC engine of the peripheral is enabled. The mode is configured through the `CRC_CTL.OPMODE` field and the CRC engine performs a 32-bit CRC operation on the incoming data stream.

The length of the data stream is configured through the `CRC_DCNT` register. The accumulated result of the CRC operation is contained in the `CRC_RESULT_CUR` register. As the CRC engine processes each 32-bit word, the `CRC_DCNT` register is decremented and `CRC_RESULT_CUR` is updated.

Once `CRC_DCNT` decrements to zero, the contents of the `CRC_RESULT_CUR` register are copied to `CRC_RESULT_FIN` and `CRC_STAT.DCNTEXP` is updated accordingly. The CRC uses the `CRC_COMP` register to store the expected result of the operation. After the CRC calculation, `CRC_COMP` is compared with `CRC_RESULT_FIN` and `CRC_STAT.CMPERR` is updated to reflect the status of the compare operation. `CRC_STAT.CMPERR` must be cleared before the next CRC operation is performed.

The CRC peripheral also contains the `CRC_DCNTRLD` register. The CRC uses this register to reload `CRC_DCNT` upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation can be configured through CRC_CTL.AUTOCLRZ and CRC_CTL.AUTOCLRF. This configuration provides a way to reset CRC_RESULT_CUR to 0x00000000, 0xFFFFFFFF or to leave the current register contents untouched for the next operation.

The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Scan Data Verify

In this mode of operation, the CRC engine of the peripheral is not required. The mode is enabled through the CRC_CTL.OPMODE field. Each 32-bit word of the data stream is compared with a constant value that is stored in the CRC_COMP register. The CRC_DCNT register contains the number of words for comparison. The CRC_DCNT register is decremented upon receiving a new 32-bit word from the data stream. If the compare operation fails, the CRC_STAT.CMPERR bit is updated and the contents of CRC_DCNT are captured in the CRC_DCNTCAP register. This information can be used to identify the location in the data stream where the error occurred. Clear the CRC_STAT.CMPERR field to reenable capturing of further errors.

Once CRC_DCNT decrements to zero, CRC_STAT.DCNTEXP is updated accordingly to signal the end of the operation. The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Compute-and-Compare Mode

In this mode of operation, the CRC engine of the peripheral is enabled. The mode is configured through the CRC_CTL.OPMODE field and the CRC engine performs a 32-bit CRC operation on the incoming data stream.

The length of the data stream is configured through the CRC_DCNT register. The accumulated result of the CRC operation is contained in the CRC_RESULT_CUR register. As the CRC engine processes each 32-bit word, the CRC_DCNT register is decremented and CRC_RESULT_CUR is updated.

Once CRC_DCNT decrements to zero, the contents of the CRC_RESULT_CUR register are copied to CRC_RESULT_FIN and CRC_STAT.DCNTEXP is updated accordingly. The CRC uses the CRC_COMP register to store the expected result of the operation. Upon completion of the CRC calculation, CRC_COMP is compared with CRC_RESULT_FIN and CRC_STAT.CMPERR is updated to reflect the status of the compare operation. Clear CRC_STAT.CMPERR before the next CRC operation is performed.

The CRC peripheral also contains CRC_DCNTRLD register. The CRC uses this register to reload CRC_DCNT upon completion of the CRC operation in preparation for the next transfer.

The initial seed of the CRC computation can be configured through CRC_CTL.AUTOCLRZ and CRC_CTL.AUTOCLRF. This configuration provides a means to reset CRC_RESULT_CUR to 0x00000000, 0xFFFFFFFF or to leave the current register contents untouched for the next operation.

The peripheral can be configured to allow for the compare error and data expiration events to generate an interrupt.

Memory Transfer Data Fill Mode

In this mode of operation, the CRC engine of the peripheral is not required. The mode is enabled through the CRC_CTL.OPMODE field. The CRC_FILLVAL register is written with a 32-bit value. The CRC uses this value to initialize a block memory through the memory-to-memory DMA destination channel. When the CRC peripheral

and the DMA destination channel are enabled, the contents of the CRC_FILLVAL register is written to the DMA channel to initialize the memory region. The CRC_DCNT register contains the number of words for the write operation.

Once CRC_DCNT decrements to zero, CRC_STAT.DCNTEXP is updated accordingly to signal the end of the operation. The peripheral can be configured to allow for the data expiration event to generate an interrupt.

CRC Event Control

The CRC peripheral can enable certain CRC status operations to generate an interrupt event to the system event controller. There, a CRC error can be qualified as a system fault.

Interrupt Signals

The CRC peripheral can generate two interrupt requests that are optionally enabled as interrupts with the ARM core, or as events to SEC1 for fault operations. One is a CRC status interrupt and the other is a CRC error interrupt.

The CRC_STAT.CMPERR status bit can be configured as an interrupt and is signaled through the CRC error interrupt signal. The CRC_STAT.CMPERR status field is set whenever the CRC peripheral performs a compare operation that fails. This status can be the result of a failed memory scan data-verify operation that compares the contents of a memory range with a constant 32-bit value. Or, it can be the result of a CRC signature calculated for a memory region that does not match the expected pre-programmed result for a memory-compare operation.

The CRC_STAT.DCNTEXP status bit is set when the CRC_DCNT register has decremented to zero. The status indicates that the CRC peripheral has now processed all the data requested for the current CRC operation. The CRC can also use this signal to generate an interrupt. The interrupt is signaled on the CRC status interrupt signal.

Both these status bits can be configured to generate and interrupt through the CRC_INEN register. The CRC_INEN register also has bit set, CRC_INEN_SET, and bit clear CRC_INEN_CLR equivalent registers that the CRC uses for the enabling and disabling of these interrupt sources.

The CRC_STAT register has two write one to clear (W1C) fields for clearing the two interrupt sources.

NOTE: Disabling the CRC peripheral through the CRC_CTL.BLKEN bit does not result in the clearing of interrupt sources. Clear the interrupt sources using a W1C operation to the CRC_STAT register.

CRC Programming Model

It is important to note the following restrictions when using the CRC peripheral with the DMA channels:

1. When enabling the CRC peripheral and the DMA channels, enable the CRC peripheral prior to enabling the DMA channels.
2. When disabling the CRC peripheral and the DMA channels, disable the DMA channels prior to disabling the CRC peripheral.

CRC Mode Configuration

Describes a number of tasks showing the various operation modes of the CRC peripheral.

- Look-up Table Generation
- Core Driven Memory Scan Compute-and-Compare Mode
- DMA Driven Memory Scan Compute-and-Compare Mode
- Core Driven Memory Scan Data Verify Mode
- DMA Driven Memory Scan Data Verify Mode
- Core Driven Memory Transfer Compute-and-Compare Mode
- DMA Driven Memory Transfer Compute-and-Compare Mode
- DMA Driven Memory Transfer Data Fill Mode

Look-up Table Generation

Describes the steps required to initialize the CRC peripheral LUT.

1. Write the 32-bit CRC polynomial of choice to the `CRC_POLY` register.

   ADDITIONAL INFORMATION: This operation results in the CRC peripheral starting the LUT initialization process. The `CRC_STAT.LUTDONE` bit is updated to reflect the operation is in progress.

2. Poll the `CRC_STAT.LUTDONE` bit until the status bit indicates that the operation is completed.

The CRC peripheral has completed initialization of all the LUT registers and is now ready for data operations. The `CRC_STAT.LUTDONE` bit remains in the current state until the `CRC_POLY` register is written again, or the peripheral or processor are reset.

Core Driven Memory Scan Compute-and-Compare Mode

Performs CRC signature calculation and verification for a region of memory using core transactions. The CRC peripheral is configured such that it operates in burst mode due to the stalling options configured through disabling the `CRC_CTL` register.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per `CRC_CTL.BLKEN`

1. Initialize the `CRC_DCNT` register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_RESULT_CUR register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan compute-and-compare mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

   - Disable the CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL bit options for this task example.
   - Configure all mirroring and bit reversal options.
   - Configure CRC auto-clear options.

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Write memory region data to the CRC peripheral.

   a. While CRC_STAT.IBR bit indicates that the input buffer is ready, write the CRC_DFIFO register with 32-bit data.

      ADDITIONAL INFORMATION: Repeat this step until all data has been written.

8. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures that all the data has been processed.

9. Poll the CRC_STAT.CMPERR bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write to the CRC_STAT register to clear both the CRC_STAT.DCNTEXP and CRC_STAT.CMPERR bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear of these status bits within the interrupt handlers for the respective interrupts.

The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The integrity check of the memory through the expected CRC signature has completed. The final result is indicated through the CRC_STAT.CMPERR bit and the corresponding interrupt when enabled.

Clear any W1C CRC status bits before performing more CRC operations.

DMA Driven Memory Scan Compute-and-Compare Mode

Performs CRC signature calculation and verification for a region of memory using DMA transactions. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured through disabling CRC_CTL.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN bit.

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_RESULT_CUR register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final operation.

5. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan compute compare mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

- Disable the CRC_CTL.OBRSTALL and CRC_CTL.IRRSTALL bit options for this task example.
- Configure all mirroring and bit reversal options.
- Configure all CRC auto clear options.

The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer from the memory region and writes the data to the CRC peripheral.

8. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if the counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the CRC_STAT.CMPERR bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the CRC_STAT register to clear both the CRC_STAT.DCNTEXP and CRC_STAT.CMPERR bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

    The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The integrity check of the memory through the expected CRC signature has completed and the final result indicated is through CRC_STAT.CMPERR and the corresponding interrupt, when enabled.

Clear any W1C CRC status bits before performing a further CRC operation. Clear any W1C status bits of the memory-to-memory source DMA channel before the next CRC operation.

Core Driven Memory Scan Data Verify Mode

Reads a region of memory using core transactions and performs a compare operation on each 32-bit word against a single pre-loaded 32-bit constant. The compare error interrupt is enabled to capture and log the location of any compare errors.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per CRC_CTL.BLKEN

The interrupt service routine for the compare error interrupt reads and stores the contents of CRC_DCNTCAP register to a buffer before clearing the compare error interrupt.

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the memory region is expected to be filled with. Each 32 bit of data presented to the peripheral is compared with this value.

4. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan data verify mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

6. Write memory region data to the CRC peripheral.

   a. Poll the CRC_STAT.IBR bit until input buffer is ready.

   b. Write the CRC_DFIFO register with 32-bit data.

      ADDITIONAL INFORMATION: Repeat these two steps until the entire memory region has been written to the CRC peripheral.

7. Poll the CRC_INEN_SET.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

8. Check if the buffer used to capture the CRC_DCNTCAP register upon a compare error has any new entries.

   ADDITIONAL INFORMATION: The values captures in the buffer provide a means to locate where in the memory region the failures occurred.

9. Write to the CRC_STAT to clear both the CRC_INEN_SET.DCNTEXP and CRC_INEN.CMPERR bits.

   ADDITIONAL INFORMATION: If interrupts were enabled, the clear these status bits within the interrupt handlers for the respective interrupts.

   The CRC memory scan-verify operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The result of the integrity check of the memory with the 32-bit constant is indicated through the CRC_INEN.CMPERR bit and the corresponding interrupt, when enabled. Each comparison error is traceable due to the logging of CRC_DCNTCAP from within the compare error interrupt handler.

Clear any W1C CRC status bits before performing a further CRC operation.

DMA Driven Memory Scan Data Verify Mode

The memory scan data verify mode reads a region of memory using DMA transactions and performs a compare operation on each 32-bit word against a single pre-loaded 32-bit constant. The compare error interrupt is enabled to capture and log the location of any compare errors.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN bit

The interrupt service routine for the compare error interrupt reads and stores the contents of the CRC_DCNTCAP register to a buffer before clearing the compare error interrupt.

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: The CRC module uses this register to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the memory region is expected to be filled with. Each 32 bit of data presented to the peripheral is compared with this value.

4. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan data verify mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.

   The CRC peripheral is now enabled and ready for the core or DMA channel to write the data.

6. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer from the memory region and writes the data to the CRC peripheral.

7. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.

8. Check if the buffer used to capture the CRC_DCNTCAP register upon a compare error has any new entries.

   ADDITIONAL INFORMATION: The values captures in the buffer provide a means to locate where in the memory region the failures occurred.

9. Write the CRC_STAT register to clear both the CRC_STAT.DCNTEXP and CRC_STAT.CMPERR bits.

   ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

   The CRC memory scan-verify operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation.

The result of the integrity check of the memory with the 32-bit constant is indicated through the CRC_STAT.CMPERR bit and the corresponding interrupt when enabled. Each comparison error is traceable due to the logging of the CRC_DCNTCAP register from within the compare error interrupt handler.

Clear any W1C CRC status bits and DMA status bits before performing a further CRC operation.

Core Driven Memory Transfer Compute-and-Compare Mode

The memory transfer compute-and-compare mode performs CRC signature calculation and verification for a region of memory using core transactions while copying the contents to another memory region. The CRC peripheral is configured such that it operates in the burst mode of operation due to the stalling options configured through disabling the CRC_CTL register.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN bit

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of the current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_RESULT_CUR register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan compute-and-compare mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

   a. Disable the CRC_CTL.OBRSTALL bit and the CRC_CTL.IRRSTALL bit options for this task example.

   b. Configure all mirroring and bit reversal options.

   c. Configure CRC auto clear options

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.

7. Write memory region data to the CRC peripheral and read it back to the new destination.

   a. While the CRC_STAT.IBR bit indicates that the input buffer is ready, write the CRC_DFIFO register with 32-bit data.

   b. While the CRC_STAT.OBR bit indicates that the output buffer is ready, read the CRC_DFIFO register and store data to new destination.

      ADDITIONAL INFORMATION: Repeat these two steps until all required data has been processed through the CRC peripheral and copied to the new destination.

8. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if the counter expired interrupt is disabled. Polling ensures all the data has been processed.

9. Poll the CRC_STAT.CMPERR bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.

10. Write the CRC_STAT register to clear both CRC_STAT.DCNTEXP and CRC_STAT.CMPERR bits.

ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

The memory region has been copied to a new location and an integrity check of the memory through the expected CRC signature has also completed. The final result is indicated through the CRC_STAT.CMPERR bit and the corresponding interrupt when enabled.

Clear any W1C CRC status bits before performing a further CRC operation.

DMA Driven Memory Transfer Compute-and-Compare Mode

The memory transfer compute-and-compare mode performs CRC signature calculation and verification for a region of memory using DMA transactions. The memory region is also copied to another memory region using memory-to-memory DMA transfers. The CRC peripheral is configured such that it operates in burst mode due to the stalling options configured through disabling CRC_CTL.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN register.

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_RESULT_CUR register.

   ADDITIONAL INFORMATION: This register can be initialized to provide an initial seed for the CRC operation that is about to take place.

4. Initialize the CRC_COMP register.

   ADDITIONAL INFORMATION: This register contains the pre-calculated final CRC signature result for the memory region that the software uses in the final compare operation.

5. Initialize the CRC_INEN register.

ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of compare errors and block completion. Configure these interrupts, as needed. If enabled, ensure that the corresponding interrupt handlers are also configured.

6. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory scan compute compare mode and CRC_CTL.BLKEN configured to enable the CRC peripheral.
   a. Disable the CRC_CTL.OBRSTALL and the CRC_CTL.IRRSTALL bit options for this task example.
   b. Configure all mirroring and bit reversal options
   c. Configure CRC auto clear options

   The CRC peripheral is now enabled and ready for the core or DMA channel to write data.
7. Configure and enable the memory-to-memory source DMA channel for memory read STOP mode and destination DMA channel for memory write STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer from one memory region to another through the memory-to-memory DMA channels and the CRC peripheral.
8. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures all the data has been processed.
9. Poll the CRC_STAT.CMPERR bit if the interrupt was disabled to check for a compare error.

   ADDITIONAL INFORMATION: Perform this step only if the compare error interrupt is not enabled.
10. Write the CRC_STAT register to clear both the CRC_STAT.DCNTEXP and the CRC_STAT.CMPERR bits.

    ADDITIONAL INFORMATION: If interrupts were enabled, then clear these status bits within the interrupt handlers for the respective interrupts.

    The CRC compute-and-compare operation is now complete. The CRC peripheral is ready to be configured for the next CRC operation. The memory region has also been copied to its new destination.

The integrity check of the memory through the expected CRC signature has completed and the final result is indicated through the CRC_STAT.CMPERR bit and the corresponding interrupt when enabled. The memory region has also been copied to its final destination.

Clear any W1C CRC status bits before performing a further CRC operation. Also, clear any W1C status bits of the memory-to-memory source and destination DMA channels before the next CRC operation.

DMA Driven Memory Transfer Data Fill Mode

This mode initializes a region of memory to a constant 32-bit value using DMA transactions.

The task assumes the following:

- The polynomial has been loaded and the look-up table is fully initialized
- All CRC interrupts have been serviced (none pending)
- The CRC block is disabled per the CRC_CTL.BLKEN bit

1. Initialize the CRC_DCNT register.

   ADDITIONAL INFORMATION: The value loaded must represent the number of 32-bit words in the memory region for which the software calculates and verifies the signature.

2. Initialize the CRC_DCNTRLD register.

   ADDITIONAL INFORMATION: This value is used to reload the CRC_DCNT register upon completion of current CRC operation. If no further operation is needed, then this register can be initialized to zero.

3. Initialize the CRC_FILLVAL register.

   ADDITIONAL INFORMATION: This register contains the 32-bit constant that the CRC module uses to fill the memory region.

4. Initialize the CRC_INEN register.

   ADDITIONAL INFORMATION: The CRC module uses this register to enable the generation of the CRC interrupts for notification of block completion. Configure these interrupts as required. If enabled, ensure that the corresponding interrupt handlers are also configured.

5. Initialize the CRC_CTL register with the CRC_CTL.OPMODE bit set to memory transfer fill mode and the CRC_CTL.BLKEN bit configured to enable the CRC peripheral.

   The CRC peripheral is now enabled and is ready for the DMA channel to write data.

6. Configure and enable the memory-to-memory destination DMA channel for memory write STOP mode.

   ADDITIONAL INFORMATION: This step starts the data transfer taking the constant 32-bit value from the CRC peripheral and writing the data to the DMA channel.

7. Poll the CRC_STAT.DCNTEXP bit if the interrupt was disabled.

   ADDITIONAL INFORMATION: Perform this step only if counter expired interrupt is disabled. Polling ensures that all the data has been processed.

8. Write the CRC_STAT register to clear the CRC_STAT.DCNTEXP bit.

   ADDITIONAL INFORMATION: If interrupts were enabled, then clear this status bit within the interrupt handlers for the respective interrupts.

   The CRC memory transfer fill operation is now complete and the CRC peripheral is ready to be configured for the next CRC operation.

The memory region is now filled with the constant data and the CRC peripheral is ready to be configured for a new operation.

Clear any W1C CRC status bits and DMA status bits before performing a further CRC operation.

CM41X_M4 CRC Register Descriptions

Cyclic Redundancy Check Unit (CRC) contains the following registers.

Table 16-4: CM41X_M4 CRC Register List

Name	Description
CRC_COMP	Data Compare Register
CRC_CTL	Control Register
CRC_DCNT	Data Word Count Register
CRC_DCNTCAP	Data Count Capture Register
CRC_DCNTRLD	Data Word Count Reload Register
CRC_DFIFO	Data FIFO Register
CRC_FILLVAL	Fill Value Register
CRC_INEN	Interrupt Enable Register
CRC_INEN_CLR	Interrupt Enable Clear Register
CRC_INEN_SET	Interrupt Enable Set Register
CRC_POLY	Polynomial Register
CRC_RESULT_CUR	CRC Current Result Register
CRC_RESULT_FIN	CRC Final Result Register
CRC_STAT	Status Register

Data Compare Register

The CRC_COMP register contains the value corresponding to the expected CRC result or signature for the current data stream. At the end of the operation, the content of this register is used to compare against the result produced by the CRC operation. In data verify mode, each incoming data value is compared with the content of this register.

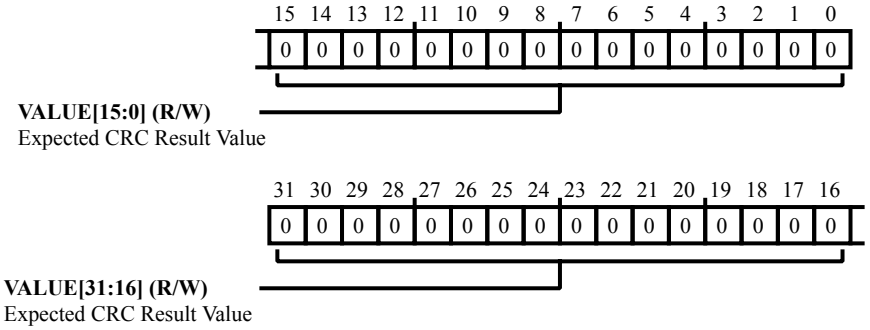


Figure 16-3: CRC_COMP Register Diagram

Table 16-5: CRC_COMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Expected CRC Result Value. The CRC_COMP.VALUE bit field contains the value corresponding to the expected CRC result or signature for the current data stream.

Control Register

The CRC_CTL register configures the operation modes and settings for the CRC.

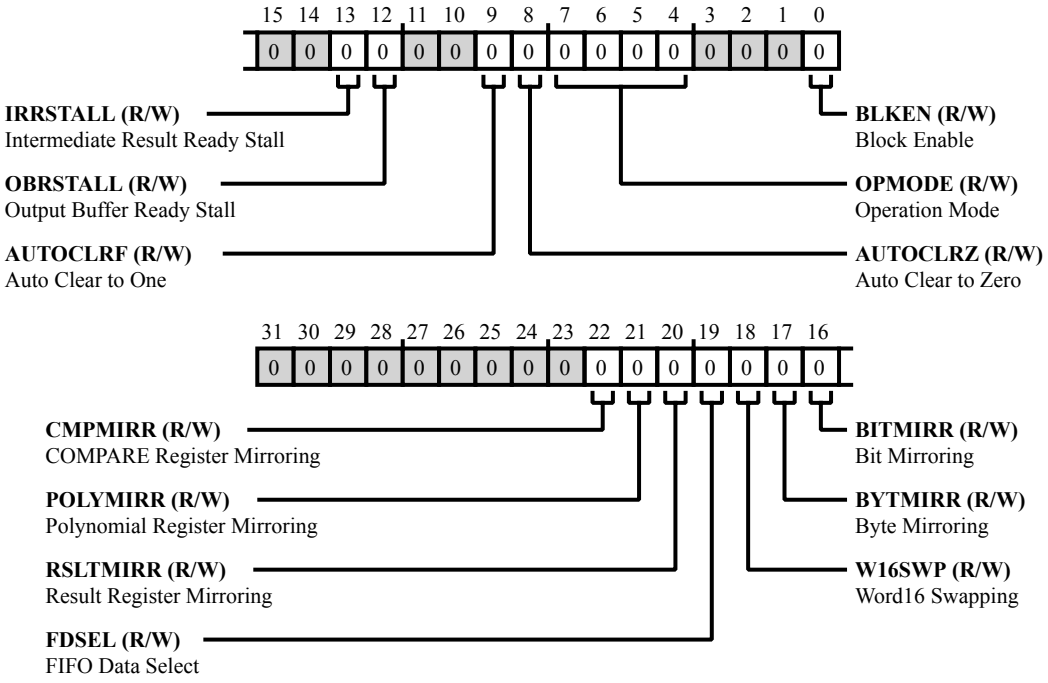


Figure 16-4: CRC_CTL Register Diagram

Table 16-6: CRC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	CMPMIRR	COMPARE Register Mirroring. The CRC_CTL.CMPMIRR bit enables data mirroring for the CRC_COMP compare register. When enabled, the 32-bit value in this register is fully bit mirrored (reversed). The bit-reversed value is used for comparison with the CRC_RESULT_FIN register.	
		0	Disable compare mirroring
		1	Enable compare mirroring
21 (R/W)	POLYMIRR	Polynomial Register Mirroring. The CRC_CTL.POLYMIRR bit enables data mirroring for the CRC_POLY polynomial register. When enabled, the 32-bit value in this register is fully bit mirrored (reversed). The bit-reversed value is used for CRC computations.	
		0	Disable polynomial mirroring
		1	Enable polynomial mirroring

Table 16-6: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W)	RSLTMIRR	Result Register Mirroring. The CRC_CTL.RSLTMIRR bit enables data mirroring for the CRC_RESULT_CUR and CRC_RESULT_FIN result registers. When enabled, the 32-bit values in these registers are fully bit mirrored (reversed).	
		0	Disable result mirroring
		1	Enable result mirroring
19 (R/W)	FDSEL	FIFO Data Select. The CRC_CTL.FDSEL bit selects whether the CRC writes modified or unmodified data to the FIFO in memory transfer mode. If enabled, the data written is affected by the state of the data mirroring selections (CRC_CTL.BITMIRR, CRC_CTL.BYTMIRR, and CRC_CTL.W16SWP) before being written to the FIFO.	
		0	Write unmodified data to FIFO
		1	Write modified data to FIFO
18 (R/W)	W16SWP	Word16 Swapping. The CRC_CTL.W16SWP bit enables the CRC's data mirror block to swap the upper and lower 16-bit words within the 32-bit input data, before further processing.	
		0	Disable word16 swapping
		1	Enable word16 swapping
17 (R/W)	BYTMIRR	Byte Mirroring. The CRC_CTL.BYTMIRR bit enables the CRC's data mirror block to mirror the bytes within the 32-bit input data, before further processing.	
		0	Disable byte mirroring
		1	Enable byte mirroring
16 (R/W)	BITMIRR	Bit Mirroring. The CRC_CTL.BITMIRR bit enables the CRC's data mirror block to mirror the bits within each byte of the 32-bit input data, before further processing.	
		0	Disable bit mirroring
		1	Enable bit mirroring
13 (R/W)	IRRSTALL	Intermediate Result Ready Stall. The CRC_CTL.IRRSTALL bit enables stalling the state machine for input data when there is a valid intermediate result to be read in the CRC_RESULT_CUR register. This feature should be used only in CRC computation modes (for example, CRC_CTL.OPMODE =1 or =3).	
		0	Do not stall
		1	Stall on IRR

Table 16-6: CRC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	OBRSTALL	Output Buffer Ready Stall. The CRC_CTL.OBRSTALL bit enables stalling the state machine for input data when there is valid data in the output buffer. This feature should be used only in memory-to-memory transfer modes (for example, CRC_CTL.OPMODE =1).	
		0	Do not stall
		1	Stall on OBR
9 (R/W)	AUTOCLRF	Auto Clear to One. The CRC_CTL.AUTOCLRF bit enables auto clear to one when the CRC is in intermediate results ready stall mode (CRC_CTL.IRRSTALL=1) and the CRC data count expires (CRC_DCNT=0). Note that the CRC_CTL.AUTOCLRZ bit must be disabled, or the CRC_CTL.AUTOCLRF bit has no effect.	
		0	No auto clear
		1	Auto clear
8 (R/W)	AUTOCLRZ	Auto Clear to Zero. The CRC_CTL.AUTOCLRZ bit enables auto clear to zero when the CRC is in intermediate results ready stall mode (CRC_CTL.IRRSTALL=1) and the CRC data count expires (CRC_DCNT=0). Note that CRC_CTL.AUTOCLRF must be disabled, or the CRC_CTL.AUTOCLRZ has no effect.	
		0	No auto clear
		1	Auto clear
7:4 (R/W)	OPMODE	Operation Mode. The CRC_CTL.OPMODE bit field selects the memory transfer or scan mode.	
		0	Reserved
		1	CRC compute/compare memory transfer
		2	Data fill memory transfer
		3	CRC compute/compare memory scan
		4	Data verify memory scan
0 (R/W)	BLKEN	Block Enable. The CRC_CTL.BLKEN bit enables and disables the CRC operation.	
		0	Disable
		1	Enable

Data Word Count Register

The CRC_DCNT register holds the word count that is used for the CRC operation. On transfer of every 32-bit word, the CRC decrements by 1 the content of this register. When the count decrements to zero, this event triggers a CRC compare action, and the CRC_DCNT register is automatically loaded from the CRC_DCNTRLD register for the next CRC operation.

Note that the initial value programmed into the CRC_DCNT register may be different from what is programmed in the CRC_DCNTRLD register.

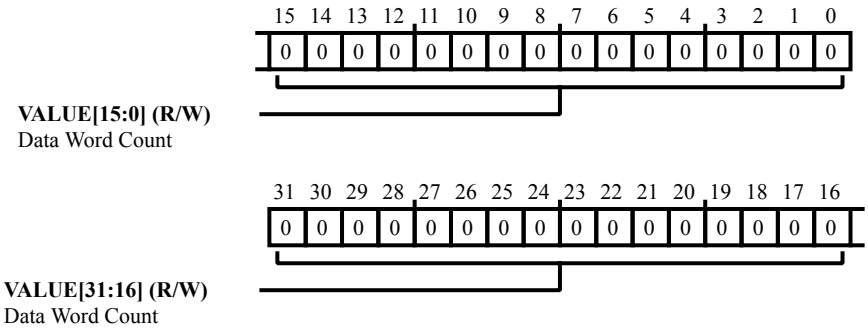


Figure 16-5: CRC_DCNT Register Diagram

Table 16-7: CRC_DCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Word Count. The CRC_DCNT.VALUE bit field holds the word count that is used for the CRC operation.

Data Count Capture Register

The CRC_DCNTCAP register captures the CRC_DCNT value when a compare operation fails in data verify mode. This capture can be used to track the position of an error in the data stream. The capture operation is enabled only if the CRC_STAT.CMPERR bit indicates no compare error. After an error occurs and the data count is captured, no further errors are logged until the CRC_STAT.CMPERR bit is cleared. To obtain the position of an error in the data stream, subtract the CRC_DCNTCAP register value from the initial CRC_DCNT.

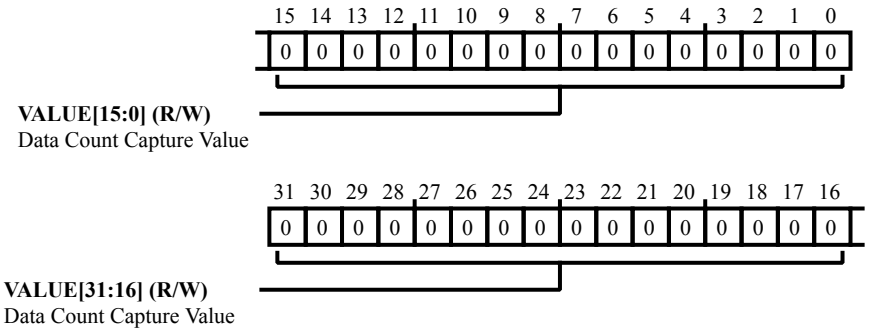


Figure 16-6: CRC_DCNTCAP Register Diagram

Table 16-8: CRC_DCNTCAP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data Count Capture Value. The CRC_DCNTCAP.VALUE bit field contains the CRC_DCNT value when a compare operation fails in data verify mode.

Data Word Count Reload Register

The CRC_DCNTRLD register holds the value that the CRC automatically loads into CRC_DCNT when the CRC_DCNT decrements to 0. At startup, the value programmed in CRC_DCNT and the CRC_DCNTRLD register could be different. So, for the first iteration, the CRC operation happens for the count initially programmed in the CRC_DCNT register. While for subsequent CRC operations, the count is taken from the CRC_DCNTRLD register.

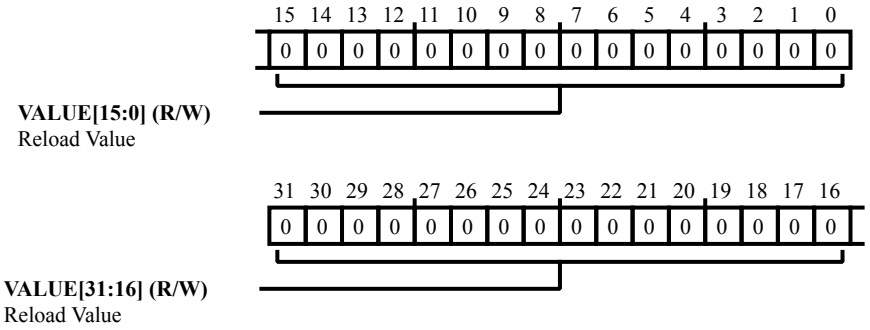


Figure 16-7: CRC_DCNTRLD Register Diagram

Table 16-9: CRC_DCNTRLD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reload Value. The CRC_DCNTRLD.VALUE bit field holds the value that automatically loads into CRC_DCNT when the CRC_DCNT decrements to 0.

Data FIFO Register

In memory transfer mode (non-data fill mode), the data from the DMA or processor core buses is written into the CRC_DFIFO on each input data grant (DMA grant or core write). Data is read from this FIFO on each output data grant (DMA grant or core read). FIFO status information is available in the CRC_STAT register. Whenever, the FIFO has valid data, output data requests are generated.

Note that in non-memory transfer mode and in data fill mode, the input data does not get written into this FIFO. So, this register should not be read in these modes.

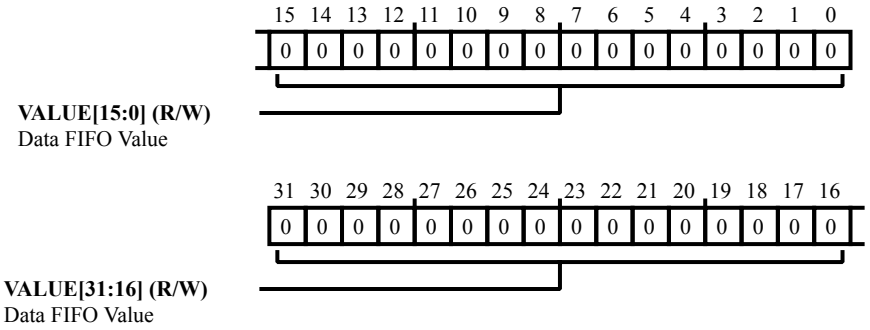


Figure 16-8: CRC_DFIFO Register Diagram

Table 16-10: CRC_DFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Data FIFO Value. The CRC_DFIFO.VALUE bit field is the data from the DMA or processor core buses.

Fill Value Register

The CRC_FILLVAL register holds the value that the CRC uses for the memory fill operation. In data fill mode, the value programmed in this register is used for the memory fill operation.

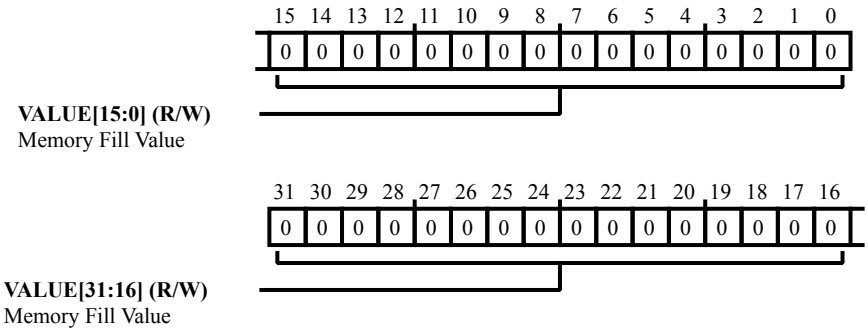


Figure 16-9: CRC_FILLVAL Register Diagram

Table 16-11: CRC_FILLVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Memory Fill Value. The CRC_FILLVAL.VALUE bit field holds the value that the CRC uses for the memory fill operation.

Interrupt Enable Register

The CRC_INEN register unmasks (enables) or masks (disables) interrupt requests generated in the CRC from going to the processor core.

Note that CRC interrupts are not disabled when the CRC is disabled (CRC_CTL.BLKEN =0).

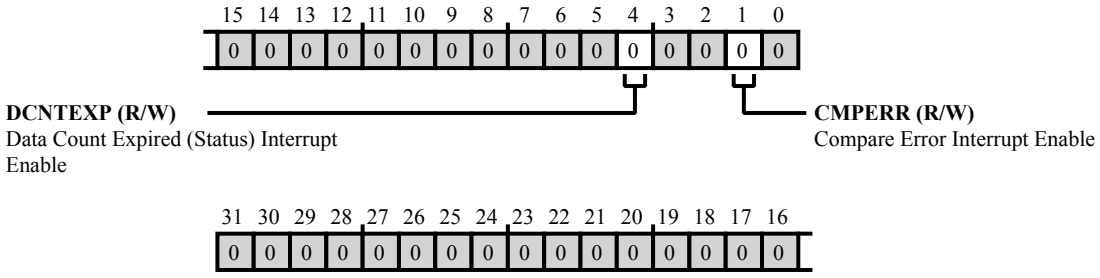


Figure 16-10: CRC_INEN Register Diagram

Table 16-12: CRC_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DCNTEXP	Data Count Expired (Status) Interrupt Enable. The CRC_INEN.DCNTEXP enables (unmasks) the data count expired (CRC status) interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
1 (R/W)	CMPERR	Compare Error Interrupt Enable. The CRC_INEN.CMPERR enables (unmasks) the data compare interrupt, which is generated when CRC data comparison fails.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

Interrupt Enable Clear Register

The CRC_INEN_CLR register permits clearing individual bits in the CRC_INEN register without affecting other bits in the register.

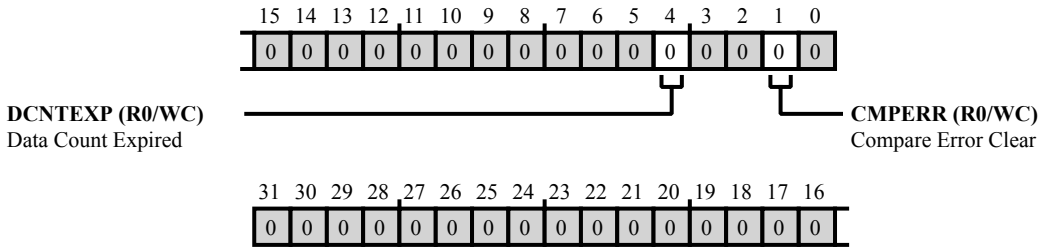


Figure 16-11: CRC_INEN_CLR Register Diagram

Table 16-13: CRC_INEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/WC)	DCNTEXP	Data Count Expired. The CRC_INEN_CLR.DCNTEXP bit clears the data count expired (status) interrupt.	
		0	No effect
		1	Clear bit
1 (R0/WC)	CMPERR	Compare Error Clear. The CRC_INEN_CLR.CMPERR bit clears the compare error interrupt.	
		0	No effect
		1	Clear bit

Interrupt Enable Set Register

The CRC_INEN_SET register permits setting individual bits in the CRC_INEN register without affecting other bits in the register.

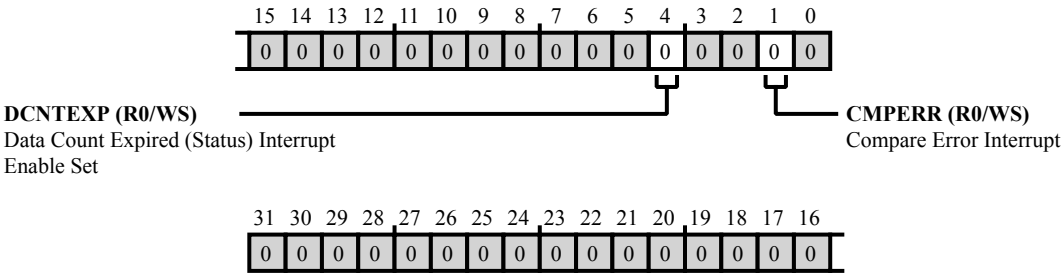


Figure 16-12: CRC_INEN_SET Register Diagram

Table 16-14: CRC_INEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/WS)	DCNTEXP	Data Count Expired (Status) Interrupt Enable Set. The CRC_INEN_SET.DCNTEXP bit sets the data count expired (status) interrupt.	
		0	No effect
		1	Set bit
1 (R0/WS)	CMPERR	Compare Error Interrupt. The CRC_INEN_SET.CMPERR bit sets the compare error interrupt.	
		0	No effect
		1	Set bit

Polynomial Register

The CRC_POLY register holds a 32-bit polynomial for CRC operations. Bit 31 corresponds to the coefficient of x31 of the CRC polynomial, bit 30 corresponds to the coefficient of x30, and so on through bit 0. A coefficient of x32 is assumed to be "1" for any polynomial that is selected. Based on the polynomial in the CRC_POLY register, the CRC generates a look-up table (LUT), which is used to compute the CRC of the incoming data stream.

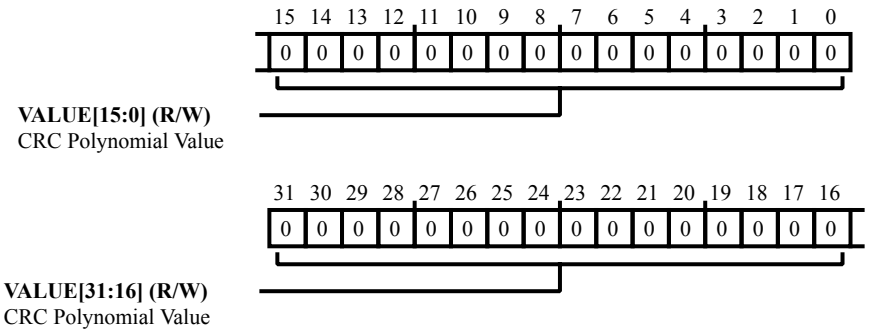


Figure 16-13: CRC_POLY Register Diagram

Table 16-15: CRC_POLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CRC Polynomial Value. The CRC_POLY.VALUE bit field holds the 32-bit polynomial for CRC operations.

CRC Current Result Register

The CRC_RESULT_CUR register holds the current or intermediate CRC result. It is updated when new data is written into the CRC. Each time the CRC_DCNT expires, the CRC loads the value from this register into the CRC_RESULT_FIN register. The CRC_RESULT_CUR register may be set to auto clear to zero or auto clear to ones when CRC_DCNT expires by configuring the CRC_CTL.AUTOCLRZ and CRC_CTL.AUTOCLRF bits. Before starting a CRC operation, the CRC_RESULT_CUR register should be programmed to the desired value.

Note that this register can be read by the processor core at any time.

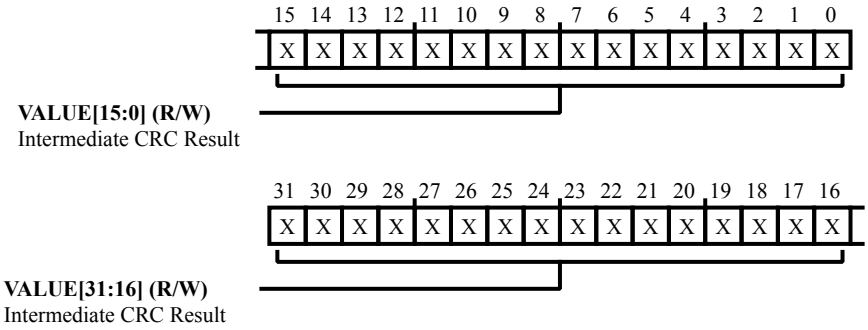


Figure 16-14: CRC_RESULT_CUR Register Diagram

Table 16-16: CRC_RESULT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Intermediate CRC Result. The CRC_RESULT_CUR.VALUE bit field holds the current or intermediate CRC result.

CRC Final Result Register

The CRC_RESULT_FIN register holds the final CRC computed for a data stream. A data stream is a DMA of CRC_DCNT number of words into the CRC. When CRC_DCNT decrements to zero for each data stream, the CRC loads the CRC_RESULT_FIN register with the value from the CRC_RESULT_CUR register.

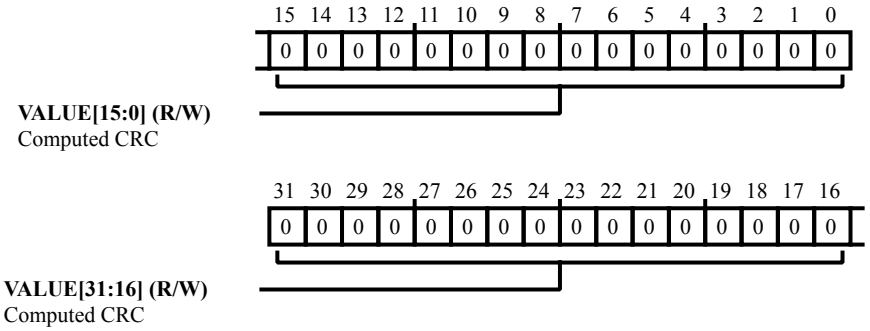


Figure 16-15: CRC_RESULT_FIN Register Diagram

Table 16-17: CRC_RESULT_FIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Computed CRC. The CRC_RESULT_FIN.VALUE bit field holds the final CRC computed for a data stream.

Status Register

The CRC_STAT register indicates the status for CRC operations and interrupt generation.

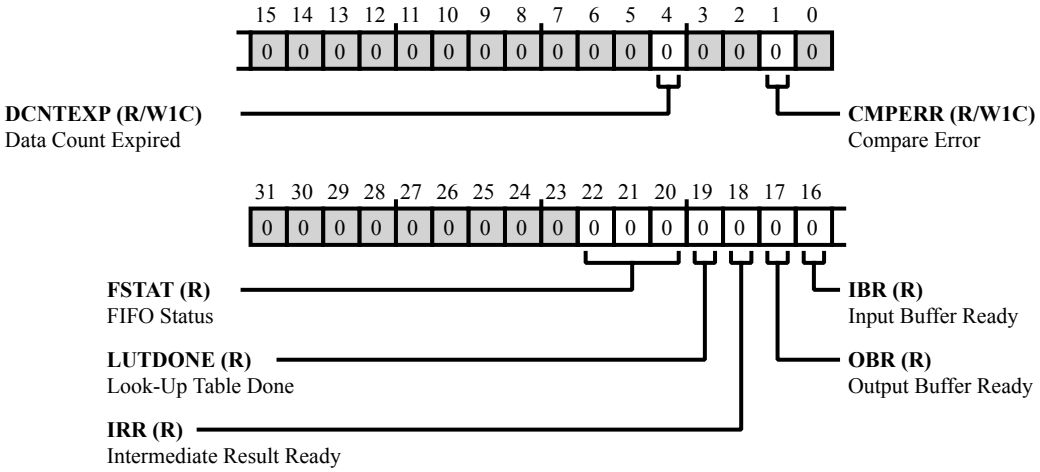


Figure 16-16: CRC_STAT Register Diagram

Table 16-18: CRC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22:20 (R/NW)	FSTAT	FIFO Status. The CRC_STAT.FSTAT indicates the current FIFO status. This field is read-only.	
		0	FIFO empty
		1	FIFO has 1 data
		2	FIFO has 2 data
		3	FIFO has 3 data
		4	FIFO has 4 data (full)
19 (R/NW)	LUTDONE	Look-Up Table Done. The CRC_STAT.LUTDONE bit indicates that the CRC has generated the look-up table for the current polynomial. This read-only bit is cleared at reset and cleared when the CRC_POLY is written.	
		0	No status
		1	LUT generation done

Table 16-18: CRC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/NW)	IRR	Intermediate Result Ready. The CRC_STAT.IRR bit indicates that the CRC has updated the CRC_RESULT_CUR register with intermediate CRC results for the new data written to the CRC. The processor core should read from the CRC_RESULT_CUR register only after detecting CRC_STAT.IRR =1. This read-only bit is cleared by CRC hardware and is valid when the CRC_CTL.IRRSTALL bit is enabled.	
		0	No status
		1	Intermediate results ready
17 (R/NW)	OBR	Output Buffer Ready. The CRC_STAT.OBR bit indicates that the CRC has data ready for the processor core to read. The processor core should read from the CRC only after detecting CRC_STAT.OBR =1. This read-only bit is cleared by CRC hardware.	
		0	No status
		1	Output buffer ready
16 (R/NW)	IBR	Input Buffer Ready. The CRC_STAT.IBR bit indicates that the CRC is ready to accept a processor core write. The processor core should write to the input register only after detecting that CRC_STAT.IBR =1. This read-only bit is cleared by CRC hardware.	
		0	No status
		1	Input buffer ready
4 (R/W1C)	DCNTEXP	Data Count Expired. The CRC_STAT.DCNTEXP bit indicates that the CRC_DCNT has expired. This W1C bit is not automatically cleared when the CRC is disabled (CRC_CTL.BLKEN =0). When the CRC sets this bit on CRC_DCNT expiry, the CRC generates the CRC_INEN.DCNTEXP interrupt.	
		0	No status
		1	Data counter expired
1 (R/W1C)	CMPERR	Compare Error. The CRC_STAT.CMPERR bit indicates that a CRC mismatch or data mismatch has been detected. This W1C bit is not automatically cleared when the CRC is disabled (CRC_CTL.BLKEN =0). When the CRC sets this bit on detecting a mismatch, the CRC generates the CRC_INEN.CMPERR interrupt. While this bit is set, the CRC_DCNTCAP register is disabled from capturing the data count values.	
		0	No status
		1	Compare error

17 Direct Memory Access (DMA)

The processor architecture distributes the DMA channels throughout the infrastructure. Often, the channels cluster together through system crossbars (SCB), sharing a single interface with the main system crossbar.

The DMA channels can perform transfers between memory and a peripheral or between one memory and another memory. Memory-to-memory DMA transfers (MDMA) require two DMA channels. One channel is the source channel, and the second, the destination channel.

All DMA channels can transport data to and from virtually all on-chip and off-chip memories.

DMA transfers on the processor use either a descriptor-based method or register-based method. Register-based DMA allows the processor directly to program DMA controller registers to initiate a DMA transfer. On completion, the controller registers can automatically update with their original setup values for continuous transfer, if needed. Descriptor-based DMA transfers require a set of parameters stored within memory to initiate a DMA sequence. Descriptor-based transfers allow the chaining together of multiple DMA sequences. In descriptor-based DMA operations, DMA channel programming can automatically set up and start another DMA transfer after the current sequence completes.

The DMA channel does not connect external memories and devices directly. Rather, data passes through an external-memory interface port. DMA operations can access any device the external memory interface supports. These interfaces typically include:

- Flash memory
- SRAM
- FIFOs
- Memory-mapped peripheral devices
- Dynamic Memory (if present)

DMA Channel Features

The processor uses Direct Memory Access (DMA) to transfer data within memory spaces or between a memory space and a peripheral. The processor can specify data transfer operations and return to normal processing while the fully integrated DMA channel carries out the data transfers independent of processor activity. The DMA channels are dispersed throughout the infrastructure and interface with the system crossbar unit (SCB).

The following is a list of DMA interface features.

- Supports integer byte strides including byte strides of 0 and negative byte strides
- Register-based configuration
  - Core writes DMA configuration
  - Supports automatic reloading for continuous operation
- Flexible descriptor-based configuration
  - DMA descriptors are fetched from memory
  - Support for variable descriptor sizes
- Flexible flow control – Transitions between the various descriptor-based modes and for DMA termination
- Orthogonal transfers
  - Support for three transfer dimensions
  - 1D and 2D transfers supported per descriptor set
  - 3D support provided by chained descriptor sets
- Configurable memory and peripheral-transfer word sizes
  - Memory interface supports 8-bit, 16-bit, 32-bit, 64-bit, 128-bit, and 256-bit transfers
  -
  - Peripheral interface supports for 8-bit, 16-bit, and 32-bit transfers
- Interrupt notification
  - Row or work unit completion
  - Error conditions
- Incoming and outgoing trigger support
  - Trigger generation for row or work unit completion
  - Work unit can wait for incoming trigger
- MMR access bus – Provides access to memory-mapped registers for configuration, monitoring, and debug
- SCB crossbar interface connects the DMA channel to the system crossbar
- Peripheral DMA bus – Interfaces the DMA channel to a peripheral or another DMA channel
- Peripheral data-request interrupt support
- Bandwidth monitoring and limiting for MDMA channels

DMA Channel Functional Description

This section provides a functional description of the DMA channel interface.

CM41X_M4 DMA Register List

The DMA channel controller (DMA) supports data transfers within memory spaces or between a memory space and a peripheral. The processor can specify data transfer operations and return to normal processing while the fully integrated DMA channel carries out the data transfers independent of processor activity. The DMA channels are dispersed throughout the infrastructure, as DMAn's. A set of registers governs DMA operations. For more information on DMA functionality, see the DMA register descriptions.

Table 17-1: CM41X_M4 DMA Register List

Name	Description
DMA_ADDRSTART	Start Address of Current Buffer Register
DMA_ADDR_CUR	Current Address Register
DMA_BWLCNT	Bandwidth Limit Count Register
DMA_BWLCNT_CUR	Bandwidth Limit Count Current Register
DMA_BWMCNT	Bandwidth Monitor Count Register
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current Register
DMA_CFG	Configuration Register
DMA_DSCPTR_CUR	Current Descriptor Pointer Register
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor Register
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer Register
DMA_STAT	Status Register
DMA_XCNT	Inner Loop Count Start Value Register
DMA_XCNT_CUR	Current Count (1D) or Intra-row XCNT (2D) Register
DMA_XMOD	Inner Loop Address Increment Register
DMA_YCNT	Outer Loop Count Start Value (2D only) Register
DMA_YCNT_CUR	Current Row Count (2D only) Register
DMA_YMOD	Outer Loop Address Increment (2D only) Register

CM41X_M4 DMA Channel List

Table 17-2: CM41X_M4 DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPI0_TXDMA	SPI0 TX DMA Channel
DMA1	SPI0_RXDMA	SPI0 RX DMA Channel

Table 17-2: CM41X_M4 DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA2	UART0_TXDMA	UART0 Transmit DMA
DMA3	UART0_RXDMA	UART0 Receive DMA
DMA4	UART1_TXDMA	UART1 Transmit DMA
DMA5	UART1_RXDMA	UART1 Receive DMA
DMA6	SPI1_TXDMA	SPI1 TX DMA Channel
DMA6	UART2_TXDMA	UART2 Transmit DMA
DMA7	SPI1_RXDMA	SPI1 RX DMA Channel
DMA7	UART2_RXDMA	UART2 Receive DMA
DMA8	HAE0_RXDMA_CH0	HAE0 RX0: DMA->HAE DATAI
DMA8	UART3_TXDMA	UART3 Transmit DMA
DMA9	HAE0_TXDMA	HAE0 TX: HAE->DMA
DMA9	UART3_RXDMA	UART3 Receive DMA
DMA10	HAE0_RXDMA_CH1	HAE0 RX1: DMA->HAE DATAV
DMA10	SPORT0_A_DMA	SPORT0 Channel A DMA
DMA10	UART4_TXDMA	UART4 Transmit DMA
DMA11	SPORT0_B_DMA	SPORT0 Channel B DMA
DMA11	UART4_RXDMA	UART4 Receive DMA
DMA12	MDMA0_SRC	MDMA0 Memory DMA Stream n Source Channel
DMA13	MDMA0_DST	MDMA0 Memory DMA Stream n Destination Channel

DMA Definitions

To make the best use of the DMA controller, it is useful to understand the following terms.

Descriptor

An individual configuration fetched from memory that maps to a single register within a DMA channel.

Descriptor Fetch

The action of retrieving descriptors from memory through memory read operations and loading then into the DMA channel registers upon their read return.

Descriptor Set

A group of descriptors associated with a single work unit. See Descriptor-Based Flow Modes .

Disabled State

The channel is disabled because the enable bit = 0 or as a result of an error.

DMAC

An acronym used for a DMA cluster.

DMA Channel

A single DMA engine that has all the capabilities and registers as defined for a given processor. A DMA channel or engine is connected to a single peripheral.

DMA Cluster

A grouping of multiple DMA channels with a shared SCB crossbar interface, controller, and arbiter. Also known as a DMAC.

Initial Descriptor

The first descriptor in the descriptor set.

MDMA

Memory-to-Memory DMA data transfer. Two DMA channels are paired to perform a memory read from one address location and a memory write of that data to another address location.

Stop State

A time where the channel is enabled but not currently programmed to perform a data transfer. Programming the flow to STOP causes the channel to enter the stop state at the end of the work unit.

User

Any person, debug, emulator, software routine, or action taken by the core that accesses the MMR registers of the DMA channel or peripherals, or sets up data and descriptors in memory.

Wait State

If instructed to wait for a trigger, the channel enters this state once it has completed a work unit. The channel remains in this state until a trigger occurs. If a trigger came in before reaching the wait state, the channel skips over the wait state upon completion of the work unit.

Work Unit

A single data transaction or series of data transactions performed based on the configuration of the DMA channel. For autobuffer mode, a new work unit is defined at the time all current count registers are initialized to start values. Once all the current count registers count down to zero, the work unit has completed.

Work Unit Chain

A single work unit or a series of work units separated by a `STOP` or disabled state. The work units in the chain are programmed to another descriptor flow. The last work unit in the chain is programmed to a flow of `STOP` or `AUTO`. `STOP` terminates the state at the end of that work unit. `AUTO` must be be terminated by disabling the DMA channel. A work unit chain is also known as a descriptor chain.

Block Diagram

The ***DMA Channel Block Diagram*** shows the functional blocks within the DMA interface.

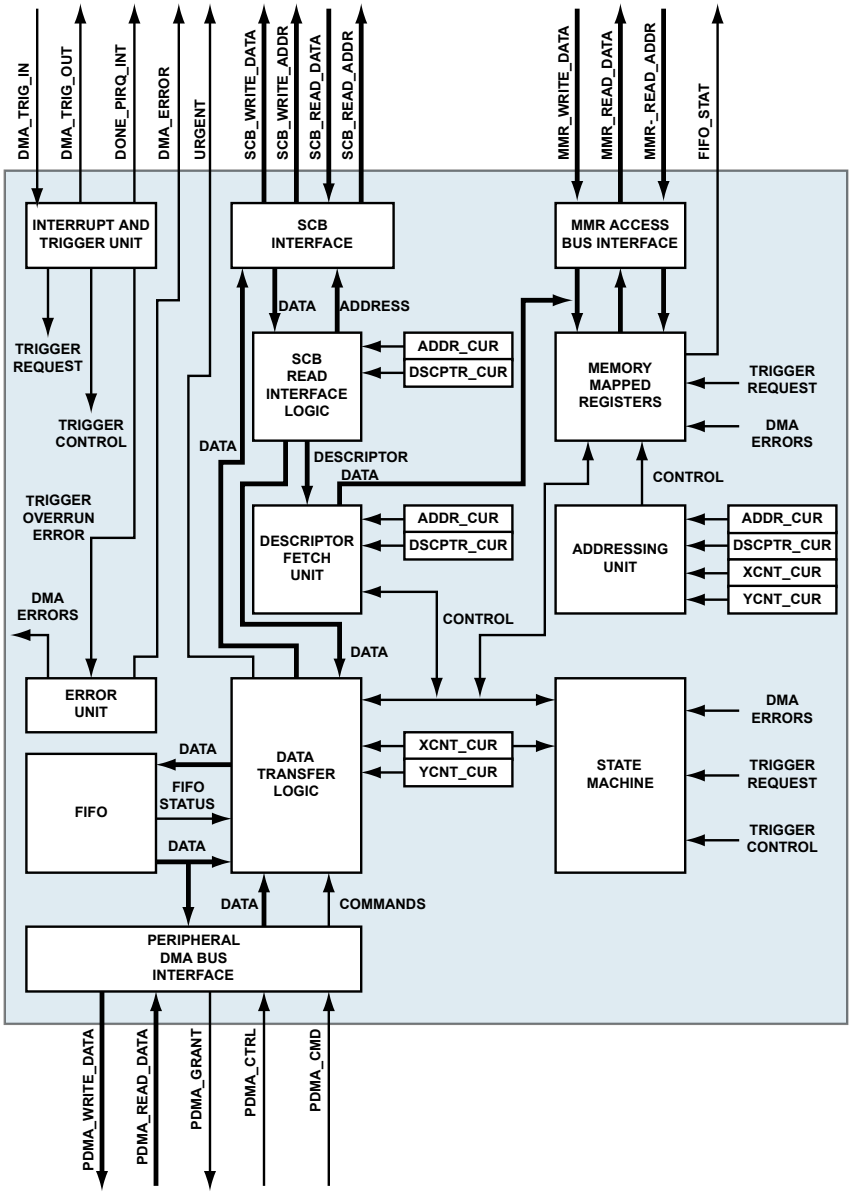


Figure 17-1: DMA Channel Block Diagram

For more information on the interfaces, see:

- DMA Channel Peripheral DMA Bus
- DMA Channel MMR Access Bus
- DMA Channel Event Control
- DMA Channel SCB Interface

Architectural Concepts

The DMA channel provides a method to transfer data between memory spaces or between memory and a peripheral using a number of system interfaces. The DMA channel provides an efficient method of distributing data

throughout the system, freeing up the processor core for other operations. Each peripheral that supports DMA transfers has its own dedicated DMA channel or channels with its own register set. The register set configures and controls the operating modes of the DMA transfers.

DMA Controller Multiplexing

The 14 DMA controllers are associated with one or more specific peripherals. Each controller supports the transfer of one independent channel of data to or from memory. To assign a shared DMA controller to a specific peripheral, program the appropriate selection in advance in the SYSBLK_DMA_MUXCTL register.

In general, all DMA-capable peripherals have more than one data channel—some have two (RX and TX), and the HAE has three. When using a peripheral, all associated data channels for the peripheral must be assigned to the appropriate DMA controller. The processor does not support assigning a subset of a peripheral's DMA controllers to that peripheral and the other controllers to another peripheral.

Some peripherals, the UART for example, support programming models for both DMA-driven and non-DMA (interrupt-driven) transport operation. When using a peripheral, a DMA controller must be assigned in all cases, even if not using a DMA-driven operation. In part this is because the DMA controller routes the DMA-request/data-interrupt signal from the peripheral to the core, and without this interrupt connection, interrupt-driven operation does not work.

Before changing the assignment of a shared DMA controller, ensure all of the DMA transactions are complete and the controller is disabled (DMA_CFG.EN =0).

DMA Channel SCB Interface

The SCB interface connects the DMA channel to the SCB crossbar allowing for transfers to and from the processors internal memory and other suitable system resources.

The DMA channel connects to the system interconnect through the SCB interface. This connection lets the DMA channel perform work-unit data transfers with memories such as L1, L2 (internal), and L3 (external). In addition to work unit data transfers, the SCB interface also is used for fetching descriptor sets for all the descriptor-based transfer modes.

The DMA channel can support data bus widths of 16, 32, 64, or 128 bits. The data bus widths for a given DMA channel on a specific processor can vary and are not configurable. Read the DMA_STAT.MBWID field to determine the assigned bus widths.

SCB Interface Signals

The DMA channel operates at one of the *SCLKn* frequencies, as does the SCB interface. SCLK0 clocks all but four DMA channels which are clocked by SCLK1. The SCB crossbar handles the internal arbitration of the transfer requests of all the masters interfaced to the SCB crossbar instance.

The processor's SCB data buses are 32 bits. System ECC memories work most efficiently with aligned 32-bit access. Word and byte accesses are fully supported but are less efficient.

Data Address Alignment

To prevent addressing errors and to maximize bandwidth of the SCB interface to the DMA channel, data addresses align with a multiple of the programmable memory size of the DMA channels configuration. These configuration options appear in the Descriptor Set Address Alignment table.

There are situations in which entire work units may not transfer at the maximum configurable memory size. In this case, the entire work unit can transfer by reducing the configured memory size at the expense of bus bandwidth using descriptor sets as follows:

- The first descriptor set can be configured to transfer data until the larger memory size alignments are met.
- A second descriptor set with a larger memory size configuration then can be used to transfer the bulk of the data in the work unit.
- Finally, a third descriptor set can be used with a smaller memory size to complete any final data transfers that cannot meet the alignment requirements of the previous descriptor set configuration.

Table 17-3: DMA Channel Address Alignment Requirements

Configured Memory Size	Address Restriction
1 Byte	No restriction
2 Bytes	ADDR[0] == 0
4 Bytes	ADDR[1:0] == 0

Descriptor Set Address Alignment

All descriptor set addresses and descriptors within a descriptor set must align to a 32-bit address. For descriptor set fetches, the DMA engine ignores the memory-size configuration of the DMA channel. This feature avoids the need to align descriptor sets based on the memory width configuration of the previous descriptor set.

For descriptor sets containing only a single descriptor, the transfer takes place as a single 32-bit transfer. For descriptor sets containing multiple descriptors, the DMA engine fetches each 32-bit descriptor individually and treats it as multiple 32-bit transfers.

DMA Channel Peripheral DMA Bus

The DMA channel connects to peripherals or other DMA channels through the peripheral DMA bus. This bus is a dedicated point-to-point interface that supports a 32-bit bus width . The data bus widths for a given DMA channel on a particular processor can vary and are not configurable. Reading the `DMA_STAT.PBWID` field permits determining the assigned bus width.

The DMA channel operates at one of the *SCLKn* frequencies, as does the peripheral DMA bus. The ***Peripheral DMA Bus Signals*** table provides descriptions of the peripheral DMA bus signals.

Table 17-4: Peripheral DMA Bus Signals

Signal	Width (bits)	Description
PDMA_WRITE_DATA	32	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.PBWID`.
PDMA_READ_DATA	32	Data bus used for read operations. The width of the bus can be determined from `DMA_STAT.PBWID`.
PDMA_DMA_GRANT		Control signals to indicate that data is valid for DMA channel read operations (peripheral transmit). These signals indicate that the DMA channel is ready to receive data for write operations (peripheral receive).
PDMA_CMD	3	The peripheral uses the signal for issuing DMA channel control commands.
PDMA_CTRL		The peripheral uses the control signals to send various commands to the DMA channel and control the direction of flow.

Peripheral Control Commands

The peripheral DMA bus of the DMA channel provides a means for peripherals on the processor to issue commands to the DMA channel. These commands provide greater control over the DMA channel operation. This control improves real-time performance and relieves control and interrupt demands on the core. Peripherals can send commands to the DMA controller over the 3-bit PERI_CMD bus. The DMA control commands extend the set of operations available to the peripheral beyond the simple “request data” command used by peripherals in general. Refer to the appropriate peripheral chapter for a description on how that peripheral uses DMA control commands.

These DMA control commands (see the ***PDMA_CMD Peripheral DMA Control Commands*** table) are not visible to or controlled by the program. But, their use by a peripheral has implications for the structure of the DMA transfers that the peripheral can support. It is important to write application software such that it complies with certain restrictions, regarding work units and descriptor chains. Complying with this guideline makes the peripheral operate properly whenever it issues DMA control commands.

The ***PDMA_CMD Peripheral DMA Control Commands*** table describes the commands the DMA controller issues. The following sections describe these commands in more detail.

Table 17-5: PDMA_CMD Peripheral DMA Control Commands

Command	Name	Description
b#000	NOP	No operation
b#001	Restart	Restarts the current work unit from the beginning
b#010	Finish	Finishes the current work unit and starts the next
b#011	Interrupt	Immediately sets the DMA completion interrupt in the DMA channel
b#100	Request Data	Typical DMA data request
b#101	Request Data Urgent	Urgent DMA data request

Table 17-5: PDMA_CMD Peripheral DMA Control Commands (Continued)

Command	Name	Description
b#110	Reserved	Reserved
b#111	Reserved	Reserved

Idle Command

The DMA channel drives this command when the enabled peripheral has no data requests required.

Restart Command

This command causes the current work unit to interrupt processing and start again, using the addresses and count values from the DMA_ADDRSTART, DMA_XCNT, and DMA_YCNT registers. The DMA controller does not signal an interrupt request when the work unit terminates.

If a channel programmed to transmit (memory read) receives a restart command, the channel momentarily pauses, permitting any pending memory reads initiated before the restart command to complete. During this period, the channel does not grant DMA requests. After all pending reads flush from the pipelines of the channel, the channel resets its counters and FIFO, and then starts pre-fetch reads from memory. The DMA controller grants data requests from the peripheral as soon as new prefetched data is available in the DMA FIFO. In this case, the peripheral can use the restart command to reattempt a failed transmission of a work unit.

If a channel programmed to receive (memory write) receives a restart command, the channel stops writing to memory, discards any data held in its DMA FIFO, and resets its counters and FIFO. As soon as this initialization is complete, the channel again grants DMA write requests from the peripheral. In this case, the peripheral can use the restart command to abort the transfer of received data into a work unit, and reuse the memory buffer for a later data transfer.

The request from the restart control command is not granted or acknowledged. The DMA controller always accepts the request.

Finish Command

The finish command causes the current work unit to terminate processing and move on to the next work unit. If enabled within the DMA_CFG register, the DMA channel signals an interrupt or a trigger event. The peripheral can then use the finish command to partition the DMA stream into work units on its own. This partitioning occurs---perhaps as a result of parsing the data currently passing though its supported communication channel---without direct real-time control by the processor.

When a DMA channel programmed to transmit (memory read) then receives a finish command, the channel momentarily pauses for the completion of any pending memory reads, which were initiated prior to the finish command. During this time, the channel does not grant DMA requests. After the flush of all pending reads from the pipelines of the channel, the channel signals an interrupt request or a trigger (if enabled) and begins fetching the next descriptor (if any). DMA data requests from the peripheral are granted as soon as new prefetched data is available in the DMA FIFO.

If a channel programmed to receive (memory write) then receives a finish command, the channel stops granting new DMA requests while it drains its FIFO. The channel writes to memory any DMA data received by the DMA channel prior to the finish command. When the FIFO reaches an empty state, the channel signals an interrupt or a trigger (if enabled) and begins fetching the next descriptor (if any). After fetching the next descriptor, the channel initializes its FIFO, then resumes granting DMA requests from the peripheral.

The finish command request is not granted or acknowledged. The request is always accepted by the DMA channel.

Interrupt Command

The interrupt command causes the DMA channel to generate an interrupt request. When programming the channel to support this command, configure the `DMA_CFG.INT` bit field to PIRQ mode. This configuration directs the channel not to generate interrupt requests based on the work unit state. Instead, the channel generates interrupts only when it receives the interrupt command from the peripheral. When the channel receives an interrupt request command, the `DMA_STAT.PIRQ` bit indicates the event under the following conditions:

- The `DMA_CFG.EN` bit enables the DMA channel.
- The DMA channel is in the stop state.
- The interrupt in `DMA_CFG.INT` is configured for PIRQ mode.

The peripheral only issues the interrupt command in response to receiving the last grant command from the DMA channel, indicating that the transfer is the last transfer in the work unit.

Request-Data Command

The request data command is a request for data transfers between the DMA channel and the peripheral. The request is held by the peripheral until granted or acknowledged by the DMA channel.

Request-Data Urgent Command

The request-data urgent command behaves identically to the request data command, except that---during the commands assertion---the DMA channel performs its memory accesses with urgent priority. This priority includes both data and descriptor fetch memory accesses. For example, a DMA management capable peripheral can use this control command if an internal FIFO approaches a critical condition.

The request is held by the peripheral until granted or acknowledged by the DMA channel.

Peripheral-Control Command Restrictions

The proper operation of the DMA channel FIFO leads to certain restrictions in the sequence of DMA peripheral control commands issued by a peripheral. The following sections describe these restrictions.

Transmit-Restart or Transmit-Finish Command

A peripheral only can issue a restart or finish control command to a channel configured for memory read under the following conditions:

- The peripheral has already performed at least one DMA transfer in the current work unit.
- The current work unit has (`FIFO_SIZE`/`DMA_CFG.MSIZE`) + 1 memory transfers remaining.

The first item ensures that the work unit has started. The second item ensures that the work unit has not completed. The second item is sufficiently large that it is always at least five more than the maximum data count before any restart or finish command. If using restart or finish commands to manage a work unit, this requirement implies that the work unit must have DMA_XCNT_CUR and DMA_YCNT_CUR register values representing at least five data items.

To satisfy the second item, ensure that the number of memory transfers described by the descriptor is (FIFO_SIZE/DMA_CFG.MSIZE) + 1 larger than the maximum number of memory transfers expected.

Receive-Restart or Receive-Finish Commands

A peripheral only can issue a restart or finish control command to a channel configured for memory write under the following conditions:

- The number of peripheral transfers completed is less than (DMA_CFG.MSIZE/DMA_CFG.PSIZE) × (transfers described by descriptor).
- In addition to the previous condition, one of the following conditions also must apply:
  - A finish command terminated the previous work unit, *and* the peripheral has done at least one transfer in the current work unit.
  - The peripheral has done (FIFO_SIZE/DMA_CFG.PSIZE) + 1 transfers in the current work unit.

The first condition ensures that the descriptor is still active. The second set of conditions ensures that data from the previous descriptor has left the FIFO and that the current descriptor has started.

Memory DMA and Triggering

A memory DMA (MDMA) channel provides a means of doing memory-to-memory DMA transfers among the various memory spaces that have DMA support.

The DMA controller implements memory DMA (MDMA) channels by interfacing two DMA channels through the peripheral DMA bus interface. One DMA channel serves for memory read operations, and the second channel servers for memory writes. Depending on the processor, a memory DMA channel can have an additional peripheral, such as a CRC peripheral. The additional peripheral is inserted into the peripheral DMA bus that optionally can be enabled.

MDMA channels utilize one predefined controller for memory read operations (MDMA_SRC) and a second predefined controller for memory write operations (MDMA_DST).

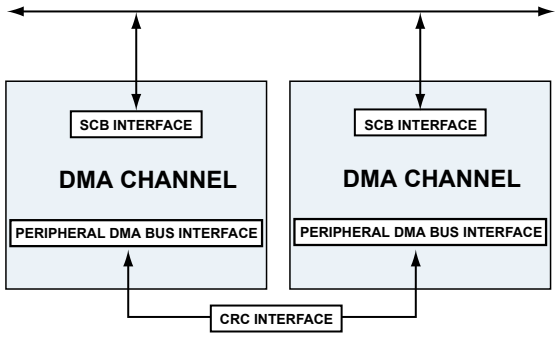


Figure 17-2: MDMA Channel Dedicated Pair

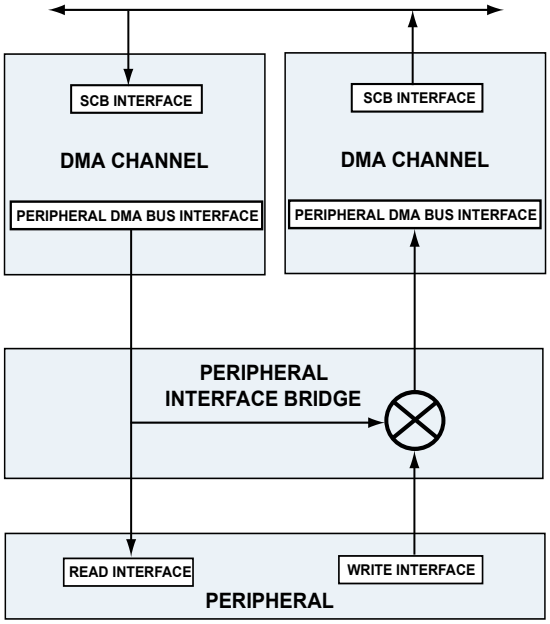


Figure 17-3: MDMA Channel Pair with Peripheral

A memory-to-memory transfer always requires enabled source and destination channels. Because the channels interface through the peripheral DMA bus and can have an additional peripheral inserted into the peripheral DMA bus, programs must make sure to set the same values in the `DMA_CFG.PSIZE` of both the source and destination channels.

The memory DMA channels support the full range of the `DMA_CFG.MSIZE` options for the DMA transfers to and from the memories.

Because the MDMA channel consists of two DMA channels, the entire MDMA channel has two sets of FIFOs, one in the read channel and one in the write channel. This FIFO usage allows for more efficient bursting of both read and write transactions using the available bandwidth. While the `DMA_CFG.PSIZE` configuration must be identical for both source and destination DMA channels, this restriction does not apply for the `DMA_CFG.MSIZE` configuration.

Configure the DMA_CFG.PSIZE bits to a value no larger than the supported bus width of the peripheral DMA bus.

The independent source and destination DMA channels also have their own dedicated interrupt and trigger events. While it is normal practice to have only event generation performed at destination DMA completion, programs also can use other means of interrupt generation.

Configuration of an MDMA transfer is done in a similar manner to peripheral DMA transfers, except for writing two DMA channel registers instead of one.

To control the pace of data transfers, use triggers on either the memory read or the memory write channel pair used in an MDMA operation. Setting the DMA_CFG.TWAIT bit in the memory read channel prevents both channels from transferring data before the system is ready. However, only configuring the memory write channel to wait for a trigger allows for data fetch from the memory in anticipation of the memory write operation.

DMA Channel MMR Access Bus

The MMR access bus provides access to all the DMA channels memory-mapped registers for DMA channel configuration, monitoring, and debug. The interface has a fixed 32-bit data bus for read and write accesses.

The ***MMR Access Bus Signals*** table provides descriptions of the MMR access bus signals.

Table 17-6: MMR Access Bus Signals

Signal	Width (bits)	Description
MMR_WRITE_DATA	32	Data bus used for write operations to the MMRs from the core
MMR_READ_DATA	32	Data bus used to return read data from the MMRs
MMR_READ_ADDR	7	Address used to select the MMR to access

DMA Channel Operation Flow

A detailed description of the flow of operation of the DMA channel appears in the following topics:

- Startup Flow
- Refresh Flow
- DMA Operating Modes
- Stop Mode
- DMA Channel Errors

Startup Flow

Enabling a DMA operation on a given channel first requires directly writing some or all of the DMA parameter registers. The minimum set of register required to be initialized depends on the desired mode of operation as described in the following sections.

Startup Minimum-Enable Requirements

To start a DMA operation on a given channel, some or all of the DMA parameter registers must first be initialized and configured to the desired DMA channels operating mode.

- For descriptor-array-based flow modes, at minimum, write the DMA_DSCPTR_CUR register prior to writing to the DMA_CFG register, which is the special action required to start the DMA channel.
- For descriptor-list-based flow modes, at minimum, write the DMA_DSCPTR_NXT register prior to writing to the DMA_CFG register, which is the special action required to start the DMA channel.
- For non-descriptor-based flow modes, write the DMA_ADDRSTART, DMA_XCNT, and DMA_XMOD registers prior to writing the DMA_CFG register.

Programs can write other registers that can remain static throughout the course of the DMA activity. The write to the DMA_CFG register begins the DMA operation.

ATTENTION: When software directly writes the DMA_CFG register, the DMA controller recognizes this action as the special startup condition. This condition occurs when starting the DMA controller for the first time on this channel or occurs after the DMA channel stops. It is possible for the channel to flag a DMA error condition regardless of the DMA_CFG.EN bit setting.

Startup Operation

The startup operation is initiated by software directly writing the DMA_CFG register when starting DMA for the first time on a channel or after the channel has entered to the stop state.

When the descriptor fetch is complete and the DMA channel is enabled, the DMA_CFG descriptor element in the DMA_CFG register assumes control. Before this point, the direct write to the DMA_CFG register had control.

At startup, the selected flow mode and the descriptor size determine the course of the DMA initialization process. The DMA_CFG.FLOW field determines whether to load more current registers from descriptor sets in memory. The DMA_CFG.NDSIZE field details how many descriptor elements to fetch before starting the DMA operation. This process does not affect DMA registers that are not in the descriptor; no modifications are made to their prior values.

For descriptor-list flow modes, the channel copies the DMA_DSCPTR_NXT register value into the DMA_DSCPTR_CUR register. Then, the channel fetches new descriptor elements from memory. The DMA_DSCPTR_CUR register indexes each fetch, and the channel increments the index after each fetch. After completion of the descriptor fetch, the DMA_DSCPTR_CUR register points to the next 32-bit word in memory past the end of the descriptor.

If the descriptor fetch is for a descriptor-array mode transfer, the channel does *not* copy the DMA_DSCPTR_NXT register into the DMA_DSCPTR_CUR register. *Instead*, the descriptor fetch indexing begins with the value in the DMA_DSCPTR_CUR register.

If DMA_CFG is not part of the fetched descriptor set, the previous value (originally as written on startup) controls the work unit operation. If the DMA_CFG register is part of the fetched descriptor set, the value programmed by the MMR access controls only the loading of the first descriptor fetched from memory. The configuration of the DMA_CFG register controls the subsequent DMA work units of the fetched descriptor set.

After the descriptor fetch is complete or if the flow configuration was originally for one of the register-based flow modes, the DMA operation begins. The DMA channel immediately fills its FIFO. For a memory-write operation, the DMA channel begins accepting data from the peripheral. For a memory-read operation, the DMA channel begins memory reads when the SCB bus grants access to the DMA channel.

When the DMA channel performs its first data-memory access, its address and count computations take their input operands from the start registers. These registers can include DMA_ADDRSTART, DMA_XCNT, and DMA_YCNT, if necessary. The channel writes results back to the current registers. These registers include DMA_ADDR_CUR, DMA_XCNT_CUR, and DMA_YCNT_CUR. Note that the current registers are not valid until the channel performs the first memory access, which can be some time after the write to the DMA_CFG register starts the channel. Once started, the channel automatically loads the current registers from the appropriate descriptor elements, overwriting their previous contents. These automatic-load operations include:

- The channel copies the DMA_ADDRSTART value to DMA_ADDR_CUR.
- The channel copies the DMA_XCNT value to DMA_XCNT_CUR.
- The channel copies the DMA_YCNT to DMA_YCNT_CUR.

Refresh Flow

When the channel completes processing of a work unit, the DMA channel performs the following operations:

- Completes the transfer of all data between memory and the DMA channel.
- Performs a synchronized transition (if the DMA channel configuration is a memory read operation with the DMA_CFG.SYNC bit enabled) *and* transfers all data to the peripheral before continuing.
- Forwards the signals from the DMA channel (if interrupts or triggers are enabled) *and* updates the DMA_STAT register to indicate the interrupt request or trigger events.
- Clears the DMA_STAT.RUN bit field to stop DMA operation (if the flow was set to stop mode) *and* transfers any remaining data in the FIFO of the DMA channel to the peripheral.
- Loads a new descriptor from memory into the DMA registers by way of the contents of the DMA_DSCPTR_CUR register (for descriptor-array mode) *and* increments the DMA_DSCPTR_CUR register

  The channel takes the descriptor size from the DMA_CFG.NDSIZE value before the fetch.
- Copies the DMA_DSCPTR_NXT register into the DMA_DSCPTR_CUR register (for descriptor-list mode), fetches the descriptor from the new contents of the DMA_DSCPTR_CUR register, *and* places these contents into the DMA registers while incrementing the DMA_DSCPTR_CUR register.
- Checks for detection of an incoming trigger event (for descriptor-on-demand array mode):
  - If the channel detects a trigger event, the DMA channel loads a new descriptor from memory into the DMA registers from the contents of the DMA_DSCPTR_CUR register, while incrementing the DMA_DSCPTR_CUR register. The channel takes the descriptor size from the DMA_CFG.NDSIZE value before the fetch.

    - If the channel detects no trigger event, the DMA channel begins the next work unit by reloading the current registers.
- Checks for detection of an incoming trigger event (for descriptor-on-demand list mode):
    - If the channel detects a trigger event, the DMA channel copies the DMA_DSCPTR_NXT register value to the DMA_DSCPTR_CUR register, fetches the descriptor memory from the DMA_DSCPTR_CUR register, *and* places the contents into the DMA registers while incrementing the DMA_DSCPTR_CUR register.
    - If the channel detects no trigger event, the DMA channel begins the next work unit by reloading the current registers as described in the next step.
- Begins the next work unit (if flow configuration is anything other than stop mode) by reloading the current registers (DMA_ADDR_CUR, DMA_XCNT_CUR, and DMA_YCNT_CUR) from their descriptor registers (DMA_ADDRSTART, DMA_XCNT, and DMA_YCNT)

Work Unit Transition Flow

The DMA_CFG.SYNC bit controls transitions from one work unit to the next work unit. In general, continuous transitions have lower latency at the cost of restrictions on changes of data format or addressed memory space in the two work units. These latency gains and data restrictions arise from the way the channel handles the DMA FIFO while fetching the next descriptor.

In continuous transitions, with disabled synchronization, the DMA FIFO pipeline continues to transfer data to and from the peripheral or destination memory. These transfers continue during the descriptor fetch and during the DMA channel pause between descriptor chains. By comparison, synchronized transitions provide better real-time synchronization of interrupts and triggers with a given peripheral state. Synchronized transitions also provide greater flexibility in the data formats and memory spaces of the two work units. This flexibility comes at the cost of higher latency in the transition. In synchronized transitions, the DMA FIFO pipeline drains to the destination or flushes (received data discarded) between work units.

NOTE: The DMA_CFG.SYNC bit of the MDMA source channel controls work unit transitions for MDMA streams. Clear this reserved bit of the MDMA destination channel, placing it in the disabled state. In transmit (memory read) channels, the DMA_CFG.SYNC bit of the last descriptor before the transition controls the transition behavior. In contrast, in receive channels, the DMA_CFG.SYNC bit of the first descriptor of the next descriptor chain controls the transition.

Work Unit Transmit and MDMA Source Transitions

In DMA transmit (memory read) and MDMA source channels, the DMA_CFG.SYNC bit controls the interrupt timing at the end of the work unit. This bit also controls the handling of the DMA FIFO between the current and the next work unit.

If the DMA_CFG.SYNC bit configuration disables synchronization, the DMA channel operates in continuous transition. In a continuous transition, just after reading the last data item from memory, the DMA channel starts all of the following operations parallel:

- Signals the interrupt request or trigger

- Updates the DMA_STAT register to indicate DMA completion status
- Begins fetching the next descriptor
- Delivers the final data items from the DMA FIFO to the destination memory or peripheral

This process lets the DMA channel provide data from the FIFO to the peripheral continuously during the descriptor fetch latency period.

If the configuration disables synchronization, the final interrupt request or trigger (if enabled) occurs when the channel reads the last data from memory. This event occurs at the earliest time that the channel safely can modify the output memory buffer without affecting the previous data transmission. There can be a number of data items remaining in the FIFO and not yet at the peripheral. This number depends on the FIFO depth of the DMA channel. In this configuration, do not use the DMA interrupt request as the sole means of synchronizing the shutdown or reconfiguration of the peripheral following a transmission.

NOTE: If the configuration selects continuous transition on a transmit (memory read) descriptor, the next descriptor must have the same:

- Peripheral transfer size (DMA_CFG.PSIZE)
- Read or write direction
- Source memory (internal versus external) as the current descriptor

It is possible to disable synchronization by selecting continuous transition on a work unit with configuration for stop-flow mode and with enabled interrupts or triggers. This approach can result in the execution of the event service routine while draining of the final data is ongoing from the FIFO to the peripheral. If data transfers are in-progress, the FIFO is not yet empty. The DMA_STAT.RUN bits of the DMA channels indicate this status. Do not start a new work unit with a different peripheral transfer size or direction while data transfers are in-progress.

CAUTION: Disabling the channel with the DMA_CFG.EN bit while data transfers are in-progress causes the loss of the data in the FIFO.

A synchronized transition configuration directs the channel to drain the DMA FIFO to the destination memory or peripheral. This FIFO operation occurs before the channel signals any interrupt and before the channel fetches any subsequent descriptor or data. This operation incurs greater latency, but provides direct synchronization between the DMA interrupt and the state of the data at the peripheral.

If the configuration enables synchronization and enables interrupts, on the last descriptor in a work unit, the interrupt occurs when the channel transfers the final data to the peripheral. This event allows the service routine to switch properly to non-DMA transmit operation. When the event vectors to the interrupt service routine, the DMA channel FIFO is empty, and the DMA channel is no longer running (indicated by the DMA_STAT.RUN bits).

A synchronized transition also allows greater flexibility in the format of the DMA descriptor chain. When enabled, the next descriptor can have any DMA_CFG.PSIZE configuration or read/write direction supported by the peripheral and can come from either memory space (internal or external). This feature can be useful in managing MDMA work unit queues, since it is no longer necessary to interrupt the queue between dissimilar work units.

Work Unit Receive and MDMA Destination Transitions

In DMA receive channels (memory write operations), the DMA_CFG.SYNC bit controls the handling of the DMA FIFO between descriptor chains (not individual descriptor sets), during the DMA channel pause. The DMA channel pauses after the descriptor sets configured with stop flow mode are complete. Restart the channel (for example, after an interrupt) by writing the DMA_CFG register of the channel with a value that enables the DMA channel. If the configuration disables synchronization in the DMA_CFG value of the new work unit, the configuration selects a continuous transition. In this mode, the DMA FIFO retains any data items received during the channel pause, and they are the first items written to memory in the new work unit. This mode of operation provides lower latency at work unit transitions and ensures no dropping of data items during a DMA pause. The channel provides this operation at the cost of certain restrictions on the DMA descriptors.

NOTE: If the DMA_CFG.SYNC bit disables synchronization on the first descriptor of a chain after a DMA pause, do not change the configuration of the DMA_CFG.PSIZE field of the new chain from the previous descriptor chain (active before the pause). This restriction applies unless the DMA channel is reset between chains by disabling and then re-enabling the DMA channel.

If the DMA_CFG.SYNC bit configuration enables synchronization, the channel uses a synchronized transition. In this mode, only the data that the DMA channel receives from the peripheral after the write to the DMA_CFG register gets to memory. The channel discards any prior data items transferred from the peripheral to the DMA FIFO before this register write occurs. This operation provides direct synchronization between the data stream received from the peripheral and the timing of the channel restart, which occurs on the write to the DMA_CFG register.

For receive DMA operations, the synchronization has no effect in transitions between work units in the same descriptor chain. When the flow mode of previous descriptor was not stopped, the DMA channel did not pause.

If a descriptor chain begins with synchronization enabled, there is no restriction on the DMA_CFG.PSIZE of the new chain in comparison with the previous chain.

NOTE: The peripheral transfer size (DMA_CFG.PSIZE) must not change between one descriptor and the next in any DMA receive (memory write) channel within a single descriptor chain, regardless of the DMA_CFG.SYNC bit setting. In other words, all memory write descriptor sets in a descriptor chain must have the same DMA_CFG.PSIZE value. For any DMA receive channel (memory write operation), there is no restriction on changes of peripheral transfer size (internal versus external) between descriptors or descriptor chains.

Transfer Termination and Shutdown Flow

This section describes channel transfer termination and shutdown in stop flow mode and in autobuffer flow mode.

Stop Flow Mode

In stop flow mode, the DMA channel stops automatically after the work unit is complete. If using a list or array of descriptors to control DMA transfers and if every descriptor contains a DMA_CFG descriptor element, configure the flow of the final DMA_CFG descriptor element to stop mode, stopping the channel gracefully. After completion, the DMA channel remains in the stop state. Do not confuse this state with the disabled state, which either occurs due to a DMA error or occurs through disabling the DMA channel by configuring the DMA_CFG.EN bit.

The intention of disabling the DMA channel through a write to the DMA_CFG.EN bit is to shut down the DMA channel and to enter the disabled state. All memory and peripheral data transfers cease, and only peripheral interrupts pass through the DMA channels interrupt signals. However, the DMA channel maintains the DMA_STAT.RUN bits. For a write to memory, the outstanding memory-transaction counter tracks returning memory write acknowledgments and updates as required.

For memory reads, the outstanding memory-transaction count also tracks returning memory reads. The channel does not write the memory reads into the FIFO. The channel updates the counter to reflect the completion of the transaction, but the channel ignores the data. The DMA_STAT.RUN bits remain in the *waiting for write ACK or FIFO drain to peripheral* state and do not change to *stop or idle state* until the return of all outstanding transactions.

When the DMA_CFG.EN bit again enables the DMA channel, the channel performs a full reset and clears all counters. If an outstanding memory transaction returns an acknowledgment or read data after this event, a memory transaction error occurred, which generates an error event. Programs must ensure that all outstanding memory transactions complete before reconfiguring the DMA channel. For example, programs can poll the DMA_STAT.RUN bits to return to the stop or idle state before proceeding.

Autobuffer Flow Mode

In this mode, the flow does not use any descriptors in stored memory. Instead, the channel performs DMA in a continuous circular buffer fashion, based on user-programmed DMA register settings. On completion of the work unit, the channel reloads the parameter registers into the current registers, and the DMA controller resumes immediately with zero overhead. Consider this mode as a succession of automatically restarted work units.

For autobuffer-flow modes, the only way to cease operations is to disable the DMA channel through the DMA_CFG.EN bit. One method of changing to a new work unit is:

- Disable the DMA channel
- Set up all the registers (and descriptors in memory, if used) except for DMA_CFG
- Poll DMA_STAT.RUN to wait for the status to reflect stop or idle state, and
- Write DMA_CFG to the new configuration to begin the next work unit

In autobuffer-flow mode or for a list or array of descriptor sets without DMA_CFG descriptors, use an MMR write to the DMA_CFG register to terminate the DMA transfer process. Configure the value of the DMA_CFG.EN bit in this register to disable the DMA channel.

CAUTION: When the configuration disables a DMA channel, the DMA controller disables interrupt logic that is based on work unit transitions. Be aware of the system environment and current actions, so that additional interrupts are not required from the DMA channel.

CAUTION: If disabled through DMA_CFG.EN in the middle of a transaction, the DMA channel completes any transactions that have begun and avoids generating bus errors. However, the channel considers the action of re-enabling the DMA as a hard reset for all internal DMA channel components. Therefore, pay attention to that particular action to avoid unexpected results.

DMA Channel Errors

When an error occurs, the DMA channel maintains all the state and register values that allow programs to diagnose error causes more thoroughly. The greatest benefit to the programmer is to know exactly what operational state the DMA channel was in at the exact moment the error occurred.

Take care to address the root cause of the error, whether or not the problem originated in the DMA channel. If not properly resolved, the error can result in an additional error shortly after operations resume. The problem can cause other errors elsewhere in the DMA channel or associated modules and circuitry. So, take care also to address those potential problems. Ensure that all outstanding memory reads and writes are complete or cleared before resuming DMA channel operation.

After addressing all issues and neutralizing all side effects of any errors, clear the DMA_STAT.ERRC status field and restart the DMA channel by disabling then re-enabling the DMA channel through the DMA_CFG.EN bit.

The following sections describe the error types.

Status and Debug Errors

DMA channel error conditions can cause the DMA process to end abnormally. The DMA channel provides error detection as a tool for system development and debug, helping to identify DMA-related programming errors. When the DMA channel detects an error, the channel immediately stops and discards any returned memory-read transactions. The DMA_STAT.RUN field of the DMA channel indicates the idle state after acknowledging all outstanding memory transactions. In addition, the channel asserts an error interrupt request and updates the DMA_STAT.IRQERR field. Also, the channel updates the DMA_STAT.ERRC field, indicating the error cause of the first detected error. Unless the error occurs at the exact moment that modification of register values occurs, the registers contain the error values.

All the DMA error interrupt requests are combined into a single shared interrupt request output. Combined error signals require reading the DMA_STAT register of each DMA channel associated with a combined error interrupt request to determine the DMA channel responsible for the generation of the interrupt.

The DMA channel error interrupt handler performs the following actions:

- Read the DMA_STAT register of each DMA channel, seeking a channel with the DMA_STAT.IRQERR set to indicate an error.
- Read the DMA_STAT.ERRC field of each DMA channel, determining the cause of the error.
- Clear the problem with the DMA channel. For example, fix the register values.
- Clear the error in the DMA channel through a write-1-to-clear operation to the DMA_STAT.IRQERR bit.

If the channel flags any uncleared error other than a bandwidth monitor error, the channel reports no other error. If the channel reports an uncleared bandwidth monitor error, the channel reports any newly detected error through updating the DMA_STAT.ERRC field.

DMA Configuration Register Errors

The channel only flags these configuration errors when the DMA_CFG.EN bit enables the DMA channel. Error flagging occurs when the configuration:

- Uses a reserved setting
- Enables DMA_CFG.TWAIT in descriptor on-demand flow mode
- Uses an illegal DMA_CFG.NDSIZE
- Uses an illegal DMA_CFG.MSIZE
- Configures DMA_XCNT = 0 or, when DMA_YCNT = 0 in 2D DMA mode
- Uses non-zero value in DMA_CFG.NDSIZE when DMA is configured in stop mode or auto mode
- Enables interrupt or outgoing triggers on DMA_YCNT when DMA is configured in 1D mode
- Use a DMA_CFG.MSIZE that exceeds the FIFO size of the DMA channel
- Uses an illegal DMA_CFG.PSIZE
- Uses a DMA_CFG.PSIZE that exceeds the FIFO size
- Uses a DMA_CFG.PSIZE that exceeds the bus width
- Attempts to change from a transmit operation (memory read) to a receive operation without properly synching in the previous work unit or when it is the first work unit in a new chain
- Attempts to change DMA_CFG.PSIZE of a transmit operation (memory read) without properly synching in previous work unit or when it is the first work unit in a new chain
- Attempts to change from receive operation (memory write) to a transmit operation during a descriptor chain.

  The channel only can change from receive to transmit if the new transmit is synchronized and is the first work unit.
- Attempts to change DMA_CFG.PSIZE of a receive operation (memory write) when the operation was not the first work unit (with DMA_CFG.SYNC enabled)

Illegal Register Write During Run

The channel generates an error when a write occurs to writable registers of an enabled, running DMA channel. The channel blocks the write. The DMA_STAT, DMA_BWLCNT, and DMA_BWMCNT registers are exempt from this behavior. The DMA_STAT register is exempt from this behavior.

Address Alignment Error

The channel generates an address alignment error when any of the following apply:

- Alignment of a descriptor address is not on a 32-bit boundary.
- The current DMA_CFG.MSIZE configuration contains an unaligned transfer address. The DMA_ADDRSTART register is not aligned according to the DMA_CFG.MSIZE field.

Memory Access Error

The channel generates a memory access error when the DMA process:

- attempts to access an unpopulated address,

- attempts to access a location that provokes a security violation

The error returned from the memory triggers the memory access error.

Trigger Overrun Error

A trigger overrun error is generated when a new trigger input occurred while an outstanding trigger is waiting. This error is only generated if `DMA_CFG.TOVEN` is enabled.

Bandwidth-Monitor Error

The channel generates this error when the bandwidth-monitor count expires. This error is not fatal, and the DMA channel continues operation.

Control Interface Error

The channel reports control-interface errors as bus errors to the bus master. This error can result from:

- An address error
- A register write error (write to a read-only register)

DMA Operating Modes

The DMA channel supports a number of different flow modes that control how the DMA channel progresses from one work unit to the next.

The flow mode of a DMA channel is not a global setting. A DMA descriptor set can include the descriptor responsible for configuring the flow of the work unit. There is no restriction, limiting the flow configuration to be the same for the entire descriptor chain. If the descriptor chain is not endless, the last descriptor set configures the flow to stop mode, which results in termination of the descriptor chain after the work unit completes. Another example for mixing flow modes is to create an endless descriptor-array. The configuration of the last descriptor set in the array selects the descriptor-list mode. The next descriptor pointer in this set of descriptors points to the first descriptor in the array.

Register-Based Flow Modes

Register-based DMA operations require configuration by directly writing to the memory-mapped registers of the DMA channel.

Register-based DMA is the traditional method of DMA operation. Software writes all of the configuration of the DMA channel into the memory-mapped registers. This configuration includes information such as the source or destination address and length of the data in the transfer. The DMA controller then starts channel operation. The DMA channel supports the following register-based flow modes.

- Stop Mode
- Autobuffer Mode

The DMA channel supports variable descriptor set sizes within the configuration. The size of a descriptor set can contain as little as a single descriptor. The supported descriptor set sizes can differ between the various descriptor-based flow modes. In addition to the descriptor set size being configurable, descriptor-based DMA also allows altering the flow mode of the next descriptor set. This feature allows for the transition from descriptor-array mode to descriptor-list mode and permits configuring the flow to stop or autobuffer mode.

Stop Mode

In stop mode, the DMA operation executes only once. If started, the DMA channel transfers the desired number of data words and stops itself again when finished. If the DMA channel is no longer used, software configures the enable bit to disable a paused channel. The channel also can generate interrupts and triggers for each row or work unit completion, depending on the desired operation.

Autobuffer Mode

In autobuffer mode, the DMA operates repeatedly in a circular manner. If the transfer of all data words completes, the channel reloads the address pointer (`DMA_ADDR_CUR`) automatically with the `DMA_ADDRSTART` value. The channel also can generate an interrupt.

The `DMA_CFG.FLOW` field enables autobuffer mode. The configuration must load the `DMA_CFG.NDSIZE` field value, such that the next descriptor size is zero.

Descriptor-Based Flow Modes

Descriptor-based DMA operations fetch descriptor sets from memory allowing for autonomous loading of work units on other work units. Software does not need to set up the DMA sequences directly by writing into the DMA controller registers. Rather, software keeps DMA descriptor sets in memory.

Descriptor-based DMA operations have the following additional attributes.

- The DMA controller autonomously loads the descriptor set from memory to the affected DMA controller registers on demand.
- The channel can fetch descriptor sets from any memory space that supports DMA read operations.
- The descriptor set describes the next operation that the DMA controller performs.
- The descriptor set can include information such as the DMA configuration word as well as data source or destination address, transfer count, and address modify values.

A descriptor set describes a single work unit. The next work unit can reuse some values from the previous one descriptor set. But, this reusage is possible only if they are not overwritten in the subsequent descriptor set fetches and only if the work unit requires the use of this descriptor.

The DMA channel supports the following flow modes with descriptor-based operations.

- Descriptor-Array Mode
- Descriptor-List Mode
- Descriptor-On-Demand Modes

The DMA channel supports variable descriptor set sizes within the configuration. The size of a descriptor set can contain as little as a single descriptor and the supported descriptor set sizes can differ between the various descriptor-based flow modes. In addition to configurable descriptor set size, descriptor-based DMA also allows for altering of the flow mode of the next descriptor set. Programs can transition from one descriptor-based mode to another descriptor-based mode and can also transition to any of the register-based flow modes.

Descriptor-Array Mode

When configured in this mode, the descriptor sets do not contain further descriptor pointers. Software writes the initial descriptor-pointer value, which points to an array of descriptors. This operation assumes that the individual descriptors reside next to each other and assumes that their addresses are known.

The ***Offsets for Descriptor-Array Mode Parameters and Descriptors*** table illustrates how to structure a descriptor set in memory. The descriptor sets must reside in a contiguous block or memory in the format shown in the table. Locate the first descriptor of the next descriptor set in the memory location immediately following the last descriptor of the current descriptor set. The values have the same order as the corresponding offset addresses of the memory-mapped register.

Table 17-7: Offsets for Descriptor -Array Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT
0x14	DMA_YMOD

All other DMA channel registers not loaded as a result of the descriptor set fetch retain their previous values. The channel reloads all of the current registers between the descriptor set fetch and the start of the DMA operation for the work unit.

NOTE: At a minimum, write the DMA_DSCPTR_CUR register prior to writing to the DMA_CFG register, which is the special action required to start the DMA channel.

Descriptor-List Mode

In this flow mode, multiple descriptors form a chained list in which each descriptor set contains a pointer to the next descriptor set, allowing greater flexibility in memory layout options. When the channel fetches the descriptor set, the operation loads this pointer value into the next descriptor pointer register of the DMA channel.

Descriptor Sets

The ***Offsets for Descriptor-List Mode Parameters and Descriptors*** table shows how to structure a descriptor set in memory. Disperse the placement of the descriptor sets throughout memory, having sets reside in different memory

blocks. But, each descriptor of the descriptor set must reside in a contiguous section of memory in the format shown in the table. The values have the same order as the corresponding offset addresses of the memory-mapped registers.

Table 17-8: Offsets for Descriptor-List Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_DSCPTR_NXT
0x04	DMA_ADDRSTART
0x08	DMA_CFG
0x0C	DMA_XCNT
0x10	DMA_XMOD
0x14	DMA_YCNT
0x18	DMA_YMOD

All other DMA channel registers not loaded as a result of the descriptor set fetch retain their previous values. The channel reloads all of the current values of the registers between the descriptor set fetch and the start of the DMA operation for the work unit.

Minimum Startup Requirements

At a minimum, write the `DMA_DSCPTR_NXT` register prior to write to the `DMA_CFG` register, which is the special action required to start the DMA channel.

Descriptor-On-Demand Modes

The Descriptor-Array Mode and Descriptor-List Mode each have an on-demand mode of operation.

In on-demand mode, at the end of the work unit, if the DMA channel has not detected an incoming trigger event, the channel repeats the current work unit. If the DMA channel receives an incoming trigger before completion of the work unit, the channel fetches a new descriptor set.

The ***Offsets for Descriptor-Array Mode Parameters and Descriptors*** and ***Offsets for Descriptor-List Mode Parameters and Descriptors*** tables illustrate how to structure each descriptor set in memory.

Table 17-9: Offsets for Descriptor-Array Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_ADDRSTART
0x04	DMA_CFG
0x08	DMA_XCNT
0x0C	DMA_XMOD
0x10	DMA_YCNT

Table 17-9: Offsets for Descriptor-Array Mode Parameters and Descriptors (Continued)

Descriptor Offset	Parameter Register
0x14	DMA_YMOD

NOTE: For descriptor-array mode, at a minimum, write the DMA_DSCPTR_CUR register prior to writing to the DMA_CFG register, which is the special action required to start the DMA channel.

Table 17-10: Offsets for Descriptor-List Mode Parameters and Descriptors

Descriptor Offset	Parameter Register
0x00	DMA_DSCPTR_NXT
0x04	DMA_ADDRSTART
0x08	DMA_CFG
0x0C	DMA_XCNT
0x10	DMA_XMOD
0x14	DMA_YCNT
0x18	DMA_YMOD

NOTE: For descriptor-list mode, at a minimum, write the DMA_DSCPTR_NXT register prior to write to the DMA_CFG register, which is the special action required to start the DMA channel.

Data Transfer Modes

In addition to supporting basic one-dimensional DMA transfers, the DMA channel also supports two-dimensional functionality.

Two-Dimensional DMA

Register-based flow modes and descriptor-based flow modes support two-dimensional data transfers.

In two-dimensional (2D) mode, the X-direction count (DMA_XCNT), the X-direction modifier (DMA_XMOD), the Y-direction count (DMA_YCNT), and the Y-direction modifier (DMA_YMOD) support arbitrary row and column sizes. Also, the modify values can be negative, allowing implementation of interleaved data streams. The DMA_XCNT value specifies the row size, and the DMA_YCNT value specifies the column size; where the DMA_XCNT value must be 2 or greater.

The DMA start address (DMA_ADDRSTART), the X-direction modifier (DMA_XMOD), and theY-direction modifier (DMA_YMOD) specifications all are in bytes. The alignment must be a multiple of the DMA transfer word size; configured using the DMA_CFG.MSIZE bit. Misalignment results in a DMA channel error.

The DMA_XMOD register value is the byte-address increment that the channel applies after each transfer, decrementing the DMA_XCNT register. The channel does not apply the DMA_XCNT when the inner loop count ends with the DMA_XCNT_CUR register decrementing to 0 from 1. Except, the channel does apply the DMA_XCNT on the final transfer, when the DMA_YCNT register is 1 and the DMA_XCNT register decrements from 1 to 0.

The DMA_YMOD register value is the byte-address increment that the channel applies after each decrement of the value in DMA_YCNT_CUR. However, the channel does not apply the DMA_YMOD value to the last item in the array on which the outer loop count (DMA_YCNT_CUR) also expires by decrementing from 1 to 0.

After the last transfer completes, DMA_YCNT_CUR is 1 and the DMA_XCNT_CUR register is 0. The DMA channels current address points to the last items address plus the DMA_XMOD register value. If the DMA channel programming selects automatic refresh (such as in autobuffer mode), the channel reloads the DMA_XCNT_CUR, DMA_YCNT_CUR, and DMA_ADDR_CUR for the first data transfer of the next work unit.

Interrupt notification is configurable for end-of-row or end-of-work unit completion.

For example, two-dimensional DMA can be used to extract interleaved data (such as RGB values for a video frame) by modifying both of the DMA_XMOD and DMA_YMOD values. The ***Capturing a Video Data Stream 2D DMA Example*** depicts the process of receiving a stream of the R, G, B values from an N*M frame. The inner loop of the 2D DMA configuration has three values (DMA_XCNT = 3) and a stride (DMA_XMOD) of N*M, chosen such that successive elements in each row are 1-2-3, 4-5-6 and so forth. The outer loop of the 2D DMA configuration has N*M values (DMA_YCNT = N*M) and a negative stride (DMA_YMOD) of 1-2*N*M chosen to instruct the DMA controller to jump from element 3 to 4, 6 to 7 and so forth at the end of each inner loop.

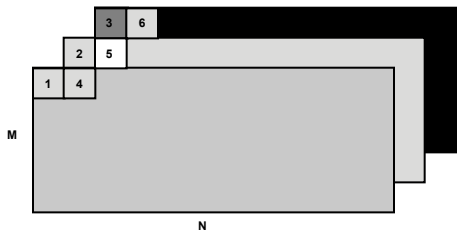


Figure 17-4: Capturing a Video Data Stream 2D DMA Example

DMA Channel Event Control

The DMA channel supports a number of events that provide notification of work unit state, peripheral data request, peripheral interrupt request and completion events, and DMA channel error conditions. In addition to flexible interrupt configuration, the DMA channel also supports incoming and outgoing triggers which are useful in synchronizing the DMA channel with other system resources.

The DMA channel has two interrupt signals for support of a number of events such as work-unit state events, peripheral interrupt request (PIRQ) events, peripheral data request (PDR) events, and DMA channel errors. The channel reports DMA channel errors on a dedicated interrupt signal. All other interrupt sources share an interrupt signal. In addition to flexible interrupt configuration, the DMA channel also supports incoming and outgoing triggers which are useful in synchronizing the DMA channel with other system resources.

The channel can signal the processor on DMA channel events using status information and optional interrupt requests. Programs can use these events to update the progress of data transfers and to request intervention from the processor core. Configure most DMA channel interrupts using bits in the DMA_CFG register. Dedicated bits in the

DMA_STAT register report the occurrence of various events. Use write-one-to-clear (W1C) operations to clear interrupt requests from the status register.

NOTE: Hardware does not clear the interrupt status bits automatically, even when programs disable then reenable the DMA channel. In this situation, the channel deasserts the interrupt signal, after the program disables the DMA channel. But, the status bit remains set until software either re-enables the DMA channel or clears the status bit.

The DMA channel supports the following categories of events on the interrupt signals:

- Work-unit state events generate interrupts on row or on work unit DMA completion.
- A peripheral uses peripheral interrupt request (PIRQ) events to signal when it has completed the transfer of all data.
- A peripheral uses peripheral data request (PDR) events to request data from a disabled or idle DMA channel.
- Error events signal a failure in the work unit.

ATTENTION: While in an error state, the DMA channel does not generate an interrupt to the processor for a work-unit state event or a PIRQ event, nor does the channel forward a PDR event.

Event Signals

The ***Event Signals*** table provides descriptions of DMA channel events.

Table 17-11: Event Signals

Signal	Width (bits)	Description
DMA_ERROR	1	Used to signal an error condition in the DMA channel. The source of the error can be determined by reading the DMA_STAT.ERRC bit.
DONE_PIRQ_INT	1	Signal used to indicate DMA completions events, PIRQ events and also for forwarding PDR events based on configuration. Read the corresponding fields in DMA_STAT to determine the source of the event.
DMA_TRIG_OUT	1	Trigger output that gets routed to the TRU and can be configured to provide notification on row or work unit completion.
DMA_TRIG_IN	1	Trigger input from the TRU that can be used to control the start of a work unit.

Work Unit State Events

Completing a row or a work unit generates a work-unit state event. For either of these events to generate an interrupt request, the configuration of the interrupt of the DMA channel must select one of the available work-unit completion modes.

- Current X count reaching 0 for row completion or 1D DMA work unit completion
- Current Y count reaching 0 for work unit completion of 2D DMA

NOTE: For 1D DMA, a DMA channel configuration error results if the configuration generates the interrupt request when the current Y counter reaches 0.

The DMA channel issues the last memory read or write transaction for the row or work unit, then pauses until the return of the read or write acknowledge. After successful acknowledge of the transfer, the DMA channel issues the interrupt request and continues to process the next row or work unit.

Waiting for acknowledgement of the memory access results in a delay. However, programs can read or modify data in the memory without adversely affecting or being affected by the DMA transfer.

NOTE: While the DMA channel pauses waiting for acknowledgement of the memory transfer, the DMA channel is still capable of fetching the next descriptor set. This fetch gets the channel ready to process the next work unit as soon as the memory access completes.

The channel configuration of the synchronization feature also affects interrupt timing. For memory-read operations with synchronization enabled, the channel delays the interrupt request until the completion of the last transfer from the DMA channel FIFO to the peripheral. The synchronization feature does not affect interrupt timing for memory write operations.

Peripheral Interrupt Request Events

For peripheral-transmit operations, a peripheral connected to the DMA channel can use peripheral interrupt request (PIRQ) events to indicate that data has left the channel FIFO and to indicate transfer completion.

In order to support PIRQ interrupts, correctly configure the interrupt of the DMA channel. This configuration disables the generation of interrupt requests based on the work unit state and, instead, results in generating an interrupt request when the DMA channel receives the command from the peripheral.

The channel only generates the interrupt request under the following conditions:

- The configuration enables the DMA channel
- The DMA channel is in the stop state
- The configuration of the DMA channel interrupt selects PIRQ operation

Peripheral Data Request Events

Peripheral data request (PDR) events occur when an interfaced peripheral requests data from the DMA channel and the DMA channel (either disabled or enabled) is in the stop state.

When a peripheral sends a data request command to a disabled DMA channel, the DMA channel generates an interrupt to the System Event Controller (SEC). There is no status information reported about this event in the status register of the DMA channel. Instead, the channel identifies the PDR event from the fact that the DMA channel generated an interrupt while disabled. It is possible to further confirm event status by verifying the status of the peripheral interfaced to the DMA channel.

This operation forwards data requests as interrupts when the DMA channel is in the disabled state. Also, the DMA channel is able to forward PDR events as an interrupt request when the DMA channel is in the stop state after the

completion of a work unit. The forwarding of this interrupt when the DMA channel is in the stop state is optional and configured by the program during DMA channel configuration.

DMA Channel Triggers

DMA channel triggers are useful for synchronizing the DMA channel with other events in the system. One usage is to combine channel triggers with each other to create ping-pong buffers. Another usage is to combine the triggers with interrupt requests to notify the processor on reaching a particular milestone that requires service. The channel also can use triggers to enforce a handshake DMA operation in which the trigger acts as a signal for a DMA request.

NOTE: Using the trigger to control the pace of data transfers, such as for handshake DMA, requires that all the data for the entire work unit is ready for transfer.

The DMA channel has a single incoming trigger that can control the pace of the data transfers performed by the DMA channel. The configuration can direct the DMA channel to wait for the incoming trigger before starting the work unit transfer or fetching a descriptor set from memory.

The DMA channel also has a single outgoing trigger signal. This configuration can direct this trigger to signal the end of row or an entire work unit. The DMA channel issues the last memory read or memory write transaction for the row or work unit, then pauses until return of the transfer acknowledge. After acknowledgement of the transfer, the DMA channel issues the trigger before processing the next row or work unit.

Issuing Triggers

The DMA channel configuration can direct the channel to generate an outgoing trigger signal at the end of row or the end of a work unit. The DMA channel issues the last memory read or memory write transaction for the row or work unit, then pauses until the return of the transfer acknowledge. After acknowledgement of the transfer, the DMA channel issues the trigger before processing the next row or work unit.

NOTE: While the DMA channel pauses waiting for acknowledgement of the memory transfer, the DMA channel is still capable of fetching the next descriptor set. This fetch gets the channel ready to process the next work unit as soon as the memory access completes.

Waiting For Triggers

Programs can use triggering to control the pace of data transfers performed by the DMA channel. The DMA channel enters a wait state before beginning the next work unit if the configuration enables `DMA_CFG.TWAIT` and either of the following apply:

- The channel receives a trigger since the last time the DMA channel left the wait state.
- The channel receives a trigger since its transition from disable to enable.

In the wait state, the DMA channel also does not perform a descriptor fetch. After receiving a trigger, the DMA channel leaves the wait state and begins the next work unit or fetches the next descriptor if configured for a descriptor-based mode of operation.

If a memory-mapped register write operation programs the channel with stop flow mode enabled (DMA_CFG.TWAIT bit) and the channel has not already received a trigger, the DMA channel enters a wait state before performing the data transfer. On receiving the trigger, the DMA channel begins the data transfer portion of the work unit. Once the data transfer is complete, the DMA channel enters the stop state.

If a memory-mapped register write operation programs the DMA channel with the flow mode configured to one of the descriptor-based modes, the DMA channel enters the wait state before performing the descriptor fetch. After completing the descriptor fetch, the DMA channel immediately proceeds to the data transfer, regardless of the value of the DMA_CFG.TWAIT bit. If another (next) descriptor fetch follows the descriptor fetch, the DMA channel enters a wait state before fetching the next descriptor.

If the descriptor fetch returns a descriptor with stop flow mode, the DMA_CFG.TWAIT value for that descriptor does not affect the DMA as the channel enters the stop state after completing the data transfer. The DMA channel only enters the wait state based on DMA_CFG.TWAIT before the next work unit or descriptor fetch.

If the descriptor fetch returns a descriptor configured for autobuffer flow mode, the DMA_CFG.TWAIT for that descriptor does not affect the DMA for the first work unit of the autobuffer transfer. After completing the first work unit and not receiving another trigger, the DMA channel enters the wait state before reinitializing its counters and address registers (if not configured for current addressing). The channel performs the next work unit after receiving the trigger.

The incoming trigger can occur when the DMA channel has not entered the wait state. The trigger can occur while the DMA channel is executing a work unit, is performing descriptor fetch, or is in the stop state. The trigger is held internally. After the work unit is complete, the DMA channel skips the wait state and proceeds directly to executing the following work unit. If the DMA_CFG.TWAIT bit is not enabled, the DMA channel also skips the wait state. However, the trigger is held internally and is used the next time the configuration enables DMA_CFG.TWAIT. This trigger retention allows programs to enable the DMA_CFG.TWAIT functionality several work units apart without concern for losing a trigger. The DMA channels trigger-overrun enable functionality can be enabled in all work units to ensure that multiple triggers do not occur between the work units with the DMA_CFG.TWAIT bit enabled.

DMA Channel Programming Model

Several synchronization and control methods are available for use in development of software tasks which manage peripheral DMA and memory DMA. Software must accept requests for new DMA transfers from other software tasks, integrate these transfers into existing transfer queues, and reliably notify other tasks when the transfers are complete.

In the processor, it is possible to manage each peripheral DMA and memory DMA stream with a separate task or to manage them together with any other stream. Each DMA channel has independent, orthogonal control registers, resources, and interrupts. So, the selection of the control scheme for one channel does not affect the choice of control scheme on other channels. For example, one peripheral can use a linked-descriptor-list, interrupt-driven scheme while another peripheral can simultaneously use a demand-driven, buffer-at-a-time scheme synchronized by polling DMA events.

The topics that follow describe the steps required to configure the DMA channel for the various modes in addition to the programming concepts required for software synchronization.

NOTE: The ADSP-CM41x has different configurations so that multiple peripherals "share" the same DMA unit, but not concurrently.

Mode Configuration

Use the step-by-step directions that follow to set up the DMA channel for operating modes.

Register-Based Linear-Buffer Stop Flow Mode

This procedure configures the DMA channel of a peripheral to read data from internal memory and to send it to the peripheral for transmission. On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

Assume that the peripheral is in a state where it is ready to transmit data received from the DMA channel.

The task involves writing to a number of DMA channel MMR registers to configure a DMA channel to:

- Read data from internal memory, and
- Send it to a peripheral connected to the peripheral DMA bus.

1. Write the DMA_ADDRSTART register.

   ADDITIONAL INFORMATION: Software can use the address to calculate the most optimum possible DMA_CFG.MSIZE.

2. Calculate the optimum DMA_CFG.MSIZE based on the DMA_ADDRSTART register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected DMA_CFG.MSIZE, and the calculation also must consider the start address alignment.

3. Write the DMA_XCNT register based on the calculated DMA_CFG.MSIZE.

   ADDITIONAL INFORMATION: The DMA_XCNT value is the number of DMA_CFG.MSIZE transfers to make up the entire work unit.

4. Write the DMA_XMOD register.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in DMA_XMOD from the selected DMA_CFG.MSIZE. Always specify this register as a number of bytes.

5. Write the DMA_CFG register with DMA_CFG.EN configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the DMA_CFG.FLOW bit for STOP mode. Configure the DMA_CFG.WNR bit for memory read operation. Configure the DMA_CFG.PSIZE bits to a value no larger than the supported bus width of the peripheral DMA bus.

- The DMA_CFG.SYNC bit can be configured to control DMA completion notification timing.
- Interrupts and triggers also can be configured at this step depending on requirements.

Now, the DMA channel is enabled, and the buffer is transferred. The DMA channel enters the IDLE state upon completion of the work unit.

Register-Based Autobuffer Flow Mode

This procedure configures the DMA channel of a peripheral to read data from internal memory and send it to the peripheral for transmission. The transmission of the buffer repeats endlessly. On DMA completion, the DMA channel restarts the DMA operation, creating an endless circular buffer transfer.

Assume the peripheral is in a state where it is ready to transmit data received from the DMA channel.

The task involves writing to a number of DMA channel MMR registers to configure a DMA channel to:

- Read data from internal memory, and
- Send it to a peripheral connected to the peripheral DMA bus.

1. Write the DMA_ADDRSTART register.

   ADDITIONAL INFORMATION: Use the address to calculate the optimum possible DMA_CFG.MSIZE.

2. Calculate the optimum DMA_CFG.MSIZE based on the DMA_ADDRSTART register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected DMA_CFG.MSIZE, and the calculation must consider the start address alignment.

3. Write the DMA_XCNT register based on calculated DMA_CFG.MSIZE.

   ADDITIONAL INFORMATION: The DMA_XCNT register value is the number of DMA_CFG.MSIZE transfers to make up the entire work unit.

4. Write the DMA_XMOD register.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in DMA_XMOD from the selected DMA_CFG.MSIZE. Always specify this register as a number of bytes.

5. Write the DMA_CFG register with the DMA_CFG.EN bit configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the DMA_CFG.FLOW bit for autobuffer mode. Configure the DMA_CFG.WNR bit for memory read operation. Configure the DMA_CFG.PSIZE bit to a value no larger than the supported bus width of the peripheral DMA bus.

   - The DMA_CFG.SYNC bit can be configured to control DMA completion notification timing.
   - Interrupts and triggers also can be configured at this step depending on requirements.

Now, the DMA channel is enabled, and the buffer transfers until the DMA channel is disabled.

Descriptor-Array Flow Mode

This procedure configures the DMA channel of a peripheral to:

- Read data from memory as described by the descriptor sets in the array, and
- Send the data to the peripheral for transmission.

Descriptor sets are read from an array in memory to configure the individual work units.

Assume the peripheral is in a state where it is ready to transmit data received from the DMA channel. Assume that the array of descriptors is to be initialized with the last descriptor set configured for STOP flow mode.

The task involves writing to a number of DMA channel MMR registers to:

- Configure a DMA channel to read the array in memory, containing the first descriptor set that configured the DMA channel to retrieve, and
- Send the data to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the DMA_DSCPTR_CUR register with the address of the array in which the descriptor sets are stored.

   ADDITIONAL INFORMATION: The array address must meet any processor alignments restrictions imposed by descriptor fetches.

2. Write the DMA_CFG register with the DMA_CFG.EN bit configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the DMA_CFG.FLOW bit for descriptor-array mode. Configure the DMA_CFG.NDSIZE bits to describe the number of descriptor elements contained within the first descriptor set. Configure the DMA_CFG.WNR bit for memory read operation. Configure the DMA_CFG.PSIZE bits to a value no larger than the supported bus width of the peripheral DMA bus.

   - The descriptor set that is fetched controls the DMA_CFG.SYNC configuration and the interrupt or trigger configurations.

   The first descriptor set is fetched from memory location provided by the DMA_DSCPTR_CUR register and loaded to the MMR registers of the DMA channel.

Now, the DMA channel is processing all the work units provided in the descriptor array. The DMA channel enters the IDLE state on completion of the final work unit that was configured for STOP flow mode.

Descriptor-List Flow Mode

This procedure configures the DMA channel of a peripheral to:

- Read data from memory as described by the descriptor sets in the list, and
- Send it to the peripheral for transmission.

The DMA controller reads the descriptor sets from a list of descriptors. With the list, each descriptor set has a descriptor that points to the next descriptor set location in memory.

Assume the peripheral must be in a state where it is ready to transmit data received from the DMA channel. Assume that the list of descriptors must be initialized with the last descriptor set in the list configured for Stop flow mode.

The task involves writing to a number of DMA channel MMR registers to:

- Configure a DMA channel to read the list in memory, containing the first descriptor set that configured the DMA channel to retrieve, and
- Send the data to a peripheral connected to the peripheral DMA bus.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

1. Write the DMA_DSCPTR_NXT register with the address of the first descriptor in the list to be processed.

   ADDITIONAL INFORMATION: The array address must meet any processor alignments restrictions imposed by descriptor fetches.

2. Write the DMA_CFG register with the DMA_CFG.EN configured to enable the DMA channel.

   ADDITIONAL INFORMATION: Set the DMA_CFG.FLOW for descriptor-list mode. Configure the DMA_CFG.NDSIZE bit to describe the number of descriptor elements contained within the first descriptor set. Configure the DMA_CFG.WNR bit for memory read operation. Configure the DMA_CFG.PSIZE bit to a value no larger than the supported bus width of the peripheral DMA bus.

   - The descriptor set that is fetched controls the DMA_CFG.SYNC configuration and controls the interrupt or trigger configurations.

   The first descriptor set is fetched from the memory location provided by DMA_DSCPTR_NXT and is loaded to the MMR registers of the DMA channel.

Now, the DMA channel is processing all the work units provided in the descriptor list. The DMA channel enters the idle state when the final work unit that was configured for stop-flow mode is complete.

Register-Based Memory-to-Memory Transfer in Stop Flow Mode

This procedure configures a memory DMA channel pair in stop flow mode. One DMA channel is configured for memory read operations, while the other DMA channel is configured for memory write.

The task involves writing to a number of DMA channels on two DMA channels that create a memory DMA pair. On DMA completion, the DMA channel enters the idle state, until either the DMA channel is disabled or is reconfigured for a new transfer.

1. Write the DMA_ADDRSTART register of the source DMA channel.

   ADDITIONAL INFORMATION: The address can be used to calculate the optimum DMA_CFG.MSIZE possible.

2. Calculate the optimum DMA_CFG.MSIZE based on the DMA_ADDRSTART register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected DMA_CFG.MSIZE and the start address alignment must also be considered.

3. Write the DMA_XCNT register of the source DMA channel based on calculated DMA_CFG.MSIZE.

   ADDITIONAL INFORMATION: DMA_XCNT is the number of DMA_CFG.MSIZE transfers to make up the entire work unit.

4. Write the DMA_XMOD register of the source DMA channel.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in DMA_XMOD from the selected DMA_CFG.MSIZE. This register is always specified in the number of bytes.

5. Write the DMA_ADDRSTART register of the destination DMA channel.

   ADDITIONAL INFORMATION: The address can be used to calculate the most optimum DMA_CFG.MSIZE possible.

6. Calculate the optimum DMA_CFG.MSIZE based on the DMA_ADDRSTART register and number of bytes in work unit.

   ADDITIONAL INFORMATION: The number of bytes in the work unit must be a multiple of the selected DMA_CFG.MSIZE and the start address alignment must also be considered.

7. Write the DMA_XCNT register of the destination DMA channel based on the calculated DMA_CFG.MSIZE.

   ADDITIONAL INFORMATION: DMA_XCNT is the number of DMA_CFG.MSIZE transfers to make up the entire work unit.

8. Write the DMA_XMOD register of the destination DMA channel.

   ADDITIONAL INFORMATION: For a linear buffer transfer, determine the value in DMA_XMOD from the selected DMA_CFG.MSIZE. This register is always specified in the number of bytes.

9. Write the DMA_CFG register of the source DMA channel with DMA_CFG.EN configured to enable the DMA channel.

   ADDITIONAL INFORMATION: The DMA_CFG.FLOW bit field must be configured for stop mode. The DMA_CFG.WNR bit must be cleared for memory read operation. The DMA_CFG.PSIZE bits must be configured to a value no larger than the supported bus width of the peripheral DMA bus.

   - The DMA_CFG.SYNC bit can be configured to control DMA completion notification timing.

- Interrupts and triggers also can be configured at this step, depending on requirements. The interrupts and triggers are enabled within the destination DMA channel configuration.

The memory read DMA transfer begins.

10. Write the DMA_CFG register of the destination DMA channel with DMA_CFG.EN configured to enable the DMA channel.

    ADDITIONAL INFORMATION: The DMA_CFG.FLOW bit field must be configured for stop mode. The DMA_CFG.WNR bit must be set for memory write operation. The DMA_CFG.PSIZE bits must be configured to a value no larger than the supported bus width of the peripheral DMA bus. This value must also match the value written for the source DMA channel configuration.

    - Interrupts and triggers also can be configured at this step depending on requirements.

    The memory write DMA transfer begins.

Both memory DMA channels are now running and the data is transferred from the source address to the destination address. The DMA channel enters the IDLE state upon completion of the work unit.

Programming Concepts

Using the features, operating modes, and event control for the DMA channel to their greatest potential requires an understanding of some DMA channel-related concepts.

Synchronization of Software and DMA

A critical element of software DMA management is the synchronization of DMA work unit completion with software. This synchronization can be achieved using DMA channel interrupt request and trigger events and using a poll of the status bits of these events within the DMA channel registers, or combining these techniques. Processor polling of DMA address/count/status for completion is not a recommended programming practice. The requirements and limitations of processor polling place significant responsibility onto the code developer to be deeply aware of the underlying hardware. The interrupt requests and triggers are designed for efficient code development and reuse.

Interrupt and Trigger Event-Based Synchronization

Interrupt and trigger based synchronization methods must avoid overrun. An overrun occurs when some events fail to invoke the event handler of a DMA channel for every event due to excessive latency in processing of events. The system design must ensure to either:

- Schedule only one event per channel (for example, at the end of a descriptor list), or
- Space the generated events sufficiently far apart in time that system processing budgets can guarantee service of every event.

The DMA channel issues status information through an interrupt request or trigger event or changes event status bits in the DMA_STAT register. This status guarantees that the last memory operation of the work unit is complete. For memory read DMA transactions, this status means that the FIFO of the DMA channel safely receives the final

memory read data. For DMA transactions writing to memory, this status indicates that the DMA channel received an acknowledge of completion of the last write transfer of the work unit.

Register Polling Based Synchronization

Do not poll the DMA channel registers (DMA_ADDR_CUR, DMA_DSCPTR_CUR, DMA_XCNT_CUR, or DMA_YCNT_CUR) as a method of precisely synchronizing DMA with data processing. This approach is inaccurate due to the operation of the DMA channel FIFOs and DMA or memory pipelining. The current address, pointer, and count registers change several cycles in advance of the completion of the corresponding memory operation. This timing is measurable from the time at which the results of the operation are first visible to the core by memory read or write instructions.

For example, in a DMA channel memory write operation to external memory, assume DMA channel *A* initiates a DMA channel write operation. For memories with access latency, this operation requires many system-clock cycles. Meanwhile, DMA channel *B* (which does not in itself incur latency) initiates a transfer, which stalls behind the slow operation of channel *A*. Software monitoring channel *B* could not safely conclude whether the memory location pointed to by the DMA_ADDR_CUR of channel *B*. Also, the software cannot conclude whether the register has been written based solely on the contents of this register.

Polling of the current address, pointer, and count registers can permit loose synchronization of DMA with software. But, the software must allow for the lengths of the DMA or memory pipeline. Also, software must consider the length of the DMA FIFO for a particular peripheral. If the FIFOs are filled with incomplete work, the DMA channel does not advance current address, pointer, or count registers. The incomplete work includes reads that have been started but have not yet finished.

Additionally, software must consider the length of the pipelines to the destination memory. If the DMA FIFO length and channel memory-pipeline length are added, software can estimate the maximum number of incomplete memory operations in progress.

NOTE: The estimate would be a maximum, as the DMA or memory pipeline can include traffic from other DMA channels.

Descriptor Queues

A system designer may want to write a DMA manager facility which accepts DMA requests from other software. The DMA manager software does not know in advance when new work requests are received or what these requests contain. The software could manage these transfers using a circular linked list of DMA descriptors. In such a list, each descriptor sets the DMA_DSCPTR_NXT descriptor, which points to the next descriptor set. And, the last descriptor set in the list points to the first descriptor set.

The code that writes into this descriptor list could use the circular addressing modes of the processor. This approach does not need to use comparison and conditional instructions to manage the circular structure. In this case, the DMA_DSCPTR_NXT descriptor of each descriptor set can be written once at startup, and skipped over as new contents are written for each descriptor.

The recommended method for synchronization of a descriptor queue is to use an interrupt or trigger. The descriptor queue is structured, such that (at least) the final valid descriptor set is always programmed to generate an interrupt or trigger event upon completion. More detail is provided in the following sections.

- Queues Using Event Generation for Every Descriptor Set
- Queues Using Minimal Events

Queues Using Event Generation for Every Descriptor Set

In this system, the DMA manager software synchronizes with the DMA channel by enabling an interrupt request or trigger on every descriptor set. Only use this method if the system design can guarantee that each work unit completion event is serviced separately (no interrupt or trigger overrun).

To maintain synchronization of the descriptor set queue, the non-interrupt software maintains a count of descriptor sets added to the queue. The event handler (either interrupt or trigger) maintains a count of completed descriptor sets removed from the queue. The counts are equal only when the DMA channel is paused after having processed all the descriptor sets.

When each new work unit event is received, the DMA manager software initializes a new descriptor set, taking care to set the flow to stop mode. Next, the software compares the descriptor set counts to determine whether the DMA channel is running. If the DMA channel is paused (counts equal), the software increments its count. Then, the software starts the DMA channel by writing the `DMA_CFG` of the new descriptor set.

If the counts are unequal, the software instead modifies the `DMA_CFG` of the next-to-last descriptor set, such that it now describes the newly queued descriptor set. This operation does not disrupt the DMA channel provided the rest of the descriptors of the set are initialized in advance. It is necessary to synchronize the software to the DMA to determine whether the DMA channel read the new or the old `DMA_CFG` value.

The event handler performs the synchronization operation. When an event is detected, the handler reads the `DMA_STAT` register of the DMA channel. If the `DMA_STAT.RUN` bit indicates that the DMA channel is running, the channel has moved on to processing another descriptor. The event handler can increment its count and exit. If the `DMA_STAT.RUN` bit indicates that the channel is not running, the channel is paused because either:

- There are no more descriptor sets to process, or
- The last descriptor set was queued too late

Where *too late* means that the modification of the `DMA_CFG` of the next-to-last descriptor set occurred *after* that descriptor was read into the DMA channel. In this case, the event handler does the following:

- Writes the `DMA_CFG` value appropriate for the last descriptor set to `DMA_CFG` register of the DMA channel,
- Increments the completed descriptor count, and
- Exits

If the event latencies of the system are large enough to cause any of the events to be dropped, this system can fail. An event handler capable of safely synchronizing multiple descriptor set interrupt requests is complex, performing several MMR accesses to ensure robust operation. In such a system environment, a minimal event synchronization method is preferred.

Queues Using Minimal Events

In this system, only one DMA interrupt request or trigger event is generated in the queue at any time. The DMA event handler for this system can also be extremely short. Here, the descriptor queue is organized into an *active* and a *waiting* portion, where events are enabled only on the last descriptor set in each portion.

When each new DMA request is processed, the software fills in the content of a new descriptor set and adds it to the waiting portion of the queue. The DMA_CFG descriptor of the descriptor set must have the flow set to stop mode. If more than one request is received before the DMA queue completion event occurs, the non-interrupt code queues later descriptor sets. It forms a waiting portion of the queue separate from the active portion of the queue that the DMA channel is processing. In other words, all but the last active descriptor sets contain flow values for a descriptor-based mode and have no event enable set.

The last active descriptor set has the stop flow mode and an event generation enabled. Also, all but the last waiting descriptor sets are configured for descriptor-based flow modes with no event generation. Only the last waiting descriptor set is configured for stop flow mode and event generation enabled. This configuration ensures that the DMA channel can automatically process the whole active queue before then issuing one event. Also, this arrangement makes it easy to start the waiting queue within the event handler by a single DMA_CFG register write.

After queuing a new waiting descriptor, the non-interrupt software leaves a message for its interrupt handler in a memory mailbox location. The location contains the desired DMA_CFG value for starting the first waiting descriptor set in the waiting queue (or 0, indicating no waiting descriptors).

The software must not modify the contents of the active descriptor set queue directly once processing by the DMA channel has started, unless careful synchronization measures are taken. In the most straightforward implementation of a descriptor set queue, the DMA manager software never modifies descriptors on the active queue. Instead, the DMA manager waits until the DMA queue completion event indicates that the processing of the entire active queue is complete.

When a DMA queue completion event is received, the event handler reads the mailbox from the non-interrupt software and writes the value to the DMA_CFG register of the DMA channel. This write to a register restarts the queue, effectively transforming the waiting queue to an active queue. The event handler then passes a message back to the non-interrupt software indicating the location of the last descriptor set accepted into the active queue.

However, the event handler can read its mailbox and find a DMA_CFG value of zero, indicating there is no more work to perform. It then passes an appropriate message back to the non-interrupt software indicating that the queue has stopped.

The non-interrupt software which accepts new DMA work unit requests must synchronize the activation of a new work unit with the interrupt handler. If the queue has stopped (the mailbox from the event handler is zero), the non-interrupt software must start the queue. (The queue starts by writing the first descriptor sets DMA_CFG value to the DMA_CFG register of the channel). If the queue is not stopped, the non-interrupt software must not write the

DMA_CFG register. (This write causes a DMA error). Instead, it must queue the descriptor onto the waiting queue and update its mailbox directed to the event handler.

CM41X_M4 DMA Register Descriptions

The Direct Memory Access module (DMA) contains the following registers.

Table 17-12: CM41X_M4 DMA Register List

Name	Description
DMA_ADDRSTART	Start Address of Current Buffer Register
DMA_ADDR_CUR	Current Address Register
DMA_BWLCNT	Bandwidth Limit Count Register
DMA_BWLCNT_CUR	Bandwidth Limit Count Current Register
DMA_BWMCNT	Bandwidth Monitor Count Register
DMA_BWMCNT_CUR	Bandwidth Monitor Count Current Register
DMA_CFG	Configuration Register
DMA_DSCPTR_CUR	Current Descriptor Pointer Register
DMA_DSCPTR_NXT	Pointer to Next Initial Descriptor Register
DMA_DSCPTR_PRV	Previous Initial Descriptor Pointer Register
DMA_STAT	Status Register
DMA_XCNT	Inner Loop Count Start Value Register
DMA_XCNT_CUR	Current Count (1D) or Intra-row XCNT (2D) Register
DMA_XMOD	Inner Loop Address Increment Register
DMA_YCNT	Outer Loop Count Start Value (2D only) Register
DMA_YCNT_CUR	Current Row Count (2D only) Register
DMA_YMOD	Outer Loop Address Increment (2D only) Register

Start Address of Current Buffer Register

The DMA_ADDRSTART register contains the start address of the work unit currently targeted for DMA. This register is read/write prior to enabling the channel, but is read-only after enabling channel.

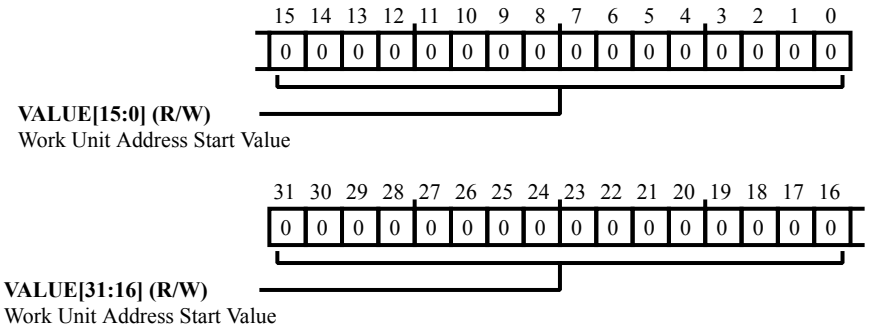


Figure 17-5: DMA_ADDRSTART Register Diagram

Table 17-13: DMA_ADDRSTART Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Address Start Value. The DMA_ADDRSTART.VALUE bit field contains the start address of the work unit currently targeted for DMA.

Current Address Register

The DMA_ADDR_CUR register contains the present memory transfer address for a given work unit. At the start of a work unit, the DMA_ADDR_CUR register is loaded from the DMA_ADDRSTART register, and the DMA_ADDR_CUR register is incremented as each transfer occurs. The DMA_ADDR_CUR register is read/write prior to enabling the channel, but is read-only after enabling the channel.

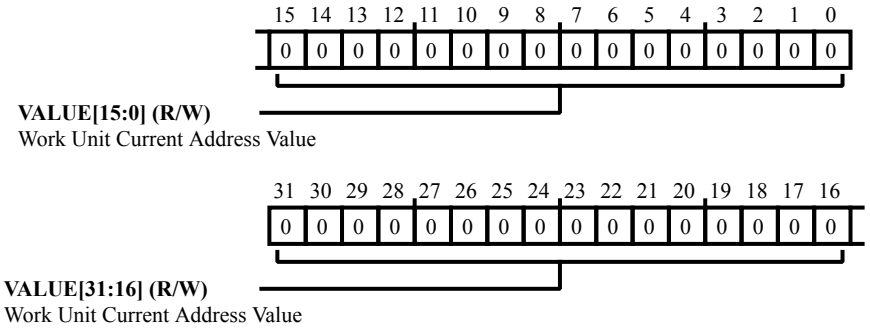


Figure 17-6: DMA_ADDR_CUR Register Diagram

Table 17-14: DMA_ADDR_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Current Address Value. The DMA_ADDR_CUR.VALUE bit field contains the present memory transfer address for a given work unit.

Bandwidth Limit Count Register

The DMA_BWLCNT register contains a count that determines how often the DMA issues memory transactions. The DMA loads the value from the DMA_BWLCNT register into the DMA_BWLCNT_CUR register and decrements the current value each SCLK cycle. When DMA_BWLCNT_CUR reaches 0x0000, the next request is issued, and the DMA reloads DMA_BWLCNT_CUR. This bandwidth limit functionality is not applied to descriptor fetch requests. Programming 0x0000 allows the DMA to request as often as possible. 0xFFFF is a special case and causes requests to stop.

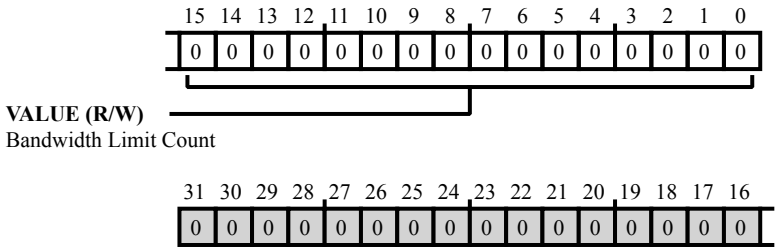


Figure 17-7: DMA_BWLCNT Register Diagram

Table 17-15: DMA_BWLCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Bandwidth Limit Count. The DMA_BWLCNT.VALUE bit field contains a count that determines how often the DMA issues memory transactions.

Bandwidth Limit Count Current Register

The DMA_BWLCNT_CUR register contains the number of SCLK count cycles remaining before the next request is issued.

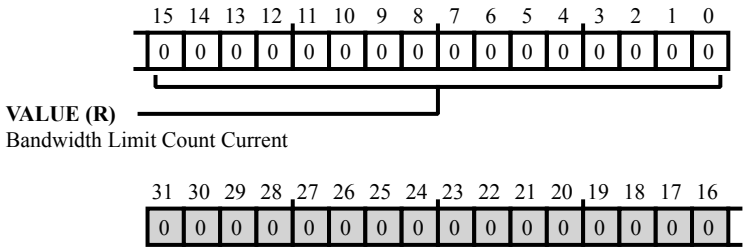


Figure 17-8: DMA_BWLCNT_CUR Register Diagram

Table 17-16: DMA_BWLCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Bandwidth Limit Count Current. The DMA_BWLCNT_CUR.VALUE bit field contains the number of SCLK count cycles remaining before the next request is issued.

Bandwidth Monitor Count Register

The DMA_BWMCNT register contains the maximum number of SCLK cycles allowed for a work unit to complete. Each time the DMA_CFG register is written (MMR access only), a work unit ends or an autobuffer wraps. The DMA loads the value in this register into the DMA_BWMCNT_CUR register.

The DMA decrements DMA_BWMCNT_CUR every SCLK a work unit is active. If the DMA_BWMCNT_CUR register reaches 0x0000_0000, the DMA_STAT.IRQERR bit is set, and the DMA_STAT.ERRC bit field is set to 0x6. The DMA_BWMCNT_CUR remains at 0x0000_0000 until it is reloaded when the work unit completes.

Unlike other errors, a bandwidth monitor error does not stop work unit processing. Programming 0x0000_0000 disables bandwidth monitor functionality.

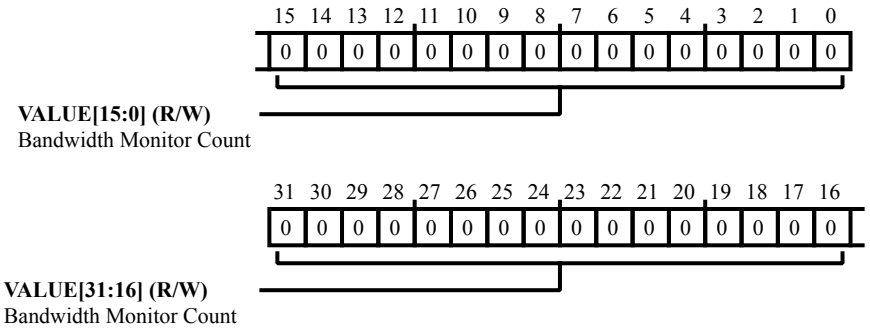


Figure 17-9: DMA_BWMCNT Register Diagram

Table 17-17: DMA_BWMCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Bandwidth Monitor Count. The DMA_BWMCNT.VALUE bit field contains the maximum number of SCLK cycles allowed for a work unit to complete.

Bandwidth Monitor Count Current Register

The DMA_BWMCNT_CUR register contains the number of cycles remaining for the current descriptor to complete.

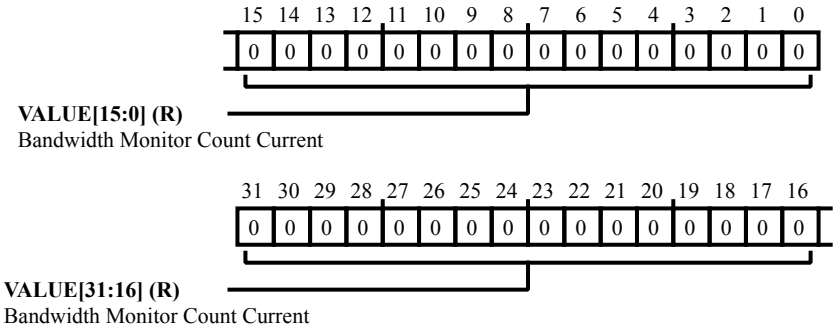


Figure 17-10: DMA_BWMCNT_CUR Register Diagram

Table 17-18: DMA_BWMCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Bandwidth Monitor Count Current. The DMA_BWMCNT_CUR.VALUE bit field contains the number of cycles remaining for the current descriptor to complete.

Configuration Register

The DMA_CFG register sets up DMA parameters and operation modes. Writing to the DMA_CFG register while a DMA process is already running causes a DMA error (except when clearing the DMA_CFG.EN bit).

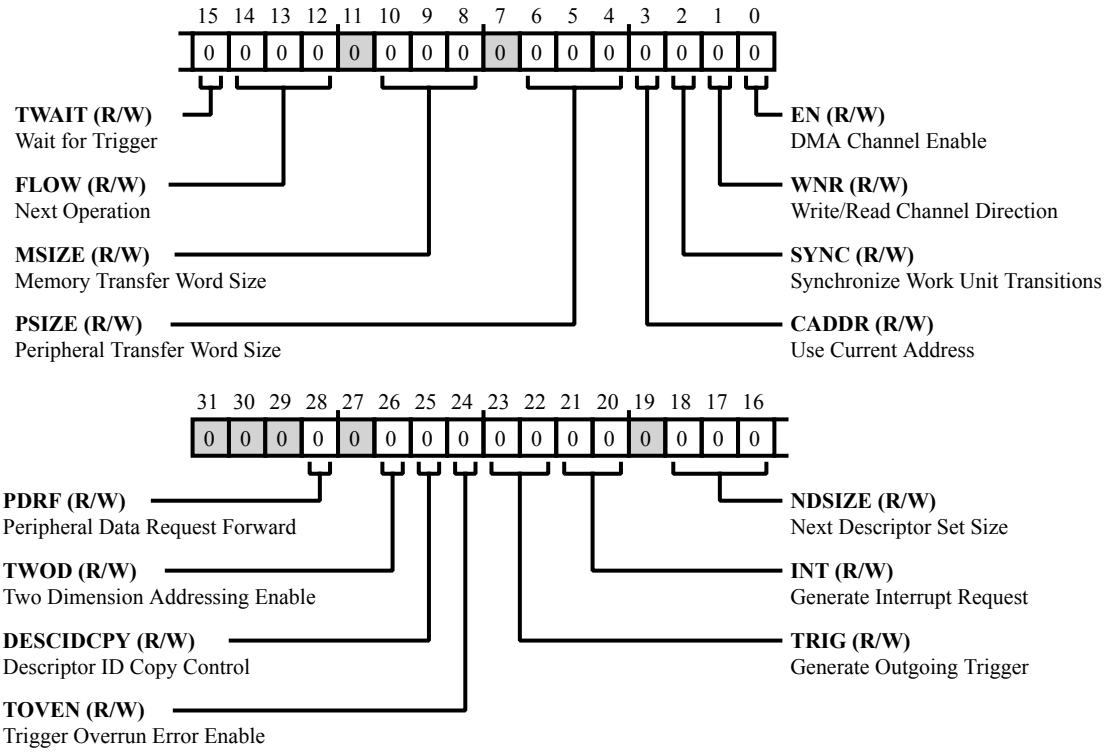


Figure 17-11: DMA_CFG Register Diagram

Table 17-19: DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	PDRF	Peripheral Data Request Forward. The DMA_CFG.PDRF bit defines how the DMA handles data requests from the peripheral while in idle state after a stop mode or memory read work unit. If set, the DMA forwards the peripheral data request as an interrupt. This bit applies only to the DMA_CFG.FLOW bits configured for stop and DMA_CFG.WNR bits configured for memory read.	
		0	Peripheral Data Request Not Forwarded
		1	Peripheral Data Request Forwarded

Table 17-19: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	TWOD	Two Dimension Addressing Enable. The DMA_CFG.TWOD bit selects whether the DMA addressing involves only DMA_XCNT and DMA_XMOD (one-dimensional DMA) or also involves DMA_YCNT and DMA_YMOD (two-dimensional DMA).	
		0	One-Dimensional Addressing
		1	Two-Dimensional Addressing
25 (R/W)	DESCIDCPY	Descriptor ID Copy Control. The DMA_CFG.DESCIDCPY bit specifies when to copy the initial descriptor pointer to the DMA_DSCPTR_PRV register. A bus write to the DMA_CFG register to clear the DMA_CFG.EN bit causes the DMA to immediately use the new value of the DMA_CFG.DESCIDCPY bit. To preserve consistency (if required by application), match the new value of this bit to the previous value.	
		0	Never Copy
		1	Copy on Work Unit Complete
24 (R/W)	TOVEN	Trigger Overrun Error Enable. A trigger overrun occurs if more than one trigger was received before the DMA reached the trigger wait state. If DMA_CFG.TOVEN is set, a trigger overrun causes the DMA to flag an error. In cases where a trigger overrun is not a problem, clearing DMA_CFG.TOVEN prevents the overrun from causing an error and halting the DMA. The DMA_CFG.TOVEN operates independently of the DMA_CFG.TWAIT bit selection.	
		0	Ignore Trigger Overrun
		1	Error on Trigger Overrun

Table 17-19: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
23:22 (R/W)	TRIG	Generate Outgoing Trigger. The DMA_CFG.TRIG selects whether the DMA issues an outgoing trigger, based on the work unit counter values. In one-dimensional mode, the only options are to trigger at the end of the whole work unit (trigger when DMA_XCNT_CUR reaches 0) or not to trigger at all. If in one-dimensional addressing mode, programming the DMA_CFG.TRIG bit field to trigger when DMA_YCNT_CUR reaches 0 (or to reserved) causes the DMA to flag a configuration error. In two-dimensional addressing mode, the trigger options are: at the end of each row of the inner loop (when DMA_XCNT_CUR reaches 0), only after completing the whole work unit (when DMA_YCNT_CUR reaches 0), or no trigger. If in two-dimensional mode and set to trigger when DMA_XCNT_CUR reaches 0, the DMA also issues a trigger at the end of the work unit. If in two-dimensional addressing mode, programming DMA_CFG.TRIG to reserved causes the DMA to flag a configuration error. If DMA_CFG.TRIG is non-zero and the peripheral issues a finish command, the DMA issues a trigger after the finish procedure is complete. For more information about trigger generation timing, see the trigger section of the DMA functional description.	
		0	Never Assert Trigger
		1	Trigger When XCNTCUR Reaches 0
		2	Trigger When YCNTCUR Reaches 0
		3	Reserved

Table 17-19: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	INT	Generate Interrupt Request. The DMA_CFG.INT bit field selects whether an interrupt request goes to the core based on work unit status or a peripheral interrupt request. For one-dimensional mode, setting the DMA_CFG.INT bits to generate an interrupt request when the DMA_YCNT_CUR register reaches 0 causes the DMA to flag a configuration error. The peripheral interrupt setting lets the DMA generate the last grant indication and to accept and or forward the peripheral interrupt command. The peripheral interrupt selection applies only to the DMA_CFG.FLOW bits set for stop and the DMA_CFG.WNR bits set for memory read. If the DMA_CFG.INT is set for interrupt request on count completion (DMA_XCNT_CUR or DMA_YCNT_CUR reach 0) and the peripheral issues a finish command, the DMA issues an interrupt request after the finish procedure is complete. For more information, see the sections on interrupt generation and peripheral control in the DMA functional description.	
		0	Never Assert Interrupt
		1	Interrupt When X Count Expires
		2	Interrupt When Y Count Expires
		3	Peripheral Interrupt request
18:16 (R/W)	NDSIZE	Next Descriptor Set Size. The DMA_CFG.NDSIZE bit field specifies the number of descriptor elements in memory to load during the next descriptor fetch. The DMA loads the descriptors in a specific order. The descriptor set contains the next descriptor pointer when it is a descriptor list. The descriptor set does not contain the next descriptor pointer when it is a descriptor array.	
		0	Fetch One Descriptor Element
		1	Fetch Two Descriptor Elements
		2	Fetch Three Descriptor Elements
		3	Fetch Four Descriptor Elements
		4	Fetch Five Descriptor Elements
		5	Fetch Six Descriptor Elements
		6	Fetch Seven Descriptor Elements
		7	Reserved

Table 17-19: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	TWAIT	Wait for Trigger. The DMA_CFG.TWAIT bit controls whether the DMA waits for an incoming trigger from another channel or user. If the DMA_CFG.TWAIT bit is set, the DMA enters the wait state before starting the next work unit, including descriptor fetch when in descriptor mode. Do not use the wait for trigger control for descriptor-on-demand mode which causes an error. For more information, see the trigger section of the DMA functional description.	
		0	Begin Work Unit Automatically (No Wait)
		1	Wait for Trigger (Halt before Work Unit)
14:12 (R/W)	FLOW	Next Operation. The DMA_CFG.FLOW bit field selects the descriptor handling options.	
		0	STOP. See the Stop Flow Mode section.
		1	AUTO. See the Autobuffer Flow Mode section.
		2	Reserved
		3	Reserved
		4	DSCL. See the Descriptor List Mode section.
		5	DSCA. See the Descriptor Array Mode section.
		6	Descriptor On-Demand List. See the Descriptor List Mode section.
		7	Descriptor On Demand Array. See the Descriptor On Demand section.
10:8 (R/W)	MSIZE	Memory Transfer Word Size. The DMA_CFG.MSIZE bits select memory transfer sizes of 8-, 16-, 32-, 64-, 128-, or 256-bit words. The transfer start address (DMA_ADDRSTART) and transfer increment values (DMA_XMOD, and, if needed, DMA_YMOD) must be a multiple of the memory transfer unit size.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
		4	16 Bytes
		5	32 Bytes

Table 17-19: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6:4 (R/W)	PSIZE	Peripheral Transfer Word Size. The DMA_CFG.PSIZE bits select peripheral transfer sizes as 8, 16, 32, or 64 bits wide. Each request and grant results in a single peripheral access. There are no bursts on the peripheral bus, so the DMA_CFG.PSIZE selection must be less than, or equal to, the width of the bus. If the selection is greater than the bus width, a configuration error occurs. The peripheral bus of the processor is dedicated to DMA and peripheral accesses.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
3 (R/W)	CADDR	Use Current Address. When the DMA_CFG.CADDR bit is cleared, the DMA loads the DMA_ADDRSTART register on the first access of the work unit. When the DMA_CFG.CADDR bit is set, the DMA uses the DMA_ADDR_CUR register value for the starting address for the work unit and writes the same value to the DMA_ADDRSTART register. This operation permits continuation of a previous work unit. When DMA uses this mode, the fetched start address value (as part of the descriptor set at the end of a descriptor list or array) is ignored.	
		0	Load Starting Address
		1	Use Current Address

Table 17-19: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	SYNC	Synchronize Work Unit Transitions. Setting the DMA_CFG.SYNC bit clears the DMA FIFO and pointers before starting the first work unit of a work unit chain. When the transfer direction is memory read/transmit (DMA_CFG.WNR =0), the DMA waits until all data transmits to a peripheral before moving on to the next work unit, clearing the FIFO and pointers. When the transfer direction is memory write/receive (DMA_CFG.WNR =1), the DMA ignores the DMA_CFG.SYNC bit value after processing the first work unit of a work unit chain. As a channel can receive data when turned on but idle, data from the peripheral can still be in the FIFO even though the channel was not programmed. When the DMA_CFG.SYNC bit field is set at the beginning of a work unit chain (during the first work unit), the DMA clears the FIFO, erasing the data put into the FIFO while the channel was idle. Syncing lets programs change the DMA_CFG.PSIZE between individual work units and (in some cases) work unit chains. The sync resets the pointers in the FIFO, preventing misaligned FIFO access. Programs can change the DMA_CFG.MSIZE field between consecutive work units, independent of the DMA_CFG.SYNC bit setting. Syncing also permits changes to transfer direction. Because the data in the FIFO is eliminated, the data that went into the FIFO from one direction (transmit or receive) is not sent back in the other direction after the direction change.	
		0	No Synchronization
		1	Synchronize Channel
1 (R/W)	WNR	Write/Read Channel Direction. The DMA_CFG.WNR bit selects receive (write to memory) or transmit (read from memory) channel direction.	
		0	Transmit (Read from memory)
		1	Receive (Write to memory)

Table 17-19: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Channel Enable. The DMA_CFG.EN bit enables the selected DMA channel. When a peripheral DMA channel is enabled, data requests from the peripheral denote DMA requests. When a channel is disabled, the DMA unit ignores the peripheral data request and passes it directly to the system event controller. To avoid unexpected results, enable the DMA channel before enabling the peripheral, and disable the peripheral before disabling the DMA channel. A transition of the DMA_CFG.EN bit from 0 to 1 creates a hard reset of all internal counters and states, including the DMA_STAT register. (All other register values remain unaffected.) A transition from 1 to 0 maintains all counters and registers for reading and analysis. Note that if a descriptor loads when this bit is cleared (see the DMA_CFG.FLOW field), the DMA transitions to the off or idle state after the descriptor load is complete.	
		0	Disable
		1	Enable

Current Descriptor Pointer Register

The DMA_DSCPTR_CUR register contains the memory address for the next descriptor to be loaded. The DMA_DSCPTR_CUR register is read/write prior to enabling the channel, but is read-only after enabling the channel. For DMA_CFG.FLOW mode settings that involve descriptor fetches, this register is used to read descriptors into appropriate MMRs before a work unit begins. For descriptor list mode, the DMA_DSCPTR_CUR register is initialized from the DMA_DSCPTR_NXT register before fetching each descriptor set. Then, the address in the DMA_DSCPTR_CUR register increments as each descriptor is read.

When the entire descriptor set has been read, the DMA_DSCPTR_CUR register contains this value:

DMA_DSCPTR_CUR = Descriptor Start Address + Descriptor Size (# of elements)

For descriptor array mode, the DMA_DSCPTR_CUR register, and not the DMA_DSCPTR_NXT register, must be programmed by a MMR access before starting DMA operation.

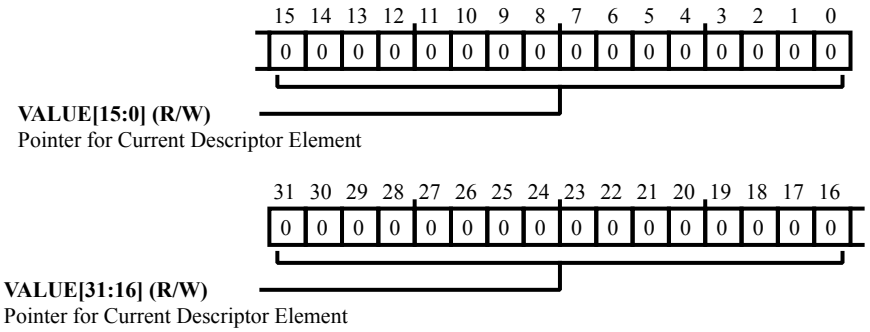


Figure 17-12: DMA_DSCPTR_CUR Register Diagram

Table 17-20: DMA_DSCPTR_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer for Current Descriptor Element. The DMA_DSCPTR_CUR.VALUE bit field contains the memory address for the next descriptor to be loaded.

Pointer to Next Initial Descriptor Register

The DMA_DSCPTR_NXT register specifies the start location of the next descriptor set, which begins when the DMA activity specified by the current descriptor set completes. This register is read/write prior to enabling the channel, but is read-only after enabling channel.

The DMA_DSCPTR_NXT register is only used in descriptor list mode. At the start of a descriptor fetch in this mode, the DMA_DSCPTR_NXT register is copied into the DMA_DSCPTR_CUR register. During descriptor fetch, the DMA increments the DMA_DSCPTR_CUR register value after reading each element of the descriptor set.

In descriptor list mode, the DMA_DSCPTR_NXT register (not the DMA_DSCPTR_CUR register) must be programmed directly through MMR access, before the DMA operation is started. In descriptor array mode, the DMA disregards the DMA_DSCPTR_NXT register and uses the DMA_DSCPTR_CUR register to control descriptor fetch.

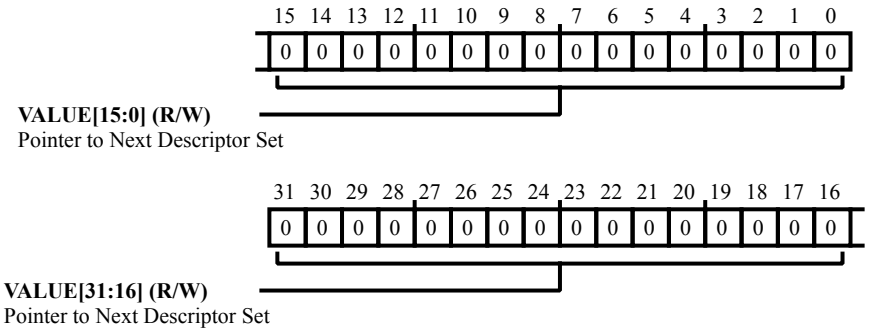


Figure 17-13: DMA_DSCPTR_NXT Register Diagram

Table 17-21: DMA_DSCPTR_NXT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Pointer to Next Descriptor Set. The DMA_DSCPTR_NXT.VALUE bit field specifies the start location of the next descriptor set.

Previous Initial Descriptor Pointer Register

The DMA_DSCPTR_PRV register contains the initial descriptor pointer for the previous work unit. If DMA_CFG.DESCIDCPYis set, the DMA copies the initial descriptor pointer to DMA_DSCPTR_PRV after the work unit completes. Otherwise, the value is not updated.

To indicate an overrun, bit 0 of the DMA_DSCPTR_PRV register is used as a previous descriptor pointer overrun (PDPO) status bit. Due to aligned addressing combined with all descriptors being 32 bits in width, bits 0 and 1 of all descriptor pointers must be zero. Otherwise, an alignment error occurs when used for descriptor fetches. As a result, bit 1 and 0 of the DMA_DSCPTR_PRV register can be used for status. For more information, see the section on descriptor pointer capture in the DMA functional description.

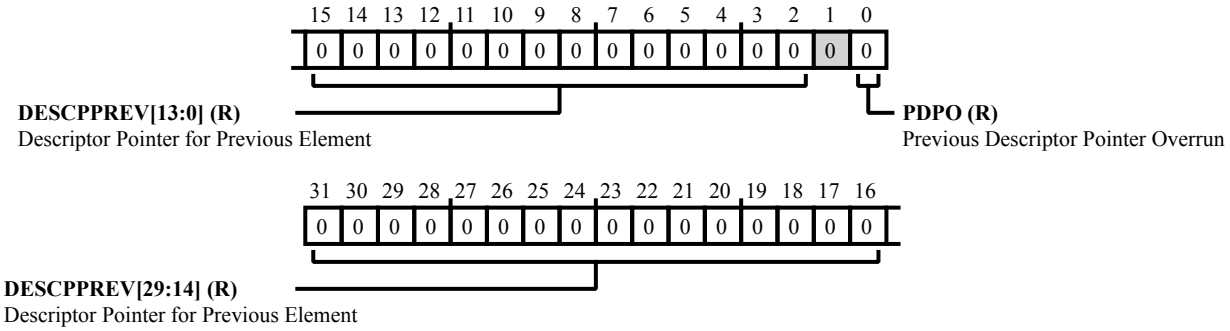


Figure 17-14: DMA_DSCPTR_PRV Register Diagram

Table 17-22: DMA_DSCPTR_PRV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:2 (R/NW)	DESCPPREV	Descriptor Pointer for Previous Element. The DMA_DSCPTR_PRV.DESCPPREV bit field contains the initial descriptor pointer for the previous work unit.	
0 (R/NW)	PDPO	Previous Descriptor Pointer Overrun. The DMA_DSCPTR_PRV.PDPO bit signifies an alignment error. Due to aligned addressing combined with all descriptors being 32 bits in width, bits 0 and 1 of all descriptor pointers must be zero. Otherwise, an alignment error would occur when used for descriptor fetches.	
		0	No Error Occurred
		1	Error Occurred

Status Register

The DMA_STAT register indicates the status of DMA work units, the FIFO, errors, interrupts, and triggers.

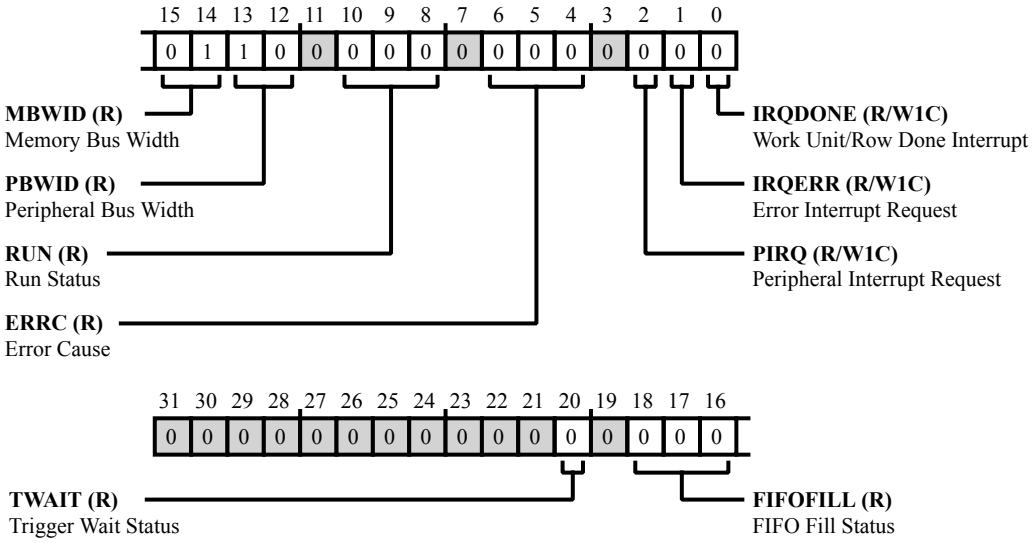


Figure 17-15: DMA_STAT Register Diagram

Table 17-23: DMA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	TWAIT	Trigger Wait Status. The DMA_STAT.TWAIT bit indicates whether the DMA has or has not received a trigger. This bit is set until it reaches the next wait state. At that point, the bit is cleared, the DMA stops processing that work unit, and the following work unit processes. The DMA does not distinguish between one or more triggers received.	
		0	No Trigger Received
		1	Trigger Received
18:16 (R/NW)	FIFOFILL	FIFO Fill Status. The DMA_STAT.FIFOFILL bit field reports the quantity of data in the FIFO relative to available space.	
		0	Empty
		1	Empty < FIFO = 1/4 Full
		2	1/4 Full < FIFO = 1/2 Full
		3	1/2 Full < FIFO = 3/4 Full
		4	3/4 Full < FIFO = Full
		5	Reserved

Table 17-23: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	Reserved
		7	Full
15:14 (R/NW)	MBWID	Memory Bus Width. The DMA_STAT.MBWID bit field indicates the width of the memory bus connected to this DMA.	
		0	2 Bytes
		1	4 Bytes
		2	8 Bytes
		3	16 Bytes
13:12 (R/NW)	PBWID	Peripheral Bus Width. The DMA_STAT.PBWID bit field indicates the width of the peripheral bus connected to this DMA.	
		0	1 Byte
		1	2 Bytes
		2	4 Bytes
		3	8 Bytes
10:8 (R/NW)	RUN	Run Status. The DMA_STAT.RUN bit field reports the DMA's current operational state. If the DMA is in idle or stop state, the DMA_CFG.EN bit is either 0 or 1. This bit field does not clear when a transition of the DMA_CFG.EN bit from 0 to 1 occurs. The DMA_STAT.RUN clears automatically when the DMA completes.	
		0	Idle/Stop State
		1	Descriptor Fetch
		2	Data Transfer
		3	Waiting for Trigger
		4	Waiting for Write ACK/FIFO Drain to Peripheral
		5	Reserved
		6	Reserved
		7	Reserved
6:4 (R/NW)	ERRC	Error Cause. When an interrupt request error occurs (DMA_STAT.IRQERR), the DMA updates the DMA_STAT.ERRC bit field to identify the type of error. For more information, see the errors section of the DMA functional description.	

Table 17-23: DMA_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Configuration Error
		1	Illegal Write Occurred While Channel Running
		2	Address Alignment Error
		3	Memory Access or Fabric Error
		4	Reserved
		5	Trigger Overrun
		6	Bandwidth Monitor Error
		7	Reserved
2 (R/W1C)	PIRQ	Peripheral Interrupt Request. The DMA_STAT.PIRQ bit indicates a peripheral interrupt request. Programs can use the DMA_STAT.PIRQ bit status to help determine which DMA asserted the interrupt request. This bit also helps to distinguish between an interrupt request caused by the state of the work unit and an interrupt request caused by the peripheral.	
		0	No Interrupt request
		1	Interrupt Request signaled by peripheral
1 (R/W1C)	IRQERR	Error Interrupt Request. The DMA_STAT.IRQERR bit indicates that the DMA has detected a documented rule violation during DMA programming or operation. The DMA cannot, however, flag all possible programming or operation issues to indicate errors. Programmers can use the DMA_STAT.IRQERR bit to help determine which DMA issued the error interrupt request. The DMA_STAT.IRQERR does not clear a transition of the DMA_CFG.EN bit from 0 to 1. Clear the DMA_STAT.IRQERR bit with a write-1-to-clear operation prior to the DMA_CFG.EN transition for the fields to be reset.	
		0	No Error
		1	Error Occurred
0 (R/W1C)	IRQDONE	Work Unit/Row Done Interrupt. The DMA_STAT.IRQDONE bit indicates that the DMA has detected the completion of a work unit or row (inner loop count) and has issued an interrupt request. Programs can use the DMA_STAT.IRQDONE status to help determine which DMA asserted the interrupt request. Programs can also use these bits to help distinguish between an interrupt request based on the state of the work unit and an interrupt request made by the peripheral. For more information, see the interrupts section of the DMA functional description.	
		0	Inactive
		1	Active

Inner Loop Count Start Value Register

For 2D DMA, the DMA_XCNT register contains the inner loop count. This value selects the number of DMA_CFG.MSIZE size data transfers that make up the length of a row. For 1D DMA, the DMA_XCNT register specifies the number of DMA_CFG.MSIZE size data transfers for the entire work unit. The DMA_XCNT register is read/write prior to enabling the channel, but is read-only after enabling channel. Note that the DMA generates a configuration error if this register is 0x0 when a work unit begins.

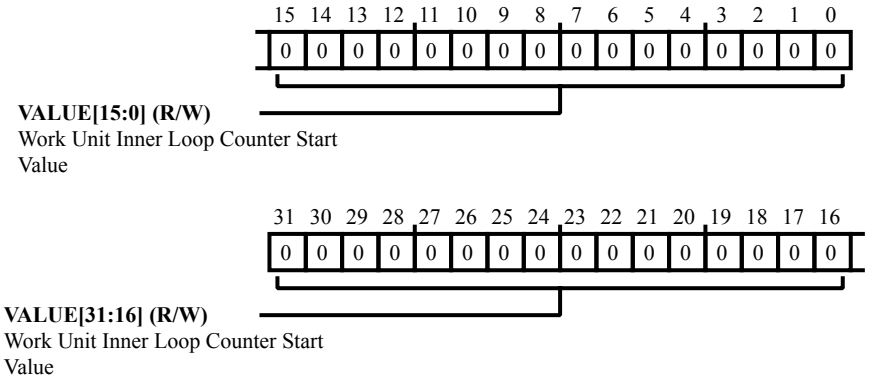


Figure 17-16: DMA_XCNT Register Diagram

Table 17-24: DMA_XCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Start Value. For 2D DMA, the DMA_XCNT.VALUE bit field contains the inner loop count. For 1D DMA, DMA_XCNT.VALUE specifies the number of DMA_CFG.MSIZE size data transfers for the entire work unit.

Current Count (1D) or Intra-row XCNT (2D) Register

For 1D DMA, the DMA loads the DMA_XCNT_CUR register from the DMA_XCNT register at the beginning of each work unit. For 2D DMA, the DMA loads the DMA_XCNT_CUR register from the DMA_XCNT register after the end of each row. The DMA decrements the value in DMA_XCNT_CUR register each time a DMA_CFG.MSIZE size data transfer occurs. When the count in DMA_XCNT_CUR register expires, the work unit is complete. In 2D DMA, the DMA_XCNT_CUR value is 0 only when the entire transfer is complete.

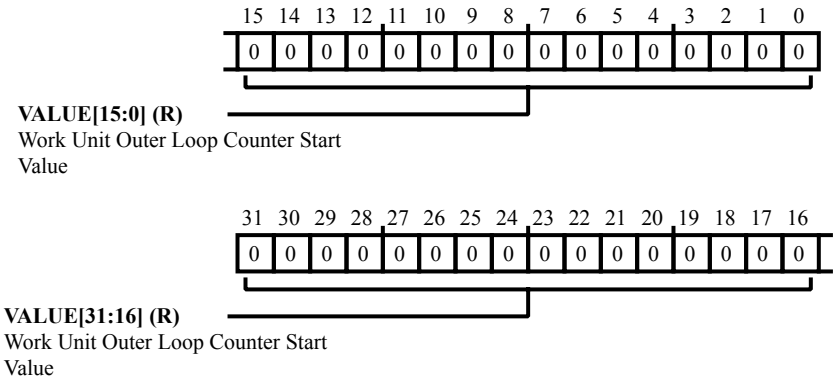


Figure 17-17: DMA_XCNT_CUR Register Diagram

Table 17-25: DMA_XCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit Outer Loop Counter Start Value. For 1D DMA, the DMA_XCNT_CUR.VALUE bit field holds the DMA_XCNT value at the beginning of each work unit. For 2D DMA, the DMA_XCNT_CUR.VALUE bit field holds the DMA_XCNT value after the end of each row.

Inner Loop Address Increment Register

The DMA_XMOD register contains a signed, two's-complement byte address increment. In 1D DMA, this increment is the stride that is applied after each DMA_CFG.MSIZE size data transfer. The DMA_XMOD register is read/write prior to enabling the channel, but is read-only after enabling the channel.

The DMA_XMOD register value is specified in bytes, regardless of the work unit size. In 2D DMA, this increment is applied after each DMA_CFG.MSIZE size data transfer in the inner loop, up to but not including the last DMA_CFG.MSIZE size data transfer in each inner loop. After the last DMA_CFG.MSIZE size data transfer in each inner loop, the DMA_YMOD register is applied instead, including the last DMA_CFG.MSIZE size data transfer of a work unit.

The DMA_XMOD field can be set to 0. In this case, DMA is performed repeatedly to or from the same address. This approach can be useful for transferring data between a data register and an external memory-mapped peripheral.

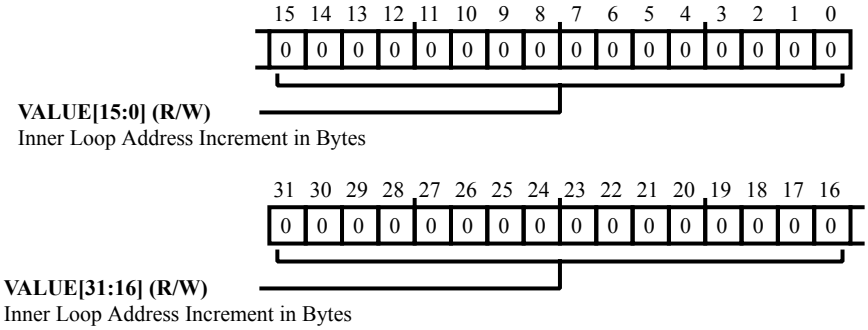


Figure 17-18: DMA_XMOD Register Diagram

Table 17-26: DMA_XMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Inner Loop Address Increment in Bytes. The DMA_XMOD.VALUE bit field contains a signed, two's-complement byte address increment.

Outer Loop Count Start Value (2D only) Register

For 2D DMA, the DMA_YCNT register contains the outer loop count. This register is not used in 1D DMA mode. The DMA_YCNT register specifies the number of rows in the outer loop of a 2D DMA sequence. The DMA_YCNT register is read/write prior to enabling the channel, but is read-only after enabling channel. Note that the DMA generates a configuration error if this register is 0x0 when a work unit begins.

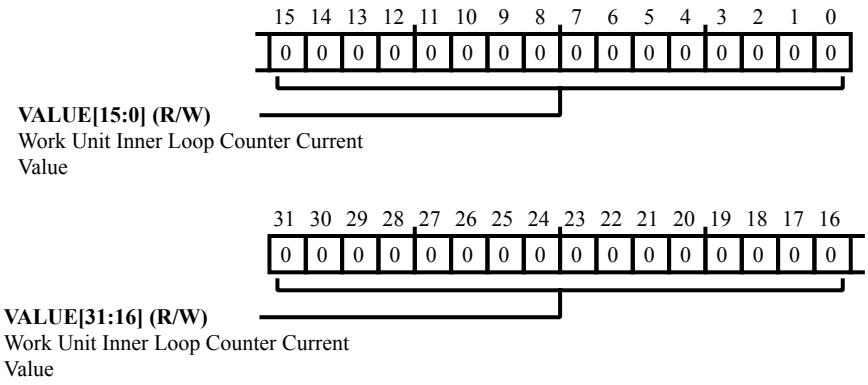


Figure 17-19: DMA_YCNT Register Diagram

Table 17-27: DMA_YCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Work Unit Inner Loop Counter Current Value. For 2D DMA, the DMA_YCNT.VALUE bit field contains the outer loop count.

Current Row Count (2D only) Register

For 2D DMA, the DMA loads the DMA_YCNT_CUR register from the DMA_YCNT register at the beginning of each 2D DMA session. The DMA_YCNT_CUR register is not used for 1D DMA. The DMA decrements the DMA_YCNT_CUR register each time the DMA_XCNT_CUR register expires during 2D DMA operation, signifying the completion of an entire row transfer.

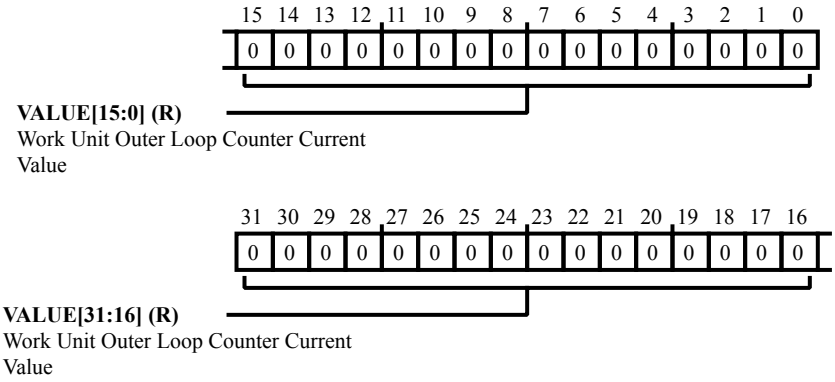


Figure 17-20: DMA_YCNT_CUR Register Diagram

Table 17-28: DMA_YCNT_CUR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Work Unit Outer Loop Counter Current Value. For 2D DMA, the DMA_YCNT_CUR.VALUE bit field holds the value from the DMA_YCNT register at the beginning of each 2D DMA session.

Outer Loop Address Increment (2D only) Register

The DMA_YMOD register contains a signed, two's-complement value. This byte address increment is applied after each decrement of the DMA_YCNT_CUR register. The value is the offset between the last word of one row and the first word of the next row. Note that DMA_YMOD is specified in bytes, regardless of the work unit size. The DMA_YMOD register is read/write prior to enabling the channel, but is read-only after enabling the channel.

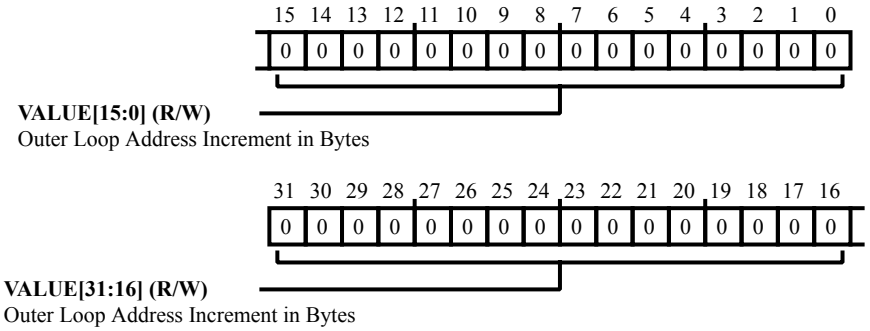


Figure 17-21: DMA_YMOD Register Diagram

Table 17-29: DMA_YMOD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Outer Loop Address Increment in Bytes. The DMA_YMOD.VALUE bit field contains a signed, two's-complement value.

18 General-Purpose Ports (PORT)

This section describes general-purpose ports, pin multiplexing, general-purpose input/output (GPIO) functionality, and pin interrupts. The general-purpose ports provide the following three functions:

- Pin multiplexing scheme
- GPIO functionality
- Pin interrupt requests

NOTE: In this chapter, the naming convention for registers and bits omits the alphabetic group enumeration to refer to any and all of the ports. For example, `PORT_FER` represents registers `PORTA_FER`, `PORTB_FER`, and so on. Likewise `PORT_FER.PX1` represents bits PA1, PB1, and so on.

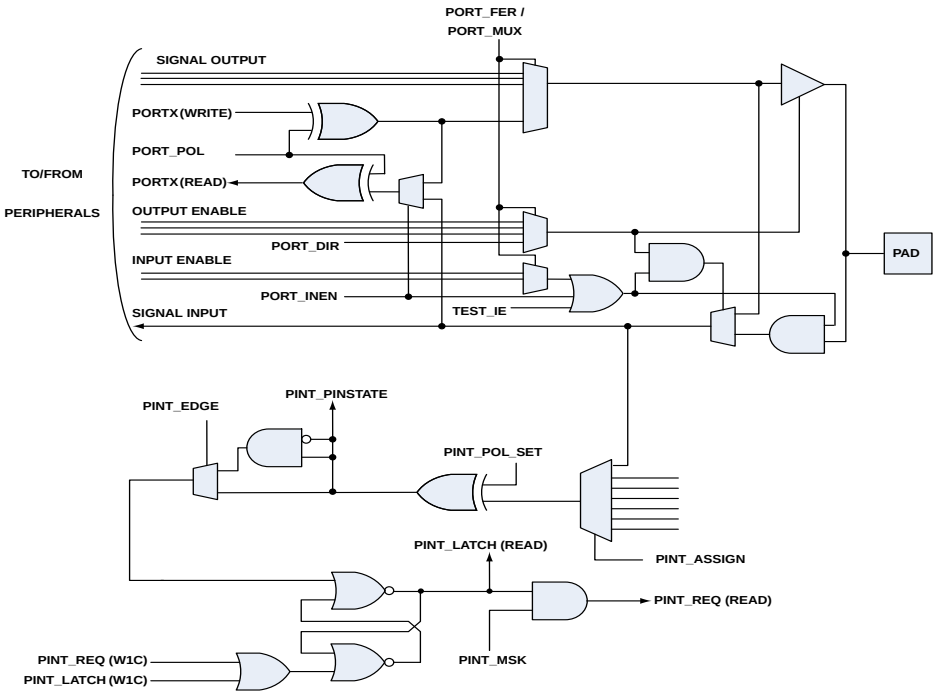


Figure 18-1: Simplified GPIO and Pin Interrupt Signal Flow

PORT Features

The PORTs include the following features:

- Input mode, output mode, and open-drain mode of GPIO operation
- Port multiplexing controlled on a pin-by-pin basis
- No external glue hardware required for unused pins
- All port pins provide interrupt request functionality
- Byte-wide pin-to-interrupt request assignment

PORT Functional Description

The number of ports and each's composition are defined in the processor datasheet. Each port has a dedicated set of MMR registers that control pin functions and operates in general-purpose I/O (GPIO) mode by default, as controlled by the port-specific PORT_FER register. Each bit in this register, as well as the other PORT MMRs, represents a specific GPIO pin on the specified port.

Input Mode, Output Mode, and Open-Drain Mode of GPIO Operation

At reset, every GPIO pin defaults to input mode with the input drivers disabled. To enable any GPIO input driver, set the bits corresponding to the individual pins in the appropriate input enable register (PORT_INEN).

The GPIO output drivers are enabled by setting the corresponding bits in the direction registers (PORT_DIR).

The PORT can use every GPIO in open-drain mode by clearing the respective bit in the PORT_DATA register or setting the respective bit in the PORT_DATA_CLR register. Then, set the corresponding bit in the PORT_INEN register. Read from the PORT_DATA register to obtain the status from the pin.

Port Multiplexing Controlled on Pin-by-Pin Basis

Each port has two dedicated MMRs that control the port multiplexing, the 16-bit function enable (PORT_FER) registers and the 32-bit port multiplexing (PORT_MUX) registers.

All Port Pins Provide Interrupt Functionality

Pin interrupts are completely decoupled from GPIO functionality. Pins are connected to the system event controller (SEC) via the PINTx modules, each of which is configurable in terms of which port pins are sensed for interrupt generation.

CM41X_M4 PORT Register List

The PORT module (PORT) regulates the use of the multiplexable processor pins. Every port pin can operate in general-purpose I/O (GPIO) mode or as an alternate function. This GPIO operation is the default after processor reset and is controlled by a set of registers that control GPIO functionality. Every bit in these registers represents a certain GPIO pin of a specific port. For more information on PORT functionality, see the PORT register descriptions.

Table 18-1: CM41X_M4 PORT Register List

Name	Description
PORT_DATA	Port x GPIO Data Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_TGL	Port x GPIO Output Toggle Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_FER	Port x Function Enable Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_FER_SET	Port x Function Enable Set Register
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_LOCK	Port x GPIO Lock Register
PORT_MUX	Port x Multiplexer Control Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register
PORT_TRIG_TGL	Port x GPIO Trigger Toggle Register

CM41X_M4 PORT Trigger List

Table 18-2: CM41X_M4 PORT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 18-3: CM41X_M4 PORT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
19	PORTB_TOGGLE	PORTB Port Toggle Trigger	Pulse
20	PORTC_TOGGLE	PORTC Port Toggle Trigger	Pulse
21	PORTD_TOGGLE	PORTD Port Toggle Trigger	Pulse
22	PORTE_TOGGLE	PORTE Port Toggle Trigger	Pulse
23	PORTF_TOGGLE	PORTF Port Toggle Trigger	Pulse

CM41X_M4 PINT Register List

The Pin Interrupt module (PINT) controls the pin-to-interrupt assignment in a byte-wide manner. The pin-interrupt assignment registers do not consist of 32 individual bits. They consist of four control bytes, each functioning as a multiplexer control. For more information, see the PINT register descriptions.

All PINT registers are 32 bits wide and can be accessed by 32-bit load/store instructions. They also support 16-bit operation where the upper 16 bits are ignored and the application uses the lower 16 bits only. Consequently, all PINT registers support 32-bit accesses as well as 16-bit accesses for the lower half words. Applications may use faster 16-bit accesses as long as they do not require functionality of upper register halves.

Table 18-4: CM41X_M4 PINT Register List

Name	Description
PINT_ASSIGN	PINT Assign Register
PINT_EDGE_CLR	PINT Edge Clear Register
PINT_EDGE_SET	PINT Edge Set Register
PINT_INV_CLR	PINT Invert Clear Register
PINT_INV_SET	PINT Invert Set Register
PINT_LATCH	PINT Latch Register
PINT_MSK_CLR	PINT Mask Clear Register
PINT_MSK_SET	PINT Mask Set Register
PINT_PINSTATE	PINT Pin State Register
PINT_REQ	PINT Request Register

CM41X_M4 PINT Interrupt List

Table 18-5: CM41X_M4 PINT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
82	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level	

Table 18-5: CM41X_M4 PINT Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
83	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level	
84	PINT3_BLOCK	PINT3 Pin Interrupt Block	Level	
85	PINT4_BLOCK	PINT4 Pin Interrupt Block	Level	
86	PINT5_BLOCK	PINT5 Pin Interrupt Block	Level	
87	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level	

CM41X_M0 PINT Interrupt List

Table 18-6: CM41X_M0 PINT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
14	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level	

CM41X_M4 PINT Trigger List

Table 18-7: CM41X_M4 PINT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
21	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level
22	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level
23	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level
24	PINT3_BLOCK	PINT3 Pin Interrupt Block	Level
25	PINT4_BLOCK	PINT4 Pin Interrupt Block	Level
26	PINT5_BLOCK	PINT5 Pin Interrupt Block	Level

Table 18-8: CM41X_M4 PINT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

18.2.7

CM41X_M0 PINT Trigger List

Table 18-9: CM41X_M0 PINT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
12	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level

Table 18-10: CM41X_M0 PINT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

PORT Architectural Concepts

These sections describe in more detail how the PORT module connects externally to pins and internally to the MMR bus. Ports are named alphabetically beginning with A.

- Internal Interfaces
- External Interfaces
- GPIO Pin Function
- Port Multiplexing Control

Internal Interfaces

All of the pin multiplexing, GPIO, and pin interrupt control block MMRs can be accessed through the MMR bus. . Each of the pin interrupt (PINTx) modules has its own dedicated interrupt request output signal.

External Interfaces

The pin multiplexing hardware can be seen as a layer between the on-chip peripherals and the silicon pads connecting to the physical pins/balls or the package, as controlled by the PORT unit.

GPIO Pin Function

By default, the PORT sets every GPIO pin to input mode. The input drivers are not enabled, which avoids the need for unnecessary current sinks and external termination resistors on unused pins.

Input Mode

The default mode of every GPIO pin after reset is input mode, but the input drivers are not enabled. To enable GPIO input drivers, set the bits corresponding to the PORT pins in the appropriate input enable register (PORT_INEN). When enabled, a read from the PORT_DATA register returns the logical state of the input pins. However, the input signal does not overwrite the state of the internal flip-flop used for providing output to the same pin. Only software can alter the state. If the input driver is enabled, a write to the PORT_DATA register can alter the state of the flip-flop, but the change cannot be read back.

Output Mode

Any GPIO pin can be configured for output mode. The GPIO output drivers are enabled by setting the bits corresponding to the PORT pins in the appropriate direction register. The PORT implements direction registers as a pair of write-1-to-set (W1S) and write-1-to-clear (W1C) MMRs called PORT_DIR_SET and PORT_DIR_CLR, respectively. As such, software can alter the direction of the signal flow on individual GPIO pins without impacting other GPIOs on the same port.

Both the PORT_DIR_SET and PORT_DIR_CLR registers return the same value when read, and a logical 1 indicates an enabled output. The PORT_DATA registers control the state of output pins. A logical 0 drives the output low while a logical 1 drives the output high.

While writes to the PORT_DATA register can alter all of the GPIOs on a specific port at once, there are also a pair of W1S and W1C MMRs called PORT_DATA_SET and PORT_DATA_CLR, respectively. These registers enable the manipulation of individual GPIO outputs. The state of the outputs can be obtained by reading the PORT_DATA registers. Because the state of the GPIO output can be controlled before the output driver is enabled, set or clear the internal flip-flop first by programming this register to avoid volatile levels on the output pin.

Trigger Toggle Mode

Any GPIO pin that has been configured for output mode can be toggled using a trigger input from the Trigger Routing Unit (TRU). To enable this functionality for a particular GPIO, set the appropriate bit in the PORT_TRIG_TGL register. Any subsequent trigger for the designated port causes all GPIO outputs set in the PORT_TRIG_TGL register to toggle.

To avoid unpredictable behavior, do not set, clear, or toggle the PORT_DATA, PORT_DATA_SET, PORT_DATA_CLR, or PORT_DATA_TGL registers when the GPIO output has the corresponding PORT_TRIG_TGL bit set. To change the state of the GPIO output using one of these registers, first clear the corresponding bit in the PORT_TRIG_TGL register.

Open-Drain Mode

Every GPIO can also be used in open-drain mode. First, either clear the respective bit in the PORT_DATA register or set the respective bit in the PORT_DATA_CLR register. Then, set the appropriate bit in the PORT_INEN register. Read from the PORT_DATA register to return the status from the pin rather than the state of the internal flip-flop.

By toggling the output driver through the PORT_DIR_SET and PORT_DIR_CLR register pair, the output signal can be pulled low or three-stated, as required. The polarity of the driven signal can be inverted when the internal flip-flop is set. When using a GPIO port in open-drain mode, take care to not exceed the V_{IH} operating condition associated with the respective pins.

Port Multiplexing Control

To configure pins properly, consult the processor datasheet to determine which bits in the PORT_FER and PORT_MUX register map to the pin of interest, and then set these registers appropriately for the desired function.

After reset, all port pins default to GPIO input mode with their output and input drivers disabled. As a result, all unused port pins can be left unconnected. Disabled pins appear as high-impedance to external circuits.

Each port has two dedicated MMRs that control the port multiplexing, the 16-bit function enable (PORT_FER) registers and the 32-bit port multiplexing (PORT_MUX) registers.

The function enable register specifies whether the pin is used as a GPIO pin or allocated for use by a specific peripheral, but it does not specify what the peripheral function is. Each bit in the 16-bit PORT_FER register corresponds to an individual port pin. For example, if bit 1 (PA1) of the PORTA_FER register is cleared, the PA_01 pin is configured as a GPIO. When set, one of the available peripheral functions becomes active on the PA_01 pin instead.

Pairs of bits in the PORT_MUX register control the multiplexing between the peripheral functions available to an individual pin, as some PORT pins provide up to four possible peripheral functions.

Refer to the Signal Muxing table in the datasheet for the specific PORT_MUX settings.

PORT Event Control

The following sections describe event generation in the PORT module.

PORT Interrupt Signals

The pin interrupts are decoupled from GPIO functionality, providing the following advantages.

- Flexible mapping scheme enables pins from up to four different ports to be grouped into one common interrupt scheme.
- Interrupt requests work on input and output pins regardless of whether the pin is functioning as a GPIO or a peripheral.

The processor has a number of interrupt channels dedicated to pin interrupts, managed by a set of pin interrupt (PINTx) blocks. Each PINTx block can sense up to 32 GPIO pins, as described in the following list and figure.

- PINT0 can sense pin activity on PORTA
- PINT1 can sense pin activity on PORTB and PORTC
- PINT2 can sense pin activity on PORTC and PORTD
- PINT3 can sense pin activity on PORTD and PORTE
- PINT4 can sense pin activity on PORTE and PORTF
- PINT1 can sense pin activity on PORTF and PORTB

The processor supports both 32-bit and 16-bit peripheral bus accesses to PINTx registers.

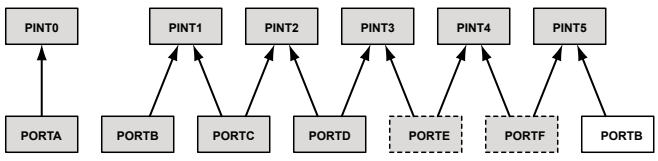


Figure 18-2: ADSP-CM41x GPIO to PINTx Assignment

Pins connect to the PINTx module and then to the system event controller (SEC), as shown in the ***PINTx Block Diagram***.

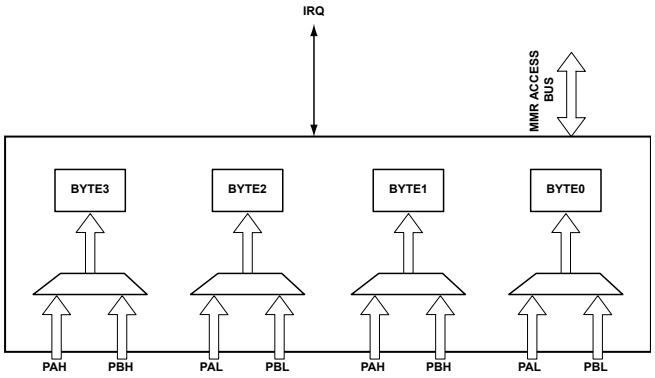


Figure 18-3: PINTx Block Diagram

As shown in the ***PINTx Block Diagram***, each port is subdivided into two 8-pin half ports, upper (PxH) and lower (PxL). The PINT_ASSIGN registers control the 8-bit multiplexers associated with these half ports, where the lower half units (eight pins) can be forwarded to either byte 0 or byte 2 of the PINTx blocks, and the upper half units (eight pins) can be forwarded to either byte 1 or byte 3 of the PINTx blocks.

When a half port is assigned to a byte in any PINTx block, the state of the eight pins appears in the PINT_PINSTATE register, regardless of whether the pin is enabled for GPIO or peripheral functions (input or output). When neither the input nor output drivers of the pin are enabled, the pin state is read as zero. The PINT_PINSTATE register reports the inverted state of the pin when the PINT_INV_SET register activates the signal inverter. The inverter can be enabled on an individual bit-by-bit basis. Each bit in the PINT_INV_SET/ PINT_INV_CLR register pair represents a pin signal.

By default, PORT interrupt request generation is level-sensitive, and an interrupt request occurs when the enabled pin is sensed as active high. Use the PINT_EDGE_SET register to change the interrupt request genearion scheme to instead be edge-sensitive (rising edge generates the interrupt request). Use the PINT_INV_SET register to invert the polarity such that the PINTx block generates the interrupt request on active-low signals or falling edges.

The PINTx modules also assist when both signal edges must generate unique interrupt requests. If two different interrupt requests are required, the PINT_ASSIGN registers can route a single input signal to two different PINTx blocks, where one block inverts the signal in the PINT_INV_SET register and the other one does not. In this fashion, a unique software routine is associated with the hardware PINTx block that is generating the unique interrupt request for each signal edge. When both signal edges can be serviced by the same interrupt request, each half port can be routed to two separate bytes within a single PINTx block using the PINT_ASSIGN register, and then one of the half ports needs to have the inversion enabled in the PINT_INV_SET register. The servicing software routine can then detect from the PINT_LATCH register whether a falling, rising, or both edges have occurred.

Regardless of whether level-sensitive or edge-sensitive mode is used, the hardware always latches an interrupt request. Latched signals can be read from the PINT_LATCH registers. Only a software or hardware reset clears the latches. To clear the latch, a W1C operation must be performed to the PINT_REQ or PINT_LATCH register. When in level-sensitive mode, the interrupt request remains asserted if the pin state does not change by the time the interrupt service routine exits.

Because every PINTx block groups up to 32 pin signals, the PINT_MSK_SET/ PINT_MSK_CLR register pair can control which of the signals can request an interrupt at the system level. Software can interrogate the PINT_REQ register for signaling pins. The PINT_REQ bits represent a logical AND between the mask and the latch. When any of these bits is set, the block output interrupt request is asserted.

GPIO Pin Safe State

The GPIOs in the ADSP-CM41x can be configured to a high, low, hold or threestate state in the event of a power failure or a bad clock signal. More information on the GPIO Pin Safe State can be found in GPIO Safe Pin States in the VMU chapter.

PORT Programming Model

The ***GPIO Programming Model Flow (Part 1)***, ***GPIO Programming Model Flow (Part 2)***, and ***GPIO Programming Model Flow (Part 3)*** figures show the programming model for the general-purpose ports. This programming includes the GPIO input and output operation, open-drain mode, and the pin interrupt PINTx modules.

NOTE: These process flow diagrams connect where call-out letters appear. For example, call-out A in the ***GPIO Programming Model Flow (Part 1)*** diagram connects to call-out A in the ***GPIO Programming Model Flow (Part 2)*** diagram.

The following flowcharts describe the processes for setting up pins for various functions. Begin the process from the start label in the ***GPIO Programming Model Flow (Part 1)*** figure. The first decision (GPIO or peripheral) determines how to program the PORT_FER register. Set the bit(s) corresponding to the pin(s) to 1 to enable peripheral functionality or to 0 to enable GPIO mode. For more information on setting up for peripheral functions, refer to Port Multiplexing Control.

If the pin is to be a GPIO pin, a subsequent series of decisions must be made that will impact how the PORT_DATA, PORT_POL, PORT_DIR, and PORT_INEN configuration registers must be programmed. Depending on the type of GPIO pin desired, some configurations do not apply and can have different meanings. The following paragraphs briefly describe the function of the different settings for each of the pin functions in the input, output, and open-drain GPIO modes. It is a best practice to use the SET or CLR versions of the PORT registers, where applicable, to effect changes on a pin-by-pin basis rather than on the full port.

For output mode, first clear the PORT_DATA register to set all the pins low. Then write the PORT_DIR register to define the direction of each pin (set the bits associated with the desired output pins to 1). In output mode, the other registers are not significant. The ***GPIO Programming Model Flow (Part 1)*** chart shows this flow starting at label 2.

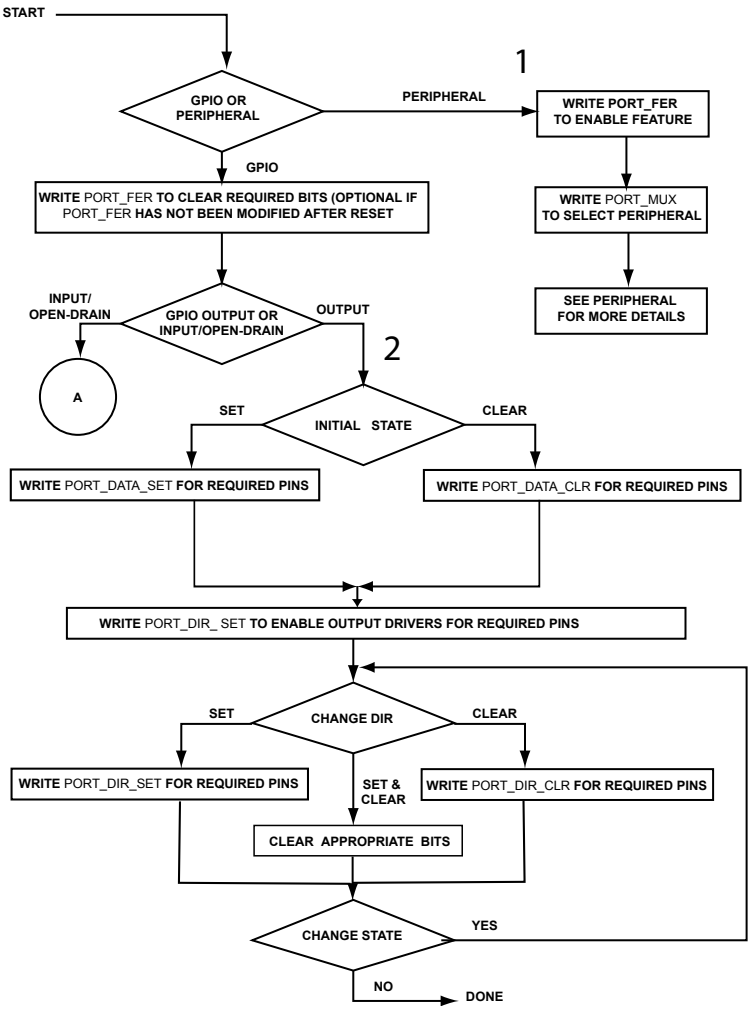


Figure 18-4: GPIO Programming Model Flow (Part 1)

For input mode, first decide the polarity for each pin using the PORT_POL register. Program the PORT_DIR register to define the appropriate pins as inputs (write a 0 to the bit location associated with the pin). If interrupt requests are desired, configure the PINT module as shown in the ***GPIO Programming Model Flow (Part 3)*** figure starting at label B. Finally, write the PORT_INEN register to enable the associated input drivers. The ***GPIO Programming Model Flow (Part 2)*** chart shows this entire flow starting at label 3.

For open-drain mode, set all pins low by clearing the PORT_DATA register. Then, use the PORT_INEN register to enable the appropriate input drivers. Set the PORT_DIR register in this mode to indicate whether the pin is in an active state or not (active being 0). The ***GPIO Programming Model Flow (Part 2)*** chart shows this flow starting at label 4.

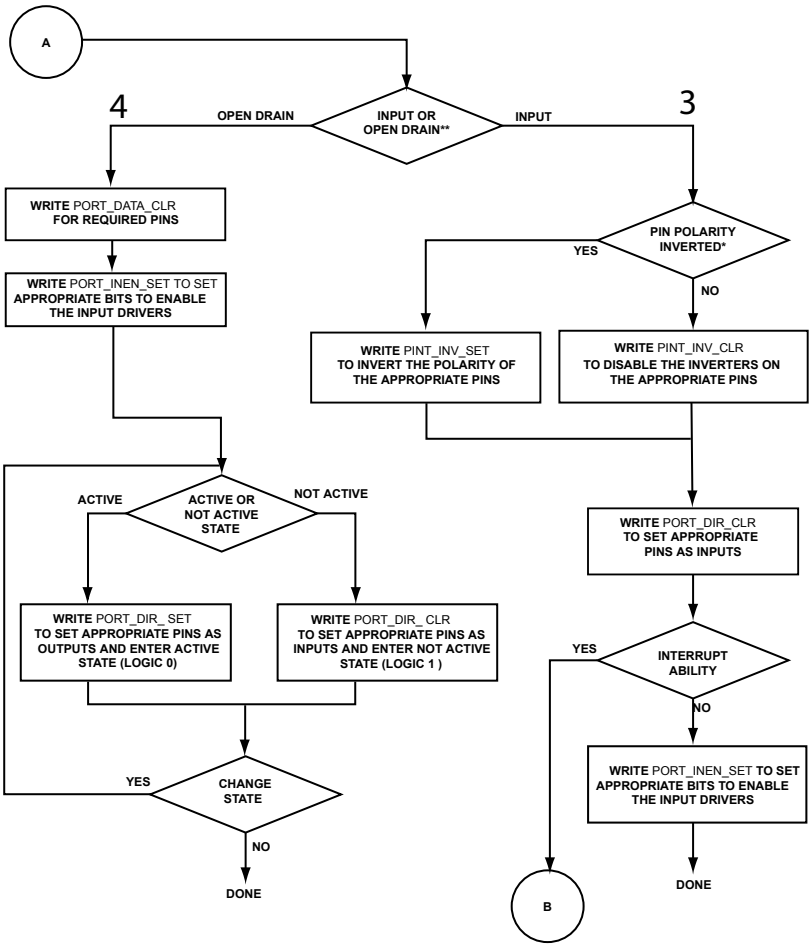


Figure 18-5: GPIO Programming Model Flow (Part 2)

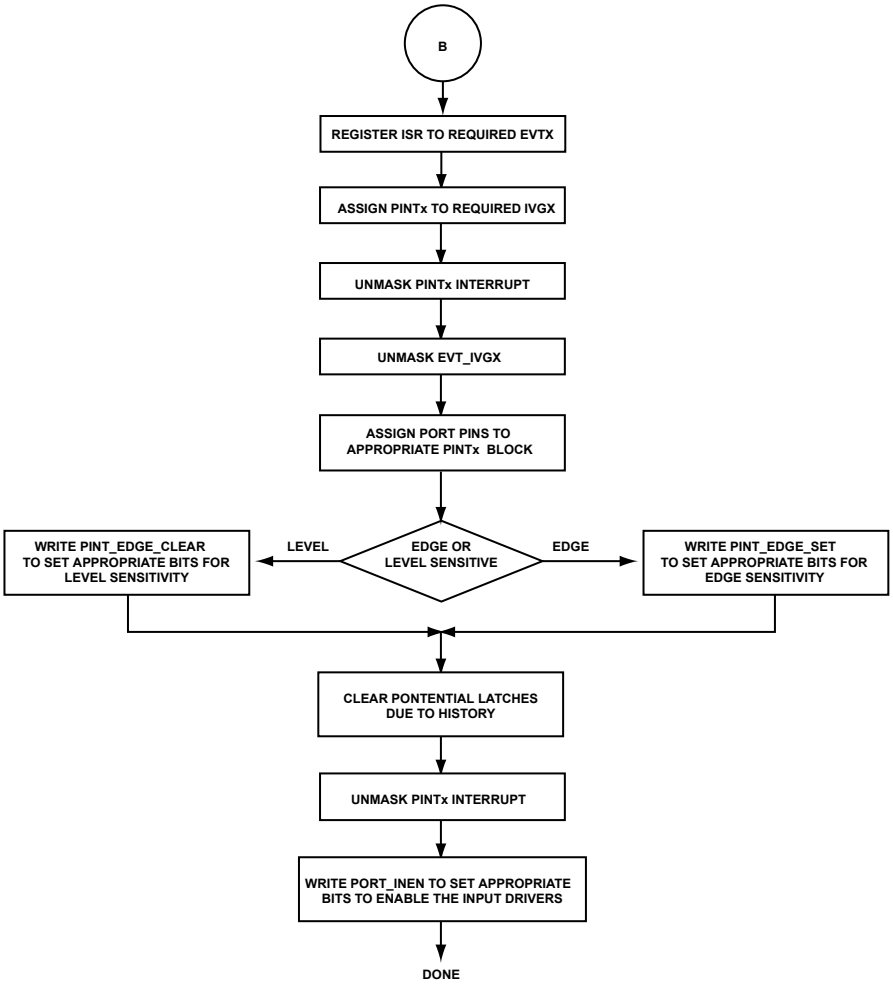


Figure 18-6: GPIO Programming Model Flow (Part 3)

CM41X_M4 PORT Register Descriptions

The General-Purpose Input/Output Port (PORT) contains the following registers.

Table 18-11: CM41X_M4 PORT Register List

Name	Description
PORT_DATA	Port x GPIO Data Register
PORT_DATA_CLR	Port x GPIO Data Clear Register
PORT_DATA_SET	Port x GPIO Data Set Register
PORT_DATA_TGL	Port x GPIO Output Toggle Register
PORT_DIR	Port x GPIO Direction Register
PORT_DIR_CLR	Port x GPIO Direction Clear Register

Table 18-11: CM41X_M4 PORT Register List (Continued)

Name	Description
PORT_DIR_SET	Port x GPIO Direction Set Register
PORT_FER	Port x Function Enable Register
PORT_FER_CLR	Port x Function Enable Clear Register
PORT_FER_SET	Port x Function Enable Set Register
PORT_INEN	Port x GPIO Input Enable Register
PORT_INEN_CLR	Port x GPIO Input Enable Clear Register
PORT_INEN_SET	Port x GPIO Input Enable Set Register
PORT_LOCK	Port x GPIO Lock Register
PORT_MUX	Port x Multiplexer Control Register
PORT_POL	Port x GPIO Polarity Invert Register
PORT_POL_CLR	Port x GPIO Polarity Invert Clear Register
PORT_POL_SET	Port x GPIO Polarity Invert Set Register
PORT_TRIG_TGL	Port x GPIO Trigger Toggle Register

Port x GPIO Data Register

The operation of the PORT_DATA register depends on whether the bit/pin is in output mode or input mode. In both modes, a set bit in the PORT_DATA register corresponds to a signal high on a GPIO pin. A cleared bit in the PORT_DATA register corresponds to a signal low on a GPIO pin.

The PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers control the state of GPIO pins in output mode. To enable output mode (and output drivers), use the PORT_DIR_SET and PORT_DIR_CLR registers.

Writes to the PORT_DATA register affect the state of all pins of the port that are in output mode. To set or clear specific pins without impacting other pins of the port, use the PORT_DATA_SET and PORT_DATA_CLR registers.

When the GPIO pins are in input mode (input driver is enabled with the PORT_INEN register), reads from the PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers return the state of the respective GPIO pins.

Note that when the input driver is not enabled, reads from the PORT_DATA, PORT_DATA_SET, and PORT_DATA_CLR registers return the value previously written to the registers.

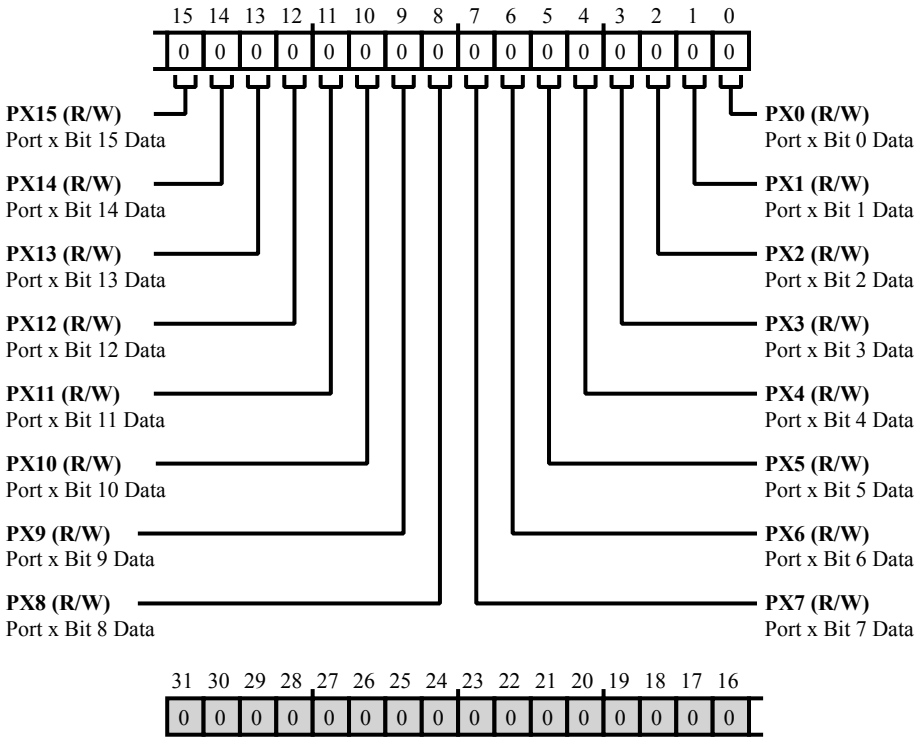


Figure 18-7: PORT_DATA Register Diagram

Table 18-12: PORT_DATA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Data. The PORT_DATA.PX15 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
14 (R/W)	PX14	Port x Bit 14 Data. The PORT_DATA.PX14 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
13 (R/W)	PX13	Port x Bit 13 Data. The PORT_DATA.PX13 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
12 (R/W)	PX12	Port x Bit 12 Data. The PORT_DATA.PX12 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
11 (R/W)	PX11	Port x Bit 11 Data. The PORT_DATA.PX11 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
10 (R/W)	PX10	Port x Bit 10 Data. The PORT_DATA.PX10 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
9 (R/W)	PX9	Port x Bit 9 Data. The PORT_DATA.PX9 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High

Table 18-12: PORT_DATA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	PX8	Port x Bit 8 Data. The PORT_DATA.PX8 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
7 (R/W)	PX7	Port x Bit 7 Data. The PORT_DATA.PX7 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
6 (R/W)	PX6	Port x Bit 6 Data. The PORT_DATA.PX6 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
5 (R/W)	PX5	Port x Bit 5 Data. The PORT_DATA.PX5 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
4 (R/W)	PX4	Port x Bit 4 Data. The PORT_DATA.PX4 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
3 (R/W)	PX3	Port x Bit 3 Data. The PORT_DATA.PX3 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
2 (R/W)	PX2	Port x Bit 2 Data. The PORT_DATA.PX2 bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High

Table 18-12: PORT_DATA Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	PX1	Port x Bit 1 Data. The `PORT_DATA.PX1` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High
0 (R/W)	PX0	Port x Bit 0 Data. The `PORT_DATA.PX0` bit indicates a signal on a GPIO pin.	
		0	Signal Low
		1	Signal High

Port x GPIO Data Clear Register

The PORT_DATA_CLR register operates differently for port bits/pins, depending on whether the bit/pin is output mode or input mode. For more information, see the PORT_DATA register description.

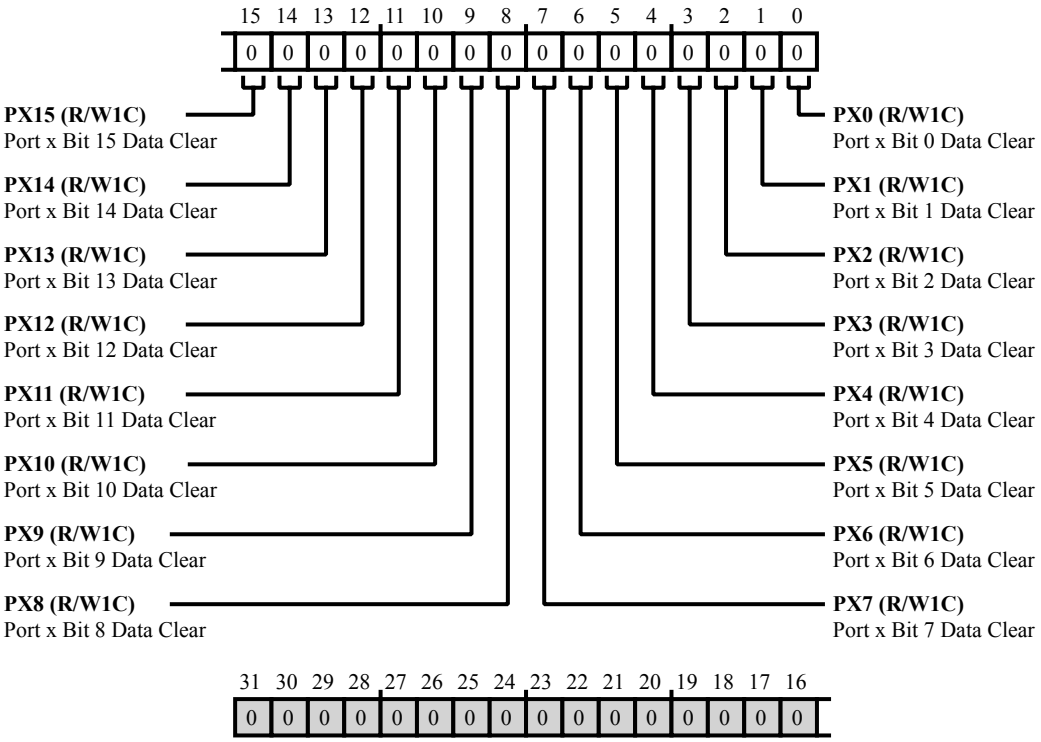


Figure 18-8: PORT_DATA_CLR Register Diagram

Table 18-13: PORT_DATA_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Data Clear. The PORT_DATA_CLR.PX15 bit clears the pin without impacting other pins of the port.	
		0	No Effect
		1	Clear Bit. Write 1 for signal low in output mode.
14 (R/W1C)	PX14	Port x Bit 14 Data Clear. The PORT_DATA_CLR.PX14 bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.

Table 18-13: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Data Clear. The `PORT_DATA_CLR.PX13` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
12 (R/W1C)	PX12	Port x Bit 12 Data Clear. The `PORT_DATA_CLR.PX12` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
11 (R/W1C)	PX11	Port x Bit 11 Data Clear. The `PORT_DATA_CLR.PX11` bit clears the pin without impacting other pins of the port.	
		0	No Effect
		1	Clear Bit. Write 1 for signal low in output mode.
10 (R/W1C)	PX10	Port x Bit 10 Data Clear. The `PORT_DATA_CLR.PX10` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
9 (R/W1C)	PX9	Port x Bit 9 Data Clear. The `PORT_DATA_CLR.PX9` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
8 (R/W1C)	PX8	Port x Bit 8 Data Clear. The `PORT_DATA_CLR.PX8` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.

Table 18-13: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	PX7	Port x Bit 7 Data Clear. The PORT_DATA_CLR.PX7 bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
6 (R/W1C)	PX6	Port x Bit 6 Data Clear. The PORT_DATA_CLR.PX6 bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
5 (R/W1C)	PX5	Port x Bit 5 Data Clear. The PORT_DATA_CLR.PX5 bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
4 (R/W1C)	PX4	Port x Bit 4 Data Clear. The PORT_DATA_CLR.PX4 bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
3 (R/W1C)	PX3	Port x Bit 3 Data Clear. The PORT_DATA_CLR.PX3 bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
2 (R/W1C)	PX2	Port x Bit 2 Data Clear. The PORT_DATA_CLR.PX2 bit clears the pin without impacting other pins of the port.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.

Table 18-13: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	PX1	Port x Bit 1 Data Clear. The `PORT_DATA_CLR.PX1` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
0 (R/W1C)	PX0	Port x Bit 0 Data Clear. The `PORT_DATA_CLR.PX0` bit clears the pin without impacting other pins of the port.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.

Port x GPIO Data Set Register

The PORT_DATA_SET register operates differently for port bits/pins, depending on whether the bit/pin is output mode or input mode. For more information, see the PORT_DATA register description.

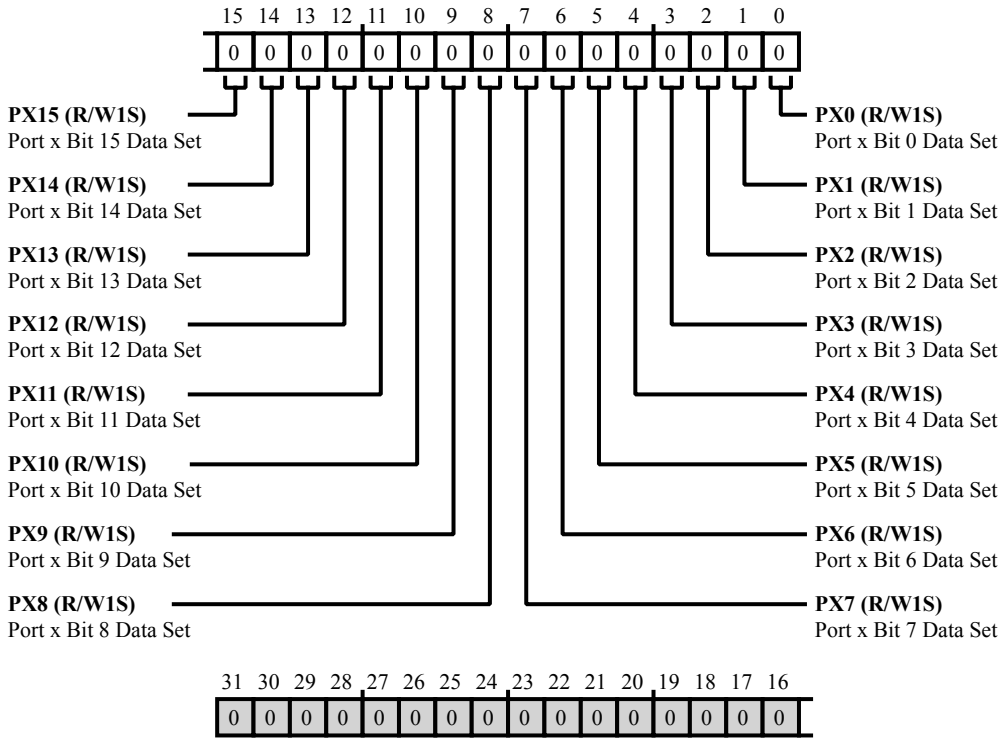


Figure 18-9: PORT_DATA_SET Register Diagram

Table 18-14: PORT_DATA_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
14 (R/W1S)	PX14	Port x Bit 14 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
13 (R/W1S)	PX13	Port x Bit 13 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.

Table 18-14: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
11 (R/W1S)	PX11	Port x Bit 11 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit.
10 (R/W1S)	PX10	Port x Bit 10 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
9 (R/W1S)	PX9	Port x Bit 9 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
8 (R/W1S)	PX8	Port x Bit 8 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
7 (R/W1S)	PX7	Port x Bit 7 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
6 (R/W1S)	PX6	Port x Bit 6 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
5 (R/W1S)	PX5	Port x Bit 5 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
4 (R/W1S)	PX4	Port x Bit 4 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
3 (R/W1S)	PX3	Port x Bit 3 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.

Table 18-14: PORT_DATA_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
1 (R/W1S)	PX1	Port x Bit 1 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.
0 (R/W1S)	PX0	Port x Bit 0 Data Set.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Set Bit. Write 1 for signal high in output mode.

Port x GPIO Output Toggle Register

The PORT_DATA_TGL register permits toggling the state of output GPIO pins. Setting bits in the PORT_DATA_TGL register affects the state of specific pins without impacting other pins of the port.

Reading the PORT_DATA_TGL returns the state of the PORT_DATA register output pin state, but does not return the input pin/signal state.

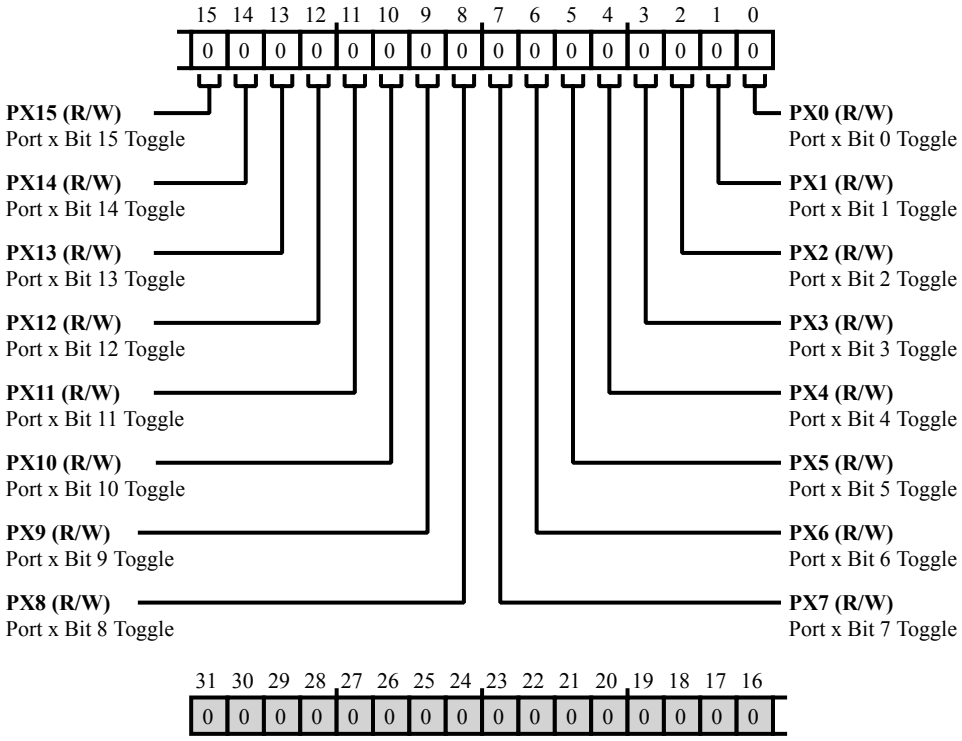


Figure 18-10: PORT_DATA_TGL Register Diagram

Table 18-15: PORT_DATA_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Toggle. The PORT_DATA_TGL.PX15 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit.
14 (R/W)	PX14	Port x Bit 14 Toggle. The PORT_DATA_TGL.PX14 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit

Table 18-15: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	PX13	Port x Bit 13 Toggle. The PORT_DATA_TGL.PX13 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
12 (R/W)	PX12	Port x Bit 12 Toggle. The PORT_DATA_TGL.PX12 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
11 (R/W)	PX11	Port x Bit 11 Toggle. The PORT_DATA_TGL.PX11 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
10 (R/W)	PX10	Port x Bit 10 Toggle. The PORT_DATA_TGL.PX10 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
9 (R/W)	PX9	Port x Bit 9 Toggle. The PORT_DATA_TGL.PX9 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
8 (R/W)	PX8	Port x Bit 8 Toggle. The PORT_DATA_TGL.PX8 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
7 (R/W)	PX7	Port x Bit 7 Toggle. The PORT_DATA_TGL.PX7 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit

Table 18-15: PORT_DATA_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	PX6	Port x Bit 6 Toggle. The PORT_DATA_TGL.PX6 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
5 (R/W)	PX5	Port x Bit 5 Toggle. The PORT_DATA_TGL.PX5 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
4 (R/W)	PX4	Port x Bit 4 Toggle. The PORT_DATA_TGL.PX4 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
3 (R/W)	PX3	Port x Bit 3 Toggle. The PORT_DATA_TGL.PX3 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
2 (R/W)	PX2	Port x Bit 2 Toggle. The PORT_DATA_TGL.PX2 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
1 (R/W)	PX1	Port x Bit 1 Toggle. The PORT_DATA_TGL.PX1 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit
0 (R/W)	PX0	Port x Bit 0 Toggle. The PORT_DATA_TGL.PX0 bit toggles the output GPIO bit/pin state.	
		0	No Effect
		1	Toggle Bit

Port x GPIO Direction Register

The PORT_DIR, PORT_DIR_SET, and PORT_DIR_CLR registers select output or input mode for GPIO pins and enable output drivers. Use the PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers to enable or disable input drivers.

Writes to the PORT_DIR register affect the state of all pins of the port. To select a direction for specific pins without impacting other pins of the port, use the PORT_DIR_SET and PORT_DIR_CLR registers.

Setting a bit in the PORT_DIR register enables output mode on the corresponding a GPIO pin. Clearing a bit in the PORT_DIR register disables output mode on the corresponding GPIO pin.

Input Mode - The default mode of every GPIO pin after reset is the input mode, but the input drivers are not enabled. To enable GPIO input drivers, set the corresponding bits in the PORT_INEN register. When enabled, a read from the PORT_DATA register returns the logical state of the input pin. The input signal does not overwrite the state of the bit used for the output case. That state can only be altered by software. If the input driver is enabled, a write to the PORT_DATA register can alter the state of the bit, but the change cannot be read back.

Output Mode - Any GPIO pin can be configured for output mode. The GPIO output drivers are enabled by setting the corresponding bits in the PORT_DIR, PORT_DIR_SET, or PORT_DIR_CLR registers. By using the PORT_DIR_SET and PORT_DIR_CLR registers, the direction of the signal flow of individual GPIO pins can be altered by separate software threads without mutually impacting other GPIOs on the same port. Both registers return the same value when read. Because the state of the GPIO output can already be controlled before the output driver is enabled, it is recommended to first set or clear the bit (using the PORT_DATA, PORT_DATA_SET, or PORT_DATA_CLR registers) to avoid any volatile levels on the output.

Open-Drain Mode - Every GPIO can also be used in open-drain mode. To accomplish this, first, clear the respective bit in the PORT_DATA or PORT_DATA_CLR register. Then, set the one bit in the PORT_INEN register. Reads from the PORT_DATA register then return the status from the pin and do not return the state of the internal flip-flop. By toggling the output driver through the PORT_DIR_SET and PORT_DIR_CLR register pair, the output signal can be pulled low or three-stated as required. Note that the polarity of the driven signal can be inverted when the internal flip-flop is set instead. When a GPIO port is used in open-drain mode, take care to not exceed the V_{IH} operating condition associated with the respective pin.

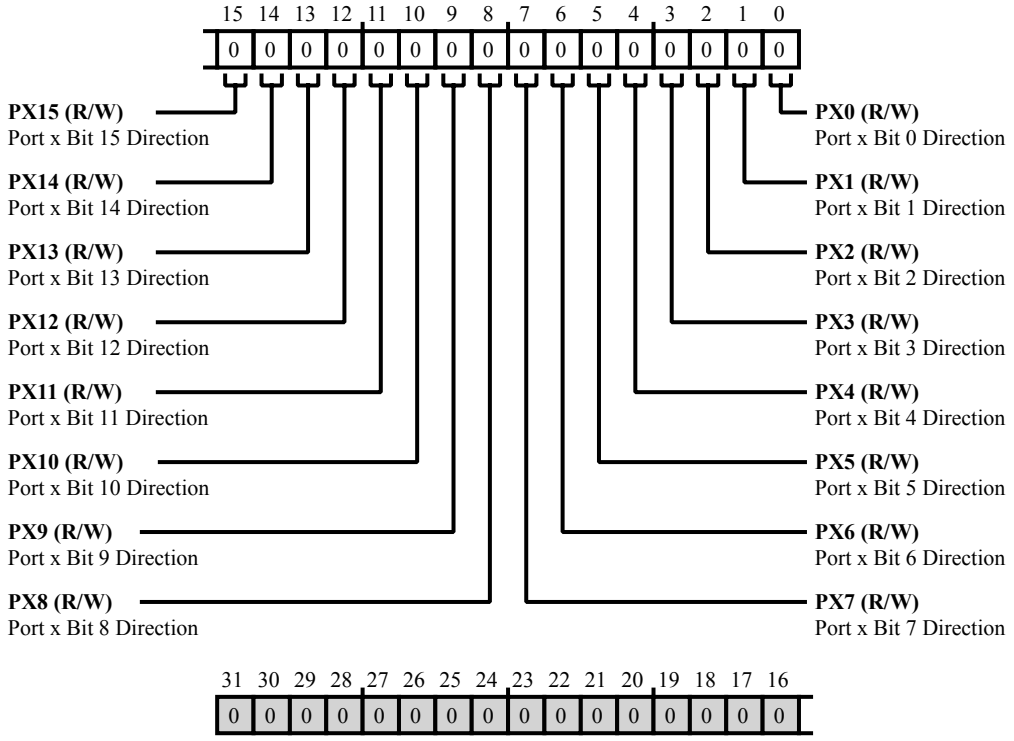


Figure 18-11: PORT_DIR Register Diagram

Table 18-16: PORT_DIR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
14 (R/W)	PX14	Port x Bit 14 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
13 (R/W)	PX13	Port x Bit 13 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
12 (R/W)	PX12	Port x Bit 12 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.

Table 18-16: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	PX11	Port x Bit 11 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
10 (R/W)	PX10	Port x Bit 10 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
9 (R/W)	PX9	Port x Bit 9 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
8 (R/W)	PX8	Port x Bit 8 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
7 (R/W)	PX7	Port x Bit 7 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
6 (R/W)	PX6	Port x Bit 6 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
5 (R/W)	PX5	Port x Bit 5 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
4 (R/W)	PX4	Port x Bit 4 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
3 (R/W)	PX3	Port x Bit 3 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
2 (R/W)	PX2	Port x Bit 2 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.

Table 18-16: PORT_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	PX1	Port x Bit 1 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.
0 (R/W)	PX0	Port x Bit 0 Direction.	
		0	Input mode. The output driver is disabled.
		1	Output mode. The output driver is enabled.

Port x GPIO Direction Clear Register

The PORT_DIR_CLR register disables output mode and disables the output drivers for GPIO pins. For more information, see the PORT_DIR register description.

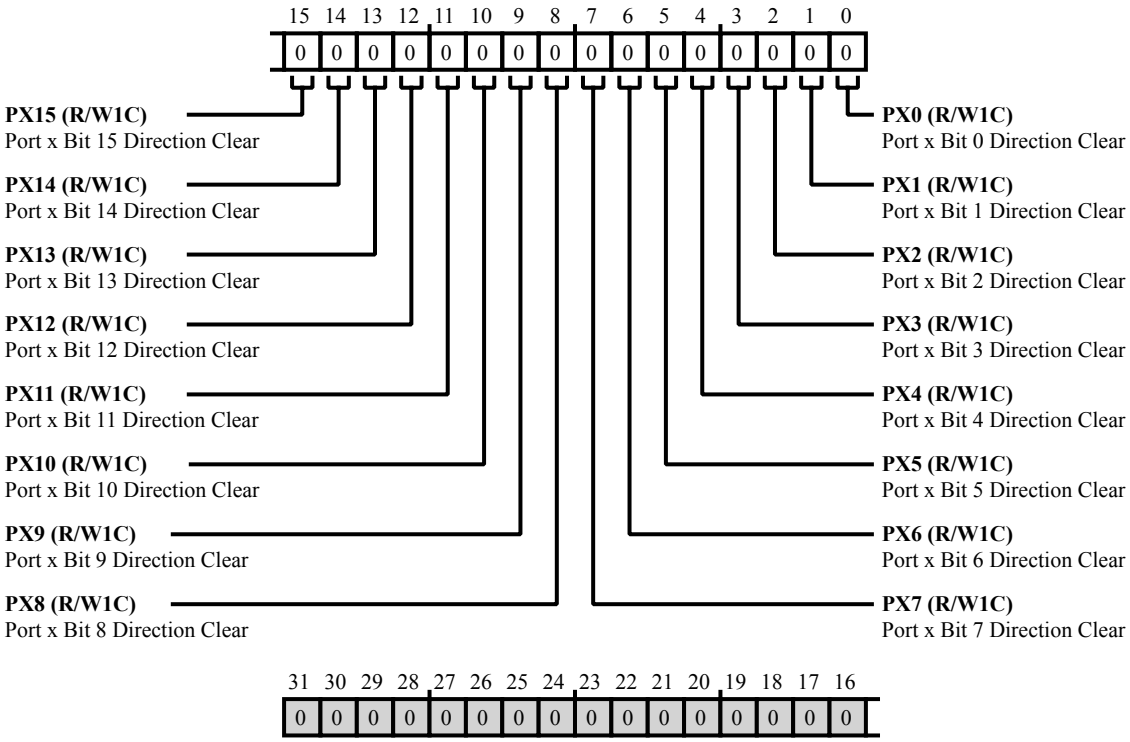


Figure 18-12: PORT_DIR_CLR Register Diagram

Table 18-17: PORT_DIR_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Direction Clear. The PORT_DIR_CLR.PX15 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
14 (R/W1C)	PX14	Port x Bit 14 Direction Clear. The PORT_DIR_CLR.PX14 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver

Table 18-17: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Direction Clear. The PORT_DIR_CLR.PX13 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
12 (R/W1C)	PX12	Port x Bit 12 Direction Clear. The PORT_DIR_CLR.PX12 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
11 (R/W1C)	PX11	Port x Bit 11 Direction Clear. The PORT_DIR_CLR.PX11 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
10 (R/W1C)	PX10	Port x Bit 10 Direction Clear. The PORT_DIR_CLR.PX10 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
9 (R/W1C)	PX9	Port x Bit 9 Direction Clear. The PORT_DIR_CLR.PX9 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
8 (R/W1C)	PX8	Port x Bit 8 Direction Clear. The PORT_DIR_CLR.PX8 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver

Table 18-17: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	PX7	Port x Bit 7 Direction Clear. The PORT_DIR_CLR.PX7 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
6 (R/W1C)	PX6	Port x Bit 6 Direction Clear. The PORT_DIR_CLR.PX6 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
5 (R/W1C)	PX5	Port x Bit 5 Direction Clear. The PORT_DIR_CLR.PX5 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
4 (R/W1C)	PX4	Port x Bit 4 Direction Clear. The PORT_DIR_CLR.PX4 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
3 (R/W1C)	PX3	Port x Bit 3 Direction Clear. The PORT_DIR_CLR.PX3 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
2 (R/W1C)	PX2	Port x Bit 2 Direction Clear. The PORT_DIR_CLR.PX2 bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver

Table 18-17: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	PX1	Port x Bit 1 Direction Clear. The `PORT_DIR_CLR.PX1` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver
0 (R/W1C)	PX0	Port x Bit 0 Direction Clear. The `PORT_DIR_CLR.PX0` bit disables output mode and the output drivers for port x.	
		0	No Effect
		1	Disable output mode/driver

Port x GPIO Direction Set Register

The PORT_DIR_SET register enables output mode and output drivers for GPIO pins. For more information, see the PORT_DIR register description.

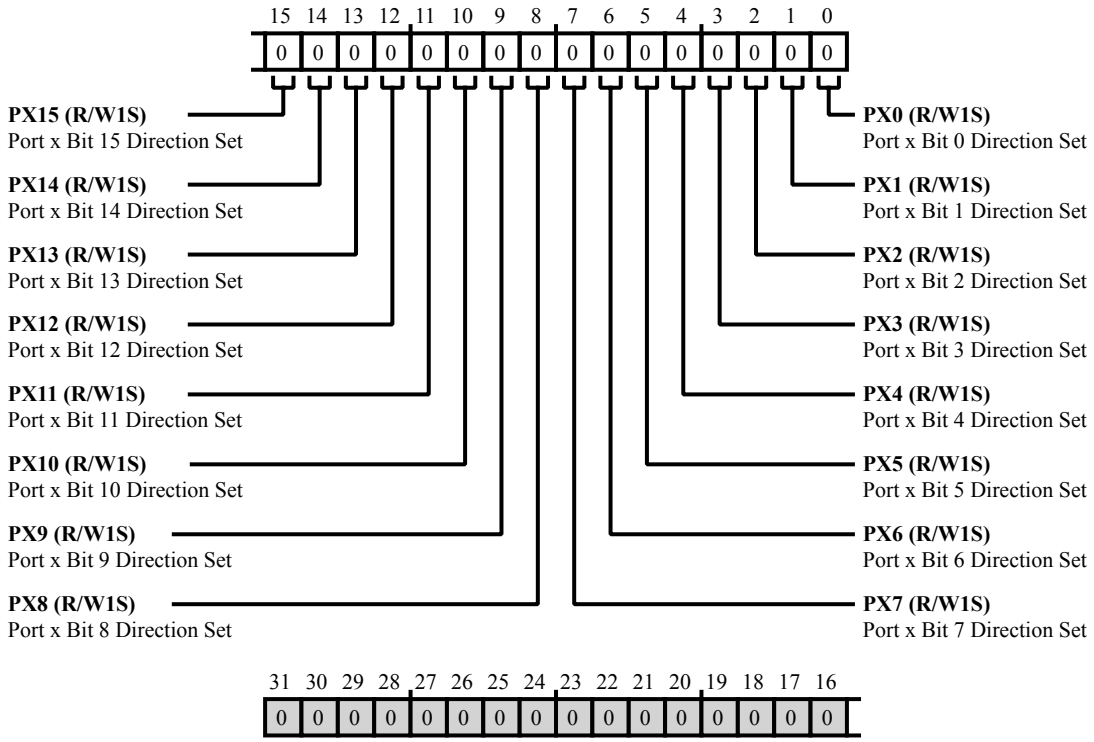


Figure 18-13: PORT_DIR_SET Register Diagram

Table 18-18: PORT_DIR_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Direction Set. The PORT_DIR_SET.PX15 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
14 (R/W1S)	PX14	Port x Bit 14 Direction Set. The PORT_DIR_SET.PX14 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Table 18-18: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Direction Set. The PORT_DIR_SET.PX13 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
12 (R/W1S)	PX12	Port x Bit 12 Direction Set. The PORT_DIR_SET.PX12 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
11 (R/W1S)	PX11	Port x Bit 11 Direction Set. The PORT_DIR_SET.PX11 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
10 (R/W1S)	PX10	Port x Bit 10 Direction Set. The PORT_DIR_SET.PX10 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
9 (R/W1S)	PX9	Port x Bit 9 Direction Set. The PORT_DIR_SET.PX9 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
8 (R/W1S)	PX8	Port x Bit 8 Direction Set. The PORT_DIR_SET.PX8 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
7 (R/W1S)	PX7	Port x Bit 7 Direction Set. The PORT_DIR_SET.PX7 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Table 18-18: PORT_DIR_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	PX6	Port x Bit 6 Direction Set. The PORT_DIR_SET.PX6 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
5 (R/W1S)	PX5	Port x Bit 5 Direction Set. The PORT_DIR_SET.PX5 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
4 (R/W1S)	PX4	Port x Bit 4 Direction Set. The PORT_DIR_SET.PX4 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
3 (R/W1S)	PX3	Port x Bit 3 Direction Set. The PORT_DIR_SET.PX3 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
2 (R/W1S)	PX2	Port x Bit 2 Direction Set. The PORT_DIR_SET.PX2 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
1 (R/W1S)	PX1	Port x Bit 1 Direction Set. The PORT_DIR_SET.PX1 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver
0 (R/W1S)	PX0	Port x Bit 0 Direction Set. The PORT_DIR_SET.PX0 bit enables the output mode/driver for port x.	
		0	No Effect
		1	Enable output mode/driver

Port x Function Enable Register

The PORT_FER register bits indicate each port bit's operating mode: general purpose I/O mode or peripheral mode. After reset, all pins default to GPIO mode. Setting a bit in the PORT_FER registers enables a peripheral module to take ownership of the pin. The function enable bits impact output control only. Regardless of the setting of the function enable bits, both GPIO and peripherals can still sense the pin input. After a function is enabled, it is up to the PORT_MUX registers as to which peripheral takes control.

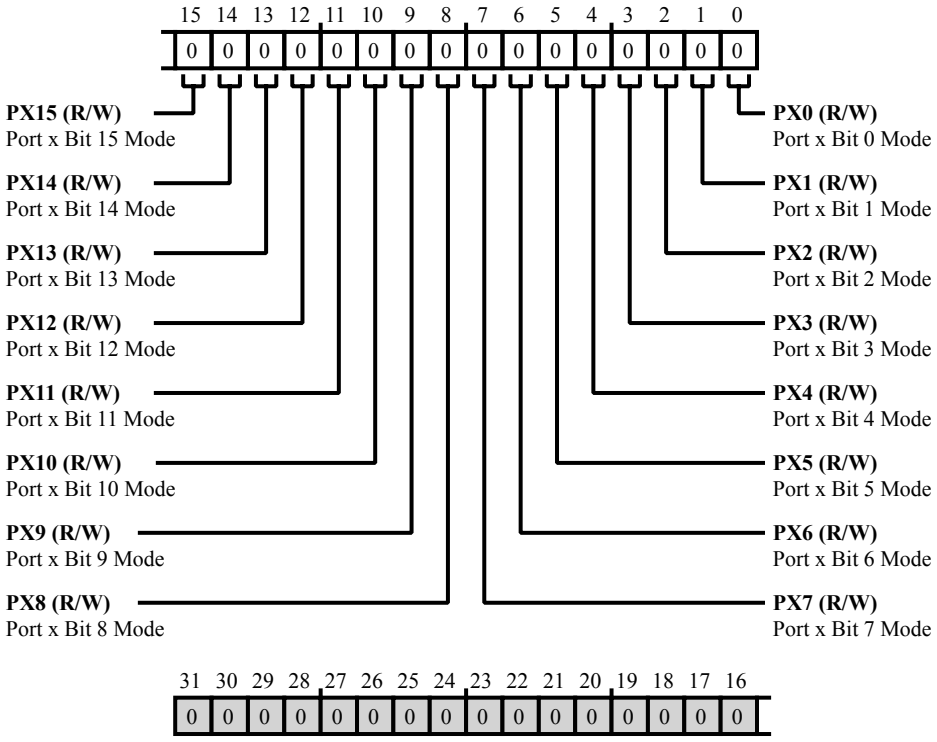


Figure 18-14: PORT_FER Register Diagram

Table 18-19: PORT_FER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Mode. The PORT_FER.PX15 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
14 (R/W)	PX14	Port x Bit 14 Mode. The PORT_FER.PX14 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode

Table 18-19: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	PX13	Port x Bit 13 Mode. The PORT_FER.PX13 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
12 (R/W)	PX12	Port x Bit 12 Mode. The PORT_FER.PX12 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
11 (R/W)	PX11	Port x Bit 11 Mode. The PORT_FER.PX11 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
10 (R/W)	PX10	Port x Bit 10 Mode. The PORT_FER.PX10 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
9 (R/W)	PX9	Port x Bit 9 Mode. The PORT_FER.PX9 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
8 (R/W)	PX8	Port x Bit 8 Mode. The PORT_FER.PX8 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
7 (R/W)	PX7	Port x Bit 7 Mode. The PORT_FER.PX7 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode

Table 18-19: PORT_FER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	PX6	Port x Bit 6 Mode. The PORT_FER.PX6 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
5 (R/W)	PX5	Port x Bit 5 Mode. The PORT_FER.PX5 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
4 (R/W)	PX4	Port x Bit 4 Mode. The PORT_FER.PX4 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
3 (R/W)	PX3	Port x Bit 3 Mode. The PORT_FER.PX3 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
2 (R/W)	PX2	Port x Bit 2 Mode. The PORT_FER.PX2 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
1 (R/W)	PX1	Port x Bit 1 Mode. The PORT_FER.PX1 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode
0 (R/W)	PX0	Port x Bit 0 Mode. The PORT_FER.PX0 bit indicates the operating mode for port x.	
		0	GPIO Mode
		1	Peripheral Mode

Port x Function Enable Clear Register

The PORT_FER_CLR register permits enabling GPIO mode for each bit and corresponding GPIO pin. Writing 1 to a bit in PORT_FER_CLR enables GPIO mode for the corresponding pin.

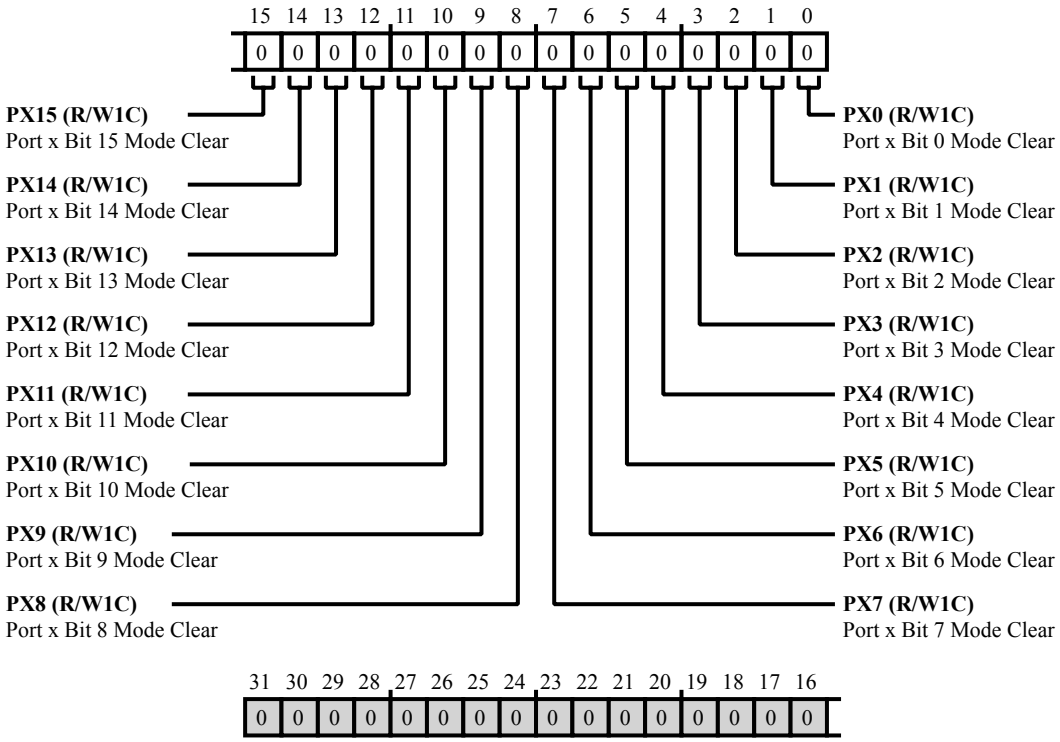


Figure 18-15: PORT_FER_CLR Register Diagram

Table 18-20: PORT_FER_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Mode Clear. The PORT_FER_CLR.PX15 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
14 (R/W1C)	PX14	Port x Bit 14 Mode Clear. The PORT_FER_CLR.PX14 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode

Table 18-20: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1C)	PX13	Port x Bit 13 Mode Clear. The PORT_FER_CLR.PX13 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
12 (R/W1C)	PX12	Port x Bit 12 Mode Clear. The PORT_FER_CLR.PX12 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
11 (R/W1C)	PX11	Port x Bit 11 Mode Clear. The PORT_FER_CLR.PX11 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
10 (R/W1C)	PX10	Port x Bit 10 Mode Clear. The PORT_FER_CLR.PX10 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
9 (R/W1C)	PX9	Port x Bit 9 Mode Clear. The PORT_FER_CLR.PX9 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
8 (R/W1C)	PX8	Port x Bit 8 Mode Clear. The PORT_FER_CLR.PX8 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
7 (R/W1C)	PX7	Port x Bit 7 Mode Clear. The PORT_FER_CLR.PX7 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode

Table 18-20: PORT_FER_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	PX6	Port x Bit 6 Mode Clear. The PORT_FER_CLR.PX6 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
5 (R/W1C)	PX5	Port x Bit 5 Mode Clear. The PORT_FER_CLR.PX5 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
4 (R/W1C)	PX4	Port x Bit 4 Mode Clear. The PORT_FER_CLR.PX4 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
3 (R/W1C)	PX3	Port x Bit 3 Mode Clear. The PORT_FER_CLR.PX3 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
2 (R/W1C)	PX2	Port x Bit 2 Mode Clear. The PORT_FER_CLR.PX2 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
1 (R/W1C)	PX1	Port x Bit 1 Mode Clear. The PORT_FER_CLR.PX1 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode
0 (R/W1C)	PX0	Port x Bit 0 Mode Clear. The PORT_FER_CLR.PX0 bit enables GPIO mode.	
		0	No Effect
		1	Set Bit for GPIO Mode

Port x Function Enable Set Register

The PORT_FER_SET register permits enabling peripheral mode for each bit and corresponding GPIO pin. Writing 1 to a bit in PORT_FER_SET enables peripheral mode for the corresponding pin.

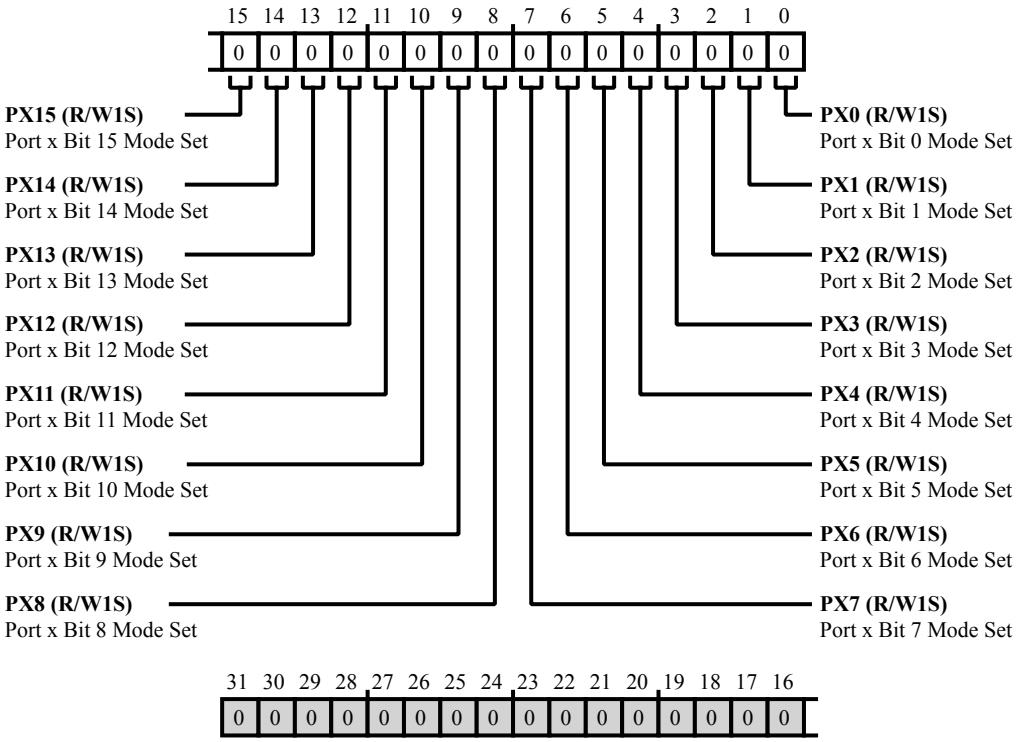


Figure 18-16: PORT_FER_SET Register Diagram

Table 18-21: PORT_FER_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Mode Set. The PORT_FER_SET.PX15 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
14 (R/W1S)	PX14	Port x Bit 14 Mode Set. The PORT_FER_SET.PX14 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Table 18-21: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Mode Set. The PORT_FER_SET.PX13 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
12 (R/W1S)	PX12	Port x Bit 12 Mode Set. The PORT_FER_SET.PX12 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
11 (R/W1S)	PX11	Port x Bit 11 Mode Set. The PORT_FER_SET.PX11 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
10 (R/W1S)	PX10	Port x Bit 10 Mode Set. The PORT_FER_SET.PX10 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
9 (R/W1S)	PX9	Port x Bit 9 Mode Set. The PORT_FER_SET.PX9 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
8 (R/W1S)	PX8	Port x Bit 8 Mode Set. The PORT_FER_SET.PX8 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
7 (R/W1S)	PX7	Port x Bit 7 Mode Set. The PORT_FER_SET.PX7 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Table 18-21: PORT_FER_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	PX6	Port x Bit 6 Mode Set. The PORT_FER_SET.PX6 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
5 (R/W1S)	PX5	Port x Bit 5 Mode Set. The PORT_FER_SET.PX5 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
4 (R/W1S)	PX4	Port x Bit 4 Mode Set. The PORT_FER_SET.PX4 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
3 (R/W1S)	PX3	Port x Bit 3 Mode Set. The PORT_FER_SET.PX3 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
2 (R/W1S)	PX2	Port x Bit 2 Mode Set. The PORT_FER_SET.PX2 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
1 (R/W1S)	PX1	Port x Bit 1 Mode Set. The PORT_FER_SET.PX1 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode
0 (R/W1S)	PX0	Port x Bit 0 Mode Set. The PORT_FER_SET.PX0 bit enables peripheral mode.	
		0	No Effect
		1	Set Bit for Peripheral Mode

Port x GPIO Input Enable Register

The PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers enable or disable input drivers, which are required for using a GPIO pin in input mode.

Writes to the PORT_INEN register affect the input drivers for all pins of the port. To set or clear specific pin drivers without impacting other pin drivers of the port, use the PORT_INEN_SET and PORT_INEN_CLR registers.

If the input is enabled, reads from the PORT_DATA, PORT_DATA_SET, or PORT_DATA_CLR registers return the state of the pins. However, the state of the output is not overwritten by the input. It is altered by software writes only. Input and output drivers can be enabled at the same time. In this case, a read of the data register returns the true value of the data register and not the pin state.

For more information, see the PORT_DATA register description and the PORT_DIR register description.

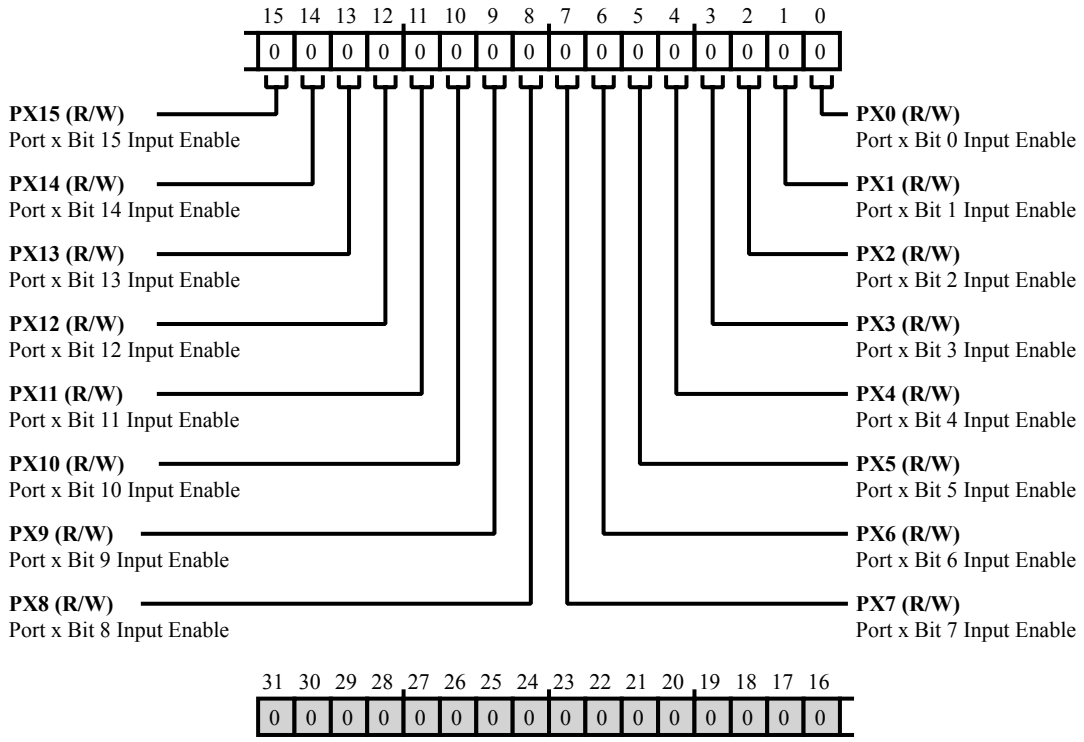


Figure 18-17: PORT_INEN Register Diagram

Table 18-22: PORT_INEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Table 18-22: PORT_INEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PX14	Port x Bit 14 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
13 (R/W)	PX13	Port x Bit 13 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
12 (R/W)	PX12	Port x Bit 12 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
11 (R/W)	PX11	Port x Bit 11 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
10 (R/W)	PX10	Port x Bit 10 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
9 (R/W)	PX9	Port x Bit 9 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
8 (R/W)	PX8	Port x Bit 8 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
7 (R/W)	PX7	Port x Bit 7 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
6 (R/W)	PX6	Port x Bit 6 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
5 (R/W)	PX5	Port x Bit 5 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Table 18-22: PORT_INEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	PX4	Port x Bit 4 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
3 (R/W)	PX3	Port x Bit 3 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
2 (R/W)	PX2	Port x Bit 2 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
1 (R/W)	PX1	Port x Bit 1 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver
0 (R/W)	PX0	Port x Bit 0 Input Enable.	
		0	Disable Input Driver
		1	Enable Input Driver

Port x GPIO Input Enable Clear Register

The PORT_INEN_CLR register disables the input drivers for GPIO pins. For more information, see the PORT_INEN register description.

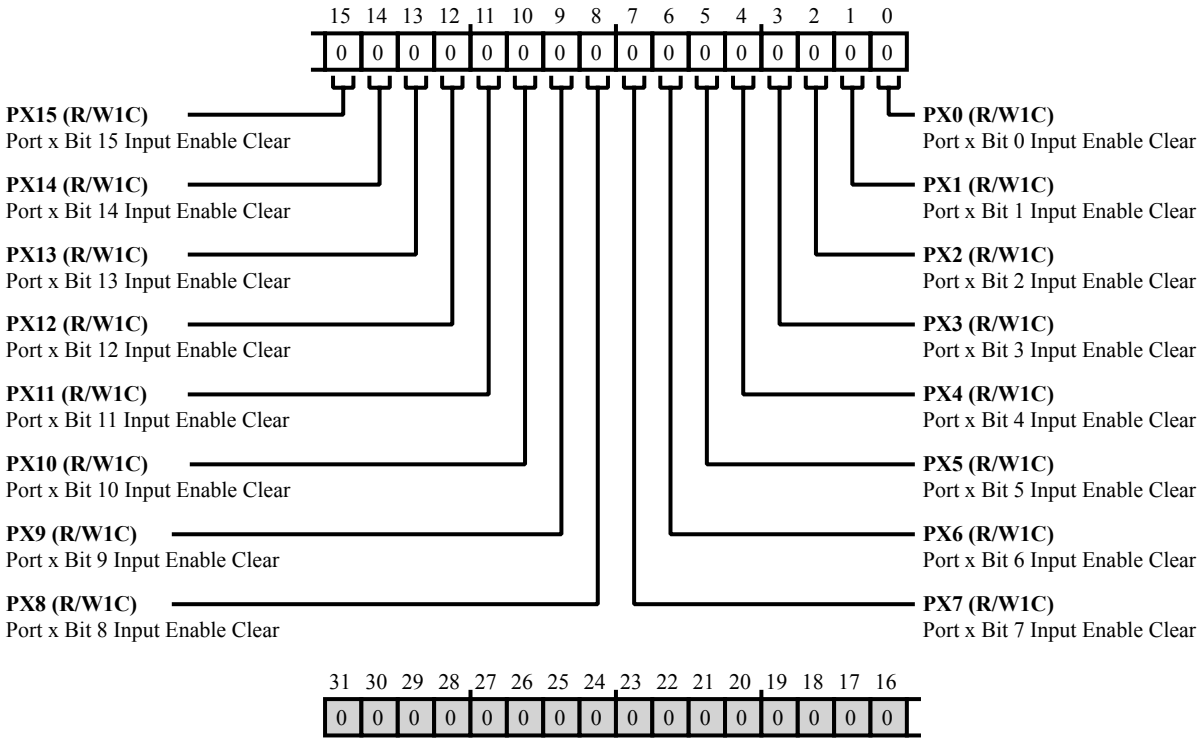


Figure 18-18: PORT_INEN_CLR Register Diagram

Table 18-23: PORT_INEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
14 (R/W1C)	PX14	Port x Bit 14 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
13 (R/W1C)	PX13	Port x Bit 13 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.

Table 18-23: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
11 (R/W1C)	PX11	Port x Bit 11 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
10 (R/W1C)	PX10	Port x Bit 10 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
9 (R/W1C)	PX9	Port x Bit 9 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
8 (R/W1C)	PX8	Port x Bit 8 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
7 (R/W1C)	PX7	Port x Bit 7 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
6 (R/W1C)	PX6	Port x Bit 6 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
5 (R/W1C)	PX5	Port x Bit 5 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
4 (R/W1C)	PX4	Port x Bit 4 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
3 (R/W1C)	PX3	Port x Bit 3 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.

Table 18-23: PORT_INEN_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
1 (R/W1C)	PX1	Port x Bit 1 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.
0 (R/W1C)	PX0	Port x Bit 0 Input Enable Clear.	
		0	No Effect
		1	Clear Bit. Set to disable the input driver.

Port x GPIO Input Enable Set Register

The PORT_INEN_SET register enables input drivers for GPIO pins. For more information, see the PORT_INEN register description.

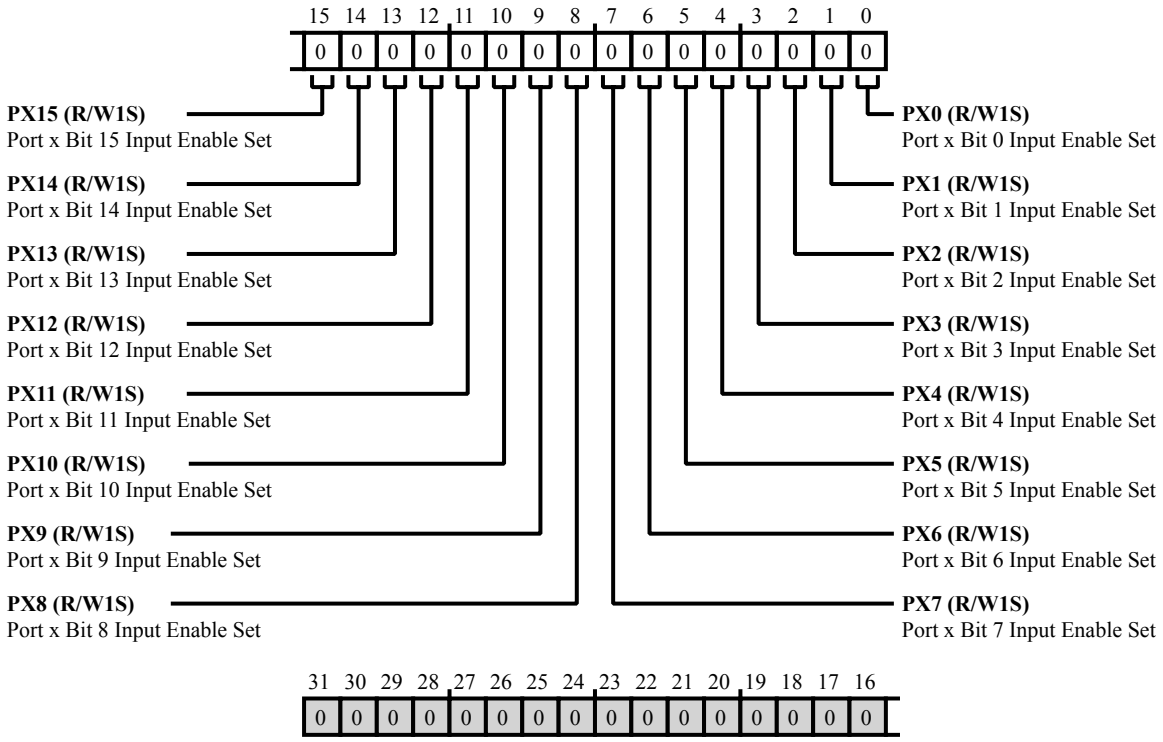


Figure 18-19: PORT_INEN_SET Register Diagram

Table 18-24: PORT_INEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
14 (R/W1S)	PX14	Port x Bit 14 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
13 (R/W1S)	PX13	Port x Bit 13 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.

Table 18-24: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1S)	PX12	Port x Bit 12 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
11 (R/W1S)	PX11	Port x Bit 11 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
10 (R/W1S)	PX10	Port x Bit 10 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
9 (R/W1S)	PX9	Port x Bit 9 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
8 (R/W1S)	PX8	Port x Bit 8 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
7 (R/W1S)	PX7	Port x Bit 7 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
6 (R/W1S)	PX6	Port x Bit 6 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
5 (R/W1S)	PX5	Port x Bit 5 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
4 (R/W1S)	PX4	Port x Bit 4 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
3 (R/W1S)	PX3	Port x Bit 3 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.

Table 18-24: PORT_INEN_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1S)	PX2	Port x Bit 2 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
1 (R/W1S)	PX1	Port x Bit 1 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.
0 (R/W1S)	PX0	Port x Bit 0 Input Enable Set.	
		0	No Effect
		1	Set Bit. Set to enable the input driver.

Port x GPIO Lock Register

The PORT_LOCK register enables (unlocks) or disables (locks) write access selectively for the PORT control registers.

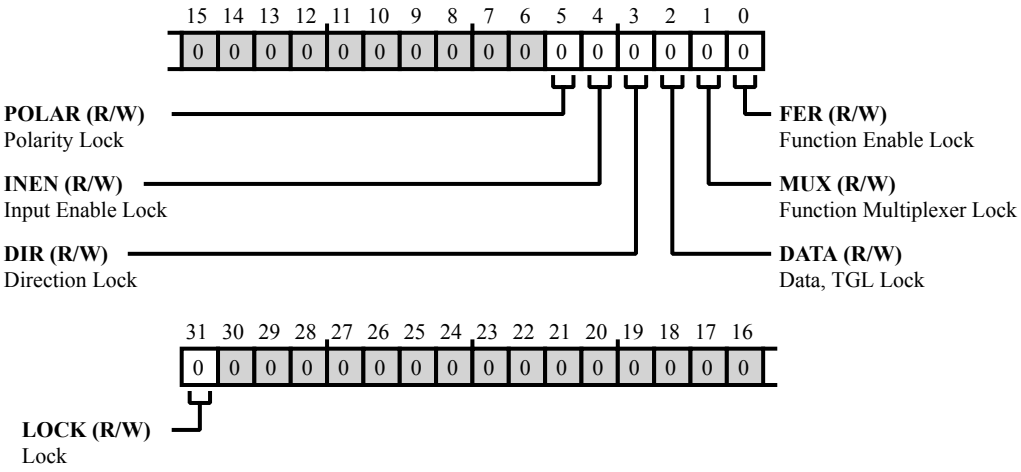


Figure 18-20: PORT_LOCK Register Diagram

Table 18-25: PORT_LOCK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the PORT_LOCK.LOCK bit is set, the PORT_LOCK register is read only (locked).	
		0	Unlock
		1	Lock
5 (R/W)	POLAR	Polarity Lock. The PORT_LOCK.POLAR disables write access to the PORT_POL, PORT_POL_SET, and PORT_POL_CLR registers.	
		0	Unlock POL
		1	Lock POL
4 (R/W)	INEN	Input Enable Lock. The PORT_LOCK.INEN disables write access to the PORT_INEN, PORT_INEN_SET, and PORT_INEN_CLR registers.	
		0	Unlock INEN
		1	Lock INEN

Table 18-25: PORT_LOCK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	DIR	Direction Lock. The PORT_LOCK.DIR disables write access to the PORT_DIR, PORT_DIR_SET, PORT_DIR_CLR registers.	
		0	Unlock DIR
		1	Lock DIR
2 (R/W)	DATA	Data, TGL Lock. The PORT_LOCK.DATA disables write access to the PORT_DATA, PORT_DATA_SET, PORT_DATA_CLR, and PORT_DATA_TGL registers.	
		0	Unlock DATA
		1	Lock DATA
1 (R/W)	MUX	Function Multiplexer Lock. The PORT_LOCK.MUX disables write accesses to the PORT_MUX register.	
		0	Unlock MUX
		1	Lock MUX
0 (R/W)	FER	Function Enable Lock. The PORT_LOCK.FER disables write access to the PORT_FER, PORT_FER_SET, and PORT_FER_CLR registers.	
		0	Unlock FER
		1	Lock FER

Port x Multiplexer Control Register

When a pin is in peripheral mode (not GPIO mode), the PORT_MUX register controls which peripheral takes ownership of a pin. Ports may have multiple, different peripheral functions. Two bits are required to describe every multiplexer on an individual pin-by-pin scheme. For example, bit 0 and bit 1 of the PORT_MUX register control the multiplexer of pin 0, bit 2 and bit 3 of PORT_MUX control the multiplexer of pin 1, and so on. The value of any PORT_MUX bit has no effect on the port pins when the associated bit in the PORT_FER register is 0 (selects GPIO mode). Even if a port has only one function, the PORT_MUX register is still present. For single function ports (no multiplexing is needed), leave the PORT_MUX bits at 0 (default). For all PORT_MUX bit fields: 00 = default/reset peripheral option, 01 = first alternate peripheral option, 10 = second alternate peripheral option, and 11 = third alternate peripheral option.

See the processor data sheet for details regarding the peripheral options associated with each port.

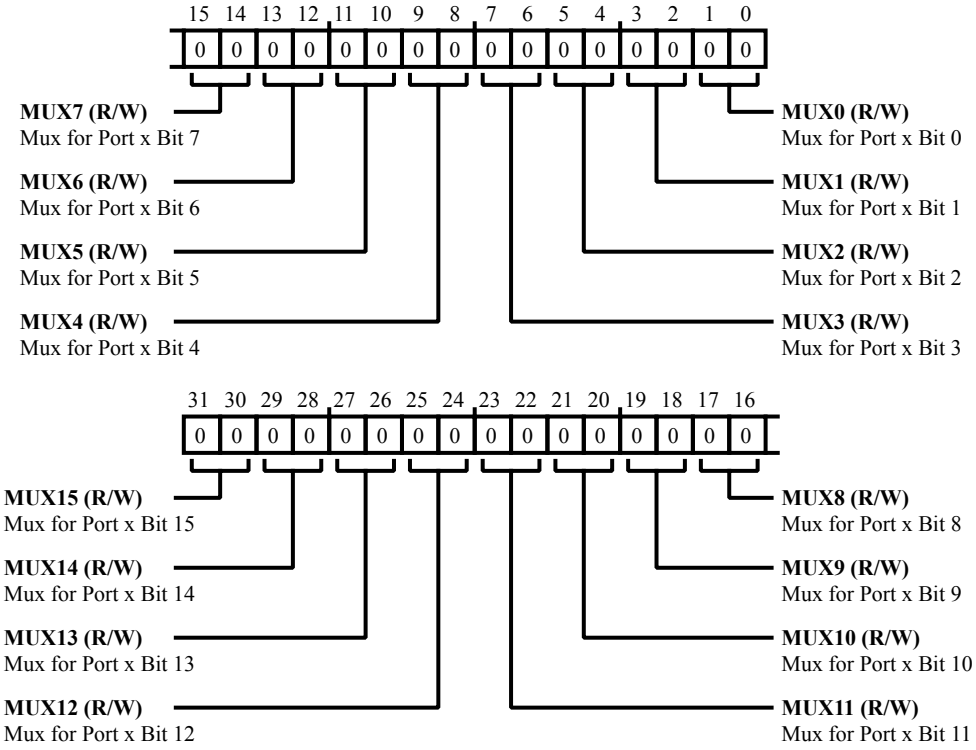


Figure 18-21: PORT_MUX Register Diagram

Table 18-26: PORT_MUX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	MUX15	Mux for Port x Bit 15. The PORT_MUX.MUX15 bit provides multiplexer control for port x bit 15.

Table 18-26: PORT_MUX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:28 (R/W)	MUX14	Mux for Port x Bit 14. The PORT_MUX.MUX14 bit provides multiplexer control for port x bit 14.
27:26 (R/W)	MUX13	Mux for Port x Bit 13. The PORT_MUX.MUX13 bit provides multiplexer control for port x bit 13.
25:24 (R/W)	MUX12	Mux for Port x Bit 12. The PORT_MUX.MUX12 bit provides multiplexer control for port x bit 12.
23:22 (R/W)	MUX11	Mux for Port x Bit 11. The PORT_MUX.MUX11 bit provides multiplexer control for port x bit 11.
21:20 (R/W)	MUX10	Mux for Port x Bit 10. The PORT_MUX.MUX10 bit provides multiplexer control for port x bit 10.
19:18 (R/W)	MUX9	Mux for Port x Bit 9. The PORT_MUX.MUX9 bit provides multiplexer control for port x bit 9.
17:16 (R/W)	MUX8	Mux for Port x Bit 8. The PORT_MUX.MUX8 bit provides multiplexer control for port x bit 8.
15:14 (R/W)	MUX7	Mux for Port x Bit 7. The PORT_MUX.MUX7 bit provides multiplexer control for port x bit 7.
13:12 (R/W)	MUX6	Mux for Port x Bit 6. The PORT_MUX.MUX6 bit provides multiplexer control for port x bit 6.
11:10 (R/W)	MUX5	Mux for Port x Bit 5. The PORT_MUX.MUX5 bit provides multiplexer control for port x bit 5.
9:8 (R/W)	MUX4	Mux for Port x Bit 4. The PORT_MUX.MUX4 bit provides multiplexer control for port x bit 4.
7:6 (R/W)	MUX3	Mux for Port x Bit 3. The PORT_MUX.MUX3 bit provides multiplexer control for port x bit 3.
5:4 (R/W)	MUX2	Mux for Port x Bit 2. The PORT_MUX.MUX2 bit provides multiplexer control for port x bit 2.
3:2 (R/W)	MUX1	Mux for Port x Bit 1. The PORT_MUX.MUX1 bit provides multiplexer control for port x bit 1.
1:0 (R/W)	MUX0	Mux for Port x Bit 0. The PORT_MUX.MUX0 bit provides multiplexer control for port x bit 0.

Port x GPIO Polarity Invert Register

The PORT_POL, PORT_POL_SET, and PORT_POL_CLR registers enable or disable inverting polarity of GPIO signals. To invert polarity of peripheral signals, use the inversion selection programming in the signal's corresponding module.

Writes to the PORT_POL register affect the polarity inversion selection of all pins of the port. To enable or disable polarity inversion for specific pins without impacting other pins of the port, use the PORT_POL_SET and PORT_POL_CLR registers.

Setting a bit in the PORT_POL register enables polarity inversion on the corresponding inversion GPIO pin, making the pin active-low or falling-edge sensitive. Clearing a bit in the PORT_POL register disables polarity (default state) on the corresponding GPIO pin, making it active-high or rising-edge sensitive.

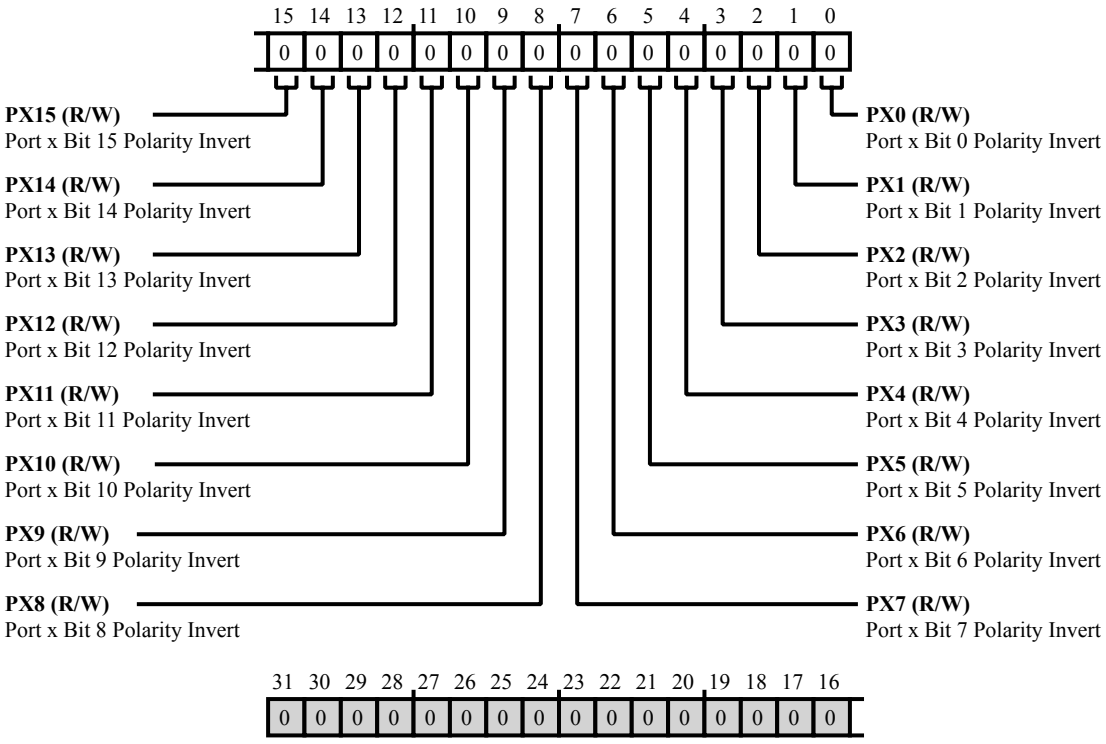


Figure 18-22: PORT_POL Register Diagram

Table 18-27: PORT_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	PX15	Port x Bit 15 Polarity Invert. The PORT_POL.PX15 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Table 18-27: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	PX14	Port x Bit 14 Polarity Invert. The PORT_POL.PX14 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
13 (R/W)	PX13	Port x Bit 13 Polarity Invert. The PORT_POL.PX13 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
12 (R/W)	PX12	Port x Bit 12 Polarity Invert. The PORT_POL.PX12 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
11 (R/W)	PX11	Port x Bit 11 Polarity Invert. The PORT_POL.PX11 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
10 (R/W)	PX10	Port x Bit 10 Polarity Invert. The PORT_POL.PX10 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
9 (R/W)	PX9	Port x Bit 9 Polarity Invert. The PORT_POL.PX9 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
8 (R/W)	PX8	Port x Bit 8 Polarity Invert. The PORT_POL.PX8 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Table 18-27: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	PX7	Port x Bit 7 Polarity Invert. The PORT_POL.PX7 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
6 (R/W)	PX6	Port x Bit 6 Polarity Invert. The PORT_POL.PX6 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
5 (R/W)	PX5	Port x Bit 5 Polarity Invert. The PORT_POL.PX5 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
4 (R/W)	PX4	Port x Bit 4 Polarity Invert. The PORT_POL.PX4 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
3 (R/W)	PX3	Port x Bit 3 Polarity Invert. The PORT_POL.PX3 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
2 (R/W)	PX2	Port x Bit 2 Polarity Invert. The PORT_POL.PX2 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.
1 (R/W)	PX1	Port x Bit 1 Polarity Invert. The PORT_POL.PX1 bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Table 18-27: PORT_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	PX0	Port x Bit 0 Polarity Invert. The `PORT_POL.PX0` bit enables polarity inversion.	
		0	No Invert. GPIO is active high or rising edge sensitive.
		1	Invert. GPIO is active low or falling edge sensitive.

Port x GPIO Polarity Invert Clear Register

The PORT_POL_CLR register disables polarity inversion for GPIO pins. For more information, see the PORT_POL register description.

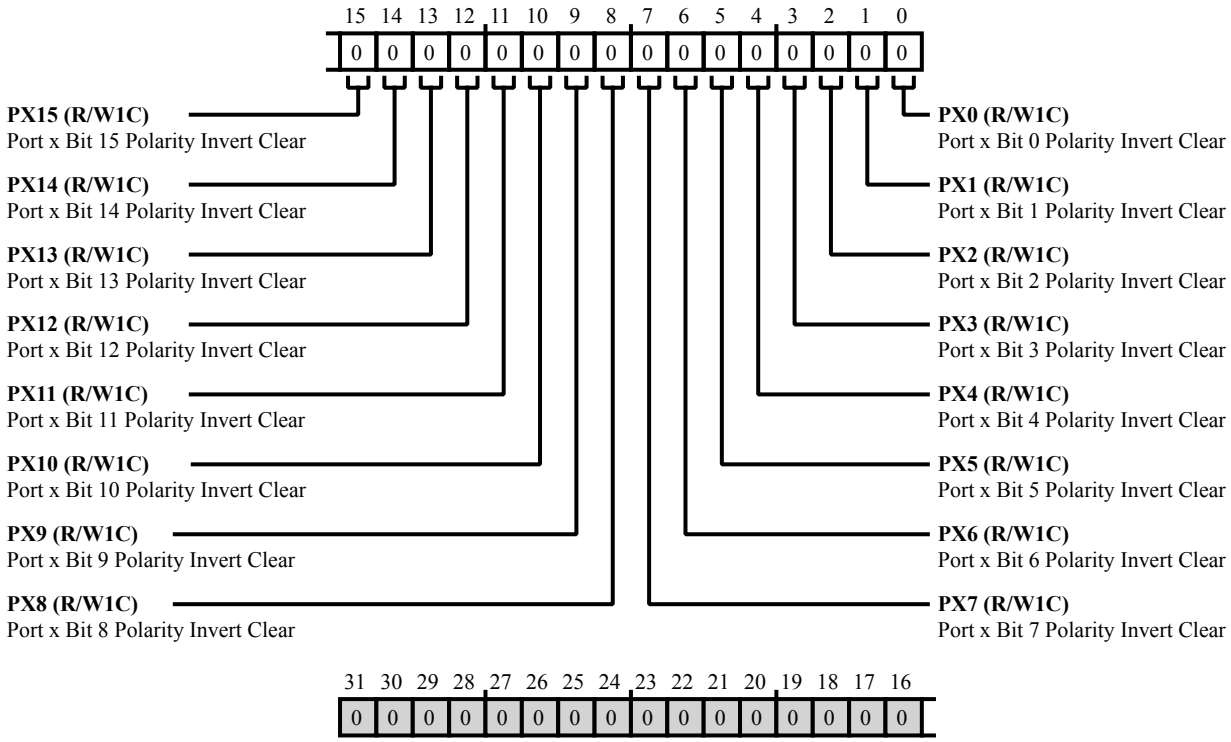


Figure 18-23: PORT_POL_CLR Register Diagram

Table 18-28: PORT_POL_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
14 (R/W1C)	PX14	Port x Bit 14 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
13 (R/W1C)	PX13	Port x Bit 13 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.

Table 18-28: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
11 (R/W1C)	PX11	Port x Bit 11 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
10 (R/W1C)	PX10	Port x Bit 10 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
9 (R/W1C)	PX9	Port x Bit 9 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
8 (R/W1C)	PX8	Port x Bit 8 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
7 (R/W1C)	PX7	Port x Bit 7 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
6 (R/W1C)	PX6	Port x Bit 6 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
5 (R/W1C)	PX5	Port x Bit 5 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
4 (R/W1C)	PX4	Port x Bit 4 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
3 (R/W1C)	PX3	Port x Bit 3 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.

Table 18-28: PORT_POL_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
1 (R/W1C)	PX1	Port x Bit 1 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.
0 (R/W1C)	PX0	Port x Bit 0 Polarity Invert Clear.	
		0	No Effect
		1	Clear Bit. Set to disable GPIO pin polarity invert.

Port x GPIO Polarity Invert Set Register

The PORT_POL_SET register enables polarity inversion for GPIO pins. For more information, see the PORT_POL register description.

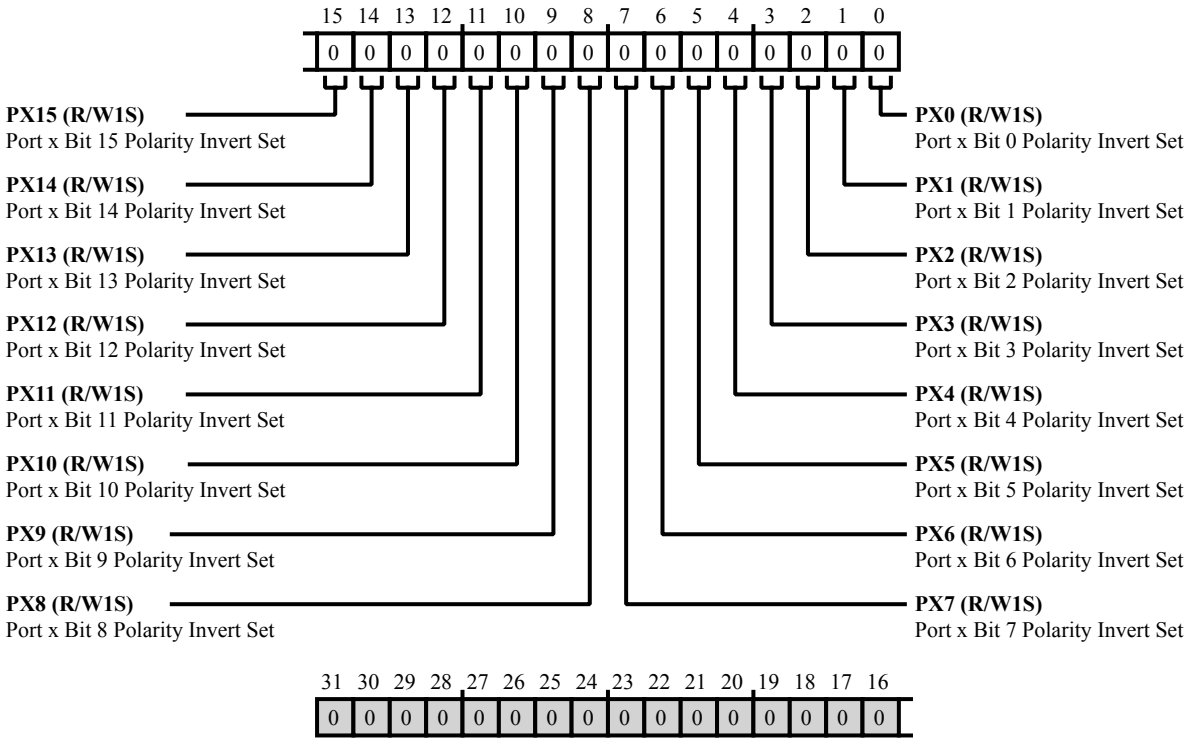


Figure 18-24: PORT_POL_SET Register Diagram

Table 18-29: PORT_POL_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1S)	PX15	Port x Bit 15 Polarity Invert Set. The PORT_POL_SET.PX15 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
14 (R/W1S)	PX14	Port x Bit 14 Polarity Invert Set. The PORT_POL_SET.PX14 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.

Table 18-29: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W1S)	PX13	Port x Bit 13 Polarity Invert Set. The PORT_POL_SET.PX13 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
12 (R/W1S)	PX12	Port x Bit 12 Polarity Invert Set. The PORT_POL_SET.PX12 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
11 (R/W1S)	PX11	Port x Bit 11 Polarity Invert Set. The PORT_POL_SET.PX11 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
10 (R/W1S)	PX10	Port x Bit 10 Polarity Invert Set. The PORT_POL_SET.PX10 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
9 (R/W1S)	PX9	Port x Bit 9 Polarity Invert Set. The PORT_POL_SET.PX9 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
8 (R/W1S)	PX8	Port x Bit 8 Polarity Invert Set. The PORT_POL_SET.PX8 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
7 (R/W1S)	PX7	Port x Bit 7 Polarity Invert Set. The PORT_POL_SET.PX7 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.

Table 18-29: PORT_POL_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	PX6	Port x Bit 6 Polarity Invert Set. The PORT_POL_SET.PX6 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
5 (R/W1S)	PX5	Port x Bit 5 Polarity Invert Set. The PORT_POL_SET.PX5 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
4 (R/W1S)	PX4	Port x Bit 4 Polarity Invert Set. The PORT_POL_SET.PX4 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
3 (R/W1S)	PX3	Port x Bit 3 Polarity Invert Set. The PORT_POL_SET.PX3 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
2 (R/W1S)	PX2	Port x Bit 2 Polarity Invert Set. The PORT_POL_SET.PX2 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
1 (R/W1S)	PX1	Port x Bit 1 Polarity Invert Set. The PORT_POL_SET.PX1 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.
0 (R/W1S)	PX0	Port x Bit 0 Polarity Invert Set. The PORT_POL_SET.PX0 bit enables pin polarity inversion.	
		0	No Effect
		1	Set Bit. Set to enable GPIO pin polarity invert.

Port x GPIO Trigger Toggle Register

The PORT_TRIG_TGL register permits toggling the state of output GPIO pins in response to a trigger from the TRU for the corresponding port. Setting bits in the PORT_TRIG_TGL register enables triggers to toggle the state of those specific pins without impacting other pins of the port.

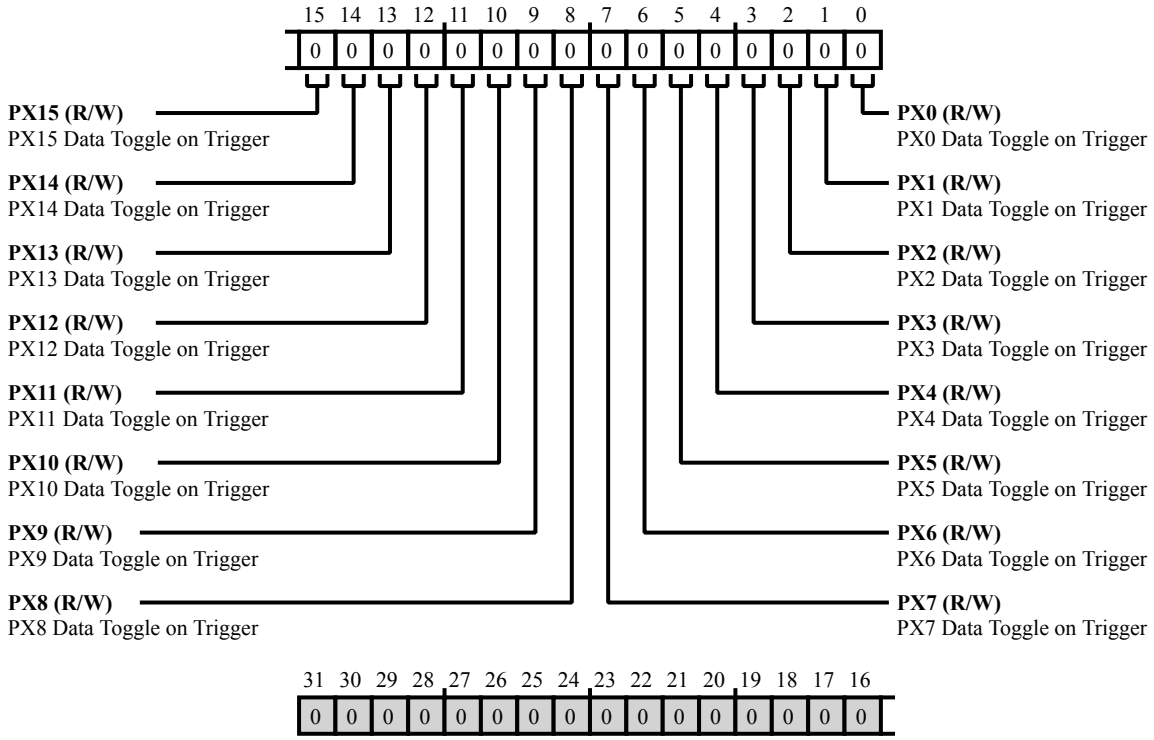


Figure 18-25: PORT_TRIG_TGL Register Diagram

Table 18-30: PORT_TRIG_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	PX15	PX15 Data Toggle on Trigger. The PORT_TRIG_TGL.PX15 bit enables triggers to toggle the state of the pin.
14 (R/W)	PX14	PX14 Data Toggle on Trigger. The PORT_TRIG_TGL.PX14 bit enables triggers to toggle the state of the pin.
13 (R/W)	PX13	PX13 Data Toggle on Trigger. The PORT_TRIG_TGL.PX13 bit enables triggers to toggle the state of the pin.
12 (R/W)	PX12	PX12 Data Toggle on Trigger. The PORT_TRIG_TGL.PX12 bit enables triggers to toggle the state of the pin.
11 (R/W)	PX11	PX11 Data Toggle on Trigger. The PORT_TRIG_TGL.PX11 bit enables triggers to toggle the state of the pin.

Table 18-30: PORT_TRIG_TGL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	PX10	PX10 Data Toggle on Trigger. The PORT_TRIG_TGL.PX10 bit enables triggers to toggle the state of the pin.
9 (R/W)	PX9	PX9 Data Toggle on Trigger. The PORT_TRIG_TGL.PX9 bit enables triggers to toggle the state of the pin.
8 (R/W)	PX8	PX8 Data Toggle on Trigger. The PORT_TRIG_TGL.PX8 bit enables triggers to toggle the state of the pin.
7 (R/W)	PX7	PX7 Data Toggle on Trigger. The PORT_TRIG_TGL.PX7 bit enables triggers to toggle the state of the pin.
6 (R/W)	PX6	PX6 Data Toggle on Trigger. The PORT_TRIG_TGL.PX6 bit enables triggers to toggle the state of the pin.
5 (R/W)	PX5	PX5 Data Toggle on Trigger. The PORT_TRIG_TGL.PX5 bit enables triggers to toggle the state of the pin.
4 (R/W)	PX4	PX4 Data Toggle on Trigger. The PORT_TRIG_TGL.PX4 bit enables triggers to toggle the state of the pin.
3 (R/W)	PX3	PX3 Data Toggle on Trigger. The PORT_TRIG_TGL.PX3 bit enables triggers to toggle the state of the pin.
2 (R/W)	PX2	PX2 Data Toggle on Trigger. The PORT_TRIG_TGL.PX2 bit enables triggers to toggle the state of the pin.
1 (R/W)	PX1	PX1 Data Toggle on Trigger. The PORT_TRIG_TGL.PX1 bit enables triggers to toggle the state of the pin.
0 (R/W)	PX0	PX0 Data Toggle on Trigger. The PORT_TRIG_TGL.PX0 bit enables triggers to toggle the state of the pin.

CM41X_M4 PINT Register Descriptions

The Pin Interrupt module (PINT) contains the following registers.

Table 18-31: CM41X_M4 PINT Register List

Name	Description
PINT_ASSIGN	PINT Assign Register
PINT_EDGE_CLR	PINT Edge Clear Register
PINT_EDGE_SET	PINT Edge Set Register
PINT_INV_CLR	PINT Invert Clear Register

Table 18-31: CM41X_M4 PINT Register List (Continued)

Name	Description
PINT_INV_SET	PINT Invert Set Register
PINT_LATCH	PINT Latch Register
PINT_MSK_CLR	PINT Mask Clear Register
PINT_MSK_SET	PINT Mask Set Register
PINT_PINSTATE	PINT Pin State Register
PINT_REQ	PINT Request Register

PINT Assign Register

The PINT_ASSIGN register controls the pin-to-interrupt request assignment in a byte-wide manner. This register consists of four control bytes that each function as a multiplexer control.

The PINT ports are subdivided into 8-bit half ports, resulting in lower and upper half 8-bit units. Using the multiplexers controlled by the PINT_ASSIGN register, the lower half units of eight pins can be forwarded to either byte 0 or byte 2 of either associated PINT block. The upper half units can be forwarded to either byte 1 or byte 3 of the PINT block, without further restrictions.

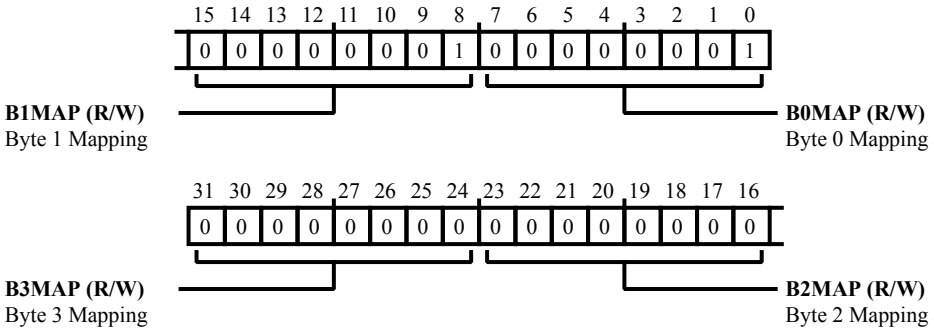


Figure 18-26: PINT_ASSIGN Register Diagram

Table 18-32: PINT_ASSIGN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	B3MAP	Byte 3 Mapping.	
		0	B3MAP_PAH. Byte 3 = PA.H
		1	B3MAP_PBH. Byte 3 = PB.H
23:16 (R/W)	B2MAP	Byte 2 Mapping.	
		0	B2MAP_PAL. Byte 2 = PA.L
		1	B2MAP_PBL. Byte 2 = PB.L
15:8 (R/W)	B1MAP	Byte 1 Mapping.	
		0	B1MAP_PAH. Byte 1 = PA.H
		1	B1MAP_PBH. Byte 1 = PB.H
7:0 (R/W)	B0MAP	Byte 0 Mapping.	
		0	B0MAP_PAL. Byte 0 = PA.L
		1	B0MAP_PBL. Byte 0 = PB.L

PINT Edge Clear Register

The PINT_EDGE_CLR register permits selecting level-sensitive interrupts. Writing 1 to a bit in PINT_EDGE_CLR enables level sensitivity for the corresponding pin interrupt.

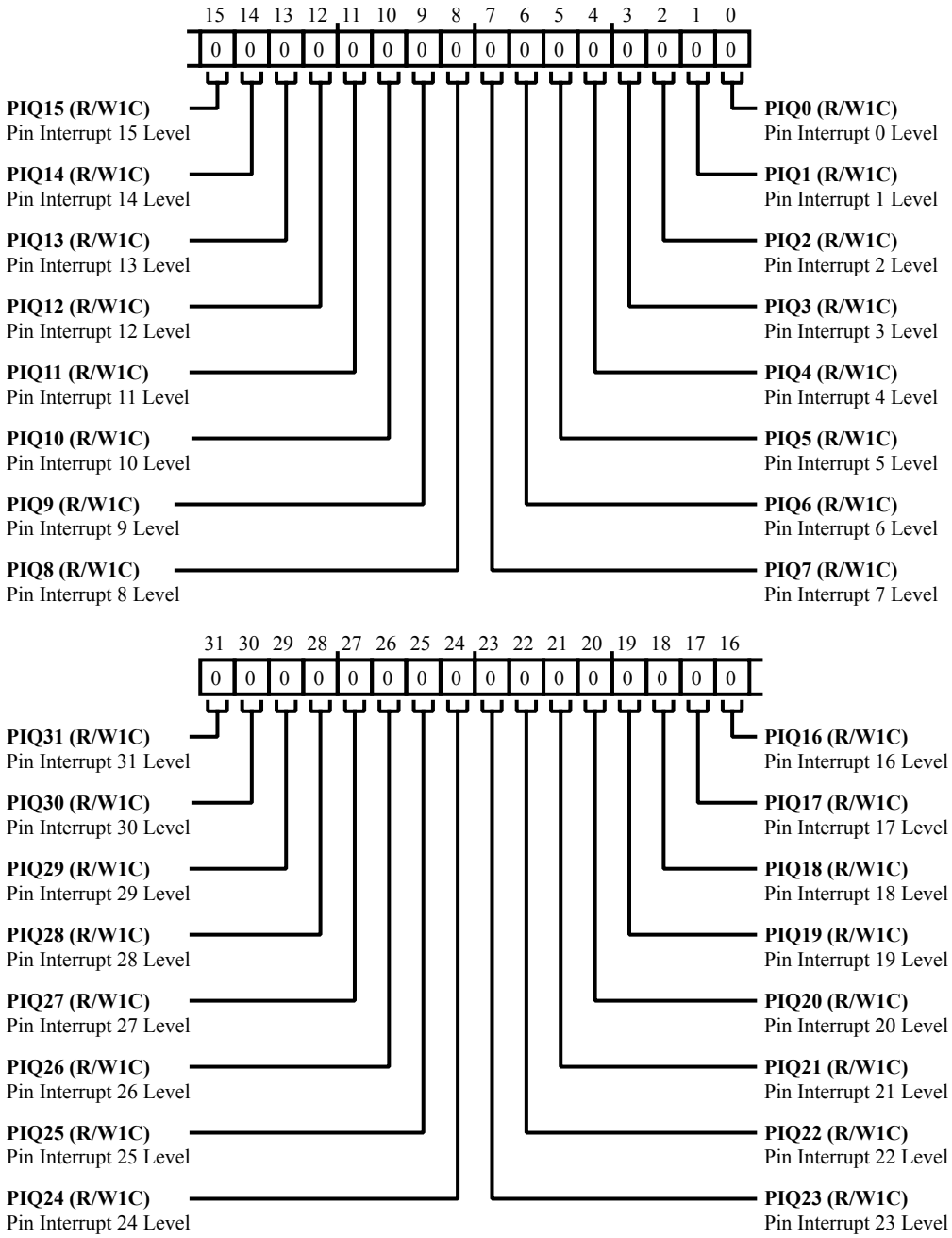


Figure 18-27: PINT_EDGE_CLR Register Diagram

Table 18-33: PINT_EDGE_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Level. Set the PINT_EDGE_CLR.PIQ31 bit to enable level sensitivity.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Level. Set the PINT_EDGE_CLR.PIQ30 bit to enable level sensitivity.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Level. Set the PINT_EDGE_CLR.PIQ29 bit to enable level sensitivity.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Level. Set the PINT_EDGE_CLR.PIQ28 bit to enable level sensitivity.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Level. Set the PINT_EDGE_CLR.PIQ27 bit to enable level sensitivity.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Level. Set the PINT_EDGE_CLR.PIQ26 bit to enable level sensitivity.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Level. Set the PINT_EDGE_CLR.PIQ25 bit to enable level sensitivity.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Level. Set the PINT_EDGE_CLR.PIQ24 bit to enable level sensitivity.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Level. Set the PINT_EDGE_CLR.PIQ23 bit to enable level sensitivity.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Level. Set the PINT_EDGE_CLR.PIQ22 bit to enable level sensitivity.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Level. Set the PINT_EDGE_CLR.PIQ21 bit to enable level sensitivity.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Level. Set the PINT_EDGE_CLR.PIQ20 bit to enable level sensitivity.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Level. Set the PINT_EDGE_CLR.PIQ19 bit to enable level sensitivity.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Level. Set the PINT_EDGE_CLR.PIQ18 bit to enable level sensitivity.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Level. Set the PINT_EDGE_CLR.PIQ17 bit to enable level sensitivity.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Level. Set the PINT_EDGE_CLR.PIQ16 bit to enable level sensitivity.

Table 18-33: PINT_EDGE_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 Level. Set the PINT_EDGE_CLR.PIQ15 bit to enable level sensitivity.
14 (R/W1C)	PIQ14	Pin Interrupt 14 Level. Set the PINT_EDGE_CLR.PIQ14 bit to enable level sensitivity.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Level. Set the PINT_EDGE_CLR.PIQ13 bit to enable level sensitivity.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Level. Set the PINT_EDGE_CLR.PIQ12 bit to enable level sensitivity.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Level. Set the PINT_EDGE_CLR.PIQ11 bit to enable level sensitivity.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Level. Set the PINT_EDGE_CLR.PIQ10 bit to enable level sensitivity.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Level. Set the PINT_EDGE_CLR.PIQ9 bit to enable level sensitivity.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Level. Set the PINT_EDGE_CLR.PIQ8 bit to enable level sensitivity.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Level. Set the PINT_EDGE_CLR.PIQ7 bit to enable level sensitivity.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Level. Set the PINT_EDGE_CLR.PIQ6 bit to enable level sensitivity.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Level. Set the PINT_EDGE_CLR.PIQ5 bit to enable level sensitivity.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Level. Set the PINT_EDGE_CLR.PIQ4 bit to enable level sensitivity.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Level. Set the PINT_EDGE_CLR.PIQ3 bit to enable level sensitivity.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Level. Set the PINT_EDGE_CLR.PIQ2 bit to enable level sensitivity.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Level. Set the PINT_EDGE_CLR.PIQ1 bit to enable level sensitivity.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Level. Set the PINT_EDGE_CLR.PIQ0 bit to enable level sensitivity.

PINT Edge Set Register

The PINT_EDGE_SET register permits selecting edge-sensitive interrupts. Writing 1 to a bit in PINT_EDGE_SET enables edge sensitivity for the corresponding pin interrupt.

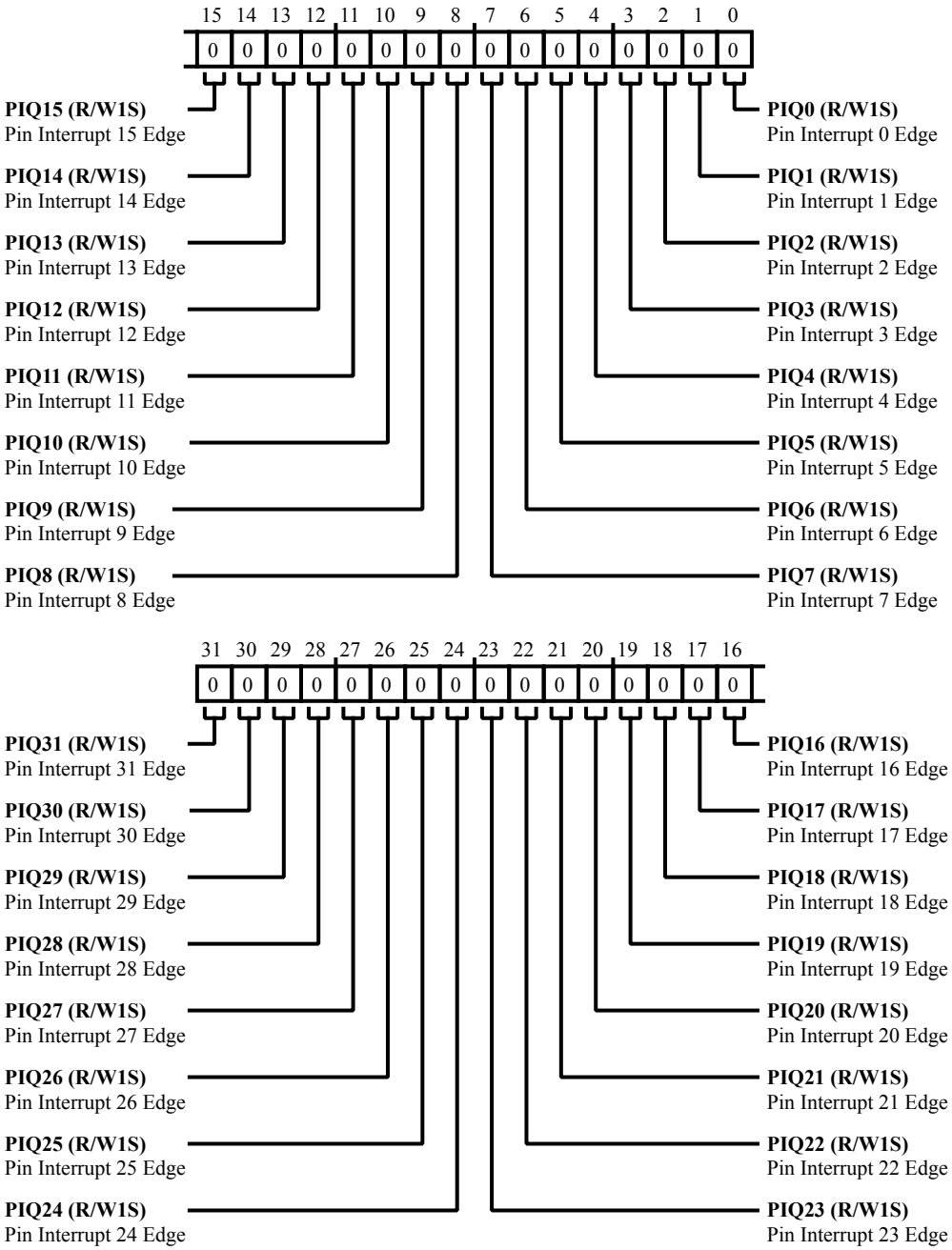


Figure 18-28: PINT_EDGE_SET Register Diagram

Table 18-34: PINT_EDGE_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Edge. Set the PINT_EDGE_SET.PIQ31 bit to enable edge sensitivity.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Edge. Set the PINT_EDGE_SET.PIQ30 bit to enable edge sensitivity.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Edge. Set the PINT_EDGE_SET.PIQ29 bit to enable edge sensitivity.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Edge. Set the PINT_EDGE_SET.PIQ28 bit to enable edge sensitivity.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Edge. Set the PINT_EDGE_SET.PIQ27 bit to enable edge sensitivity.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Edge. Set the PINT_EDGE_SET.PIQ26 bit to enable edge sensitivity.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Edge. Set the PINT_EDGE_SET.PIQ25 bit to enable edge sensitivity.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Edge. Set the PINT_EDGE_SET.PIQ24 bit to enable edge sensitivity.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Edge. Set the PINT_EDGE_SET.PIQ23 bit to enable edge sensitivity.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Edge. Set the PINT_EDGE_SET.PIQ22 bit to enable edge sensitivity.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Edge. Set the PINT_EDGE_SET.PIQ21 bit to enable edge sensitivity.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Edge. Set the PINT_EDGE_SET.PIQ20 bit to enable edge sensitivity.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Edge. Set the PINT_EDGE_SET.PIQ19 bit to enable edge sensitivity.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Edge. Set the PINT_EDGE_SET.PIQ18 bit to enable edge sensitivity.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Edge. Set the PINT_EDGE_SET.PIQ17 bit to enable edge sensitivity.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Edge. Set the PINT_EDGE_SET.PIQ16 bit to enable edge sensitivity.

Table 18-34: PINT_EDGE_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Edge. Set the PINT_EDGE_SET.PIQ15 bit to enable edge sensitivity.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Edge. Set the PINT_EDGE_SET.PIQ14 bit to enable edge sensitivity.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Edge. Set the PINT_EDGE_SET.PIQ13 bit to enable edge sensitivity.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Edge. Set the PINT_EDGE_SET.PIQ12 bit to enable edge sensitivity.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Edge. Set the PINT_EDGE_SET.PIQ11 bit to enable edge sensitivity.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Edge. Set the PINT_EDGE_SET.PIQ10 bit to enable edge sensitivity.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Edge. Set the PINT_EDGE_SET.PIQ9 bit to enable edge sensitivity.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Edge. Set the PINT_EDGE_SET.PIQ8 bit to enable edge sensitivity.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Edge. Set the PINT_EDGE_SET.PIQ7 bit to enable edge sensitivity.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Edge. Set the PINT_EDGE_SET.PIQ6 bit to enable edge sensitivity.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Edge. Set the PINT_EDGE_SET.PIQ5 bit to enable edge sensitivity.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Edge. Set the PINT_EDGE_SET.PIQ4 bit to enable edge sensitivity.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Edge. Set the PINT_EDGE_SET.PIQ3 bit to enable edge sensitivity.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Edge. Set the PINT_EDGE_SET.PIQ2 bit to enable edge sensitivity.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Edge. Set the PINT_EDGE_SET.PIQ1 bit to enable edge sensitivity.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Edge. Set the PINT_EDGE_SET.PIQ0 bit to enable edge sensitivity.

PINT Invert Clear Register

The PINT_INV_CLR register disables inverting input polarity. Writing 1 to a bit in PINT_INV_CLR disables an inverter for input on the corresponding pin.

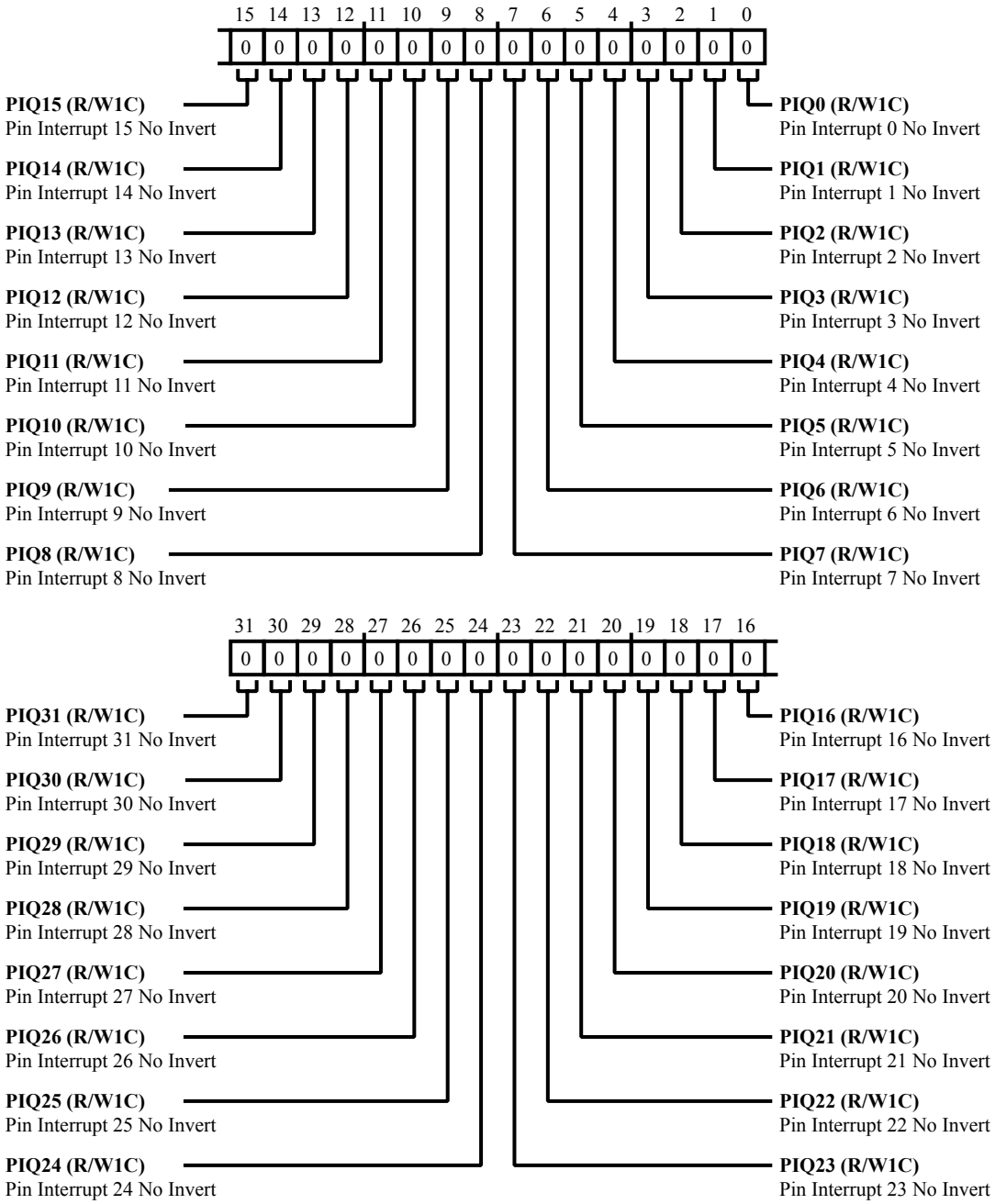


Figure 18-29: PINT_INV_CLR Register Diagram

Table 18-35: PINT_INV_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 No Invert. Set the PINT_INV_CLR.PIQ31 bit to disable inverted input.
30 (R/W1C)	PIQ30	Pin Interrupt 30 No Invert. Set the PINT_INV_CLR.PIQ30 bit to disable inverted input.
29 (R/W1C)	PIQ29	Pin Interrupt 29 No Invert. Set the PINT_INV_CLR.PIQ29 bit to disable inverted input.
28 (R/W1C)	PIQ28	Pin Interrupt 28 No Invert. Set the PINT_INV_CLR.PIQ28 bit to disable inverted input.
27 (R/W1C)	PIQ27	Pin Interrupt 27 No Invert. Set the PINT_INV_CLR.PIQ27 bit to disable inverted input.
26 (R/W1C)	PIQ26	Pin Interrupt 26 No Invert. Set the PINT_INV_CLR.PIQ26 bit to disable inverted input.
25 (R/W1C)	PIQ25	Pin Interrupt 25 No Invert. Set the PINT_INV_CLR.PIQ25 bit to disable inverted input.
24 (R/W1C)	PIQ24	Pin Interrupt 24 No Invert. Set the PINT_INV_CLR.PIQ24 bit to disable inverted input.
23 (R/W1C)	PIQ23	Pin Interrupt 23 No Invert. Set the PINT_INV_CLR.PIQ23 bit to disable inverted input.
22 (R/W1C)	PIQ22	Pin Interrupt 22 No Invert. Set the PINT_INV_CLR.PIQ22 bit to disable inverted input.
21 (R/W1C)	PIQ21	Pin Interrupt 21 No Invert. Set the PINT_INV_CLR.PIQ21 bit to disable inverted input.
20 (R/W1C)	PIQ20	Pin Interrupt 20 No Invert. Set the PINT_INV_CLR.PIQ20 bit to disable inverted input.
19 (R/W1C)	PIQ19	Pin Interrupt 19 No Invert. Set the PINT_INV_CLR.PIQ19 bit to disable inverted input.
18 (R/W1C)	PIQ18	Pin Interrupt 18 No Invert. Set the PINT_INV_CLR.PIQ18 bit to disable inverted input.
17 (R/W1C)	PIQ17	Pin Interrupt 17 No Invert. Set the PINT_INV_CLR.PIQ17 bit to disable inverted input.
16 (R/W1C)	PIQ16	Pin Interrupt 16 No Invert. Set the PINT_INV_CLR.PIQ16 bit to disable inverted input.

Table 18-35: PINT_INV_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 No Invert. Set the PINT_INV_CLR.PIQ15 bit to disable inverted input.
14 (R/W1C)	PIQ14	Pin Interrupt 14 No Invert. Set the PINT_INV_CLR.PIQ14 bit to disable inverted input.
13 (R/W1C)	PIQ13	Pin Interrupt 13 No Invert. Set the PINT_INV_CLR.PIQ13 bit to disable inverted input.
12 (R/W1C)	PIQ12	Pin Interrupt 12 No Invert. Set the PINT_INV_CLR.PIQ12 bit to disable inverted input.
11 (R/W1C)	PIQ11	Pin Interrupt 11 No Invert. Set the PINT_INV_CLR.PIQ11 bit to disable inverted input.
10 (R/W1C)	PIQ10	Pin Interrupt 10 No Invert. Set the PINT_INV_CLR.PIQ10 bit to disable inverted input.
9 (R/W1C)	PIQ9	Pin Interrupt 9 No Invert. Set the PINT_INV_CLR.PIQ9 bit to disable inverted input.
8 (R/W1C)	PIQ8	Pin Interrupt 8 No Invert. Set the PINT_INV_CLR.PIQ8 bit to disable inverted input.
7 (R/W1C)	PIQ7	Pin Interrupt 7 No Invert. Set the PINT_INV_CLR.PIQ7 bit to disable inverted input.
6 (R/W1C)	PIQ6	Pin Interrupt 6 No Invert. Set the PINT_INV_CLR.PIQ6 bit to disable inverted input.
5 (R/W1C)	PIQ5	Pin Interrupt 5 No Invert. Set the PINT_INV_CLR.PIQ5 bit to disable inverted input.
4 (R/W1C)	PIQ4	Pin Interrupt 4 No Invert. Set the PINT_INV_CLR.PIQ4 bit to disable inverted input.
3 (R/W1C)	PIQ3	Pin Interrupt 3 No Invert. Set the PINT_INV_CLR.PIQ3 bit to disable inverted input.
2 (R/W1C)	PIQ2	Pin Interrupt 2 No Invert. Set the PINT_INV_CLR.PIQ2 bit to disable inverted input.
1 (R/W1C)	PIQ1	Pin Interrupt 1 No Invert. Set the PINT_INV_CLR.PIQ1 bit to disable inverted input.
0 (R/W1C)	PIQ0	Pin Interrupt 0 No Invert. Set the PINT_INV_CLR.PIQ0 bit to disable inverted input.

PINT Invert Set Register

The PINT_INV_SET register enables inverting input polarity. Writing 1 to a bit in PINT_INV_SET enables an inverter for input on the corresponding pin.

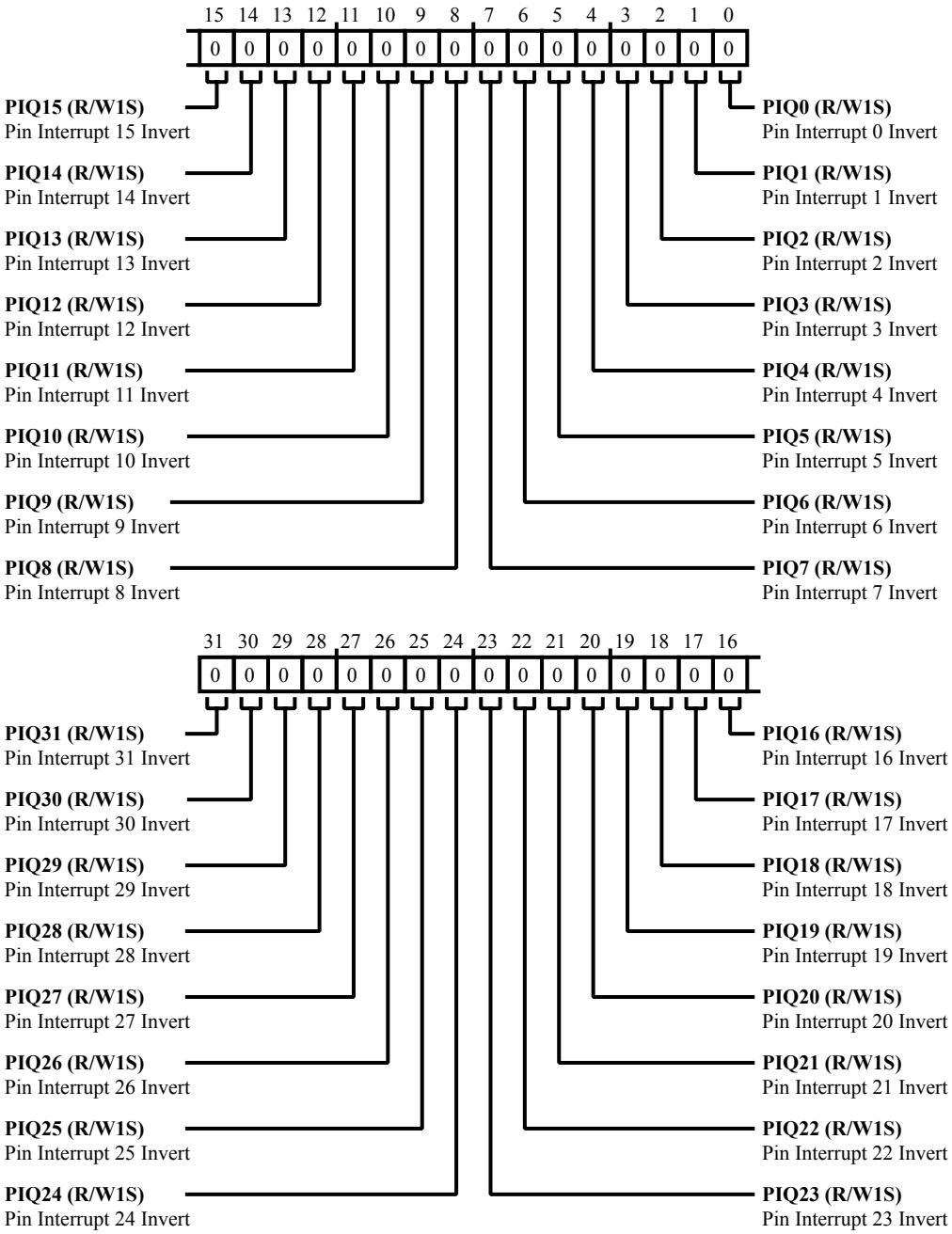


Figure 18-30: PINT_INV_SET Register Diagram

Table 18-36: PINT_INV_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Invert. Set the PINT_INV_SET.PIQ31 bit to enable inverted input.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Invert. Set the PINT_INV_SET.PIQ30 bit to enable inverted input.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Invert. Set the PINT_INV_SET.PIQ29 bit to enable inverted input.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Invert. Set the PINT_INV_SET.PIQ28 bit to enable inverted input.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Invert. Set the PINT_INV_SET.PIQ27 bit to enable inverted input.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Invert. Set the PINT_INV_SET.PIQ26 bit to enable inverted input.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Invert. Set the PINT_INV_SET.PIQ25 bit to enable inverted input.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Invert. Set the PINT_INV_SET.PIQ24 bit to enable inverted input.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Invert. Set the PINT_INV_SET.PIQ23 bit to enable inverted input.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Invert. Set the PINT_INV_SET.PIQ22 bit to enable inverted input.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Invert. Set the PINT_INV_SET.PIQ21 bit to enable inverted input.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Invert. Set the PINT_INV_SET.PIQ20 bit to enable inverted input.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Invert. Set the PINT_INV_SET.PIQ19 bit to enable inverted input.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Invert. Set the PINT_INV_SET.PIQ18 bit to enable inverted input.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Invert. Set the PINT_INV_SET.PIQ17 bit to enable inverted input.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Invert. Set the PINT_INV_SET.PIQ16 bit to enable inverted input.

Table 18-36: PINT_INV_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Invert. Set the PINT_INV_SET.PIQ15 bit to enable inverted input.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Invert. Set the PINT_INV_SET.PIQ14 bit to enable inverted input.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Invert. Set the PINT_INV_SET.PIQ13 bit to enable inverted input.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Invert. Set the PINT_INV_SET.PIQ12 bit to enable inverted input.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Invert. Set the PINT_INV_SET.PIQ11 bit to enable inverted input.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Invert. Set the PINT_INV_SET.PIQ10 bit to enable inverted input.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Invert. Set the PINT_INV_SET.PIQ9 bit to enable inverted input.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Invert. Set the PINT_INV_SET.PIQ8 bit to enable inverted input.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Invert. Set the PINT_INV_SET.PIQ7 bit to enable inverted input.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Invert. Set the PINT_INV_SET.PIQ6 bit to enable inverted input.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Invert. Set the PINT_INV_SET.PIQ5 bit to enable inverted input.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Invert. Set the PINT_INV_SET.PIQ4 bit to enable inverted input.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Invert. Set the PINT_INV_SET.PIQ3 bit to enable inverted input.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Invert. Set the PINT_INV_SET.PIQ2 bit to enable inverted input.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Invert. Set the PINT_INV_SET.PIQ1 bit to enable inverted input.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Invert. Set the PINT_INV_SET.PIQ0 bit to enable inverted input.

PINT Latch Register

The PINT_LATCH register indicates the interrupt latch status for pin interrupts. When set, an interrupt request is latched. When cleared, there is no interrupt request latched.

Both the PINT_REQ and PINT_LATCH registers indicate whether an interrupt request is latched on the respective pin. The PINT_LATCH register is a latch that operates regardless of the interrupt masks. Bits of the PINT_REQ register depend on the mask register. The PINT_REQ register is a logical AND of the PINT_LATCH register and the interrupt mask.

Having two separate registers here enables the user to interrogate certain pins in polling mode while others work in interrupt mode. The PINT_LATCH registers can be used for edge detection or pin activity detection.

Both registers have W1C behavior. Writing a 1 to either register clears the respective bits in both registers. For interrupt operation, the user may prefer to W1C the PINT_REQ register (address still loaded in Px pointer). In polling mode, it might be cleaner to W1C the PINT_LATCH register.

Whether in edge-sensitive mode or level-sensitive mode, PINT_LATCH bits are never cleared by hardware except at system reset. Even in level-sensitive mode, the PINT_LATCH register functions as latch.

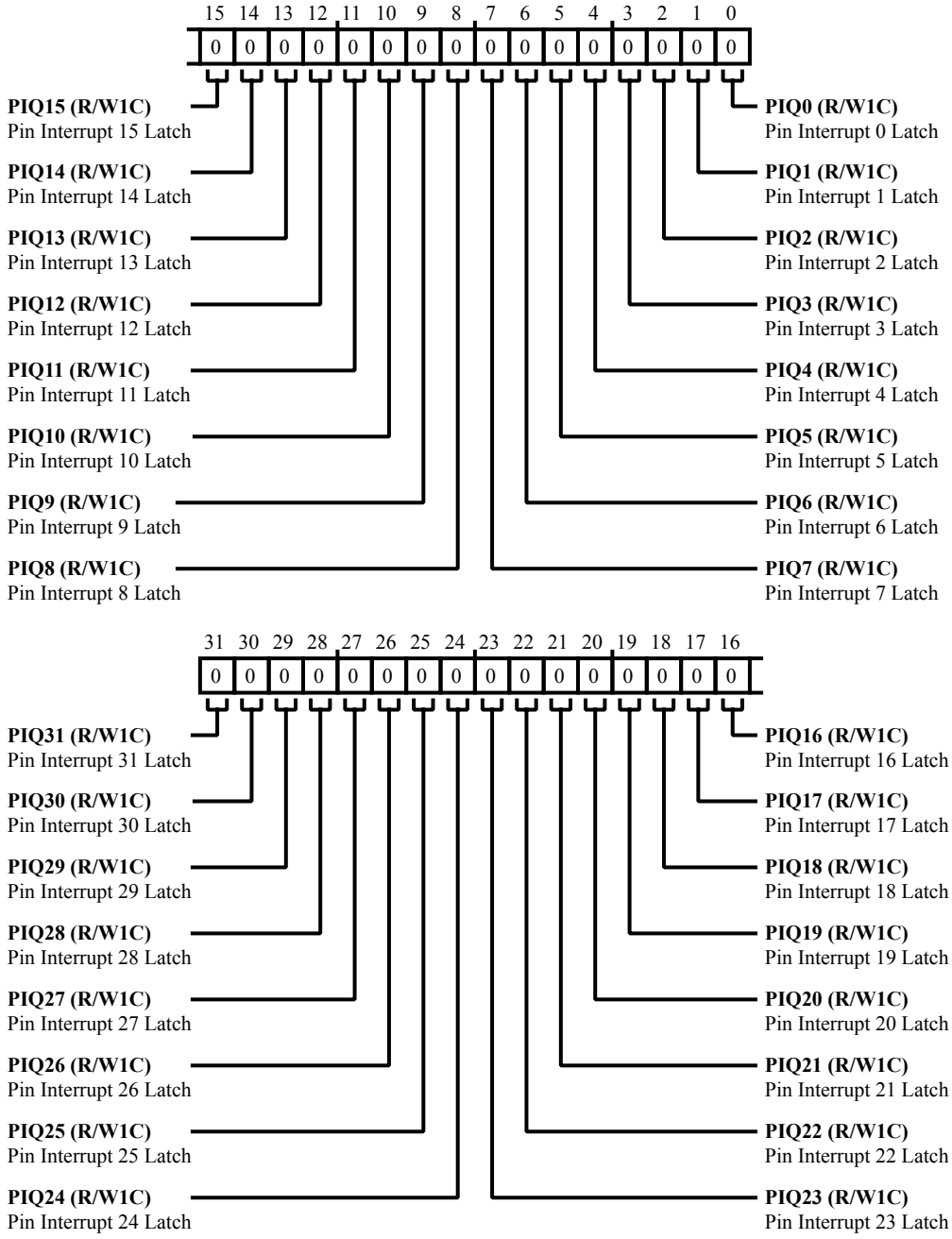


Figure 18-31: PINT_LATCH Register Diagram

Table 18-37: PINT_LATCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Latch. If the `PINT_LATCH.PIQ31` bit is set, the request is latched.

Table 18-37: PINT_LATCH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	PIQ30	Pin Interrupt 30 Latch. If the `PINT_LATCH.PIQ30` bit is set, the request is latched.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Latch. If the `PINT_LATCH.PIQ29` bit is set, the request is latched.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Latch. If the `PINT_LATCH.PIQ28` bit is set, the request is latched.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Latch. If the `PINT_LATCH.PIQ27` bit is set, the request is latched.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Latch. If the `PINT_LATCH.PIQ26` bit is set, the request is latched.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Latch. If the `PINT_LATCH.PIQ25` bit is set, the request is latched.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Latch. If the `PINT_LATCH.PIQ24` bit is set, the request is latched.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Latch. If the `PINT_LATCH.PIQ23` bit is set, the request is latched.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Latch. If the `PINT_LATCH.PIQ22` bit is set, the request is latched.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Latch. If the `PINT_LATCH.PIQ21` bit is set, the request is latched.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Latch. If the `PINT_LATCH.PIQ20` bit is set, the request is latched.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Latch. If the `PINT_LATCH.PIQ19` bit is set, the request is latched.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Latch. If the `PINT_LATCH.PIQ18` bit is set, the request is latched.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Latch. If the `PINT_LATCH.PIQ17` bit is set, the request is latched.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Latch. If the `PINT_LATCH.PIQ16` bit is set, the request is latched.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Latch. If the `PINT_LATCH.PIQ15` bit is set, the request is latched.

Table 18-37: PINT_LATCH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Latch. If the `PINT_LATCH.PIQ14` bit is set, the request is latched.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Latch. If the `PINT_LATCH.PIQ13` bit is set, the request is latched.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Latch. If the `PINT_LATCH.PIQ12` bit is set, the request is latched.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Latch. If the `PINT_LATCH.PIQ11` bit is set, the request is latched.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Latch. If the `PINT_LATCH.PIQ10` bit is set, the request is latched.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Latch. If the `PINT_LATCH.PIQ9` bit is set, the request is latched.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Latch. If the `PINT_LATCH.PIQ8` bit is set, the request is latched.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Latch. If the `PINT_LATCH.PIQ7` bit is set, the request is latched.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Latch. If the `PINT_LATCH.PIQ6` bit is set, the request is latched.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Latch. If the `PINT_LATCH.PIQ5` bit is set, the request is latched.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Latch. If the `PINT_LATCH.PIQ4` bit is set, the request is latched.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Latch. If the `PINT_LATCH.PIQ3` bit is set, the request is latched.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Latch. If the `PINT_LATCH.PIQ2` bit is set, the request is latched.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Latch. If the `PINT_LATCH.PIQ1` bit is set, the request is latched.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Latch. If the `PINT_LATCH.PIQ0` bit is set, the request is latched.

PINT Mask Clear Register

The PINT_MSK_CLR register permits masking (disabling) of interrupt requests. Writing 1 to a bit in PINT_MSK_CLR masks the corresponding pin interrupt.

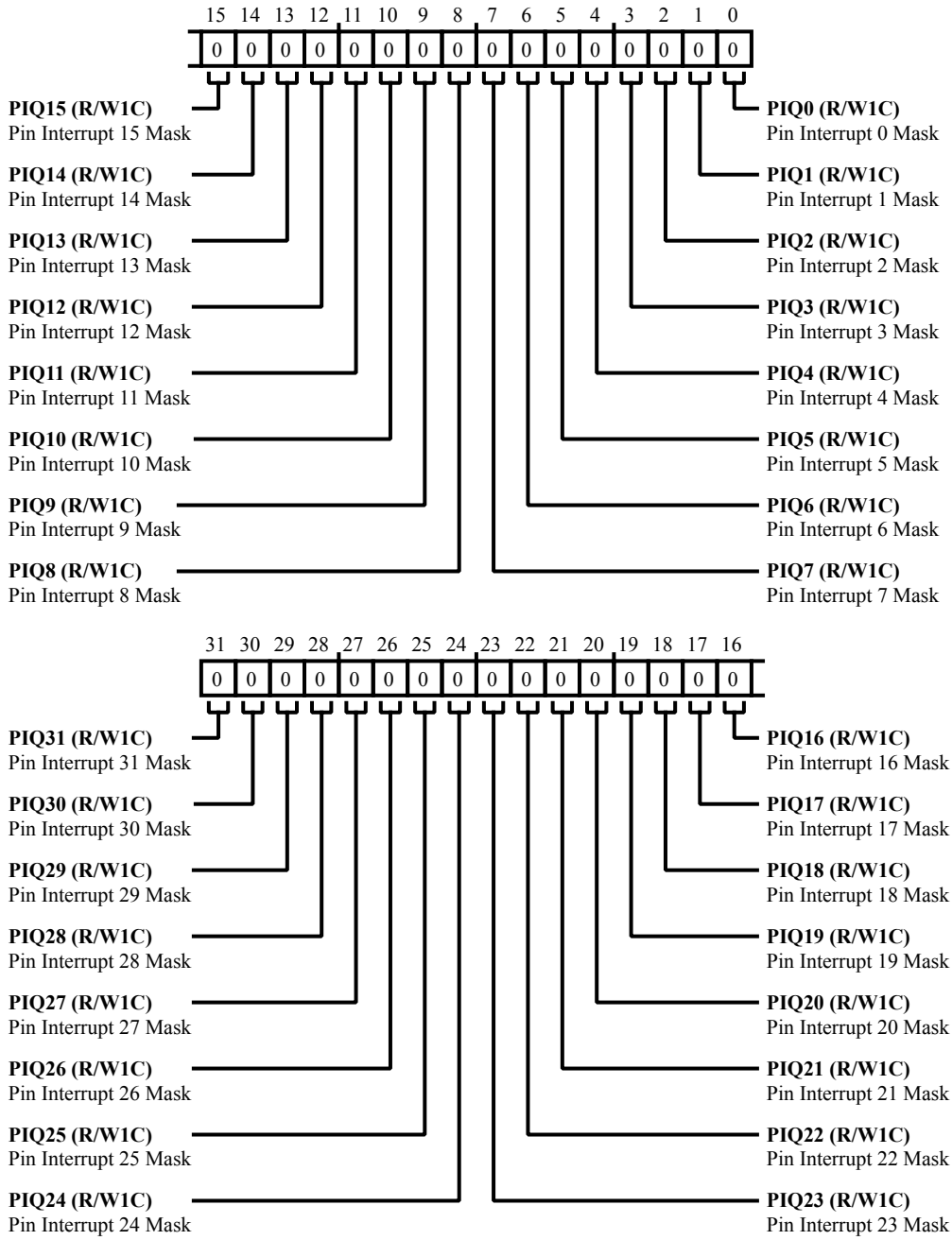


Figure 18-32: PINT_MSK_CLR Register Diagram

Table 18-38: PINT_MSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Mask. Set the PINT_MSK_CLR.PIQ31 bit to disable the interrupt.
30 (R/W1C)	PIQ30	Pin Interrupt 30 Mask. Set the PINT_MSK_CLR.PIQ30 bit to disable the interrupt.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Mask. Set the PINT_MSK_CLR.PIQ29 bit to disable the interrupt.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Mask. Set the PINT_MSK_CLR.PIQ28 bit to disable the interrupt.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Mask. Set the PINT_MSK_CLR.PIQ27 bit to disable the interrupt.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Mask. Set the PINT_MSK_CLR.PIQ26 bit to disable the interrupt.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Mask. Set the PINT_MSK_CLR.PIQ25 bit to disable the interrupt.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Mask. Set the PINT_MSK_CLR.PIQ24 bit to disable the interrupt.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Mask. Set the PINT_MSK_CLR.PIQ23 bit to disable the interrupt.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Mask. Set the PINT_MSK_CLR.PIQ22 bit to disable the interrupt.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Mask. Set the PINT_MSK_CLR.PIQ21 bit to disable the interrupt.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Mask. Set the PINT_MSK_CLR.PIQ20 bit to disable the interrupt.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Mask. Set the PINT_MSK_CLR.PIQ19 bit to disable the interrupt.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Mask. Set the PINT_MSK_CLR.PIQ18 bit to disable the interrupt.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Mask. Set the PINT_MSK_CLR.PIQ17 bit to disable the interrupt.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Mask. Set the PINT_MSK_CLR.PIQ16 bit to disable the interrupt.

Table 18-38: PINT_MSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1C)	PIQ15	Pin Interrupt 15 Mask. Set the PINT_MSK_CLR.PIQ15 bit to disable the interrupt.
14 (R/W1C)	PIQ14	Pin Interrupt 14 Mask. Set the PINT_MSK_CLR.PIQ14 bit to disable the interrupt.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Mask. Set the PINT_MSK_CLR.PIQ13 bit to disable the interrupt.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Mask. Set the PINT_MSK_CLR.PIQ12 bit to disable the interrupt.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Mask. Set the PINT_MSK_CLR.PIQ11 bit to disable the interrupt.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Mask. Set the PINT_MSK_CLR.PIQ10 bit to disable the interrupt.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Mask. Set the PINT_MSK_CLR.PIQ9 bit to disable the interrupt.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Mask. Set the PINT_MSK_CLR.PIQ8 bit to disable the interrupt.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Mask. Set the PINT_MSK_CLR.PIQ7 bit to disable the interrupt.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Mask. Set the PINT_MSK_CLR.PIQ6 bit to disable the interrupt.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Mask. Set the PINT_MSK_CLR.PIQ5 bit to disable the interrupt.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Mask. Set the PINT_MSK_CLR.PIQ4 bit to disable the interrupt.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Mask. Set the PINT_MSK_CLR.PIQ3 bit to disable the interrupt.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Mask. Set the PINT_MSK_CLR.PIQ2 bit to disable the interrupt.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Mask. Set the PINT_MSK_CLR.PIQ1 bit to disable the interrupt.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Mask. Set the PINT_MSK_CLR.PIQ0 bit to disable the interrupt.

PINT Mask Set Register

The PINT_MSK_SET register permits unmasking (enabling) of interrupt requests. Writing 1 to a bit in PINT_MSK_SET unmasks the corresponding pin interrupt.

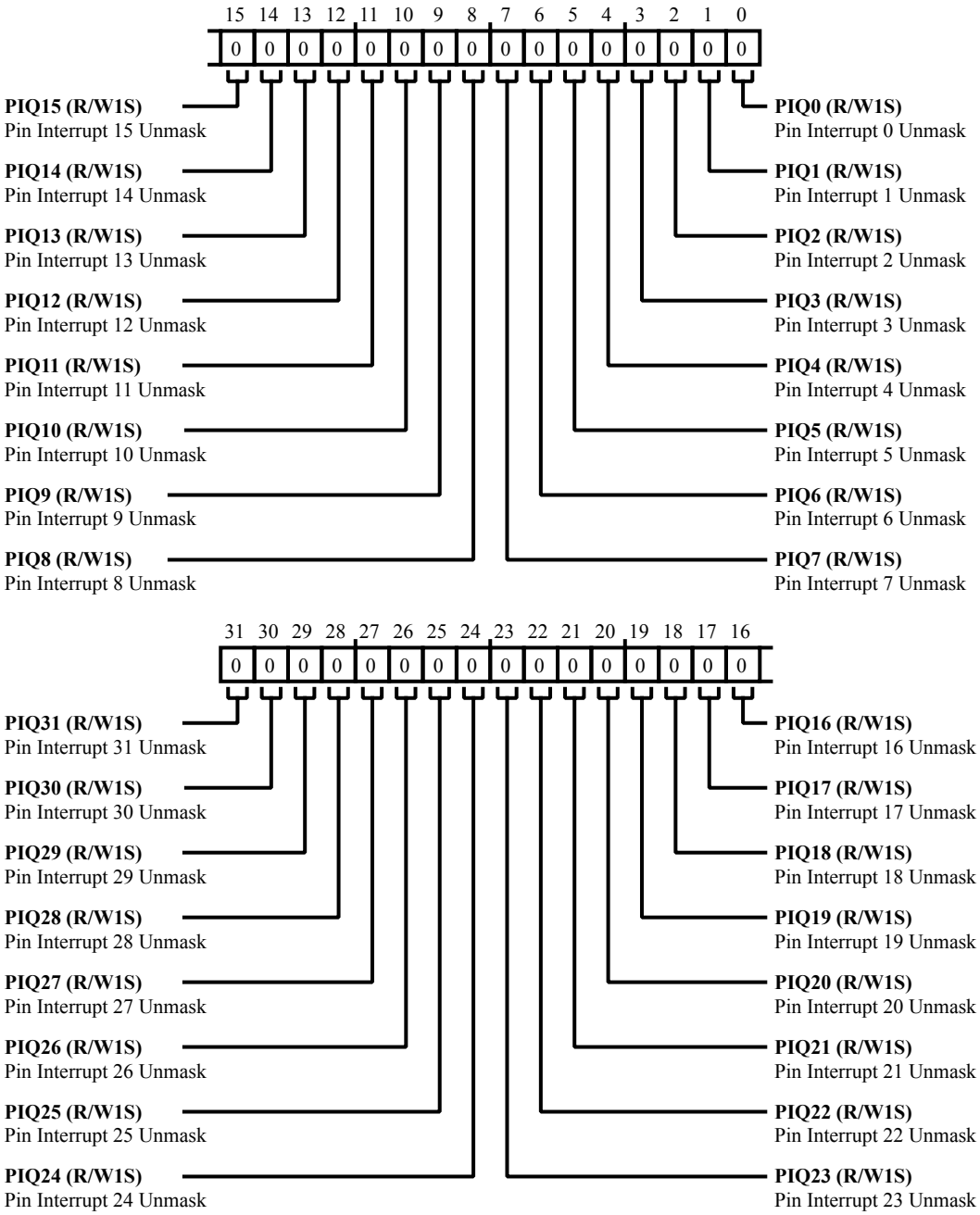


Figure 18-33: PINT_MSK_SET Register Diagram

Table 18-39: PINT_MSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1S)	PIQ31	Pin Interrupt 31 Unmask. Set the PINT_MSK_SET.PIQ31 bit to enable the interrupt.
30 (R/W1S)	PIQ30	Pin Interrupt 30 Unmask. Set the PINT_MSK_SET.PIQ30 bit to enable the interrupt.
29 (R/W1S)	PIQ29	Pin Interrupt 29 Unmask. Set the PINT_MSK_SET.PIQ29 bit to enable the interrupt.
28 (R/W1S)	PIQ28	Pin Interrupt 28 Unmask. Set the PINT_MSK_SET.PIQ28 bit to enable the interrupt.
27 (R/W1S)	PIQ27	Pin Interrupt 27 Unmask. Set the PINT_MSK_SET.PIQ27 bit to enable the interrupt.
26 (R/W1S)	PIQ26	Pin Interrupt 26 Unmask. Set the PINT_MSK_SET.PIQ26 bit to enable the interrupt.
25 (R/W1S)	PIQ25	Pin Interrupt 25 Unmask. Set the PINT_MSK_SET.PIQ25 bit to enable the interrupt.
24 (R/W1S)	PIQ24	Pin Interrupt 24 Unmask. Set the PINT_MSK_SET.PIQ24 bit to enable the interrupt.
23 (R/W1S)	PIQ23	Pin Interrupt 23 Unmask. Set the PINT_MSK_SET.PIQ23 bit to enable the interrupt.
22 (R/W1S)	PIQ22	Pin Interrupt 22 Unmask. Set the PINT_MSK_SET.PIQ22 bit to enable the interrupt.
21 (R/W1S)	PIQ21	Pin Interrupt 21 Unmask. Set the PINT_MSK_SET.PIQ21 bit to enable the interrupt.
20 (R/W1S)	PIQ20	Pin Interrupt 20 Unmask. Set the PINT_MSK_SET.PIQ20 bit to enable the interrupt.
19 (R/W1S)	PIQ19	Pin Interrupt 19 Unmask. Set the PINT_MSK_SET.PIQ19 bit to enable the interrupt.
18 (R/W1S)	PIQ18	Pin Interrupt 18 Unmask. Set the PINT_MSK_SET.PIQ18 bit to enable the interrupt.
17 (R/W1S)	PIQ17	Pin Interrupt 17 Unmask. Set the PINT_MSK_SET.PIQ17 bit to enable the interrupt.
16 (R/W1S)	PIQ16	Pin Interrupt 16 Unmask. Set the PINT_MSK_SET.PIQ16 bit to enable the interrupt.

Table 18-39: PINT_MSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W1S)	PIQ15	Pin Interrupt 15 Unmask. Set the PINT_MSK_SET.PIQ15 bit to enable the interrupt.
14 (R/W1S)	PIQ14	Pin Interrupt 14 Unmask. Set the PINT_MSK_SET.PIQ14 bit to enable the interrupt.
13 (R/W1S)	PIQ13	Pin Interrupt 13 Unmask. Set the PINT_MSK_SET.PIQ13 bit to enable the interrupt.
12 (R/W1S)	PIQ12	Pin Interrupt 12 Unmask. Set the PINT_MSK_SET.PIQ12 bit to enable the interrupt.
11 (R/W1S)	PIQ11	Pin Interrupt 11 Unmask. Set the PINT_MSK_SET.PIQ11 bit to enable the interrupt.
10 (R/W1S)	PIQ10	Pin Interrupt 10 Unmask. Set the PINT_MSK_SET.PIQ10 bit to enable the interrupt.
9 (R/W1S)	PIQ9	Pin Interrupt 9 Unmask. Set the PINT_MSK_SET.PIQ9 bit to enable the interrupt.
8 (R/W1S)	PIQ8	Pin Interrupt 8 Unmask. Set the PINT_MSK_SET.PIQ8 bit to enable the interrupt.
7 (R/W1S)	PIQ7	Pin Interrupt 7 Unmask. Set the PINT_MSK_SET.PIQ7 bit to enable the interrupt.
6 (R/W1S)	PIQ6	Pin Interrupt 6 Unmask. Set the PINT_MSK_SET.PIQ6 bit to enable the interrupt.
5 (R/W1S)	PIQ5	Pin Interrupt 5 Unmask. Set the PINT_MSK_SET.PIQ5 bit to enable the interrupt.
4 (R/W1S)	PIQ4	Pin Interrupt 4 Unmask. Set the PINT_MSK_SET.PIQ4 bit to enable the interrupt.
3 (R/W1S)	PIQ3	Pin Interrupt 3 Unmask. Set the PINT_MSK_SET.PIQ3 bit to enable the interrupt.
2 (R/W1S)	PIQ2	Pin Interrupt 2 Unmask. Set the PINT_MSK_SET.PIQ2 bit to enable the interrupt.
1 (R/W1S)	PIQ1	Pin Interrupt 1 Unmask. Set the PINT_MSK_SET.PIQ1 bit to enable the interrupt.
0 (R/W1S)	PIQ0	Pin Interrupt 0 Unmask. Set the PINT_MSK_SET.PIQ0 bit to enable the interrupt.

PINT Pin State Register

When a half port is assigned to a byte in any PINT block, the state of the eight pins (regardless of GPIO or function, input or output) can be seen in the PINT_PINSTATE register. While neither input nor output drivers of the pin are enabled, reads of the pin state in PINT_PINSTATE return zero. The PINT_PINSTATE register reports the inverted state of the pin if the signal inverter is activated by the PINT_INV_SET register. The inverter can be enabled on an individual bit-by-bit basis. Every bit in the PINT_INV_SET and PINT_INV_CLR register pair represents a pin signal.

The pin interrupt pin state registers enable the service routine to read the current state of the pin without reading from GPIO space. If there was an edge-sensitive interrupt, the service routine can check whether the state of the pin is still high or turned low.

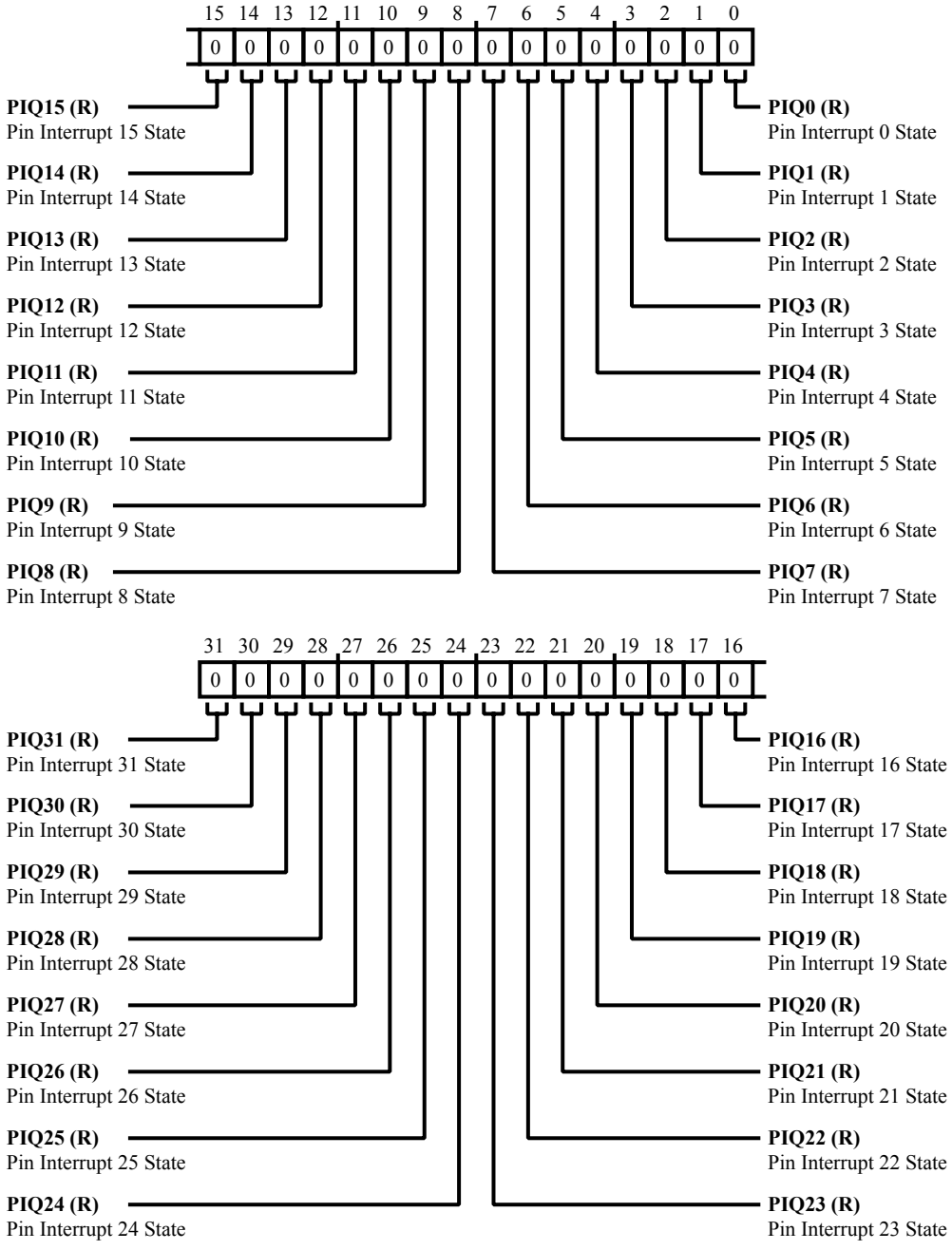


Figure 18-34: PINT_PINSTATE Register Diagram

Table 18-40: PINT_PINSTATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	PIQ31	Pin Interrupt 31 State. A read of the `PINT_PINSTATE.PIQ31` bit returns the pin state.

Table 18-40: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/NW)	PIQ30	Pin Interrupt 30 State. A read of the PINT_PINSTATE.PIQ30 bit returns the pin state.
29 (R/NW)	PIQ29	Pin Interrupt 29 State. A read of the PINT_PINSTATE.PIQ29 bit returns the pin state.
28 (R/NW)	PIQ28	Pin Interrupt 28 State. A read of the PINT_PINSTATE.PIQ28 bit returns the pin state.
27 (R/NW)	PIQ27	Pin Interrupt 27 State. A read of the PINT_PINSTATE.PIQ27 bit returns the pin state.
26 (R/NW)	PIQ26	Pin Interrupt 26 State. A read of the PINT_PINSTATE.PIQ26 bit returns the pin state.
25 (R/NW)	PIQ25	Pin Interrupt 25 State. A read of the PINT_PINSTATE.PIQ25 bit returns the pin state.
24 (R/NW)	PIQ24	Pin Interrupt 24 State. A read of the PINT_PINSTATE.PIQ24 bit returns the pin state.
23 (R/NW)	PIQ23	Pin Interrupt 23 State. A read of the PINT_PINSTATE.PIQ23 bit returns the pin state.
22 (R/NW)	PIQ22	Pin Interrupt 22 State. A read of the PINT_PINSTATE.PIQ22 bit returns the pin state.
21 (R/NW)	PIQ21	Pin Interrupt 21 State. A read of the PINT_PINSTATE.PIQ21 bit returns the pin state.
20 (R/NW)	PIQ20	Pin Interrupt 20 State. A read of the PINT_PINSTATE.PIQ20 bit returns the pin state.
19 (R/NW)	PIQ19	Pin Interrupt 19 State. A read of the PINT_PINSTATE.PIQ19 bit returns the pin state.
18 (R/NW)	PIQ18	Pin Interrupt 18 State. A read of the PINT_PINSTATE.PIQ18 bit returns the pin state.
17 (R/NW)	PIQ17	Pin Interrupt 17 State. A read of the PINT_PINSTATE.PIQ17 bit returns the pin state.
16 (R/NW)	PIQ16	Pin Interrupt 16 State. A read of the PINT_PINSTATE.PIQ16 bit returns the pin state.
15 (R/NW)	PIQ15	Pin Interrupt 15 State. A read of the PINT_PINSTATE.PIQ15 bit returns the pin state.

Table 18-40: PINT_PINSTATE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	PIQ14	Pin Interrupt 14 State. A read of the PINT_PINSTATE.PIQ14 bit returns the pin state.
13 (R/NW)	PIQ13	Pin Interrupt 13 State. A read of the PINT_PINSTATE.PIQ13 bit returns the pin state.
12 (R/NW)	PIQ12	Pin Interrupt 12 State. A read of the PINT_PINSTATE.PIQ12 bit returns the pin state.
11 (R/NW)	PIQ11	Pin Interrupt 11 State. A read of the PINT_PINSTATE.PIQ11 bit returns the pin state.
10 (R/NW)	PIQ10	Pin Interrupt 10 State. A read of the PINT_PINSTATE.PIQ10 bit returns the pin state.
9 (R/NW)	PIQ9	Pin Interrupt 9 State. A read of the PINT_PINSTATE.PIQ9 bit returns the pin state.
8 (R/NW)	PIQ8	Pin Interrupt 8 State. A read of the PINT_PINSTATE.PIQ8 bit returns the pin state.
7 (R/NW)	PIQ7	Pin Interrupt 7 State. A read of the PINT_PINSTATE.PIQ7 bit returns the pin state.
6 (R/NW)	PIQ6	Pin Interrupt 6 State. A read of the PINT_PINSTATE.PIQ6 bit returns the pin state.
5 (R/NW)	PIQ5	Pin Interrupt 5 State. A read of the PINT_PINSTATE.PIQ5 bit returns the pin state.
4 (R/NW)	PIQ4	Pin Interrupt 4 State. A read of the PINT_PINSTATE.PIQ4 bit returns the pin state.
3 (R/NW)	PIQ3	Pin Interrupt 3 State. A read of the PINT_PINSTATE.PIQ3 bit returns the pin state.
2 (R/NW)	PIQ2	Pin Interrupt 2 State. A read of the PINT_PINSTATE.PIQ2 bit returns the pin state.
1 (R/NW)	PIQ1	Pin Interrupt 1 State. A read of the PINT_PINSTATE.PIQ1 bit returns the pin state.
0 (R/NW)	PIQ0	Pin Interrupt 0 State. A read of the PINT_PINSTATE.PIQ0 bit returns the pin state.

PINT Request Register

The PINT_REQ register indicates the interrupt request status for pin interrupts. When set, an interrupt request is pending. When cleared, there is no interrupt request pending.

Both the PINT_REQ and PINT_LATCH registers indicate whether an interrupt request is latched on the respective pin. The PINT_LATCH register is a latch that operates regardless of the interrupt masks. Bits of the PINT_REQ register depend on the mask register. The PINT_REQ register is a logical AND of the PINT_LATCH register and the interrupt mask.

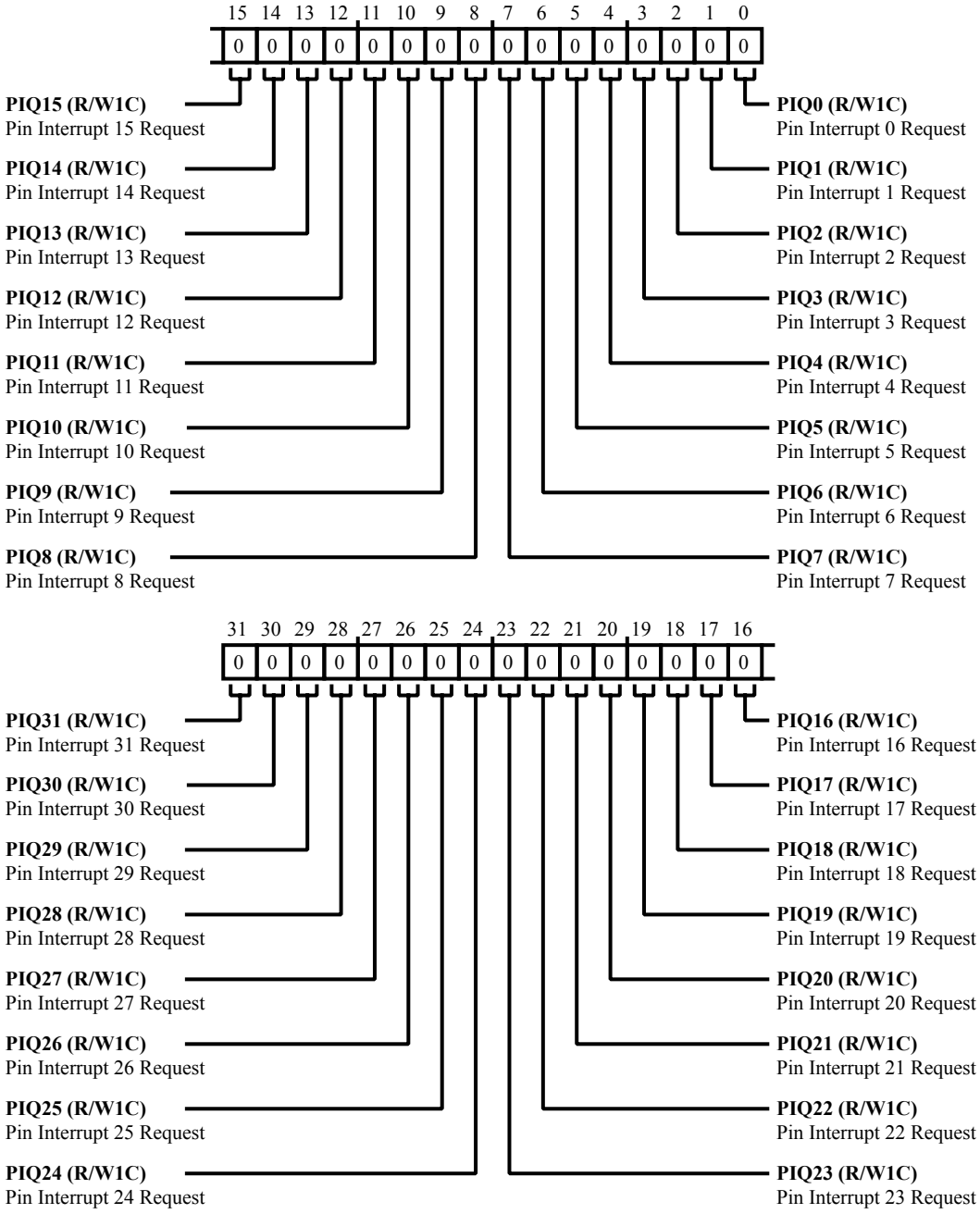


Figure 18-35: PINT_REQ Register Diagram

Table 18-41: PINT_REQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	PIQ31	Pin Interrupt 31 Request. If the `PINT_REQ.PIQ31` bit is set, a request is pending.

Table 18-41: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
30 (R/W1C)	PIQ30	Pin Interrupt 30 Request. If the PINT_REQ.PIQ30 bit is set, a request is pending.
29 (R/W1C)	PIQ29	Pin Interrupt 29 Request. If the PINT_REQ.PIQ29 bit is set, a request is pending.
28 (R/W1C)	PIQ28	Pin Interrupt 28 Request. If the PINT_REQ.PIQ28 bit is set, a request is pending.
27 (R/W1C)	PIQ27	Pin Interrupt 27 Request. If the PINT_REQ.PIQ27 bit is set, a request is pending.
26 (R/W1C)	PIQ26	Pin Interrupt 26 Request. If the PINT_REQ.PIQ26 bit is set, a request is pending.
25 (R/W1C)	PIQ25	Pin Interrupt 25 Request. If the PINT_REQ.PIQ25 bit is set, a request is pending.
24 (R/W1C)	PIQ24	Pin Interrupt 24 Request. If the PINT_REQ.PIQ24 bit is set, a request is pending.
23 (R/W1C)	PIQ23	Pin Interrupt 23 Request. If the PINT_REQ.PIQ23 bit is set, a request is pending.
22 (R/W1C)	PIQ22	Pin Interrupt 22 Request. If the PINT_REQ.PIQ22 bit is set, a request is pending.
21 (R/W1C)	PIQ21	Pin Interrupt 21 Request. If the PINT_REQ.PIQ21 bit is set, a request is pending.
20 (R/W1C)	PIQ20	Pin Interrupt 20 Request. If the PINT_REQ.PIQ20 bit is set, a request is pending.
19 (R/W1C)	PIQ19	Pin Interrupt 19 Request. If the PINT_REQ.PIQ19 bit is set, a request is pending.
18 (R/W1C)	PIQ18	Pin Interrupt 18 Request. If the PINT_REQ.PIQ18 bit is set, a request is pending.
17 (R/W1C)	PIQ17	Pin Interrupt 17 Request. If the PINT_REQ.PIQ17 bit is set, a request is pending.
16 (R/W1C)	PIQ16	Pin Interrupt 16 Request. If the PINT_REQ.PIQ16 bit is set, a request is pending.
15 (R/W1C)	PIQ15	Pin Interrupt 15 Request. If the PINT_REQ.PIQ15 bit is set, a request is pending.

Table 18-41: PINT_REQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	PIQ14	Pin Interrupt 14 Request. If the `PINT_REQ.PIQ14` bit is set, a request is pending.
13 (R/W1C)	PIQ13	Pin Interrupt 13 Request. If the `PINT_REQ.PIQ13` bit is set, a request is pending.
12 (R/W1C)	PIQ12	Pin Interrupt 12 Request. If the `PINT_REQ.PIQ12` bit is set, a request is pending.
11 (R/W1C)	PIQ11	Pin Interrupt 11 Request. If the `PINT_REQ.PIQ11` bit is set, a request is pending.
10 (R/W1C)	PIQ10	Pin Interrupt 10 Request. If the `PINT_REQ.PIQ10` bit is set, a request is pending.
9 (R/W1C)	PIQ9	Pin Interrupt 9 Request. If the `PINT_REQ.PIQ9` bit is set, a request is pending.
8 (R/W1C)	PIQ8	Pin Interrupt 8 Request. If the `PINT_REQ.PIQ8` bit is set, a request is pending.
7 (R/W1C)	PIQ7	Pin Interrupt 7 Request. If the `PINT_REQ.PIQ7` bit is set, a request is pending.
6 (R/W1C)	PIQ6	Pin Interrupt 6 Request. If the `PINT_REQ.PIQ6` bit is set, a request is pending.
5 (R/W1C)	PIQ5	Pin Interrupt 5 Request. If the `PINT_REQ.PIQ5` bit is set, a request is pending.
4 (R/W1C)	PIQ4	Pin Interrupt 4 Request. If the `PINT_REQ.PIQ4` bit is set, a request is pending.
3 (R/W1C)	PIQ3	Pin Interrupt 3 Request. If the `PINT_REQ.PIQ3` bit is set, a request is pending.
2 (R/W1C)	PIQ2	Pin Interrupt 2 Request. If the `PINT_REQ.PIQ2` bit is set, a request is pending.
1 (R/W1C)	PIQ1	Pin Interrupt 1 Request. If the `PINT_REQ.PIQ1` bit is set, a request is pending.
0 (R/W1C)	PIQ0	Pin Interrupt 0 Request. If the `PINT_REQ.PIQ0` bit is set, a request is pending.

19 General-Purpose Timer (TIMER)

The general-purpose timer (GP Timer) module serves as a collection of system timers that support various system-level functions. These functions include:

- Synchronized PWM waveform output capability
- External signal capture
- External event count
- General time-base functionality

Additionally, interrupt requests can be generated upon completion of timer events. Moreover, GP timers can act both as trigger masters and trigger slaves.

GP Timer Features

Each timer can be individually configured in any of these modes:

- Pin interrupt capture mode
- Windowed watchdog mode
- Pulse-width count and capture (WDTH_CAP) mode
- External Event (EXT_CLK) mode
- Pulse-width modulation (PWM_OUT) mode

Other features include:

- Synchronous operation
- Consistent management of period and pulse width values
- Autobaud detection for UART module (where available)
- Graceful bit pattern termination when stopping
- Support for center-aligned PWM patterns
- Error detection on implausible pattern values

- All read and write accesses to 32-bit registers are atomic
- Every timer has its dedicated interrupt request output

CM41X_M4 TIMER Register List

The General-Purpose Timer block (TIMER) provides timers that may be used for external event capture and measurement, system timing, and PWM waveform generation. A set of registers governs TIMER operations. For more information on TIMER functionality, see the TIMER register descriptions.

Table 19-1: CM41X_M4 TIMER Register List

Name	Description
TIMER_BCAST_DLY	Broadcast Delay Register
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_ERR_TYPE	Error Type Status Register
TIMER_RUN	Run Register
TIMER_RUN_CLR	Run Clear Register
TIMER_RUN_SET	Run Set Register
TIMER_STAT_ILAT	Status Interrupt Latch Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_TMR[n]_CFG	Timer n Configuration Register
TIMER_TMR[n]_CNT	Timer n Counter Register
TIMER_TMR[n]_DLY	Timer n Delay Register
TIMER_TMR[n]_PER	Timer n Period Register
TIMER_TMR[n]_WID	Timer n Width Register
TIMER_TRG_IE	Trigger Slave Enable Register
TIMER_TRG_MSK	Trigger Master Mask Register

CM41X_M0 TIMER Interrupt List

Table 19-2: CM41X_M0 TIMER Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
15	TIMER0_STAT	TIMER0 Status	Level	
15	TIMER0_TMR0	TIMER0 Timer 0	Level	
15	TIMER0_TMR1	TIMER0 Timer 1	Level	
15	TIMER0_TMR2	TIMER0 Timer 2	Level	
15	TIMER0_TMR3	TIMER0 Timer 3	Level	
15	TIMER0_TMR4	TIMER0 Timer 4	Level	
15	TIMER0_TMR5	TIMER0 Timer 5	Level	
15	TIMER0_TMR6	TIMER0 Timer 6	Level	
15	TIMER0_TMR7	TIMER0 Timer 7	Level	

CM41X_M4 TIMER Interrupt List

Table 19-3: CM41X_M4 TIMER Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
88	TIMER1_TMR0	TIMER1 Timer 0	Level	
89	TIMER1_TMR1	TIMER1 Timer 1	Level	
90	TIMER1_TMR2	TIMER1 Timer 2	Level	
91	TIMER1_TMR3	TIMER1 Timer 3	Level	
92	TIMER1_TMR4	TIMER1 Timer 4	Level	
93	TIMER1_TMR5	TIMER1 Timer 5	Level	
94	TIMER1_TMR6	TIMER1 Timer 6	Level	
95	TIMER1_TMR7	TIMER1 Timer 7	Level	
96	TIMER0_TMR0	TIMER0 Timer 0	Level	
97	TIMER0_TMR1	TIMER0 Timer 1	Level	
98	TIMER0_TMR2	TIMER0 Timer 2	Level	
99	TIMER0_TMR3	TIMER0 Timer 3	Level	
100	TIMER0_TMR4	TIMER0 Timer 4	Level	
101	TIMER0_TMR5	TIMER0 Timer 5	Level	
102	TIMER0_TMR6	TIMER0 Timer 6	Level	
103	TIMER0_TMR7	TIMER0 Timer 7	Level	

Table 19-3: CM41X_M4 TIMER Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
104	TIMER1_STAT	TIMER1 Status	Level	
105	TIMER0_STAT	TIMER0 Status	Level	

CM41X_M0 TIMER Trigger List

Table 19-4: CM41X_M0 TIMER Trigger List Masters

Trigger ID	Name	Description	Sensitivity
4	TIMER0_TMR0_MST	TIMER0 TMR 0 Trigger Master	Edge
5	TIMER0_TMR1_MST	TIMER0 TMR 1 Trigger Master	Edge
6	TIMER0_TMR2_MST	TIMER0 TMR 2 Trigger Master	Edge
7	TIMER0_TMR3_MST	TIMER0 TMR 3 Trigger Master	Edge
8	TIMER0_TMR4_MST	TIMER0 TMR 4 Trigger Master	Edge
9	TIMER0_TMR5_MST	TIMER0 TMR 5 Trigger Master	Edge
10	TIMER0_TMR6_MST	TIMER0 TMR 6 Trigger Master	Edge
11	TIMER0_TMR7_MST	TIMER0 TMR 7 Trigger Master	Edge

Table 19-5: CM41X_M0 TIMER Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
1	TIMER0_TMR0_SLV0	TIMER0 TMR 0 Trigger Slave 0	Pulse
2	TIMER0_TMR0_SLV1	TIMER0 TMR 0 Trigger Slave 1	Pulse
3	TIMER0_TMR1_SLV0	TIMER0 TMR 1 Trigger Slave 0	Pulse
4	TIMER0_TMR1_SLV1	TIMER0 TMR 1 Trigger Slave 1	Pulse
5	TIMER0_TMR2_SLV0	TIMER0 TMR 2 Trigger Slave 0	Pulse
6	TIMER0_TMR2_SLV1	TIMER0 TMR 2 Trigger Slave 1	Pulse
7	TIMER0_TMR3_SLV0	TIMER0 TMR 3 Trigger Slave 0	Pulse
8	TIMER0_TMR3_SLV1	TIMER0 TMR 3 Trigger Slave 1	Pulse
9	TIMER0_TMR4_SLV0	TIMER0 TMR 4 Trigger Slave 0	Pulse
10	TIMER0_TMR4_SLV1	TIMER0 TMR 4 Trigger Slave 1	Pulse
11	TIMER0_TMR5_SLV0	TIMER0 TMR 5 Trigger Slave 0	Pulse
12	TIMER0_TMR5_SLV1	TIMER0 TMR 5 Trigger Slave 1	Pulse
13	TIMER0_TMR6_SLV0	TIMER0 TMR 6 Trigger Slave 0	Pulse

Table 19-5: CM41X_M0 TIMER Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
14	TIMER0_TMR6_SLV1	TIMER0 TMR 6 Trigger Slave 1	Pulse
15	TIMER0_TMR7_SLV0	TIMER0 TMR 7 Trigger Slave 0	Pulse
16	TIMER0_TMR7_SLV1	TIMER0 TMR 7 Trigger Slave 1	Pulse

CM41X_M4 TIMER Trigger List

Table 19-6: CM41X_M4 TIMER Trigger List Masters

Trigger ID	Name	Description	Sensitivity
5	TIMER0_TMR0_MST	TIMER0 TMR 0 Trigger Master	Edge
6	TIMER0_TMR1_MST	TIMER0 TMR 1 Trigger Master	Edge
7	TIMER0_TMR2_MST	TIMER0 TMR 2 Trigger Master	Edge
8	TIMER0_TMR3_MST	TIMER0 TMR 3 Trigger Master	Edge
9	TIMER0_TMR4_MST	TIMER0 TMR 4 Trigger Master	Edge
10	TIMER0_TMR5_MST	TIMER0 TMR 5 Trigger Master	Edge
11	TIMER0_TMR6_MST	TIMER0 TMR 6 Trigger Master	Edge
12	TIMER0_TMR7_MST	TIMER0 TMR 7 Trigger Master	Edge
13	TIMER1_TMR0_MST	TIMER1 TMR 0 Trigger Master	Edge
14	TIMER1_TMR1_MST	TIMER1 TMR 1 Trigger Master	Edge
15	TIMER1_TMR2_MST	TIMER1 TMR 2 Trigger Master	Edge
16	TIMER1_TMR3_MST	TIMER1 TMR 3 Trigger Master	Edge
17	TIMER1_TMR4_MST	TIMER1 TMR 4 Trigger Master	Edge
18	TIMER1_TMR5_MST	TIMER1 TMR 5 Trigger Master	Edge
19	TIMER1_TMR6_MST	TIMER1 TMR 6 Trigger Master	Edge
20	TIMER1_TMR7_MST	TIMER1 TMR 7 Trigger Master	Edge

Table 19-7: CM41X_M4 TIMER Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
3	TIMER1_TMR0_SLV0	TIMER1 TMR 0 Trigger Slave 0	Pulse
4	TIMER1_TMR0_SLV1	TIMER1 TMR 0 Trigger Slave 1	Pulse
5	TIMER1_TMR1_SLV0	TIMER1 TMR 1 Trigger Slave 0	Pulse
6	TIMER1_TMR1_SLV1	TIMER1 TMR 1 Trigger Slave 1	Pulse
7	TIMER1_TMR2_SLV0	TIMER1 TMR 2 Trigger Slave 0	Pulse

Table 19-7: CM41X_M4 TIMER Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
8	TIMER1_TMR2_SLV1	TIMER1 TMR 2 Trigger Slave 1	Pulse
9	TIMER1_TMR3_SLV0	TIMER1 TMR 3 Trigger Slave 0	Pulse
10	TIMER1_TMR3_SLV1	TIMER1 TMR 3 Trigger Slave 1	Pulse
11	TIMER1_TMR4_SLV0	TIMER1 TMR 4 Trigger Slave 0	Pulse
12	TIMER1_TMR4_SLV1	TIMER1 TMR 4 Trigger Slave 1	Pulse
13	TIMER1_TMR5_SLV0	TIMER1 TMR 5 Trigger Slave 0	Pulse
14	TIMER1_TMR5_SLV1	TIMER1 TMR 5 Trigger Slave 1	Pulse
15	TIMER1_TMR6_SLV0	TIMER1 TMR 6 Trigger Slave 0	Pulse
16	TIMER1_TMR6_SLV1	TIMER1 TMR 6 Trigger Slave 1	Pulse
17	TIMER1_TMR7_SLV0	TIMER1 TMR 7 Trigger Slave 0	Pulse
18	TIMER1_TMR7_SLV1	TIMER1 TMR 7 Trigger Slave 1	Pulse
76	TIMER1_ACI1	TIMER1 Alternate Capture Input 1	Pulse
77	TIMER1_ACI7	TIMER1 Alternate Capture Input 7	Pulse

Internal Interface

The processor core always accesses the timer registers through the MMR access bus. Hardware ensures that all read and write operations from and to 32-bit timer registers are atomic. Every timer has a dedicated data interrupt request. There is also one common timer status and error interrupt request output that connects to the system event controller. Whenever a data interrupt request is generated, a data trigger master pulse is also driven out, if enabled. Each timer has an individual trigger input line, and each timer can be either started or stopped as a trigger slave.

In total, the GP timer module can have up to (N + 1) interrupt request output lines and N data trigger lines.

External Interface

Each GP timer module can support up to 16 individual timers. However, most processors have less than this number. The exact number of timers available on a given processor is available in the data sheet for the processor.

Every timer has one main input/output signal (`TIMER_TMR[n]`) and, usually, one auxiliary input pin, used as an alternate capture input (`TIMER_ACI[n]`). Each timer can either run with a time base of SCLK or can reference an external clock on one of two `TIMER_ACLK[n]` pins. The TMR_ALT_CLK0 signal maps to individual alternate clock (`TIMER_ACLK[n]`) pins for one or more timers. For instance, a TM_ACLK3 pin would provide an alternate site to supply an external signal that would serve as reference clock for TMR3. Likewise, the TMR_ALT_CLK1 signal from each timer unit connects together internally to provide a single global timer clock pin (`TIMER_CLK`) for the GP timer module. It is used as an additional time base.

In the ***Timer Alternate Input and Alternate Clock*** table, column 4 is the alternative capture input list and columns 6 and 7 are alternative clock input0 and alternative clock input1, respectively.

Table 19-8: Timer Alternate Input and Alternate Clock

Timer Group	Timer	TMn_TMRm	TMn_ACIm	Peripheral connection	TMn_ACLKm	TMn_ACLKm
			Alternate Capture Input	on same GPIO:	Auxiliary Clock 0	Auxiliary Clock 1
0	0	PA_12	PA_04	UART0_RXb	PA_09	PA_00(TM0_CLK)
0	1	PA_13	PA_06	CAN0_RX	PA_10	PA_00
0	2	unused(0)	PA_01		PA_07	PA_00
0	3	unused(0)	PA_02		PA_08	PA_00
0	4	unused(0)	PA_03		PA_05	PA_00
0	5	unused(0)	M0 Trigger SlaveTIMER0_ACI5		AUXCLK (OSCWD 1Mhz)	PA_00
0	6	AUXCLK (OSCWD 1Mhz)	M0 Trigger SlaveTIMER0_ACI6		SYS_CLKIN0	PA_00
0	7	unused(0)	M0 Trigger Slave TIMER0_ACI7		SYS_CLKIN1	PA_00
1	0	PE_14, PB_14	PE_12	CAN1_RX	SYS_CLKIN0	PC_06(TM1_CLK)
1	1	PE_15, PB_15	M4 Trigger Slave TIMER1_ACI1		SYS_CLKIN0	PC_06
1	2	PB_13	PC_08	UART3_RXb	AUXCLK (OSCWD 1Mhz)	PC_06
1	3	PC_10	CNT_TO		PC_15	PC_06
1	4	PE_04	PC_09	UART2_RXb	PC_00	PC_06
1	5	PF_06	PE_09	UART1_RXb	PE_08	PC_06
1	6	PE_02	PF_05	UART4_RXb	SYS_CLKIN0	PC_06
1	7	PC_12	TIMER1_ACI7 M4 Trigger Slave		SYS_CLKIN1	PC_06

GP Timer Operating Modes

The following sections provide information on the various operating modes of the GP timer.

General Operation

The core of every timer is a 32-bit counter that can be interrogated through the read-only `TIMER_TMR[n]_CNT` register. Once the module enables a timer, it loads the timer `TIMER_TMR[n]_CNT` register with a starting value.

A timer can operate in one of several different modes, configured through the TIMER_TMR[n]_CFG register for that timer. These modes are: PWMOUT, EXTCLK, WIDCAP, WATCHDOG, PININT, and IDLE. The ***Timer Mode Descriptions*** table summarizes the modes.

Table 19-9: Timer Mode Descriptions

Timer Mode	Description
PWMOUT	Generates single or continuous PWM waveforms with programmable pulse width, period, and delay
EXTCLK	Counts edges of an externally applied waveform
WIDCAP	Captures pulse width or period of an externally applied waveform
WATCHDOG	Monitors pulse width or period of an external signal and compares against a window of acceptable values, optionally generating an interrupt when it falls inside or outside of that window
PININT	Can generate an interrupt request on an active edge applied to a timer pin
IDLE	Idle; no activity

Period, Width and Delay Register Interaction

When the timer is started, writes to the buffer registers are immediately copied through to the double-buffered period, pulse width, and delay registers. These values are then ready for use in the first timer period. When a timer is already running, software can write new values to the TIMER_TMR[n]_PER, TIMER_TMR[n]_WID, and TIMER_TMR[n]_DLY registers. The written values are buffered and do not update into the registers until the end of the current period. (The update occurs when the value in the TIMER_TMR[n]_CNT register equals the value in the TIMER_TMR[n]_PER register.)

If new values are not written to these registers, the value from the previous period is reused. Writes to these registers are atomic; it is not possible for the high word to be written without the low word also being written. Values written to the period, pulse width, and delay registers are always stored in the buffer registers. Reads from the same register always return the current, active value of period, pulse width, or delay value. Written values are not readback until they become active.

The usage of the TIMER_TMR[n]_PER, TIMER_TMR[n]_WID, and TIMER_TMR[n]_DLY registers varies, depending on the mode of the timer specified by the TIMER_TMR[n]_CFG.TMODE bits. See the ***Usage of the Period, Width, and Delay Registers in Different Timer Modes*** table for more information.

Table 19-10: Usage of the Period, Width, and Delay Registers in Different Timer Modes

Timer Mode	**TIMER_TMR[n]_PER**	**TIMER_TMR[n]_WID**	**TIMER_TMR[n]_DLY**
IDLE	Not writable	Not writable	Not writable
WATCHDOG	Update on-the-fly. New value takes effect either at timer start or when an asserting edge on the input signal is sensed.	Read-only. Retains value of last measured width or period of the input signal.	Update on-the-fly. New value takes effect either at timer start or when an asserting edge on the input signal is sensed.

Table 19-10: Usage of the Period, Width, and Delay Registers in Different Timer Modes (Continued)

Timer Mode	TIMER_TMR[n]_PER	TIMER_TMR[n]_WID	TIMER_TMR[n]_DLY
WIDCAP	Read-only. Period value captured at the appropriate time and updated from its buffer register simultaneously with the Width register.	Read-only. Width value captured at the appropriate time and updated from its buffer register simultaneously with the Period register.	Not used
PWMOUT	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Update on-the-fly. New value takes effect either at timer start or at the end of the current period. A write followed by immediate read returns the current operating values.
EXTCLK	Can be updated on-the-fly.	Not used	Not used
PININT	Not used	Not used	Not used

If any of the period, pulse width, and delay registers are not used, then programs cannot write into that register. For example, in WIDCAP mode, the delay registers are not used. So, the program is not allowed to write any value to the TIMER_TMR[n]_DLY register. To prevent undesired operation, program the TIMER_TMR[n]_CFG.TMODE bits before programming the period, width, or delay registers.

If a program changes the TIMER_TMR[n]_CFG.TMODE bits from a status register to writable register (for example in PWMOUT mode), hardware does not clear these registers. These values are automatically overwritten by new values specified by software.

In PWMOUT mode with small periods, there may not be enough time between updates from the buffer registers to write these registers. The next period can use one old value and one new value. To prevent (width + pulse delay) > period errors, write the width and delay registers before the period register when decreasing the values. Write the period register before the width and delay registers when increasing the value.

Single-Pulse PWMOUT Mode

In single-pulse PWMOUT mode, the timer generates a single pulse on the TIMER_TMR[n] pin. This mode is frequently used to implement a precise delay, often with generating an output trigger. The timer module uses the value in the TIMER_TMR[n]_DLY register to control the assertion of a pulse. The value in the TIMER_TMR[n]_WID register defines the pulse width. The TIMER_TMR[n]_PER is not used and cannot be written in this mode. After completion of the pulse, the timer is automatically stopped, and optionally generates an interrupt. The timer uses the TIMER_TMR[n]_CFG.PULSEHI bit to control pulse polarity.

The timer can be configured to generate a data interrupt request after satisfying various conditions specified by the TIMER_TMR[n]_CFG.IRQMODE bits.

It is not necessary to clear the relevant TIMER_RUN bit to stop the timer cleanly. At the end of the pulse, the timer stops automatically and the corresponding TIMER_RUN bit is cleared. To generate multiple discrete pulses (as opposed to a continuous PWM waveform), write a 1 to the appropriate TIMER_RUN bit, and wait for the timer to stop. Then, write another 1 to the same TIMER_RUN bit.

Continuous PWMOUT Mode

In continuous PWMOUT mode, the timer generates a repetitive pulse with a well-defined period, duty cycle, and pulse position. The TIMER_TMR[n]_DLY, TIMER_TMR[n]_PER, and TIMER_TMR[n]_WID registers are programmed with the values of the required PWM pulse. After the timer is started, the counter counts towards the value programmed in the TIMER_TMR[n]_PER register. Initially, the TIMER_TMR[n] pin remains in a deasserted state. The pin toggles to an asserted state when the value in the TIMER_TMR[n]_CNT register equals the value in the TIMER_TMR[n]_DLY register.

The timer can control the assertion sense of the TIMER_TMR[n] pin with the TIMER_TMR[n]_CFG.PULSEHI bit. The TIMER_TMR[n] pin holds this value for the number of clock cycles specified in the TIMER_TMR[n]_WID register. Then, the pin deasserts and holds this value until the completion of the programmed period. The same waveform is generated repeatedly until the timer is disabled.

The timer can be configured to generate a data interrupt request after satisfying any of various conditions specified by the TIMER_TMR[n]_CFG.IRQMODE bits.

It is important to guarantee that the programmed period is greater than or equal to the sum of width and delay. Similarly, delay must be less than period. Violating either of these criteria results in an unpredictable waveform on the TIMER_TMR[n] pin until the situation is rectified by writing proper values to these registers.

The maximum frequency possible to generate on the TIMER_TMR[n] pin is achieved by setting TIMER_TMR[n]_PER to 2 and TIMER_TMR[n]_WID to 1. This operation makes the TIMER_TMR[n] pin toggle each SCLK clock cycle (assuming the timer is configured to clock internally), producing a duty cycle of 50%.

When the TIMER_STOP_CFG.TMR[nn] bit of a timer is 0, the timer treats a stop operation as a stop-is-pending condition. When terminated with this setting, the timer automatically completes the current waveform and then stops cleanly, remaining in a deasserted state. This funtionality prevents truncation of the current pulse and unwanted PWM patterns at the TIMER_TMR[n] pin. The processor can determine when the timer stops running by polling the corresponding TIMER_RUN.TMR[nn] bit until it reads 0 or by waiting for the last interrupt (if enabled).

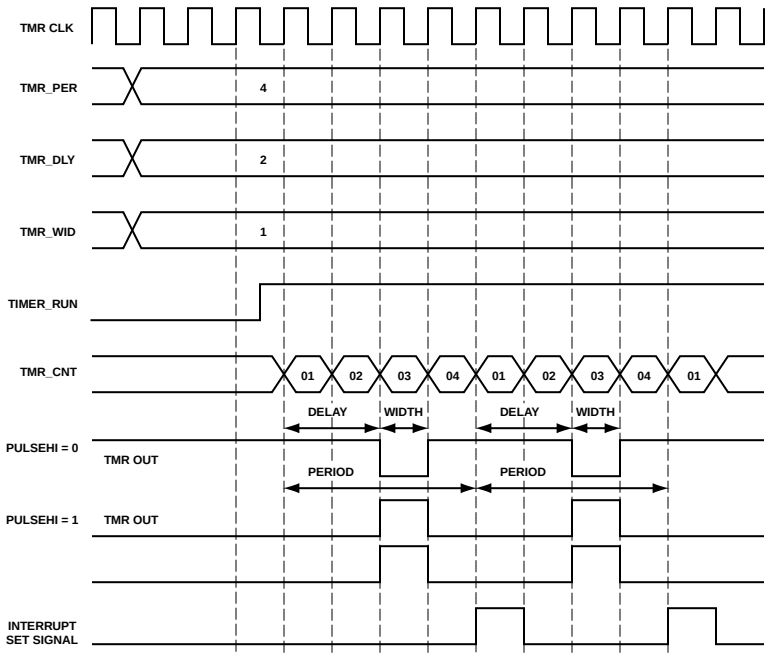


Figure 19-1: Signal Generation in Continuous PWMOUT Mode

The TIMER_TMR[n]_CFG register cannot be reconfigured until after the timer stops and the TIMER_RUN register reads 0.

Programs can force a timer to stop immediately in PWMOUT mode by writing a 1 to the TIMER_STOP_CFG register followed by writing a 1 to the TIMER_RUN_CLR register. (Or, a program can stop a timer by writing a 0 to the appropriate TIMER_RUN.TMR[nn] bit.) This operation stops the timer whether the pending stop is waiting for the end of the current period or the end of the current pulse width. The timer can use this feature to regain immediate control of a timer during an error recovery sequence.

Use this feature carefully, as it can corrupt the PWM pattern generated at the TIMER_TMR[n] pin, though after such a stop the pin deasserts automatically. Each timer samples its TIMER_RUN.TMR[nn] bit at the end of each period. It stops cleanly at the end of the first period after the TIMER_RUN.TMR[nn] bit is low. A timer that is disabled and then restarted (before the end of the current period), continues to run as if nothing happened. Typically, the program disables a PWMOUT timer and then waits for it to stop itself.

Width Capture (WIDCAP) Mode

The timer uses WIDCAP mode, often called capture mode, to measure pulse widths on the TIMER_TMR[n] or TIMER_ACI[n] inputs. The polarity (active high or low) of the input signal can be selected with the TIMER_TMR[n]_CFG.PULSEHI bit. The ***Timer Signal Flow in Width Capture Mode*** figure shows the control signal flow for WIDCAP_CAP mode.

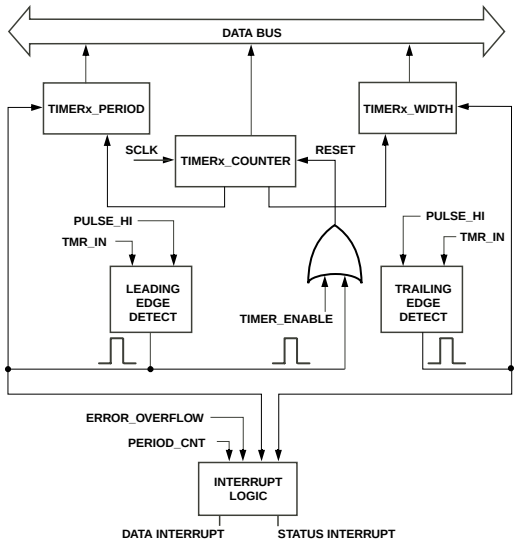


Figure 19-2: Timer Signal Flow in Width Capture Mode

NOTE: SCLK in the ***Timer Signal Flow in Width Capture Mode*** figure is SCLK.

In this mode, the timer uses the `TIMER_TMR[n]_CFG.TINSEL` bit to select between the `TIMER_TMR[n]` or `TIMER_ACI[n]` input. The internally clocked timer is used to determine the period and pulse width of the externally applied rectangular waveforms.

NOTE: In WIDCAP_CAP mode, when TMR_AUX_IN input is selected, a number of the timers sense internal signals from the GP counter-module through their alternate input. (These internal signals appear as "TO GP TIMER TMR_AUX_IN (IF ENABLED)" in the Figure 22-1 GP Counter Block Diagram). This feature lets the timer capture the period between counter-events and supports autobaud detection for the UART and the CAN modules.

When a timer is enabled in this mode, the timer resets the count in its `TIMER_TMR[n]_CNT` register to 0x0000 0001. It does not start counting until it detects a leading edge on the selected input pin.

When the timer detects the first leading edge, it starts incrementing. When it detects a trailing edge of a waveform, it captures the current 32-bit value of its `TIMER_TMR[n]_CNT` register into its width buffer register. At the next leading edge, the timer transfers the current 32-bit value of its `TIMER_TMR[n]_CNT` register into its period buffer register. The `TIMER_TMR[n]_CNT` register is reset to 0x0000 0001 again, and the timer continues counting and capturing until it is disabled.

In this mode, programs can measure both the pulse width and the pulse period of a waveform. The timer does not use the `TIMER_TMR[n]_DLY` register in this mode. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to control the definition of leading edge and trailing edge of the `TIMER_TMR[n]/TIMER_ACI[n]` pin.

In WIDCAP mode, the following events always occur at the same time as one unit:

1. The `TIMER_TMR[n]_PER` register is updated from the period buffer register.
2. The `TIMER_TMR[n]_WID` register is updated from the width buffer register.

3. The TIMER_DATA_ILAT.TMR[nn] bit is set (if enabled).
4. A timer data trigger pulse is generated (if enabled).

The TIMER_TMR[n]_CFG.TMODE bit 0 controls the point in time at which this set of events is executed. Taken together, these four events are called a measurement report. The TIMER_STAT_ILAT register is not set at a measurement report. A measurement report occurs, at most, once per input signal period. The current TIMER_TMR[n]_CNT value is always copied to the width buffer and period buffer registers at the trailing and leading edges of the input signal, respectively. But, these values are not visible to software. A measurement report event samples the captured values into visible registers and sets the timer interrupt request to signal that the TIMER_TMR[n]_PER and the TIMER_TMR[n]_WID registers are ready to be read.

When the TIMER_TMR[n]_CFG.TMODE bit =b#1011, the measurement report occurs just after the width buffer register captures its value at a falling edge. Then, the TIMER_TMR[n]_WID register reports the pulse width measured in the pulse that has ended, but the TIMER_TMR[n]_PER register reports the pulse period measured at the end of the previous period. If only the first trailing edge has occurred, then the first period value has not yet been measured at the first measurement report. So, the period value is not valid. A read of the TIMER_TMR[n]_PER value in this case returns 0. See the ***Example of Width Capture Deasserted Mode (TMODE=b#1011)*** figure for more information.

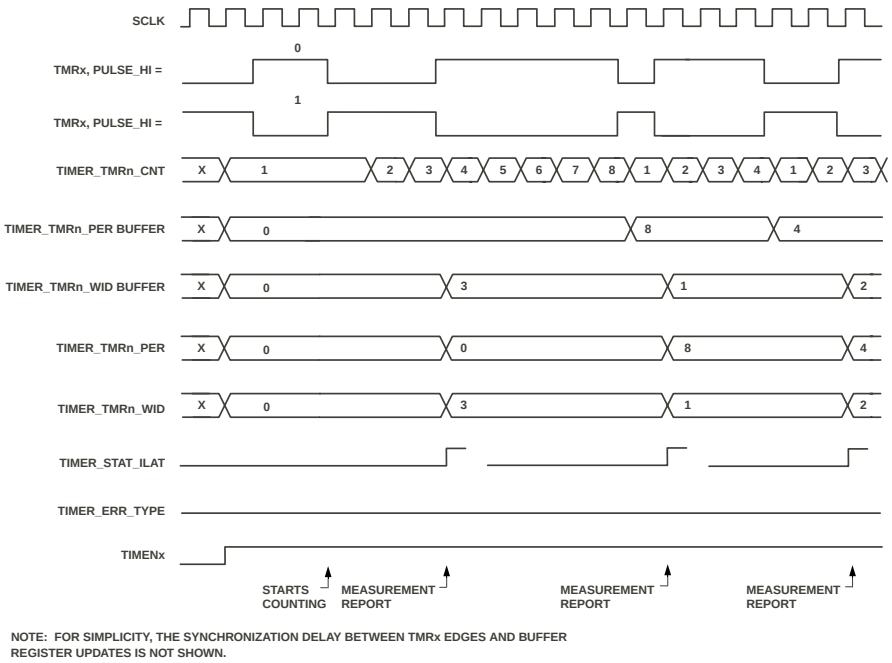


Figure 19-3: Example of Width Capture Deasserted Mode (TMODE=b#1011)

NOTE: SCLK in the ***Example of Width Capture Deasserted Mode (TMODE=b#1011)*** figure is SCLK.

When the TIMER_TMR[n]_CFG.TMODE bit =b#1010, the measurement report occurs just after the period buffer register captures its value at a leading edge. Then, the TIMER_TMR[n]_PER and TIMER_TMR[n]_WID

registers report the pulse period and pulse width measured in the period that has ended. Refer to the ***Example of Width Capture Asserted Mode (TMODE=b#1010)*** figure for more information.

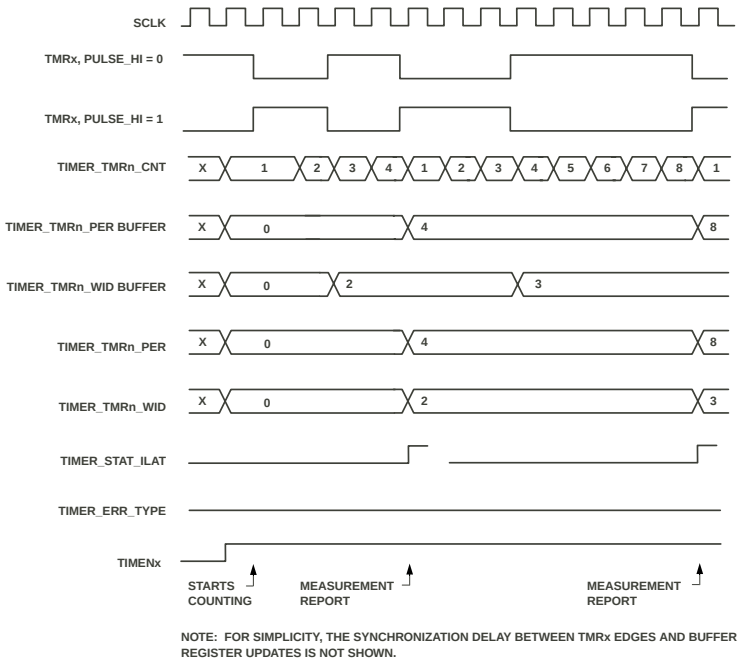


Figure 19-4: Example of Width Capture Asserted Mode (TMODE=b#1010)

NOTE: SCLK in the ***Example of Width Capture Asserted Mode (TMODE=b#1010)*** figure is SCLK.

To measure the pulse width of a waveform that has only one leading edge and one trailing edge, set `TIMER_TMR[n]_CFG.TMODE` = b#1011. If `TIMER_TMR[n]_CFG.TMODE` = b#1010 for this case, no period value is captured in the period buffer register. Instead, the timer genererates an error report interrupt request (if enabled) when the `TIMER_TMR[n]_CNT` range is exceeded and the counter wraps around. In this case, both the `TIMER_TMR[n]_PER` and `TIMER_TMR[n]_WID` registers read 0 (because no measurement report occurred to copy the value captured in the width buffer register to the `TIMER_TMR[n]_WID` register).

If using the `TIMER_TMR[n]_CFG.TMODE` bit =b#1010 mode to measure the width of a single pulse, programs can disable the timer after taking the interrupt that ends the measurement interval. If desired, restart the timer as appropriate in preparation for another measurement. This procedure prevents the timer from free-running after the width measurement and logging errors generated by the timer count overflowing.

Width Capture Mode Overflow

A timer status interrupt request (when enabled) is generated when the `TIMER_TMR[n]_CNT` register wraps around from 0xFFFF FFFF to 0 in the absence of a leading edge. At that point, the `TIMER_STAT_ILAT` bit is set and the `TIMER_ERR_TYPE` bits change to indicate a count overflow due to a period greater than the range of the counter. This indication is referred to as an error report. A data interrupt request in WIDCAP mode indicates that a new measurement is ready to be read (a measurement report). Similarly, an interrupt request on the timer status interrupt line (shared interrupt request for all timers) indicates an overflow error when generated in this mode.

The TIMER_TMR[n]_PER and TIMER_TMR[n]_WID registers are never updated at the time an overflow error is signaled. If the timer overflows and the TIMER_TMR[n]_CFG.TMODE bit =b#1010, the TIMER_TMR[n]_PER and TIMER_TMR[n]_WID registers are not updated. If the timer overflows and the TIMER_TMR[n]_CFG.TMODE bit =b#1011, the TIMER_TMR[n]_PER and TIMER_TMR[n]_WID registers are updated only if a trailing edge is detected at a previous measurement report.

Software can count the number of error reports between measurement report interrupt requests to measure input signal periods longer than 0xFFFF FFFF. Each error report interrupt request adds a full 2^{32}SCLK counts to the total for the period, but the width is ambiguous. Ensure that if software monitors only the status interrupt request, then status interrupt requests from all other timers are masked.

Refer to the ***Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)*** figure. The period is 0x1 0000 0004, but the pulse width could be either 0x0 0000 0002 or 0x1 0000 0002.

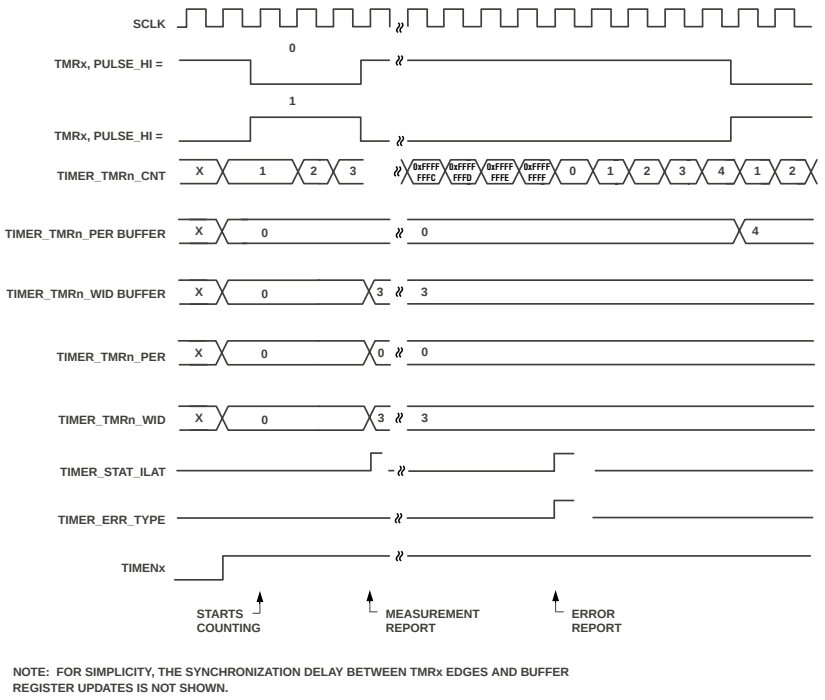


Figure 19-5: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)

NOTE: SCLK in the ***Example Timing for Width Capture Followed by Period Overflow*** figure is SCLK.

The waveform applied to the TIMER_TMR[n] (or TIMER_ACI[n]) pin is not required to have a 50% duty cycle. The minimum input low time is little more than one SCLK period. The minimum input high time is a little more than one SCLK period. (Refer to the product data sheet for details.) The maximum TIMER_TMR[n] input frequency is less than SCLK/2, with a 50% duty cycle. Under these conditions, the WIDCAP mode timer measures: period =2 and pulse width =1.

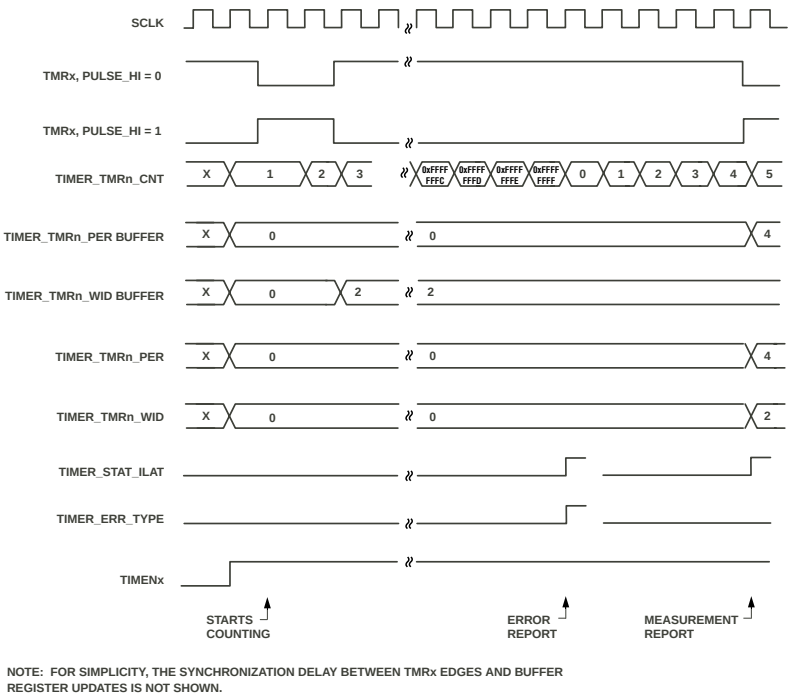


Figure 19-6: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1011)

NOTE: SCLK in the ***Example Timing for Width Capture Followed by Period Overflow*** figure is SCLK.

Windowed Watchdog (WATCHDOG) Modes

In windowed watchdog (WATCHDOG) modes, a timer can take inputs from either the `TIMER_TMR[n]` pin or the `TIMER_ACI[n]` pin. With this mode, the timer can monitor pulse width (width watchdog mode) or pulse period (period watchdog mode) on the input line. It also compares the measured value against a minimum required value and maximum allowed value and generates an interrupt request appropriately. The timer uses the `TIMER_TMR[n]_CFG.PULSEHI` bit to select polarity of the input signal.

The waveform applied to the input pin in watchdog mode is not required to have a 50% duty cycle. The minimum input pulse low time, high time, and total period specifications are available in the product data sheet.

Windowed Watchdog Width Mode

In windowed watchdog width mode, the timer counter monitors the pulse width of an input signal on either the `TIMER_TMR[n]` pin or one of the alternate clock pins (`TIMER_ACLK[n]`). Program the minimum pulse width (p_{MIN}) in the `TIMER_TMR[n]_DLY` register and the maximum pulse width (p_{MAX}) in the `TIMER_TMR[n]_PER` register. Both values are programmed in terms of number of clock cycles (SCLK or alternate clock). The timer can generate an interrupt if the deasserting pulse edge occurs:

- Inside the window (p_{MIN} < pulse width ≤ p_{MAX}), or
- Outside the window (pulse width ≤ p_{MIN} or pulse width > p_{MAX})

After enabling the timer in this mode, it always starts counting at the asserting edge of the input signal. Any pulse that is already active when the timer is enabled is ignored.

With the TIMER_TMR[n]_CFG.IRQMODE bit =b#11, the timer generates an interrupt if the timed pulse width exceeds p_{MAX}, or if the pulse width is less than p_{MIN}. After attaining p_{MAX}, the pulse stays at an active level, and the counter keeps on counting until it sees a deasserting edge. When the input pulse is not active, the counter holds its current value until it again sees an asserting edge, or it restarts. An interrupt can also be generated for when the pulse occurs within the specified window condition, by setting TIMER_TMR[n]_CFG.IRQMODE =b#10.

In this mode, a trailing edge on the input pin triggers capturing of pulse width into the TIMER_TMR[n]_WID register. During the inactive portion of the input signal, the internal counter does not increment. The ***Watchdog Width Mode Timing*** figure shows the signal flow in this mode.

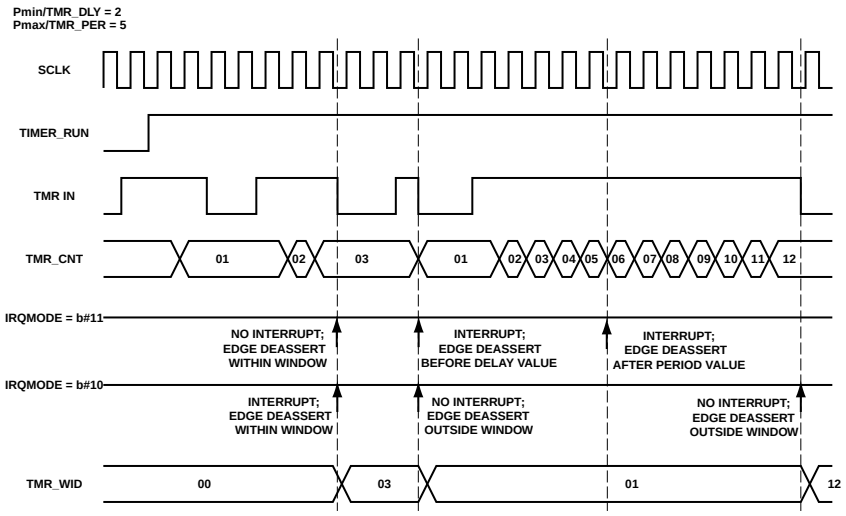


Figure 19-7: Watchdog Width Mode Timing

NOTE: SCLK in the ***Watchdog Width Mode Timing*** figure is SCLK.

To check only the upper limit on pulse width (p_{MAX} but not p_{MIN}), then program p_{MIN} as 0 or 1. In such a case, it is better to use TIMER_TMR[n]_CFG.IRQMODE =b#11. With TIMER_TMR[n]_CFG.IRQMODE = b#10, a pulse width of 1 clock cycle results in an interrupt. For details, see the ***Windowed Watchdog Width Mode Interpretation*** table.

Table 19-11: Windowed Watchdog Width Mode Interpretation

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE= b#10	IRQMODE= b#11	Error Interrupt?
0 or 1	Anything ≥ 1	PW = 1	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW ≤ TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) Value	No Error Interrupt
> 1 but ≤ (Period -1)	Anything > 1	PW ≤ TMR_DLY	No Interrupt	Interrupt at deasserting edge of input signal	No Error Interrupt
		TMR_DLY < PW≤ TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		PW > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) Value	No Error Interrupt
≥ Period	-	PW ≤ TMR_PER	Undefined	Undefined	No Error Interrupt
	-	PW > TMR_PER	Undefined	Undefined	b#11 Error Type
-	0	-	Undefined	Undefined	b#10 Error Type

Windowed Watchdog Period Mode

In this mode, the timer monitors the number of clock cycles between two consecutive rising or falling edges of an input signal on either the `TIMER_TMR[n]` or `TIMER_ACI[n]` pin. Program the required minimum number of clock cycles (t_{MIN}) in the `TIMER_TMR[n]_DLY` register and the required maximum allowed number of clock cycles (t_{MAX}) in the `TIMER_TMR[n]_PER` register. Both values are programmed in terms of number of clock cycles (SCLK) or alternate time clock (`TIMER_ACLK[n]`). The timer can generate an interrupt when two consecutive occurrences of an active edge are:

- Within a specified window (t_{MIN} < Pulse Period ≤ t_{MAX}), or
- Outside a specified window (pulse width ≤ (t_{MIN} or t_{MAX} < pulse width)

When the `TIMER_TMR[n]_CFG.IRQMODE` bit =b#11 and the pulse period > t_{MAX} or is ≤ t_{MIN}, the timer generates an interrupt (if unmasked). After attaining the t_{MAX} value, the counter keeps on counting until it sees an active edge on the input line. An interrupt can also be generated for when the pulse occurs within the specified

window condition, by setting `TIMER_TMR[n]_CFG.IRQMODE` =b#10. Refer to the ***Watchdog Period Mode Timing*** figure for timer functionality in period watchdog mode.

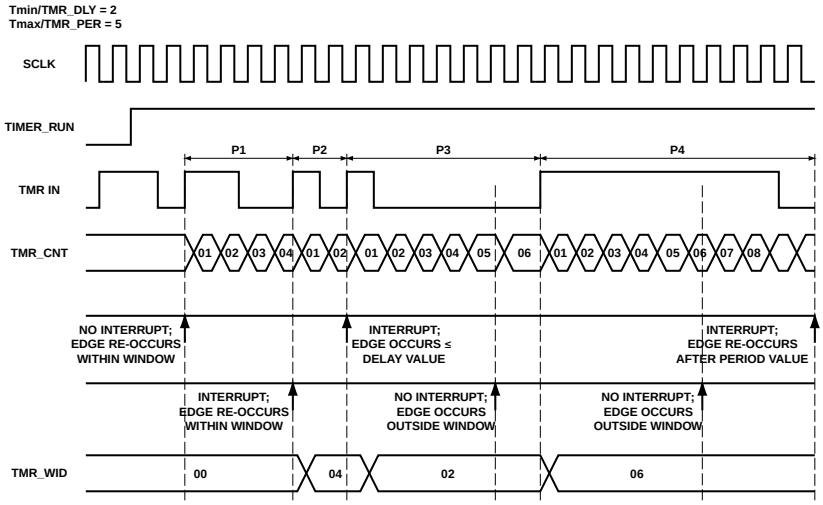


Figure 19-8: Watchdog Period Mode Timing

NOTE: SCLK in the ***Watchdog Period Mode Timing*** figure is SCLK.

To check only the upper limit on period (the t_{MAX} value, not the t_{MIN} value), program t_{MIN} as 0 or 1. For details, refer to the ***Windowed Watchdog Period Mode Interpretation*** table.

Table 19-12: Windowed Watchdog Period Mode Interpretation

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE=b#10	IRQMODE =b#11	Error Interrupt?
0 or 1	Anything ≥ 2	Pulse Period ≤ TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		Pulse Period > TMR_PER	No Interrupt	Interrupt when pulse period crosses Pmax (Period Register) value	No Error Interrupt
> 1 but ≤ Period -1	Anything ≥ 2	Pulse Period ≤ TMR_DLY	No Interrupt	Interrupt at deasserting edge of input signal	No Error Interrupt
		TMR_DLY < Pulse Period ≤TMR_PER	Interrupt at deasserting edge of input signal	No Interrupt	No Error Interrupt
		Pulse Period > TMR_PER	No Interrupt	Interrupt when pulse width exceeds Pmax (Period Register) value	No Error Interrupt

Table 19-12: Windowed Watchdog Period Mode Interpretation (Continued)

Timer Delay	Timer Period	Incoming Pulse Width	IRQMODE=b#10	IRQMODE =b#11	Error Interrupt?
≥ Period	-	Pulse Period < TMR_PER	Undefined	Undefined	No Error Interrupt
		Pulse Period ≥ TMR_PER	Undefined	Undefined	b#11 Error Type
-	0 or 1	-	Undefined	Undefined	b#10 Error Type

Pin Interrupt (PININT) Mode

In PININT mode, any active edges on either the TIMER_TMR[n] pin or the TIMER_ACI[n] pin can cause an edge-based interrupt, if enabled. (The timer uses the TIMER_TMR[n]_CFG.TINSEL register to select the pin). The event on the input pin can set the TIMER_DATA_ILAT.TMR[nn] bit and issue a system interrupt request. Program the TIMER_TMR[n]_CFG.PULSEHI bit to change active edge polarity.

Since the interrupt request is generated in the SCLK clock domain, the width of the input signal must be more than one SCLK period. Along with generating the interrupt request, the timer also generates a trigger pulse (configured using the TIMER_TRG_MSK register). Due to the configuration of polarity, glitches can cause the generation of an undesired interrupt request at the input. To avoid this problem, programs must ensure that interrupt requests are unmasked only after configuring the desired polarity.

External Clock (EXTCLK) Mode

The timer uses EXTCLK mode, sometimes referred to as the counter mode, to count external events (signal edges), on either the TIMER_TMR[n] or TIMER_ACI[n] input pin. The timer works as a counter clocked by an external source (the signal at the pin), which can be asynchronous to SCLK. The current count in the TIMER_TMR[n]_CNT register represents the number of leading-edge events detected. The TIMER_TMR[n]_PER register is programmed with the value of the maximum timer external count before stopping or issuing an interrupt request or trigger.

The TIMER_TMR[n]_CFG.PULSEHI bit determines the polarity of the leading edge on the input pin. The timer uses the TIMER_TMR[n]_CFG.TINSEL bit to select whether the event is counted on the TIMER_TMR[n] or on the TIMER_ACI[n] pin. The TIMER_STAT_ILAT.TMR[nn] and TIMER_ERR_TYPE bits are set if *one* of these conditions is met:

- TIMER_TMR[n]_CNT wraps around from 0xFFFF FFFF to 0
- The period = 0 at startup
- TIMER_TMR[n]_CNT register rolls over (from count = period to count = 0x1)

The TIMER_TMR[n]_WID and TIMER_TMR[n]_DLY registers are unused in this mode and must not be written.

The ***EXTCLK Mode Control Flow*** figure shows a flow diagram for EXTCLK mode.

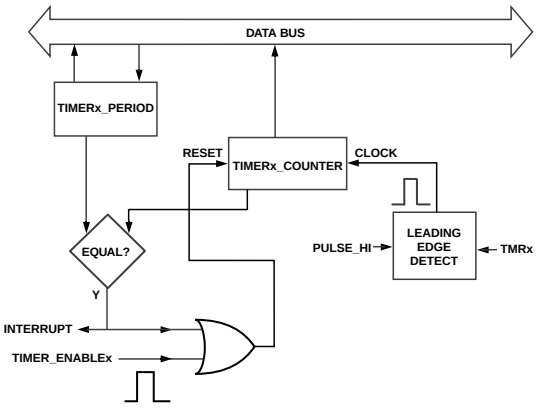


Figure 19-9: EXTCLK Mode Control Flow

The waveform applied to the input pin is not required to have a 50% duty cycle. The minimum input pulse low time, high time, and total period specifications are available in the product data sheet. Program the period to any value from 1 to (2^{32} – 1), inclusive.

After the timer has started, it resets the `TIMER_TMR[n]_CNT` register to 0x0 and then waits for the first leading edge on the input pin. This edge causes `TIMER_TMR[n]_CNT` to be incremented to the value 0x1, and every subsequent leading edge increments it by one. After the `TIMER_TMR[n]_CNT` register reaches the value programmed in the `TIMER_TMR[n]_PER` register, the corresponding `TIMER_DATA_ILAT` bit is set, and an interrupt and trigger are both generated (if enabled). The next leading-edge reloads the `TIMER_TMR[n]_CNT` register with 0x1, and the timer continues counting until it is disabled.

GP Timer Programming Concepts

Using the features, operating modes, and event control for the GP timer to their greatest potential requires an understanding of some GP timer-related concepts.

Setting Up Constantly Changing Timer Conditions

This task shows how to use different period, pulse width, and delay settings for each of the first three timer periods after the timer starts.

1. Program the first set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values.
2. Enable the timer using the `TIMER_RUN` register.
3. Immediately program the second set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values, as needed.
4. Wait for the first timer interrupt request.
5. Program the third set of `TIMER_TMR[n]_PER`, `TIMER_TMR[n]_WID`, and `TIMER_TMR[n]_DLY` register values.

Each new setting is then programmed when the preceding timer interrupt request is received.

Configuring, Enabling, and Disabling One or More Timers

1. Configure the relevant timers for the operating mode and other properties using the TIMER_TMR[n]_CFG register.
2. Write a 1 to the representative TIMER_RUN.TMR[nn] bit. Or, use the TIMER_RUN_SET register to avoid disturbing the settings of other timers that are not going through configuration.

   The timer is enabled and operating.
3. To stop one or more timers, first program the TIMER_STOP_CFG register to determine whether to stop immediately or gracefully upon receiving a stop command.

   ADDITIONAL INFORMATION: PWMOUT modes are the only modes where a timer can be configured for graceful termination.
4. Write a 0 to the representative TIMER_RUN.TMR[nn] bits to stop the timer according to their TIMER_STOP_CFG settings. Alternately, write a 1 to the appropriate TIMER_RUN_CLR.TMR[nn] bits to avoid disturbing the settings of other timers that are not terminating.

   The timers stop.

Configuring Timer Data and Status Interrupts

1. Configure the TIMER_TMR[n]_CFG.IRQMODE bit field with the desired interrupt properties.
2. Unmask the interrupt source.
3. Set the TIMER_TMR[n]_CFG.IRQMODE field but leave the interrupt masked at the system level to poll the TIMER_DATA_ILAT.TMR[nn] bit of the timer without generating an interrupt.
4. Use the TIMER_STAT_IMSK register to generate interrupt requests by overflow or error conditions (incorrect programming values). The timer uses the TIMER_STAT_ILAT.TMR[nn] bits to report interrupt errors, when the timer status interrupt source is unmasked.
5. To poll the TIMER_STAT_ILAT.TMR[nn] bit of the timer without generating an interrupt, unmask the corresponding bit in the TIMER_STAT_IMSK register, but leave the interrupt masked at the system level.

Configuring the Timer as a Trigger Slave

The timer can be configured to either start or stop or toggle between these two states on the input trigger pulse depending on the configuration of the TIMER_TMR[n]_CFG.TGLTRIG and TIMER_TMR[n]_CFG.SLAVETRIG bits.

- If TIMER_TMR[n]_CFG.TGLTRIG bit =0 and TIMER_TMR[n]_CFG.SLAVETRIG bit =1 then the trigger pulse starts timer, if it is stopped.

- If TIMER_TMR[n]_CFG.TGLTRIG bit =0 and TIMER_TMR[n]_CFG.SLAVETRIG bit =0 then the trigger pulse stops timer, if it is running.
- When TIMER_TMR[n]_CFG.TGLTRIG bit =0, the trigger pulse has no effect when the timer is already in the requested state.

If TIMER_TMR[n]_CFG.TGLTRIG bit is 1, the trigger pulse starts the timer if it is stopped, or stops the timer if it is running. The TIMER_TMR[n]_CFG.SLAVETRIG bit has no effect on trigger mechanism. In continuous PWMOUT mode, the timer stops gracefully or abruptly depending on the stop mechanism programmed in the TIMER_STOP_CFG_CLR register. In other modes, the timer stops immediately.

Using the Timer Broadcast Feature

The broadcast feature provides a means to update period, width, and delay registers simultaneously across more than one timer.

Timer triggers TIMER_TMR[n]_SLV0 and TIMER_TMR[n]_SLV1 are logically OR'd so each individual timer can have two input triggers.

1. Enable the appropriate broadcast bits (TIMER_TMR[n]_CFG.BPEREN, TIMER_TMR[n]_CFG.BWIDEN are TIMER_TMR[n]_CFG.BDLYEN) for the timers involved in the broadcast. The use of these bits depends on which broadcast registers the timer uses (TIMER_BCAST_PER, TIMER_BCAST_WID, or TIMER_BCAST_DLY).
2. Program the TIMER_BCAST_PER register (for example), to broadcast the period setting across the multiple timers enabled.

   The enabled timers load their TIMER_TMR[n]_PER registers with the value specified in the TIMER_BCAST_PER register.
3. Repeat Step 2 as needed for the TIMER_BCAST_WID and TIMER_BCAST_DLY register settings.

Timer Illegal States

The following sections use these definitions:

- Startup. The first clock period during which the timer counter is running after the timer is started by writing the TIMER_RUN register.
- Rollover. The time when the current count in TIMER_TMR[n]_CNT matches the value in TIMER_TMR[n]_PER and the counter is reloaded with the value 1.
- Overflow. The timer counter was incremented instead of doing a rollover when it was holding the maximum count value of 0xFFFF FFFF. The counter does not have a large enough range to express the next greater value and so it erroneously loads a new value of 0x0000 0000.
- Unchanged. No new error.

When the TIMER_ERR_TYPE register is designated unchanged, it displays the previously reported error code orb# 00 when there has been no error since this timer was enabled.

When the TIMER_STAT_ILAT register is unchanged, it reads 0 when there has been no error or overflow since this timer was enabled. Or, it reads 0 if software has performed a W1C to clear any previous error. If software has not acknowledged a previous error, the TIMER_STAT_ILAT register reads 1. Software can read the TIMER_STAT_ILAT register to check for errors. If a particular bit of a timer is set in this register, software can then read the TIMER_ERR_TYPE register for more information. Once detected, software can W1C the appropriate TIMER_STAT_ILAT bit to acknowledge the error.

Read the following tables as:

- In mode __ at event __,
- if TIMER_TMR[n]_PER is __ and TIMER_TMR[n]_WID is __ and TIMER_TMR[n]_DLY is __,
- then TIMER_ERR_TYPE is __ and TIMER_STAT_ILAT is __.

Startup error conditions do not prevent the timer from starting. Similarly, overflow and rollover error conditions do not stop the timer. Illegal cases can cause unwanted behavior of the TIMER_TMR[n] pin.

NOTE: For PININT mode, the timer does not use error functionality.

Continuous PWMOUT Mode

Table 19-13: Startup Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ 1	Anything other than period[8]	Anything	Anything	b#10	Set
≥ 2	Anything including 0, excluding TMR_PER value	Anything including 0	≤ PERIOD	Unchanged	Unchanged
	Anything including 0	Anything including 0	> PERIOD	Unchanged[9] (Detected at rollover)	Unchanged (Detected at rollover)
	Anything	Anything	> $2^{32} - 1$	b#11	Set
	=Period	=0	=Period	No error	Unchanged (Detected at rollover)

Table 19-14: Rollover Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≥ 1	Anything	Anything	Anything	b#10[timer running at SCLK] b#11 [timer running at ALT_CLKx]	Set
≥ 2	Anything including 0, excluding TMR_PER value	Anything including 0	≤PERIOD	Unchanged	Unchanged
	Anything including 0, excluding TMR_PER value	Anything >0	>PERIOD	b#11	Set
	Anything	Anything	> 2^{32} – 1	b#11	Set
	= Period[10]	=0	=Period	b#11	Set
	>Period	=0	>Period	Unchanged	Unchanged

Table 19-15: Overflow Event (On TMR_PER Register Programming Error Only)

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

Single Pulse PWMOUT Mode

For single pulse PWMOUT mode, there are no rollover events.

Table 19-16: Startup Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_STAT_ILAT (if enabled)	TIMER_STAT_ILAT (if enabled)
N/A	Anything	== 0	Anything	b#11[11]	Set
N/A	Anything including 0	≥1	> 2^{32} – 1	Unchanged	Unchanged
N/A	Anything including 0	≥1	> 2^{32} – 1	b#11	Set

Table 19-17: Overflow Event (On another error, such as DELAY + WIDTH ≥ 2^{32}– 1)

	TIMER_TMR[n]_DLY		TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY		TIMER_STAT_ILAT (if enabled)
Anything	Anything	Anything	Anything	b#01	Set

WIDCAP Mode

For WIDCAP mode, the TIMER_TMR[n]_PER and TIMER_TMR[n]_WID registers are read-only and the TIMER_TMR[n]_DLY register is not used. Therefore, no startup or rollover errors are possible.

Table 19-18: Overflow Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	N/A	Anything	N/A	b#01	Set

EXTCLK Mode

Table 19-19: Startup Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	N/A	N/A	N/A	b#01	Set
≥1	N/A	N/A	N/A	Unchanged	Unchanged

Table 19-20: Rollover Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
=0	N/A	N/A	N/A	b#01	Set
≥1	N/A	N/A	N/A	Unchanged	Unchanged

Table 19-21: Overflow Event (On TMR_PER Register = 0 Only)

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	N/A	N/A	N/A	b#01	Set

WATCHDOG Events

Table 19-22: Startup Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ Allowed MIN[12]	Anything < PERIOD	N/A	N/A	b#01	Set
> Allowed MIN	Anything < PERIOD	N/A	N/A	Unchanged	Unchanged
> Allowed MIN	Anything ≥ PERIOD	Refer to WATCHDOG Mode tables			

Table 19-23: Rollover Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
≤ Allowed MIN[10]	Anything < PERIOD	N/A	N/A	b#01	Set
> Allowed MIN	Anything	N/A	N/A	Unchanged	Unchanged
> Allowed MIN	Anything ≥ PERIOD	Refer to WATCHDOG Mode tables			

Table 19-24: Overflow Event

TIMER_TMR[n]_PER	TIMER_TMR[n]_DLY	TIMER_TMR[n]_WID	TIMER_TMR[n]_WID + TIMER_TMR[n]_DLY	TIMER_ERR_TYPE	TIMER_STAT_ILAT (if enabled)
Anything	Anything	N/A	N/A	b#01	Set

CM41X_M4 TIMER Register Descriptions

General-Purpose Timer Block (TIMER) contains the following registers.

Table 19-25: CM41X_M4 TIMER Register List

Name	Description
TIMER_BCAST_DLY	Broadcast Delay Register

Table 19-25: CM41X_M4 TIMER Register List (Continued)

Name	Description
TIMER_BCAST_PER	Broadcast Period Register
TIMER_BCAST_WID	Broadcast Width Register
TIMER_DATA_ILAT	Data Interrupt Latch Register
TIMER_DATA_IMSK	Data Interrupt Mask Register
TIMER_ERR_TYPE	Error Type Status Register
TIMER_RUN	Run Register
TIMER_RUN_CLR	Run Clear Register
TIMER_RUN_SET	Run Set Register
TIMER_STAT_ILAT	Status Interrupt Latch Register
TIMER_STAT_IMSK	Status Interrupt Mask Register
TIMER_STOP_CFG	Stop Configuration Register
TIMER_STOP_CFG_CLR	Stop Configuration Clear Register
TIMER_STOP_CFG_SET	Stop Configuration Set Register
TIMER_TMR[n]_CFG	Timer n Configuration Register
TIMER_TMR[n]_CNT	Timer n Counter Register
TIMER_TMR[n]_DLY	Timer n Delay Register
TIMER_TMR[n]_PER	Timer n Period Register
TIMER_TMR[n]_WID	Timer n Width Register
TIMER_TRG_IE	Trigger Slave Enable Register
TIMER_TRG_MSK	Trigger Master Mask Register

Broadcast Delay Register

For timers with TIMER_TMR[n]_CFG.BDLYEN enabled, a write to the TIMER_BCAST_DLY register concurrently updates the delay (TIMER_TMR[n]_DLY) registers of only those timers. A read of the TIMER_BCAST_DLY register returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's TIMER_TMR[n]_DLY register.

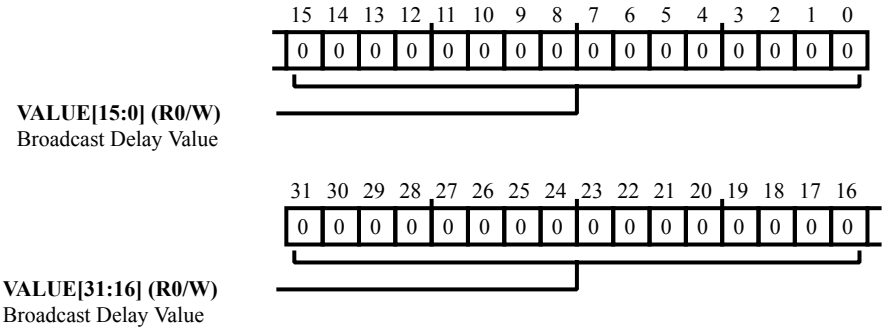


Figure 19-10: TIMER_BCAST_DLY Register Diagram

Table 19-26: TIMER_BCAST_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Delay Value. A write to the TIMER_BCAST_DLY.VALUE bit field concurrently updates the delay (TIMER_TMR[n]_DLY) registers of only those timers. A read of the TIMER_BCAST_DLY.VALUE bit field returns 0x0000 0000, and no bus error is generated.

Broadcast Period Register

For timers with TIMER_TMR[n]_CFG.BPEREN enabled, a write to the TIMER_BCAST_PER register concurrently updates the period (TIMER_TMR[n]_PER) registers of only those timers. A read of TIMER_BCAST_PER returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's TIMER_TMR[n]_PER register.

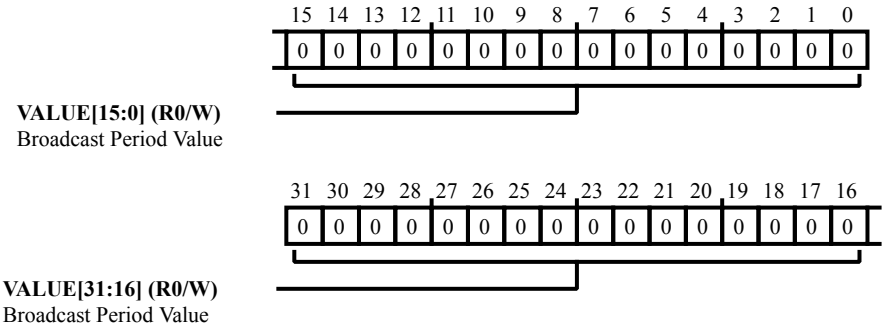


Figure 19-11: TIMER_BCAST_PER Register Diagram

Table 19-27: TIMER_BCAST_PER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Period Value. A write to the TIMER_BCAST_PER.VALUE bit field concurrently updates the period (TIMER_TMR[n]_PER) registers of only those timers. A read of the TIMER_BCAST_PER.VALUE bit fields returns 0x0000 0000, and no bus error is generated.

Broadcast Width Register

For timers with TIMER_TMR[n]_CFG.BWIDEN enabled, a write to the TIMER_BCAST_WID register concurrently updates the width (TIMER_TMR[n]_WID) registers of only those timers. A read of the TIMER_BCAST_WID register returns 0x00000000, and no bus error is generated. To read back a written value, read that TMR's TIMER_TMR[n]_WID register.

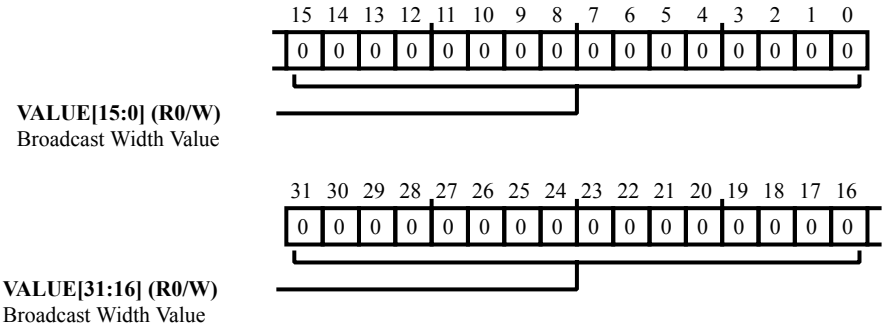


Figure 19-12: TIMER_BCAST_WID Register Diagram

Table 19-28: TIMER_BCAST_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	VALUE	Broadcast Width Value. A write to the TIMER_BCAST_WID.VALUE bit field concurrently updates the width (TIMER_TMR[n]_WID) registers of only those timers. A read of the TIMER_BCAST_WID.VALUE bit field returns 0x0000 0000, and no bus error is generated.

Data Interrupt Latch Register

The TIMER_DATA_ILAT holds the latched interrupt status for interrupt requests that have been unmasked (enabled) by the TIMER_DATA_IMSK register and generated according to the conditions selected by the TIMER_TMR[n]_CFG.IRQMODE bits. If a bit in TIMER_DATA_ILAT is already set and the corresponding interrupt is masked in TIMER_DATA_IMSK, the latch holds its old value, leaving the interrupt request asserted until it is reset by software with a W1C operation.

Note that interrupt service routines (ISRs) should clear the appropriate bits in TIMER_DATA_ILAT before returning from the ISR, to ensure that the interrupt is not re-issued. To make sure that no timer event is missed, the latch should be reset at the very beginning of the ISR when in EXTCLK mode.

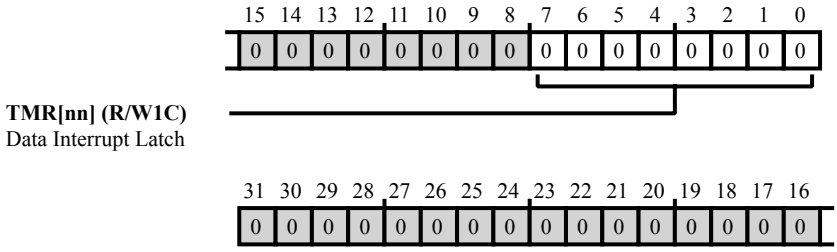


Figure 19-13: TIMER_DATA_ILAT Register Diagram

Table 19-29: TIMER_DATA_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	TMR[nn]	Data Interrupt Latch. For all TIMER_DATA_ILAT.TMR[nn] bits, status of =0 indicates no interrupt is latched, and status of =1 indicates a latched interrupt (indicating an unmasked interrupt request from a timer with a condition matching the one selected with corresponding TIMER_TMR[n]_CFG.IRQMODE bit has occurred).

Data Interrupt Mask Register

Each timer may generate a unique processor data interrupt request signal. The TIMER_DATA_IMSK register contains an interrupt mask for these requests, masking (disabling) or unmasking (enabling) the interrupts as programmed. The reset value of the TIMER_DATA_IMSK register is 0xFFFF, masking these interrupts after reset.

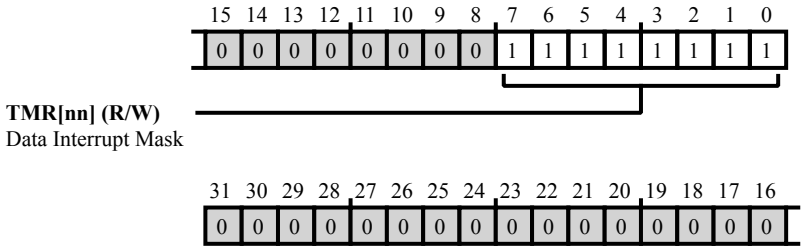


Figure 19-14: TIMER_DATA_IMSK Register Diagram

Table 19-30: TIMER_DATA_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Data Interrupt Mask. For all TIMER_DATA_IMSK.TMR[nn] bits, write =0 unmasks (enables) the corresponding data interrupt request, and write =1 masks (disables) the corresponding data interrupt request.

Error Type Status Register

The TIMER_ERR_TYPE register contains error type status bits for each timer. These bits indicate the type of error (if any) in a running timer. This register is read-only. These status bits are cleared at reset and when a particular timer is enabled.

Each time an error request interrupt is latched in the TIMER_STAT_ILAT register, the corresponding TERRx bits in the TIMER_ERR_TYPE register are loaded with a code that identifies the type of error that was detected. This status value is held until the next error or until a particular timer is restarted. No bus error is generated if a write is performed on the TIMER_ERR_TYPE register.

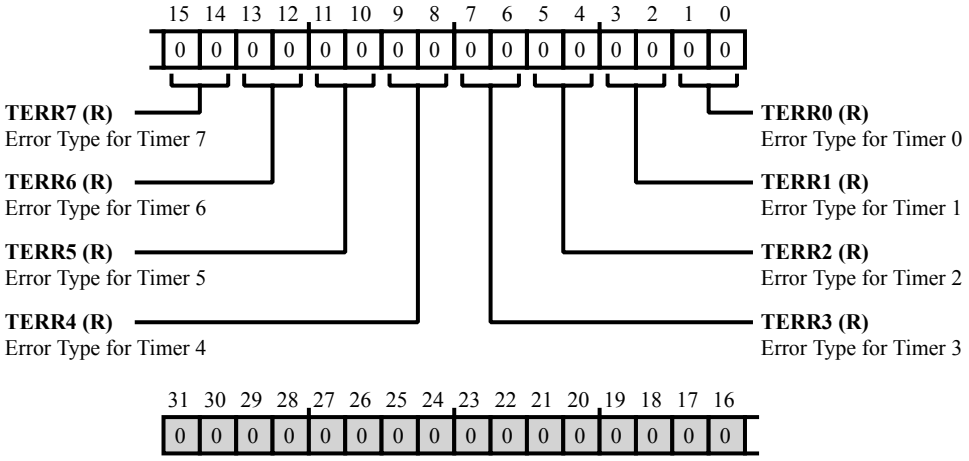


Figure 19-15: TIMER_ERR_TYPE Register Diagram

Table 19-31: TIMER_ERR_TYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:14 (R/NW)	TERR7	Error Type for Timer 7.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
13:12 (R/NW)	TERR6	Error Type for Timer 6.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error

Table 19-31: TIMER_ERR_TYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/NW)	TERR5	Error Type for Timer 5.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
9:8 (R/NW)	TERR4	Error Type for Timer 4.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
7:6 (R/NW)	TERR3	Error Type for Timer 3.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
5:4 (R/NW)	TERR2	Error Type for Timer 2.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
3:2 (R/NW)	TERR1	Error Type for Timer 1.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error
1:0 (R/NW)	TERR0	Error Type for Timer 0.	
		0	No Error
		1	Counter Overflow Error
		2	PER Register Programming Error
		3	WID or DLY Register Programming Error

Run Register

The TIMER_RUN allows all timers to be enabled simultaneously, permitting them to run synchronously. For each timer, there is a single start/stop control bit. Writing a 1 to this bit starts the corresponding timer; writing a 0 stops the timer with mechanism specified in the timer stop configuration TIMER_STOP_CFG register.

The start/stop control bits can be set/reset individually or in any combination. While starting or stopping one particular timer directly with this register, software must perform a read-modify write, so the bits corresponding to other timers remain unchanged. To avoid this need, software can use the TIMER_RUN_CLR register.

Reading the TIMER_RUN register shows the start status for the corresponding timer. A 1 indicates that the timer is running.

If a timer is in run state (corresponding run bit is =1), a software write of 1 in this bit does not have any effect on the timer state. The write does not result in restarting the timer.

Note that the TIMER_RUN register is not used in PININT mode. PININT mode starts as soon as the TIMER_TMR[n]_CFG.TMODE bits are set to 111.

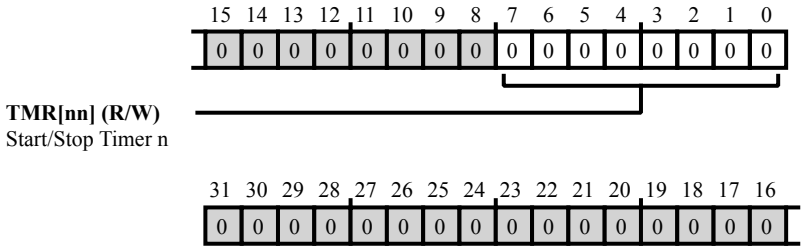


Figure 19-16: TIMER_RUN Register Diagram

Table 19-32: TIMER_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Start/Stop Timer n. For all TIMER_RUN.TMR[nn] bits, write =0 for stop, and write =1 for start. Read =1 when timer is running.

Run Clear Register

The TIMER_RUN_CLR register is an alias register, providing a mechanism to clear a specific start/stop bit in the TIMER_RUN register without affecting other bits in TIMER_RUN. To stop a particular timer, software must write a 1 into the corresponding TIMER_RUN_CLR bit. Writing a 0 has no effect. Because TIMER_RUN_CLR is a write-only register, the result of any write to this register must be checked by reading the TIMER_RUN register. A read of the TIMER_RUN_CLR returns 0x0000.

Note that the stopping mechanism of a timer may be abrupt or graceful (after completion of current waveform period) depending on the selection in the TIMER_STOP_CFG register.

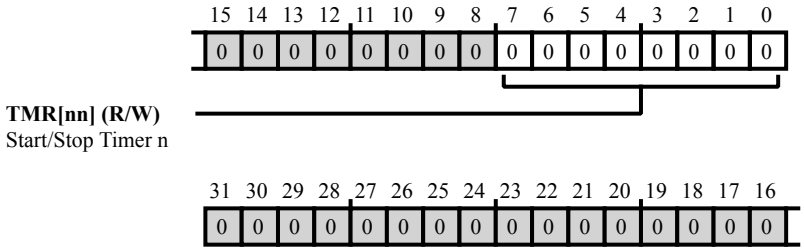


Figure 19-17: TIMER_RUN_CLR Register Diagram

Table 19-33: TIMER_RUN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1C)	TMR[nn]	RUN Clear Alias. For all TIMER_RUN_CLR.TMR[nn] bits, write =0 has no effect, and write =1 for stop (clearing the corresponding in start/stop bit in the TIMER_RUN register). Using TIMER_RUN_CLR to clear start/stop bits permits stopping specific timers without influencing run status of other timers.

Run Set Register

The TIMER_RUN_SET register is an alias register, providing a mechanism to set a specific start/stop bit in the TIMER_RUN register without affecting other bits in TIMER_RUN. To start a particular timer, software must write a 1 into the corresponding TIMER_RUN_SET bit. Writing a zero has no effect. For an example, to start timer 3 without affecting any other timer, write 0x0008 into TIMER_RUN_SET. Because TIMER_RUN_SET is a write-only register, the result of any write to this register must be checked by reading the TIMER_RUN register. A read of the TIMER_RUN_SET returns 0x0000.

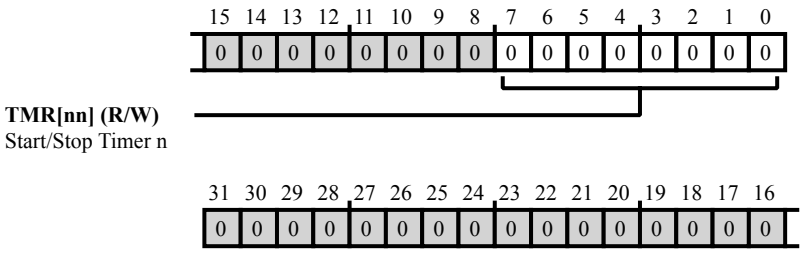


Figure 19-18: TIMER_RUN_SET Register Diagram

Table 19-34: TIMER_RUN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1S)	TMR[nn]	RUN Set Alias. For all TIMER_RUN_SET.TMR[nn] bits, write =0 has no effect, and write =1 for start (setting the corresponding start/stop bit in the TIMER_RUN register). Using TIMER_RUN_SET to set start/stop bits permits starting specific timers without influencing the run status of other timers.

Status Interrupt Latch Register

The TIMER_STAT_ILAT holds the latched interrupt status for error interrupt requests, indicating a timer overflow condition or indicating that prohibited programming has occurred for a timer. These interrupt status bits are sticky and are W1C. The bits in the TIMER_STAT_ILAT register provide information regarding each timer interrupt request source.

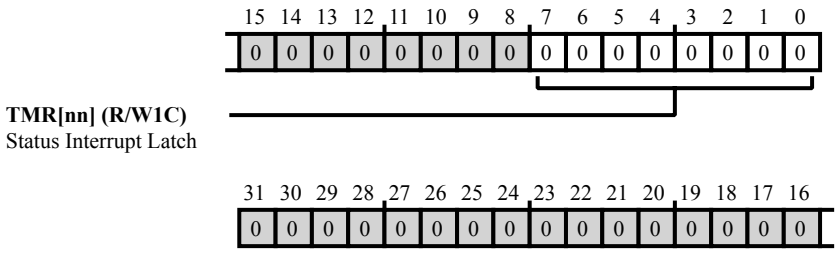


Figure 19-19: TIMER_STAT_ILAT Register Diagram

Table 19-35: TIMER_STAT_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	TMR[nn]	Status Interrupt Latch. For all TIMER_STAT_ILAT.TMR[nn] bits, status of 0 indicates no error interrupt request is latched, and status of 1 indicates a timer counter overflow or programming error interrupt request is latched.

Status Interrupt Mask Register

While each timer may generate a status interrupt request, these requests are OR'ed to generate a single status interrupt signal to the system event controller. The TIMER_STAT_IMSK register contains an interrupt mask for these requests, masking (disabling) or unmasking (enabling) the interrupts as programmed. The reset value of the TIMER_STAT_IMSK register is 0xFFFF, masking these interrupts after reset.

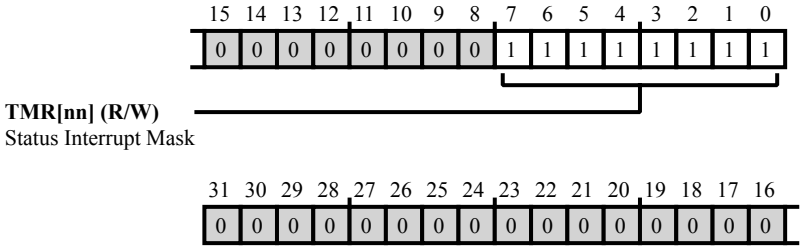


Figure 19-20: TIMER_STAT_IMSK Register Diagram

Table 19-36: TIMER_STAT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Status Interrupt Mask. For all TIMER_STAT_IMSK.TMR[nn] bits, write =0 unmasks (enables) the corresponding status interrupt request, and write =1 masks (disables) the corresponding status interrupt request.

Stop Configuration Register

The TIMER_STOP_CFG register selects the stop mode for each timer. Timers may be stopped abruptly (immediate halt - all modes) or gracefully in PWMOUT modes (single pulse and continuous). The halt is achieved through either a write =0 to the corresponding bit in TIMER_RUN or a write =1 to the corresponding bit in TIMER_RUN_CLR. A read of TIMER_STOP_CFG returns the last value written.

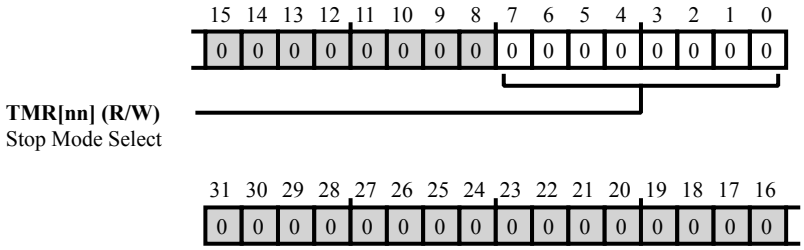


Figure 19-21: TIMER_STOP_CFG Register Diagram

Table 19-37: TIMER_STOP_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Stop Mode Select. For all TIMER_STOP_CFG.TMR[nn] bits, write =0 for graceful termination (PWMOUT modes only), and write =1 for abrupt (immediate halt) on stop.

Stop Configuration Clear Register

This is an alias register, providing a mechanism to clear a specific bit in the TIMER_STOP_CFG register without affecting other bits in TIMER_STOP_CFG. To clear a bit in TIMER_STOP_CFG, software must write a 1 to the corresponding bit of TIMER_STOP_CFG_CLR register. Writing a zero has no effect. Because the TIMER_STOP_CFG_CLR register is a write-only register, the result of any write to this register must be checked by reading the TIMER_STOP_CFG register. A read of the TIMER_STOP_CFG_CLR register returns 0x0000.

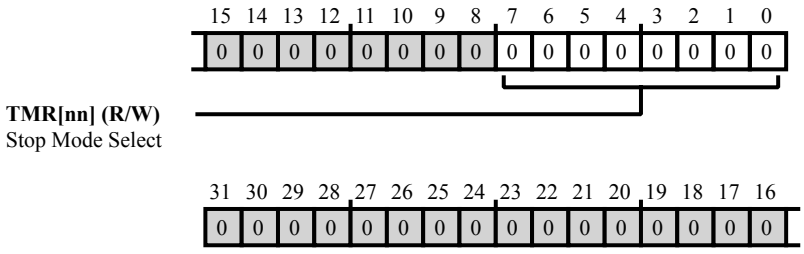


Figure 19-22: TIMER_STOP_CFG_CLR Register Diagram

Table 19-38: TIMER_STOP_CFG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1C)	TMR[nn]	STOP_CFG Clear Alias. For all TIMER_STOP_CFG_CLR.TMR[nn] bits, write =0 has no effect, and write =1 for graceful stop in PWMOUT modes (clearing the corresponding stop mode select bit in the TIMER_STOP_CFG register). Using TIMER_STOP_CFG_CLR to clear stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

Stop Configuration Set Register

This is an alias register, providing a mechanism to set a specific bit in the TIMER_STOP_CFG register without affecting other bits in TIMER_STOP_CFG. To set a bit in the TIMER_STOP_CFG register, software must write a 1 to the corresponding bit of the TIMER_STOP_CFG_SET register. Writing a zero has no effect. Because the TIMER_STOP_CFG_SET register is a write-only register, the result of any write to this register must be checked by reading the TIMER_STOP_CFG register. A read of the TIMER_STOP_CFG_SET register returns 0x0000.

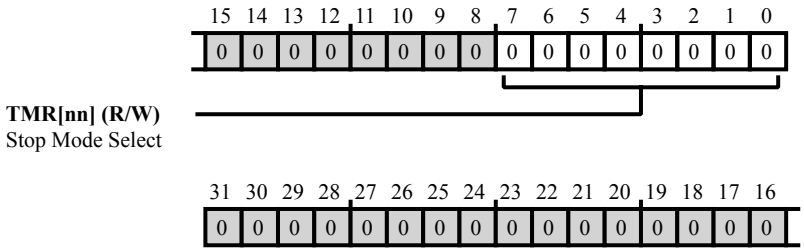


Figure 19-23: TIMER_STOP_CFG_SET Register Diagram

Table 19-39: TIMER_STOP_CFG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W1S)	TMR[nn]	STOP_CFG Set Alias. For all TIMER_STOP_CFG_SET.TMR[nn] bits, write =0 has no effect, and write =1 for abrupt stop (setting the corresponding stop mode select bit in the TIMER_STOP_CFG register). Using TIMER_STOP_CFG_SET to set stop mode bits permits configuring specific timers without influencing the stop mode configuration of other timers.

Timer n Configuration Register

Each timer has a TIMER_TMR[n]_CFG register that specifies its operating mode. Only write to a TIMER_TMR[n]_CFG register when the corresponding timer is not running.

After disabling a timer operating in PWMOUT mode, verify that the timer has stopped running by checking the start/stop status of the timer in the TIMER_RUN register before writing to the timer's TIMER_TMR[n]_CFG register.

Note that a timer's TIMER_TMR[n]_CFG register may be read at any time.

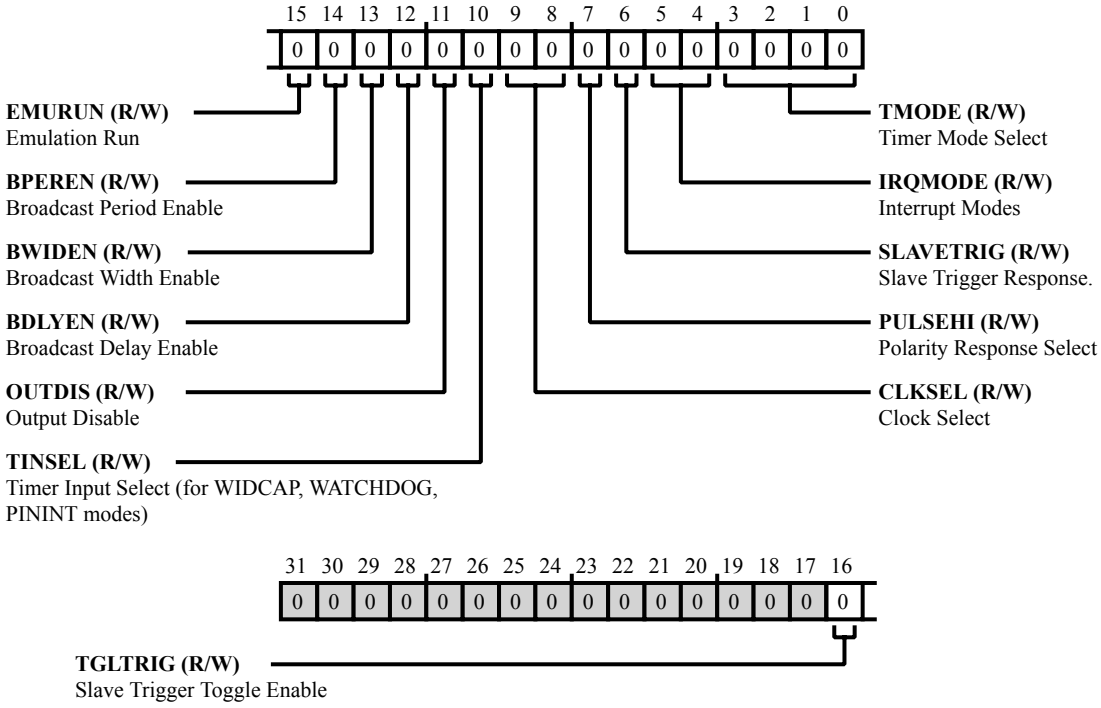


Figure 19-24: TIMER_TMR[n]_CFG Register Diagram

Table 19-40: TIMER_TMR[n]_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	TGLTRIG	Slave Trigger Toggle Enable. The TIMER_TMR[n]_CFG.TGLTRIG bit stops the timer if it is running and starts the timer if it is halted (in the stop state). If the TIMER_TMR[n]_CFG.TGLTRIG bit is set, then the setting of the TIMER_TMR[n]_CFG.SLAVETRIG bit is ignored.	
		0	Slave Trigger Response Depends on SLAVETRIG Bit Setting
		1	Slave Trigger Toggles Timer State

Table 19-40: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EMURUN	Emulation Run. The TIMER_TMR[n]_CFG.EMURUN bit causes the timer to run (count) during emulation.	
		0	Stop Timer During Emulation
		1	Run Timer During Emulation
14 (R/W)	BPEREN	Broadcast Period Enable. The TIMER_TMR[n]_CFG.BPEREN bit enables updates to the TIMER_TMR[n]_PER register simultaneously across more than one timer.	
		0	Disable Broadcast to PER Register
		1	Enable Broadcast to PER Register
13 (R/W)	BWIDEN	Broadcast Width Enable. The TIMER_TMR[n]_CFG.BWIDEN bit enables updates to the TIMER_TMR[n]_WID register simultaneously across more than one timer.	
		0	Disable Broadcast to WID Register
		1	Enable Broadcast to WID Register
12 (R/W)	BDLYEN	Broadcast Delay Enable. The TIMER_TMR[n]_CFG.BDLYEN bit enables updates to the TIMER_TMR[n]_DLY register simultaneously across more than one timer.	
		0	Disable Broadcast to DLY Register
		1	Enable Broadcast to DLY Register
11 (R/W)	OUTDIS	Output Disable. The TIMER_TMR[n]_CFG.OUTDIS bit enables or disables the timer pin output buffer.	
		0	Enable TMR Pin Output Buffer
		1	Disable TMR Pin Output Buffer
10 (R/W)	TINSEL	Timer Input Select (for WIDCAP, WATCHDOG, PININT modes).	
		0	Use TMR Pin Input
		1	Use TMR Alternate Capture Input
9:8 (R/W)	CLKSEL	Clock Select. The TIMER_TMR[n]_CFG.CLKSEL bit field selects the TIMER clock to use.	
		0	Use SCLK
		1	Use TMR_ALT_CLK0 as TMR Clock
		3	Use TMR_ALT_CLK1 as TMR Clock

Table 19-40: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	PULSEHI	Polarity Response Select. The `TIMER_TMR[n]_CFG.PULSEHI` bit defines specific behaviors of the timer based on the operating mode. For more information, see the specific operating mode in the Programming Concepts section.	
		0	Negative Response or Pulse. A Negative Edge Response or Negative Action Pulse on the TMR pin.
		1	Positive Response or Pulse. A Positive Edge Response or Positive Action Pulse on the TMR pin.
6 (R/W)	SLAVETRIG	Slave Trigger Response.. The `TIMER_TMR[n]_CFG.SLAVETRIG` bit controls the trigger response. The trigger pulse has no effect (to stop or start the timer) if the timer is already in the requested state.	
		0	Pulse Stops Timer if it is Running
		1	Pulse Starts Timer if it is Stopped

Table 19-40: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:4 (R/W)	IRQMODE	Interrupt Modes. The TIMER_TMR[n]_CFG.IRQMODE bit field selects the interrupt request mode. Note that any mismatched combination of the TIMER_TMR[n]_CFG.IRQMODE and the TIMER_TMR[n]_CFG.TMODE bits results in no interrupt being generated. In WIDCAP modes, the position of the interrupt is controlled with the TIMER_TMR[n]_CFG.TMODE bit, and the TIMER_TMR[n]_CFG.IRQMODE bit is ignored. Whenever an interrupt is generated, a trigger master pulse is also generated, if enabled in the TIMER_TRG_MSK register.	
		0	Active Edge Mode. The timer generates an interrupt at every active edge. The active edge polarity depends on the state of the TIMER_TMR[n]_CFG.PULSEHI bit. Valid for PININT mode only.
		1	Delay Expired Mode. The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_DLY register. This mode is valid for all PWMOUT modes.
		2	Width Plus Delay Expired Mode. The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_WID register plus the value in the TIMER_TMR[n]_DLY register. (PWMOUT modes only) The timer generates an interrupt if the deasserting edge is within the specified window. (WATCHDOG modes only)
		3	Period Expired Mode. The timer generates an interrupt when the TIMER_TMR[n]_CNT value reaches the value in the TIMER_TMR[n]_PER register. (Continuous PWMOUT and EXTCLK modes only) The timer generates an interrupt if the deasserting edge is outside the specified window.(WATCHDOG modes only)
3:0 (R/W)	TMODE	Timer Mode Select. The TIMER_TMR[n]_CFG.TMODE bit field selects the operating mode of each timer.	
		0-7	Idle Mode
		8	Period Watchdog Mode
		9	Width Watchdog Mode

Table 19-40: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		10	Measurement Report at Asserting Edge of Waveform
		11	Measurement Report at Deasserting Edge of Waveform
		12	Continuous PWMOUT Mode
		13	Single Pulse PWMOUT Mode
		14	EXTCLK Mode
		15	PININT (pin interrupt) Mode

Timer n Counter Register

The TIMER_TMR[n]_CNT register holds the current timer count. After enabling, the count is re-initialized to either 0x0 or 0x1, depending on the configuration and mode. The TIMER_TMR[n]_CNT register is read-only and may be read at any time (whether the timer is running or stopped). Reading the TIMER_TMR[n]_CNT register returns an atomic 32-bit value.

Depending on the timer operation mode, the counter increment can be clocked by a number of sources, including SCLK, the TMR or alternate capture input pins, TIMER_ACLK[n]. The counter retains its value after the timer is disabled.

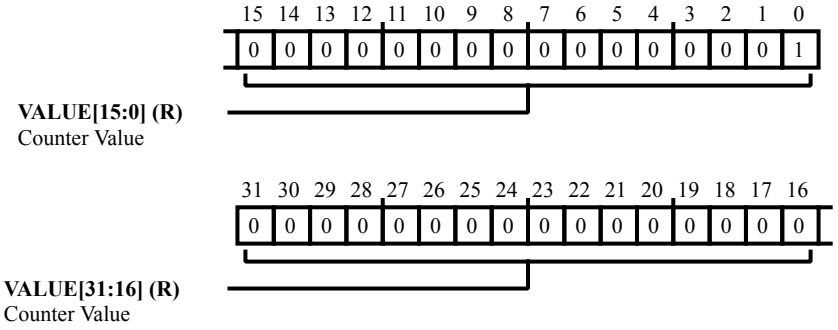


Figure 19-25: TIMER_TMR[n]_CNT Register Diagram

Table 19-41: TIMER_TMR[n]_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Counter Value. The TIMER_TMR[n]_CNT.VALUE bit field holds the current timer count.

Timer n Delay Register

The TIMER_TMR[n]_DLY register holds the delay value for the corresponding timer. This register's use is based on the selected timer mode.

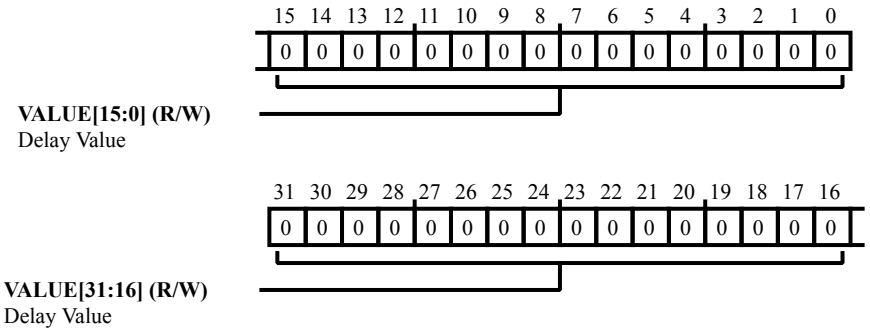


Figure 19-26: TIMER_TMR[n]_DLY Register Diagram

Table 19-42: TIMER_TMR[n]_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Delay Value. The TIMER_TMR[n]_DLY.VALUE bit field holds the delay value for the corresponding timer.

Timer n Period Register

The TIMER_TMR[n]_PER register holds the period value for the corresponding timer. This register's use is based on the selected timer mode.

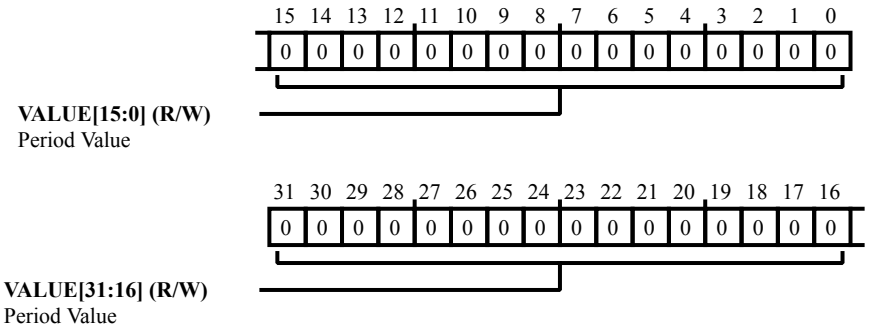


Figure 19-27: TIMER_TMR[n]_PER Register Diagram

Table 19-43: TIMER_TMR[n]_PER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Period Value. The TIMER_TMR[n]_PER.VALUE bit field holds the period value for the corresponding timer.

Timer n Width Register

The TIMER_TMR[n]_WID register holds the width value for the corresponding timer. This register's use is based on the selected timer mode.

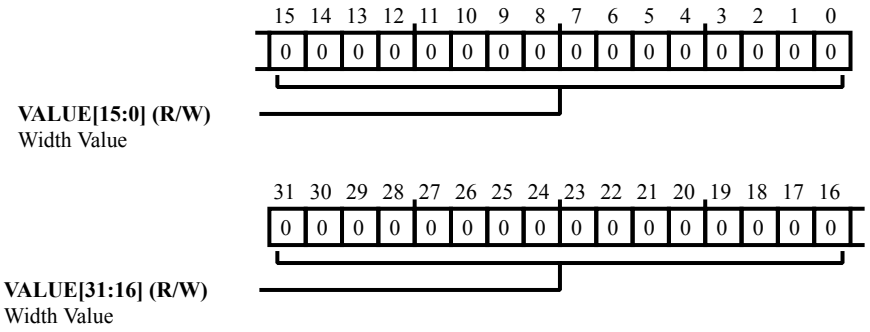


Figure 19-28: TIMER_TMR[n]_WID Register Diagram

Table 19-44: TIMER_TMR[n]_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Width Value. The TIMER_TMR[n]_WID.VALUE bit field holds the width value for the corresponding timer.

Trigger Slave Enable Register

As a trigger slave, each timer can generate a unique data trigger pulse signal. The TIMER_TRG_IE contains trigger input enable bits for these signals, disabling or enabling the triggers as programmed. The reset value of the TIMER_TRG_IE register is 0xFFFF, masking these triggers after reset.

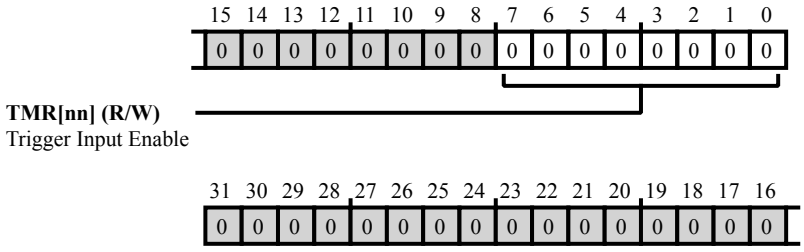


Figure 19-29: TIMER_TRG_IE Register Diagram

Table 19-45: TIMER_TRG_IE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Trigger Input Enable. For all TIMER_TRG_IE.TMR[nn] bits, write =0 disables the corresponding trigger input, and write =1 enables the corresponding trigger input.

Trigger Master Mask Register

As a trigger master, each timer can generate a unique data trigger pulse signal. The TIMER_TRG_MSK register contains a trigger mask for these outputs, masking (disabling) or unmasking (enabling) the triggers as programmed. The reset value of the TIMER_TRG_MSK register is 0xFFFF, masking these triggers after reset.

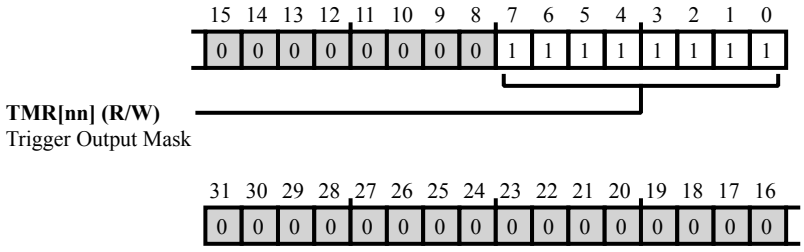


Figure 19-30: TIMER_TRG_MSK Register Diagram

Table 19-46: TIMER_TRG_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	TMR[nn]	Trigger Output Mask. For all TIMER_TRG_MSK.TMR[nn] bits, write =0 unmasks (enables) the corresponding data trigger output, and write =1 masks (disables) the corresponding data trigger output.

20 Capture Timer (CPTMR)

The Capture Timer (CPTMR) module measures the total on-time of the input signal between two triggers. The module performs this measurement by counting clock pulses during the on-time of the input signal. The total on-time is captured when the stop trigger is received.

The CPTMR is used in inverter applications to measure the total inverter on-time. The application can capture the total on-time of the inverter output between PWM cycles. The triggers required to start and stop the capture timer are received from the TRU, where the PWM is the master.

CPTMR Functional Description

The processor has three capture timers. Each capture timer block can be enabled and configured to capture the total on-time of the `CPTMR_IN[n]` input pin between two trigger assertions (`CPTMR_CPT[n]_SLV0` or `CPTMR_CPT[n]_SLV1`). The core of the capture timer is a 32-bit counter that can be interrogated through the read-only `CPTMR_CNT[n]` register. The counter is reset to 0x1 when the start trigger is received. Between the start and stop triggers, the total on-time of the `CPTMR_IN[n]` input pin is captured in the `CPTMR_TON[n]` register.

CPTMR Block Diagram

The ***Capture Timer Block Diagram*** illustrates the block diagram of the CPTMR module. The block diagram is comprised of two primary sections:

Internal Interface. The core always accesses the capture timer registers through the peripheral bus interface. The capture timer has dedicated data and status interrupt request outputs that connect to the SEC module.

External Interface. The capture timer has an asynchronous input pin (`CPTMR_IN[n]`) and asynchronous trigger inputs (`CPTMR_CPT[n]_SLV0` and `CPTMR_CPT[n]_SLV1`). The requirement for minimum input pulse width timing on the external pins is $2 \times t_{SCLK}$, to guarantee that SCLK samples the asynchronous pulse.

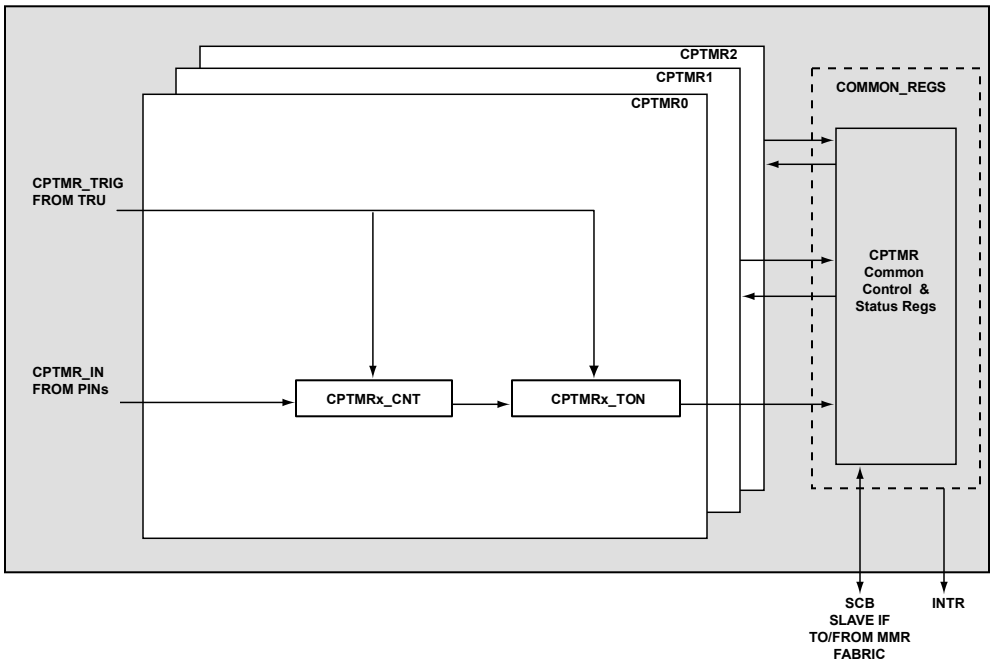


Figure 20-1: Capture Timer Block Diagram

CM41X_M4 CPTMR Register List

The Capture Timer (CPTMR) measures total on-time of the input signal between two triggers.

Table 20-1: CM41X_M4 CPTMR Register List

Name	Description
CPTMR_CFG[n]	Configuration Register
CPTMR_CNT[n]	Counter Register
CPTMR_DATA_ILAT	Data Interrupt Latch Status Register
CPTMR_DATA_IMSK	Data Interrupt Mask Register
CPTMR_DATA_IMSK_CLR	Data Interrupt Mask Clear Register
CPTMR_DATA_IMSK_SET	Data Interrupt Mask Set Register
CPTMR_RUN	Run Register
CPTMR_RUN_CLR	Run Clear Register
CPTMR_RUN_SET	Run Set Register
CPTMR_STAT_ILAT	Interrupt Latch Status Register
CPTMR_STAT_IMSK	Status Interrupt Mask Register
CPTMR_STAT_IMSK_CLR	Status Interrupt Mask Clear Register
CPTMR_STAT_IMSK_SET	Status Interrupt Mask Set Register

Table 20-1: CM41X_M4 CPTMR Register List (Continued)

Name	Description
CPTMR_TON[n]	On-time Capture Register

CM41X_M4 CPTMR Trigger List

Table 20-2: CM41X_M4 CPTMR Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 20-3: CM41X_M4 CPTMR Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
46	CPTMR0_CPT0_SLV0	CPTMR0 CPT 0, Trigger Slave 0	Pulse
47	CPTMR0_CPT0_SLV1	CPTMR0 CPT 0, Trigger Slave 1	Pulse
48	CPTMR0_CPT1_SLV0	CPTMR0 CPT 1, Trigger Slave 0	Pulse
49	CPTMR0_CPT1_SLV1	CPTMR0 CPT 1, Trigger Slave 1	Pulse
50	CPTMR0_CPT2_SLV0	CPTMR0 CPT 2, Trigger Slave 0	Pulse
51	CPTMR0_CPT2_SLV1	CPTMR0 CPT 2, Trigger Slave 1	Pulse

CM41X_M0 CPTMR Interrupt List

Table 20-4: CM41X_M0 CPTMR Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
16	CPTMR0_CPT0_MEAS	CPTMR0 CPT 0 Measure Interrupt	Level	
16	CPTMR0_CPT1_MEAS	CPTMR0 CPT 1 Measure Interrupt	Level	
16	CPTMR0_CPT2_MEAS	CPTMR0 CPT 2 Measure Interrupt	Level	

CM41X_M4 CPTMR Interrupt List

Table 20-5: CM41X_M4 CPTMR Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
54	CPTMR0_CPT0_ERR	CPTMR0 CPT 0 Error Interrupt	Level	
55	CPTMR0_CPT1_ERR	CPTMR0 CPT 1 Error Interrupt	Level	

Table 20-5: CM41X_M4 CPTMR Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
56	CPTMR0_CPT2_ERR	CPTMR0 CPT 2 Error Interrupt	Level	
106	CPTMR0_CPT0_MEAS	CPTMR0 CPT 0 Measure Interrupt	Level	
107	CPTMR0_CPT1_MEAS	CPTMR0 CPT 1 Measure Interrupt	Level	
108	CPTMR0_CPT2_MEAS	CPTMR0 CPT 2 Measure Interrupt	Level	

CPTMR Operation

The CPTMR is used to measure the total on-time of the input waveform between two trigger assertions. When the capture timer is enabled, the start trigger resets the count in the `CPTMR_CNT[n]` register to 1, and the timer starts counting and incrementing on the following SCLK rising edges, if the `CPTMR_IN[n]` input pin is asserted.

The `CPTMR_IN[n]` input pin is used directly or with inverted polarity, based on the `CPTMR_CFG[n].TINPOL` bit setting. Similarly, the capture timer trigger is used directly or with inverted polarity, based on the `CPTMR_CFG[n].TRIGPOL` bit setting. On the start trigger assertion, the contents of the `CPTMR_CNT[n]` register are transferred to the `CPTMR_TON[n]` register and the `CPTMR_CNT[n]` register is reset to 0x1. The capture timer counter continues counting and capturing until the timer is disabled.

When the 32-bit counter range exceeds 0xFFFFFFFF, the counter rolls over to 0, and an error report interrupt is generated. The capture timer status interrupt latch register (`CPTMR_STAT_ILAT`) bits indicate the occurrence of the overflow, if enabled in the status interrupt mask register (`CPTMR_STAT_IMSK`) bits. To measure on-time longer than 0xFFFF FFFF cycles, software can count the number of errors reported between measurement report interrupts. Since the reset value of the counter is 1, the first overflow error report interrupt adds 2^{32}– 1 SCLK cycles to the total on-time captured. As the counter rolls over to 0 after the overflow, 2^{32}SCLK cycles are added to the total on-time from the second overflow interrupt onward.

The timer data interrupt latch bits in the `CPTMR_DATA_ILAT` register are set (if enabled) to indicate an interrupt request from a particular capture timer instance. The on-time data interrupt is generated when the stop trigger is asserted. The bits in the `CPTMR_DATA_ILAT` and `CPTMR_STAT_ILAT` registers are sticky bits, and software has to explicitly clear them. Disabling the CPTMR module clears the `CPTMR_DATA_ILAT` and `CPTMR_STAT_ILAT` register bits along with the corresponding IRQ.

Since the timer can also accept external asynchronous inputs, the external input signal is shifted by close to 2 SCLK cycles to align the input edge with the SCLK edge. This synchronized input is then provided to the capture timer module for on-time calculations. There can be a maximum accuracy loss of 2 SCLK cycles in on-time calculations, but only if the input level is active and very near the leading edge of a trigger. The input given in the following figure is a synchronized signal going to the capture timer module.

Refer to the ***On-Time Capture Timing (TINPOL=0 and TRIGPOL=0)*** figure. For `CPTMR_CFG[n].TINPOL` =0 and `CPTMR_CFG[n].TRIGPOL` =0, the counter value in `CPTMR_CNT[n]` is reset to 1 at the assertion of the start trigger. Thereafter, the `CPTMR_CNT[n]` counter value increments with every SCLK rising edge, if the

CPTMR_IN[n] input is active. When the stop trigger is then asserted, the CPTMR_TON[n] total on-time is copied from the CPTMR_CNT[n] counter value, and the CPTMR_CNT[n] count value is reset to 1. The operation continues this way for subsequent trigger assertions until the timer is disabled.

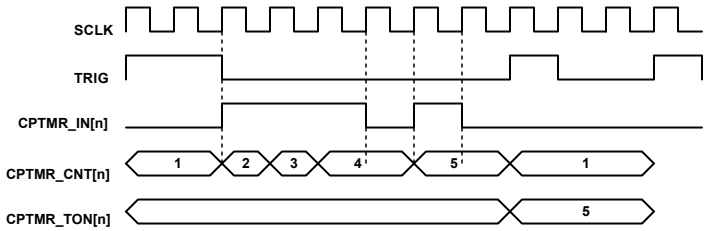


Figure 20-2: On-Time Capture Timing (TINPOL=0 and TRIGPOL=0)

CPTMR Interrupt Signals

The CPTMR is capable of signaling the system about its state and error conditions that occur during its operation, by providing status and data interrupt requests.

Data Interrupts

The capture timer can generate a processor data interrupt request signal if the data interrupt is unmasked by programming a 0 in the mask bit of the timer data interrupt mask register (CPTMR_DATA_IMSK). For a data interrupt to be serviced, clear the CPTMR_DATA_ILAT register with software using a W1C operation. The total on-time can be monitored when the data interrupt is generated at the assertion of the stop trigger.

Status Interrupts

The capture timer indicates a counter overflow condition through the status interrupt latch register (CPTMR_STAT_ILAT). The capture timer can generate a status interrupt for an overflow condition if the status interrupt is unmsasked by programming 0 in the mask bit of the status interrupt mask register (CPTMR_STAT_IMSK). The interrupt is serviced by performing a W1C operation to the CPTMR_STAT_ILAT register.

CM41X_M4 CPTMR Register Descriptions

Capture Timer (CPTMR) contains the following registers.

Table 20-6: CM41X_M4 CPTMR Register List

Name	Description
CPTMR_CFG[n]	Configuration Register
CPTMR_CNT[n]	Counter Register
CPTMR_DATA_ILAT	Data Interrupt Latch Status Register
CPTMR_DATA_IMSK	Data Interrupt Mask Register

Table 20-6: CM41X_M4 CPTMR Register List (Continued)

Name	Description
CPTMR_DATA_IMSK_CLR	Data Interrupt Mask Clear Register
CPTMR_DATA_IMSK_SET	Data Interrupt Mask Set Register
CPTMR_RUN	Run Register
CPTMR_RUN_CLR	Run Clear Register
CPTMR_RUN_SET	Run Set Register
CPTMR_STAT_ILAT	Interrupt Latch Status Register
CPTMR_STAT_IMSK	Status Interrupt Mask Register
CPTMR_STAT_IMSK_CLR	Status Interrupt Mask Clear Register
CPTMR_STAT_IMSK_SET	Status Interrupt Mask Set Register
CPTMR_TON[n]	On-time Capture Register

Configuration Register

The CPTMR_CFG[n] register contains bits that are used to configure various module options.

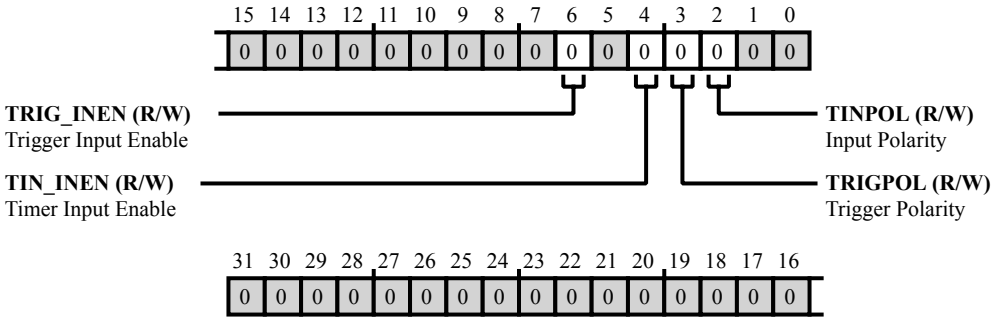


Figure 20-3: CPTMR_CFG[n] Register Diagram

Table 20-7: CPTMR_CFG[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	TRIG_INEN	Trigger Input Enable. The CPTMR_CFG[n].TRIG_INEN bit enables trigger input (=1).
4 (R/W)	TIN_INEN	Timer Input Enable. The CPTMR_CFG[n].TIN_INEN bit enables timer input (=1).
3 (R/W)	TRIGPOL	Trigger Polarity. The CPTMR_CFG[n].TRIGPOL bit selects whether a trigger is not inverted (active high/rising edge as leading edge, =0) or a trigger is inverted (active low/falling edge as leading edge, =1).
2 (R/W)	TINPOL	Input Polarity. The CPTMR_CFG[n].TINPOL bit selects whether input is not inverted (active high/rising edge as leading edge, =0) or input is inverted (active low/falling edge as leading edge, =1).

Counter Register

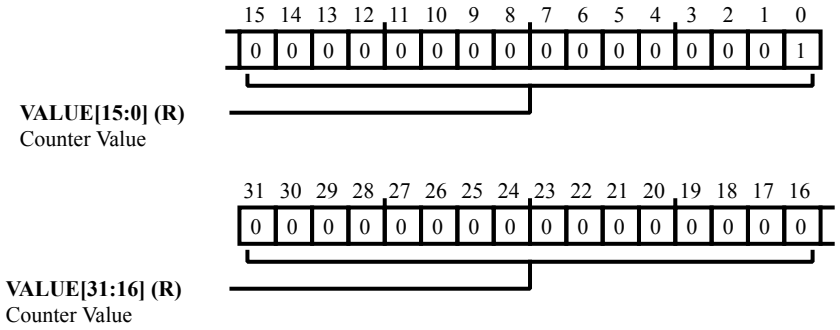


Figure 20-4: CPTMR_CNT[n] Register Diagram

Table 20-8: CPTMR_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Counter Value.

Data Interrupt Latch Status Register

The CPTMR_DATA_ILAT register indicates data interrupt requests.

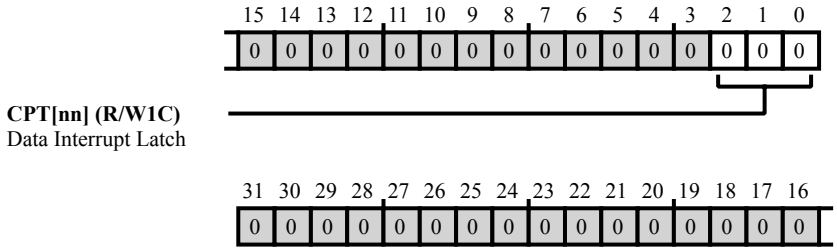


Figure 20-5: CPTMR_DATA_ILAT Register Diagram

Table 20-9: CPTMR_DATA_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Data Interrupt Latch. A 1 in any of the CPTMR_DATA_ILAT.CPT[nn] bits indicates a data interrupt request from timer n.

Data Interrupt Mask Register

The CPTMR_DATA_IMSK register is used to mask data interrupts.

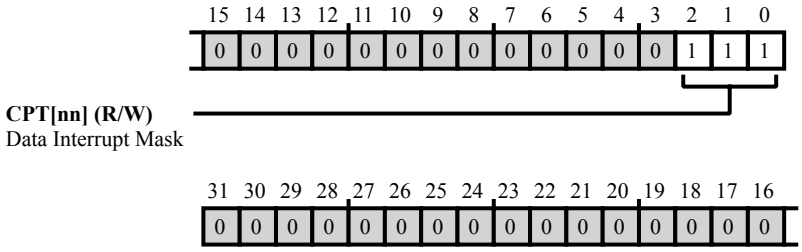


Figure 20-6: CPTMR_DATA_IMSK Register Diagram

Table 20-10: CPTMR_DATA_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	CPT[nn]	Data Interrupt Mask. Writing a 0 to any of the CPTMR_DATA_IMSK.CPT[nn] bits enables the data interrupt requests for timer n.

Data Interrupt Mask Clear Register

The CPTMR_DATA_IMSK_CLR register enables the data interrupt request for timer n

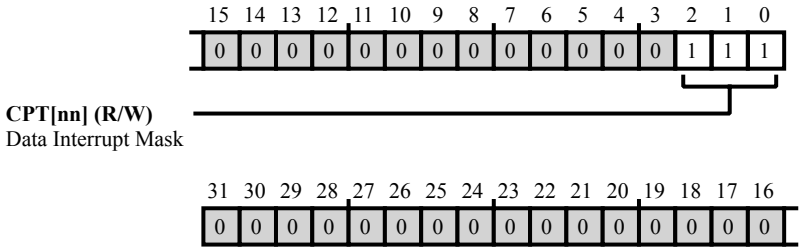


Figure 20-7: CPTMR_DATA_IMSK_CLR Register Diagram

Table 20-11: CPTMR_DATA_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Data Interrupt Mask Clear. Writing a 1 to any of the CPTMR_DATA_IMSK_CLR.CPT[nn] bits enables the data interrupt request for timer n. Writing a 0 has no effect.

Data Interrupt Mask Set Register

The CPTMR_DATA_IMSK_SET register disables the data interrupt for timer n.

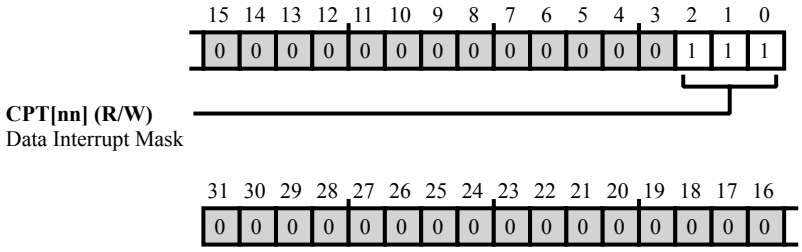


Figure 20-8: CPTMR_DATA_IMSK_SET Register Diagram

Table 20-12: CPTMR_DATA_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1S)	CPT[nn]	Data Interrupt Mask Set. Writing a 1 to any of the CPTMR_DATA_IMSK_SET.CPT[nn] bits disables the data interrupt for timer n. Writing a 0 has no effect.

Run Register

Writing a 1 to bit n of the CPTMR_RUN register starts timer n. Writing a 0 to bit n halts timer n.

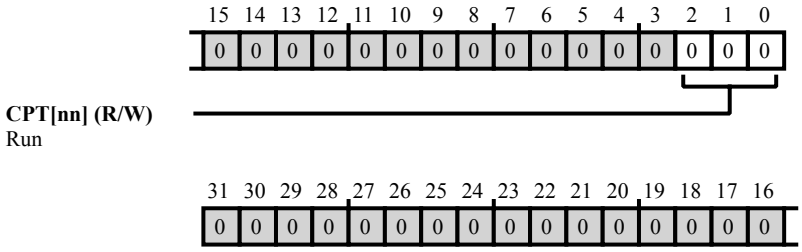


Figure 20-9: CPTMR_RUN Register Diagram

Table 20-13: CPTMR_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	CPT[nn]	Run. Writing 1 to any of the CPTMR_RUN.CPT[nn] bits starts timer n. Writing 0 stops the timer.

Run Clear Register

Writing a 1 to bit n of the CPTMR_RUN_CLR register stops timer n. Writing a 0 has no effect.

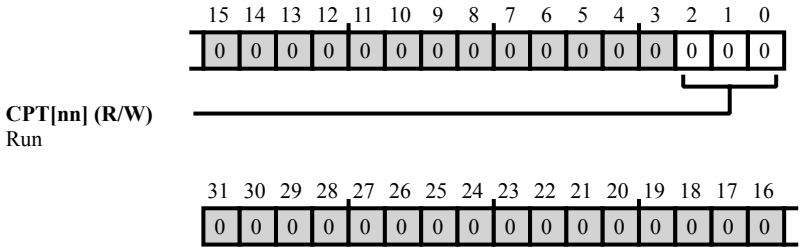


Figure 20-10: CPTMR_RUN_CLR Register Diagram

Table 20-14: CPTMR_RUN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Run Clear. Writing a 1 to any of the CPTMR_RUN_CLR.CPT[nn] bits stops timer n. Writing a 0 has no effect.

Run Set Register

Writing a 1 to bit n of the CPTMR_RUN_SET register starts timer n. Writing a 0 has no effect.

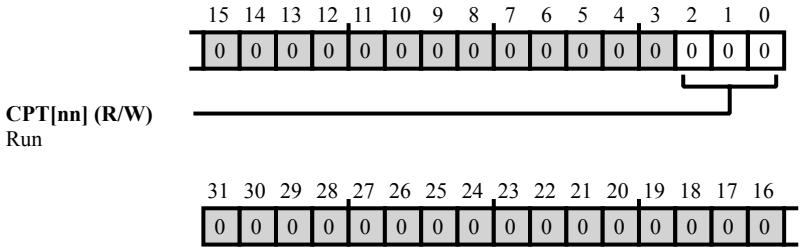


Figure 20-11: CPTMR_RUN_SET Register Diagram

Table 20-15: CPTMR_RUN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1S)	CPT[nn]	Run Set Register. Write a 1 to any of the CPTMR_RUN_SET.CPT[nn] bits to start timer n.

Interrupt Latch Status Register

The CPTMR_STAT_ILAT register indicates counter overflow errors.

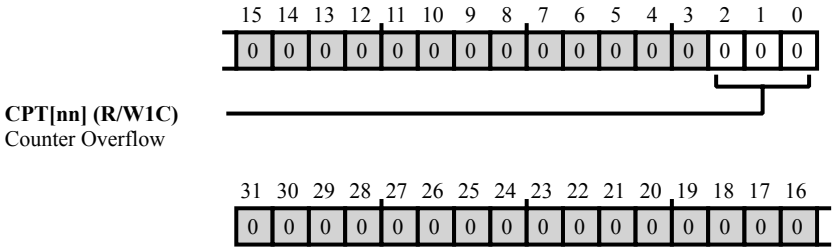


Figure 20-12: CPTMR_STAT_ILAT Register Diagram

Table 20-16: CPTMR_STAT_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Counter Overflow. A 1 in any of the CPTMR_STAT_ILAT.CPT[nn] bits indicates an overflow error for timer n.

Status Interrupt Mask Register

The CPTMR_STAT_IMSK register is used to mask status interrupts.

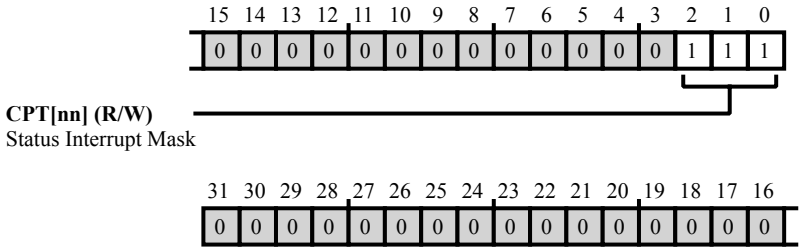


Figure 20-13: CPTMR_STAT_IMSK Register Diagram

Table 20-17: CPTMR_STAT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	CPT[nn]	Status Interrupt Mask. Writing a 0 to any of the CPTMR_STAT_IMSK.CPT[nn] bits enables the status interrupt for timer n.

Status Interrupt Mask Clear Register

The CPTMR_STAT_IMSK_CLR register is used to unmask status interrupts.

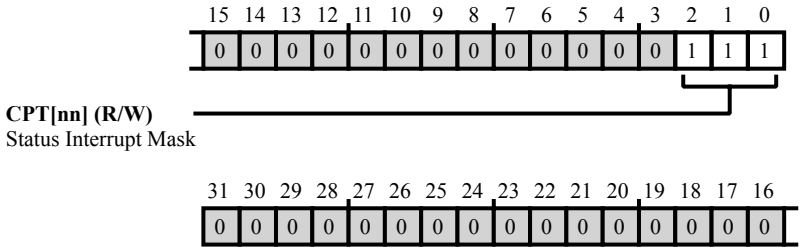


Figure 20-14: CPTMR_STAT_IMSK_CLR Register Diagram

Table 20-18: CPTMR_STAT_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1C)	CPT[nn]	Status Interrupt Mask Clear. Writing a 1 to any of the CPTMR_STAT_IMSK_CLR.CPT[nn] bits enables the status interrupt request for timer n. Writing a 0 has no effect.

Status Interrupt Mask Set Register

The CPTMR_STAT_IMSK_SET register is used to mask status interrupt requests.

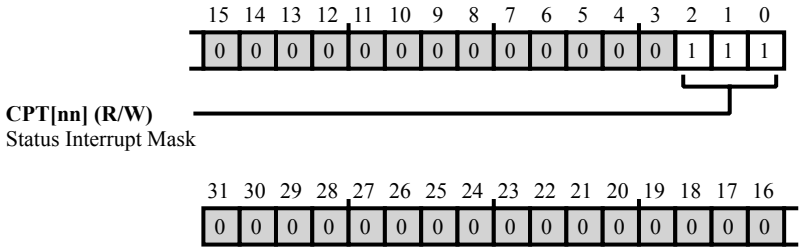


Figure 20-15: CPTMR_STAT_IMSK_SET Register Diagram

Table 20-19: CPTMR_STAT_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W1S)	CPT[nn]	Status Interrupt Mask Set. Writing a 1 to any of the CPTMR_STAT_IMSK_SET.CPT[nn] bits disables the status interrupt request for timer n. Writing a 0 has no effect.

On-time Capture Register

The CPTMR_TON[n] register captures the total on-time of the input waveform.

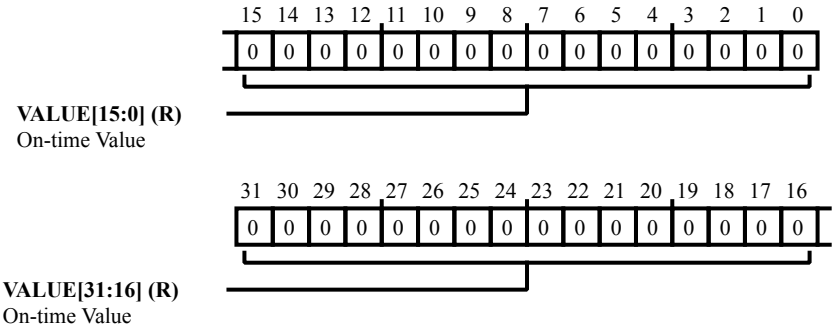


Figure 20-16: CPTMR_TON[n] Register Diagram

Table 20-20: CPTMR_TON[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	On-time Value.

21 Watchdog Timer (WDOG)

The processor has a 32-bit watchdog timer that can be used to verify system reliability by generating an event to the processor core when the watchdog expires before software updates it.

There are two watchdog timers, one that uses SYSCLK one that uses SCLK0.

WDOG Features

The watchdog timer has the following features:

- Programmable 32-bit watchdog count value
- 8-bit disable bit pattern
- General-purpose system event generation

The watchdog timer can supervise system software stability by periodically reloading it to prevent expiration of the downward-counting timer (such that the count never becomes 0). When used in this fashion, an expiring timer can indicate the system software is not running normally.

Expiration of the WDOG counter generates a general-purpose interrupt, which can be used in a variety of ways:

- as an interrupt vector sent through the System Event Controller (SEC) to the core to be serviced by a handler function, providing full software control of device resources (for example, GPIO management and reset control).
- as a fault condition through the SEC to provide hardware-automated:
    - signalling of the fault condition on external pins to the system,
    - system reset requests to the Reset Control Unit (RCU), and/or
    - trigger outputs (SEC0_FAULT trigger master) through the Trigger Routing Unit (TRU) to initiate activities in a variety of potential trigger slaves (for example, GPIO control).

To facilitate debugging, the system may optionally prevent the watchdog from decrementing during emulation halt.

WDOG Functional Description

When enabled, the 32-bit watchdog timer counts downward every SCLK0_0 cycle. If it reaches zero, the watchdog expiration event is generated, which can be used in many ways.

To start the watchdog timer:

1. Program the watchdog timeout period (in SCLK0_0 cycles) in the WDOG_CNT register. With the watchdog disabled, this write also preloads the WDOG_STAT register.
2. Enable the watchdog timer by writing any value other than 0xAD to the WDOG_CTL.WDEN field.

Once the watchdog is enabled, writes to the WDOG_CNT register are ignored. The counter begins decrementing, and the current counter value can be read from the 32-bit WDOG_STAT register at any time.

To prevent the counter from expiring, software must "kick" the watchdog by writing any value to the WDOG_STAT register while the current count is non-zero. While the value written is irrelevant and ignored, this action resets the current counter in the WDOG_STAT register to the programmed WDOG_CNT value, and decrementing continues. The internal counter continues decrementing until it reaches zero, at which point the expiration event is generated, and the WDOG_CTL.WDRO rollover bit is set.

Watchdog operation continues in this manner unless disabled by explicitly writing 0xAD to the WDOG_CTL.WDEN field.

The watchdog expiration event can be used in a variety of ways, as listed below.

- The watchdog expiration event itself is one of numerous interrupt sources that is managed by the System Event Controller. Like other peripheral sources, this event can be used to cause a vector to a handler function that executes based on interrupt priority.
- The watchdog expiration event can be used to initiate automated hardware response through the SEC Fault Interface (SFI).

For this, the WDOG expiry has to be configured as the fault source in the SEC. Then the response to the WDOG expiry can be set to any of the below:

- Signalling through the external fault pin.
- System reset
- Trigger outputs to numerous potential trigger slaves.

For further details on how watchdog expiration event can be used with the SFI, see Configuring the WDOG Expiry Event to Issue a System Reset.

CM41X_M4 WDOG Register List

The Watchdog Timer unit (WDOG) provides a software-based watchdog timer that can improve system reliability by generating an event to the processor core if the watchdog expires before being updated by software. A set of registers governs WDOG operations. For more information on WDOG functionality, see the WDOG register descriptions.

Table 21-1: CM41X_M4 WDOG Register List

Name	Description
WDOG_CNT	Count Register
WDOG_CTL	Control Register
WDOG_STAT	Watchdog Timer Status Register

CM41X_M4 WDOG Interrupt List

Table 21-2: CM41X_M4 WDOG Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
34	WDOG1_EXP	WDOG1 Expiration	Level	
35	WDOG0_EXP	WDOG0 Expiration	Level	

CM41X_M0 WDOG Interrupt List

Table 21-3: CM41X_M0 WDOG Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
9	WDOG0_EXP	WDOG0 Expiration	Level	

WDOG Block Diagram

The ***Watchdog Timer Block Diagram*** figure shows the detailed block diagram for the watchdog timer.

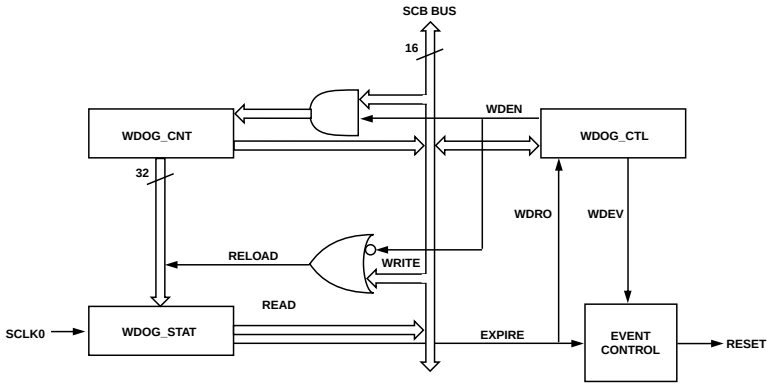


Figure 21-1: Watchdog Timer Block Diagram

Internal Interface

There are two watchdog units. One is clocked by SCLK0 and the other by SYSCLK The registers are accessed through the 32-bit peripheral MMR access bus. 32-bit read/write operations always access the 32-bit WDOG_CNT and WDOG_STAT registers. Hardware ensures that those accesses are atomic. When the counter expires, the WDOG expiration event is generated.

External Interface

The watchdog timer does not directly interact with any external pins.

CM41X_M4 WDOG Register Descriptions

Watchdog Timer Unit (WDOG) contains the following registers.

Table 21-4: CM41X_M4 WDOG Register List

Name	Description
WDOG_CNT	Count Register
WDOG_CTL	Control Register
WDOG_STAT	Watchdog Timer Status Register

Count Register

The WDOG_CNT register holds the programmable, unsigned count value. A valid write to this register also pre-loads the WDOG counter. For added safety, the WDOG_CNT register can be updated only when the WDOG timer is disabled. A write to the WDOG_CNT register while the timer is enabled does not modify the contents of this register. This register must be accessed with 32-bit read/writes only.

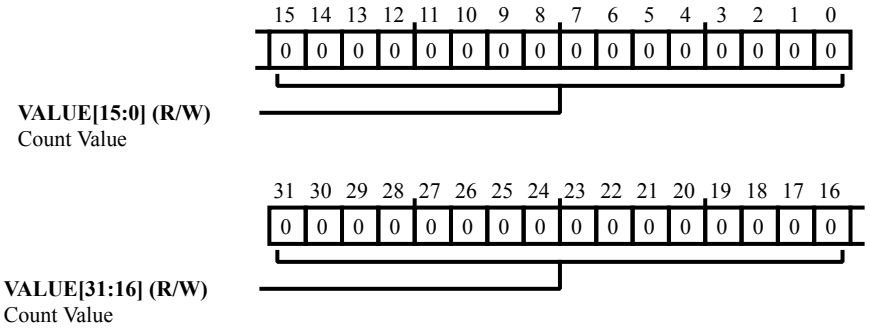


Figure 21-2: WDOG_CNT Register Diagram

Table 21-5: WDOG_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Count Value. The WDOG_CNT.VALUE bit field holds the programmable, unsigned count value.

Control Register

The WDOG_CTL register controls the watchdog timer. This register supports enabling/disabling the watchdog timer and supports checking the timer rollover status. Note that when the processor is in emulation mode, the watchdog timer counter will not decrement even if it is enabled.

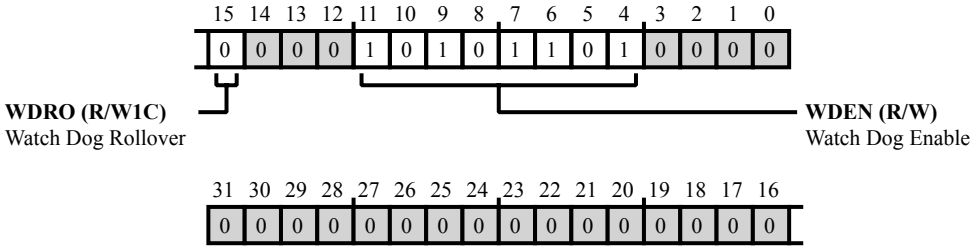


Figure 21-3: WDOG_CTL Register Diagram

Table 21-6: WDOG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	WDRO	Watch Dog Rollover. Software can determine whether the timer has rolled over by interrogating the WDOG_CTL.WDRO status bit. This is a sticky bit that is set whenever the watch dog timer count reaches 0 and cleared only by disabling the watch dog timer and then writing a 1 to the bit.	
		0	WDT Has Not Expired
		1	WDT Has Expired
11:4 (R/W)	WDEN	Watch Dog Enable. The WDOG_CTL.WDEN field is used to enable and disable the watchdog timer. Writing any value other than the disable value into this field enables the watchdog timer. This multi-bit disable key minimizes the chance of inadvertently disabling the watchdog timer.	
		173	Counter Disabled. All other values mean that the counter is enabled.

Watchdog Timer Status Register

The WDOG_STAT register contains the current count value of the watchdog timer. Reads of this register return the current count value. When the watchdog timer is enabled, the WDOG_STAT register is decremented by 1 on each SCLK cycle. When the count value reaches 0, the watchdog timer stops counting, and the expiry event is generated. The WDOG_STAT register is a 32-bit unsigned system MMR that must be accessed with 32-bit reads and writes.

Values cannot be stored directly in this register but are instead copied from the WDOG_CNT register. This copy process can happen in two ways:

- While the watchdog timer is disabled, writing the WDOG_CNT register pre-loads the WDOG_STAT register.
- While the watchdog timer is enabled, writing the WDOG_STAT register loads it with the value in the WDOG_CNT register.
- While the watchdog timer is disabled, writing to the WDOG_STAT register also reloads it with the value in the WDOG_CNT register.

When the processor executes a write (of an arbitrary value) to the WDOG_STAT register, the value in the WDOG_CNT register is copied into the WDOG_STAT register. Typically, software sets the value of WDOG_CNT at initialization, then periodically writes to the WDOG_STAT register before the watchdog timer expires. This reloads the watchdog timer with the value from WDOG_CNT and prevents generation of the expiry event.

If the program does not reload the counter before SCLK x count register cycles, an expiry event is generated, and the WDOG_CTL.WDRO bit is set. When this happens, the counter stops decrementing and remains at zero. If the counter is enabled with a zero loaded to the counter, the WDOG_CTL.WDRO bit is set immediately and the counter remains at zero and does not decrement.

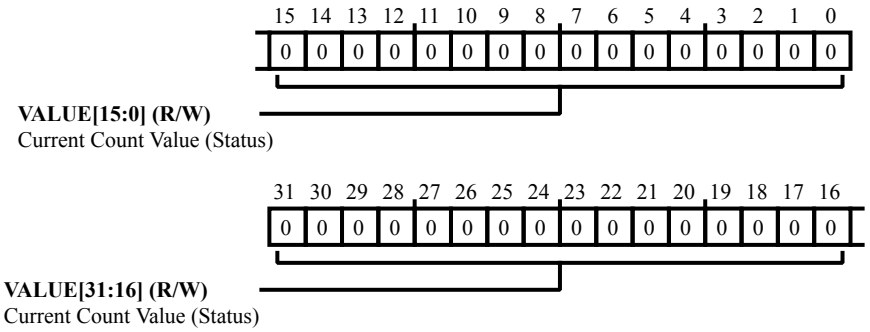


Figure 21-4: WDOG_STAT Register Diagram

Table 21-7: WDOG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Current Count Value (Status). The `WDOG_STAT.VALUE` bit field contains the current count value of the watchdog timer.

22 General-Purpose Counter (CNT)

The GP counter converts pulses from incremental position encoders into data that is representative of the actual position of the pulse. This conversion is done by integrating (counting) pulses on one or two inputs. Since integration provides relative position, some devices also feature a zero-position input (zero marker). The GP counter can use the zero position input feature to establish a reference point for verifying that the acquired position does not drift over time. In addition, the GP counter can use the incremental position information to determine speed, if the time intervals are measured.

The GP counter provides flexible ways to establish position information. When used with the GP timer block, the GP counter can allow for the acquisition of coherent position or time stamp information that enables speed calculation.

GP Counter Features

The GP counter includes the following features:

- 32-bit up or down counter
- Quadrature encode mode (Gray code)
- Binary encoder mode
- Alternative frequency-direction mode
- Timed direction and up or down counting modes
- Zero marker or push-button support
- Capture event timing in association with GP Timer
- Boundary comparison and boundary setting features
- M/N frequency scaling of the inputs CUD/CDG

GP Counter Functional Description

The ***GP Counter Block Diagram*** shows a block diagram of the GP counter. The CNT_UD and CNT_DG pins accept various forms of incremental inputs. The 32-bit counter processes the inputs. The GP counter uses the CNT_ZM pin to sense the pressing of a push button.

NOTE: When enabled, the GP counter requires 3 SCLK cycles of initialization before recognizing valid toggles on its input pins.

The three input pins can be filtered (debounced) before the GP counter evaluates them.

The GP counter features a flexible boundary comparison. In all of the operating modes, the counter can be compared to an upper and lower limit. It takes various actions when reaching these limits.

The module can optionally generate an interrupt request to the system through its IRQ line. On many processors, a GP timer module can use an output to generate time stamps on certain events.

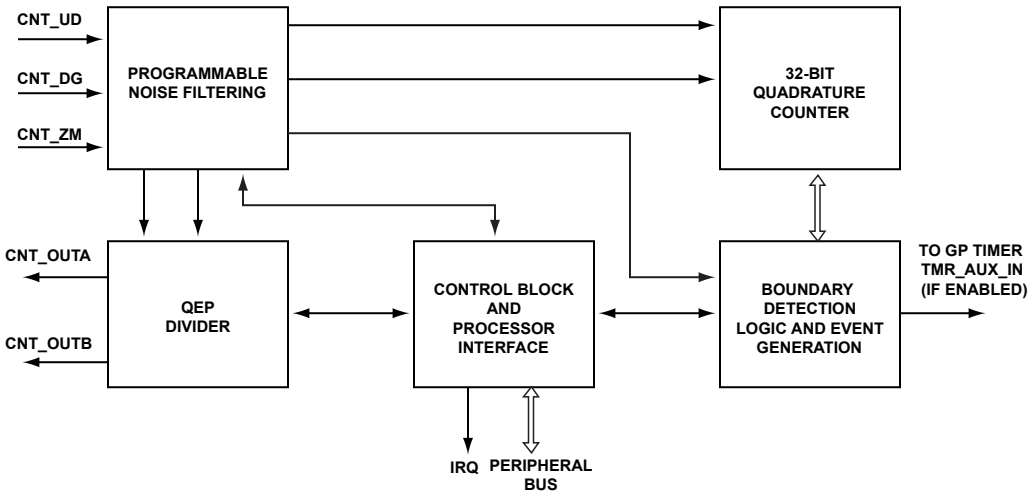


Figure 22-1: GP Counter Block Diagram

The inputs CNT_UD (A-In) and CNT_DG (B-In) to the rotary counter may be selectively connected to the following sources, under control of the SYSBLK_ROT_UPDN_CFG.SELA and SYSBLK_ROT_UPDN_CFG.SELB register bit fields with the following options.

- Input from the CNT_UD and CNT_DG pins
- LBA SYS_OUT outputs
- TRU Trigger slaves for CNT_UD and CNT_DG

CM41X_M4 CNT Register List

The GP Counter (CNT) provides support for manually controlled rotary controllers, such as the volume wheel on a radio device. This unit also supports industrial encoders.

The CNT converts pulses from incremental position encoders into data that is representative of the actual position. To complete this task, the CNT integrates (counts) pulses on one or two inputs. Because integration provides relative position, a zero position input (zero marker) is usually provided that establishes a reference point, verifying that the acquired position does not drift over time. The incremental position information may also be used to determine speed, if the relevant time intervals are measured. The CNT provides flexible ways to establish position information. When used in conjunction with the General-purpose Timer (TIMER), the CNT allows acquisition of coherent position/time stamp information, enabling speed calculation.

A set of registers govern CNT operations. For more information on CNT functionality, see the CNT register descriptions.

Table 22-1: CM41X_M4 CNT Register List

Name	Description
CNT_CFG	Configuration Register
CNT_CMD	Command Register
CNT_CNTR	Counter Register
CNT_DEBNCE	Debounce Register
CNT_IMSK	Interrupt Mask Register
CNT_MAX	Maximum Count Register
CNT_MDIV	M Value for Divider
CNT_MIN	Minimum Count Register
CNT_NDIV	N Value for Divider
CNT_STAT	Status Register

CM41X_M4 CNT Interrupt List

Table 22-2: CM41X_M4 CNT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
152	CNT0_STAT	CNT0 Status	Level	

CM41X_M4 CNT Trigger List

Table 22-3: CM41X_M4 CNT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
27	CNT0_STAT	CNT0 Status	Level
85	CNT0_TO	CNT0 Counter Output to Timer Block	Level
86	CNT0_OUTA	CNT0 Output Divider A	Level

Table 22-3: CM41X_M4 CNT Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
87	CNT0_OUTB	CNT0 Output Divider B	Level

Table 22-4: CM41X_M4 CNT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
78	CNT0_DG	CNT0 Count Down and Gate	Pulse
79	CNT0_UD	CNT0 Count Up and Direction	Pulse

GP Counter Operating Modes

The GP counter has the following five modes of operation.

1. Quadrature Encoder
2. Binary Encoder
3. Up/Down Counter
4. Direction Counter
5. Timed Direction

With the exception of the timed direction mode, the GP counter can operate with the GP timer block to capture additional timing information (time stamps) associated with events detected by this block.

Quadrature Encoder Mode

In this mode, the CNT_UD and CNT_DG inputs expect a quadrature-encoded signal that is interpreted as a two-bit Gray code. The order of transitions of the CNT_UD and CNT_DG inputs determines whether the counter increments or decrements. The CNT_CNTR register contains the number of transitions that have occurred as shown in the ***Quadrature Events and Counting Mechanism*** table. Optionally, an interrupt is generated when both inputs change within one SCLK cycle. Gray coding prohibits such transitions. Therefore, the CNT_CNTR register remains unchanged, and an error condition is signaled.

Table 22-5: Quadrature Events and Counting Mechanism

CNT_COUNTER Register Value	-4	-3	-2	-1	0	+1	+2	+3	+4
CDG, CUD Inputs	00	01	11	10	00	01	11	10	00

It is possible to reverse the count direction of the Gray-coded signal by enabling the polarity inverter of either the CNT_UD pin or the CNT_DG pin. Inverting both pins does not alter the behavior. The GP counter can enable this feature with the CNT_CFG.CDGINV and CNT_CFG.CUDINV bits.

As an example, the CNT_DG and CNT_UD inputs are 00 and the next transition is to 01. These inputs normally change the counter in increments as shown in the table. If the CNT_UD polarity is inverted, this condition generates a received input of 01 followed by 00. The results is a decrement of the counter, altering the behavior of the connected hardware.

Binary Encoder Mode

This mode is almost identical to quadrature encoder mode, with the exception that the CNT_UD: CNT_DG inputs expect a binary-encoded signal. The order of transitions of the CNT_UD and CNT_DG inputs determines whether the counter increments or decrements. The CNT_CNTR register contains the number of transitions that have occurred as shown in the ***Binary Events and Counting Mechanism*** table. Optionally, an interrupt is generated when the detected code steps by more than 1 (in binary arithmetic) within one SCLK cycle. Such transitions are erroneous. Therefore, the CNT_CNTR register remains unchanged, and an error condition is signaled.

Table 22-6: Binary Events and Counting Mechanism

CNT_COUNTER Register Value	-4	-3	-2	-1	0	+1	+2	+3	+4
CDG:CUD Inputs	00	01	10	11	00	01	10	11	00

Reversing the CNT_UD and CNT_DG pin polarity has a different effect in binary encoder mode than for the quadrature encoder mode. Inverting the polarity of the CNT_UD pin only, or inverting both the CNT_UD and CNT_DG pins, results in reversing the count direction.

Up/Down Counter Mode

In this mode, the counter increments or decrements at every active edge of the input pins. The GP counter uses the CNT_CFG.CUDINV bit to select an active edge and has the following results.

- If the GP counter module detects an active edge at the CNT_UD input, the counter increments.
- If the GP counter module detects an active edge at the CNT_DG input, the counter decrements.
- If simultaneous edges occur on the CNT_DG and CNT_UD pins, the counter remains unchanged, and both up-count and down-count events are signaled in the CNT_STAT register.

Direction Counter Mode

In this mode, the counter is incremented or decremented at every active edge of the CNT_DG input pin. The state of the CNT_UD input determines whether the counter increments or decrements. The GP counter uses the CNT_CFG.CUDINV bit to select the polarity.

If the GP counter detects an active edge at the CNT_DG input, the counter value changes by one in the selected direction.

Timed Direction Mode

In this mode, the counter is incremented or decremented at each SCLK cycle. The state of the CNT_UD input determines whether the counter increments or decrements. The GP counter uses the CNT_CFG.CUDINV bit to select

the polarity. The CNT_DG pin can be used to gate the clock. The GP counter uses the CNT_CFG.CDGINV bit to select the polarity.

M/N Scaling

The GP counter provides programmable M/N frequency scaling of the CNT_UD and CNT_DG inputs. It supports frequency scaling only when the inputs are in quadrature encoded mode, with support for both incrementing and decrementing gray code, and with on-the-fly direction changing. The divided outputs are available on the GP counter output pins CNT_OUTA and CNT_OUTB.

To use M/N frequency scaling, set the CNT_CFG.DIVEN bit and program the M and N values in the CNT_MDIV and CNT_NDIV registers. With division enabled, the output frequency on the CNT_OUTA and CNT_OUTB pins is:

$f_{QOUT} = f_{QIN} \times (M/N)$

Where the frequency of the inputs on the CNT_UD and CNT_DG input pins is:

$f_{QIN}=1/t_{PERIOD}$

The ***M/N Scaling*** figure shows the M/N scaling implemented by the QEP dividers.

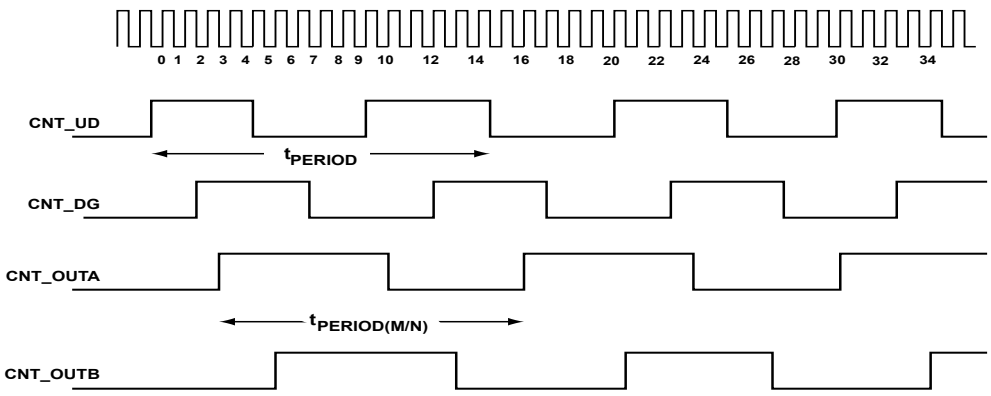


Figure 22-2: M/N Scaling

The divided outputs (CNT_OUTA and CNT_OUTB pins) start after 1 QEP input pulse period as shown in the figure (and 1 or 2 system clocks (SCLK) based on the implementation). If debouncing is enabled and the CNT_CFG.DIVNTV bit is not set, then the divided outputs start after 1 QEP pulse + the debouncing period (t_{FILTER}).

The ***M/N Scaling*** figure shows the full period scaled M/N (t_{PERIOD}/(M/N). However, the value for scaling is measured from the rising edge of either of the inputs (CNT_UD or CNT_DG) to the rising edge of the other input. This value gives the width of the QEP pulse and the value measured is scaled to create the M/N version. Each QEP pulse is continuously measured and the values are constantly updated to create the divided outputs. The QEP width seen on the output is not the exact M/N- scaled version of a measured QEP input width because the input frequency changes. There is not a one-to-one correspondence for the QEP measured-width and the actual QEP M/N output.

This difference is because the design constantly updates the measured values to the dividers. The output reflects a type of weighted-average of the multiple QEP pulses measured. The ***Scaled Outputs with Increasing Frequency at Inputs (Increasing Motor Speed)*** shows waveforms for an increasing input frequency.

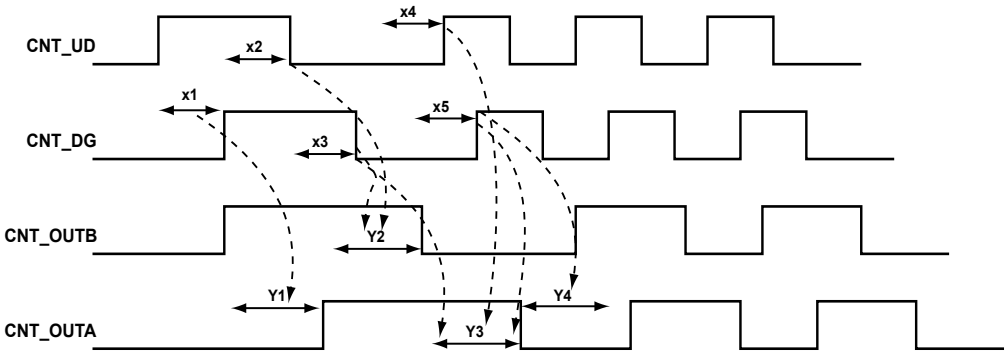


Figure 22-3: Scaled Outputs with Increasing Frequency at Inputs (Increasing Motor Speed)

In the ***Scaled Outputs with Increasing Frequency at Inputs*** figure, X1, X2, X3, X4 and X5 are the input QEP widths measured. (After X5 measurements, the widths remain constant. This period is also known as frequency stabilized.) Y1, Y2, Y3 and Y4 are the QEP output widths (and all outputs keep to Y4 as the input frequency stabilized). Importantly,

- Y1 = X1/(M/N)
- Y2 = (Weighted Average of {X2 ,X3})/(M/N)
- Y3 = (Weighted Average of {X3, X4, X5})/(M/N)
- Y4 = X5/(M/N)

The weight applied for each measurement depends on when the measurement completes based on the outputs. It is difficult to predict the exact weight. Similar behavior occurs with the decreasing frequency of inputs.

Set the `CNT_CFG.DIVMODE` bit to avoid weighted averaging. Once the bit is set, the output values are:

- Y1 = X1/(M/N)
- Y2 = X2/(M/N)
- Y3 = X3/(M/N)
- Y4 = X5/(M/N)

The default behavior of the QEP dividers is weighted-averaging.

The maximum error in the input width measured by the QEP dividers is ± 1 SCLK cycles. The maximum error on the QEP divided outputs is ± N/M SCLK cycles.

NOTE: The count of output pulses created by this M/N scaling is not always M/N times the input pulses. The input pulse counts are scaled N/M without many errors for large values of M/N. For small values, there are errors. These errors can accumulate as the number of input pulses increases.

M/N Stop Detection

If there are no QEP input pulses after the last measured and translated QEP input pulse, then the QEP divider responds as if the input (and therefore, the motor) has stopped. Once the stop condition is detected, the QEP divider-outputs do not switch. A W1C status bit (`CNT_STAT.STP`) is set in the status register, and an interrupt is generated when the `CNT_IMSK.STP` bit is unmasked. Once the input QEP pulses restart, the QEP output pulses restart after a delay of 1 QEP pulse.

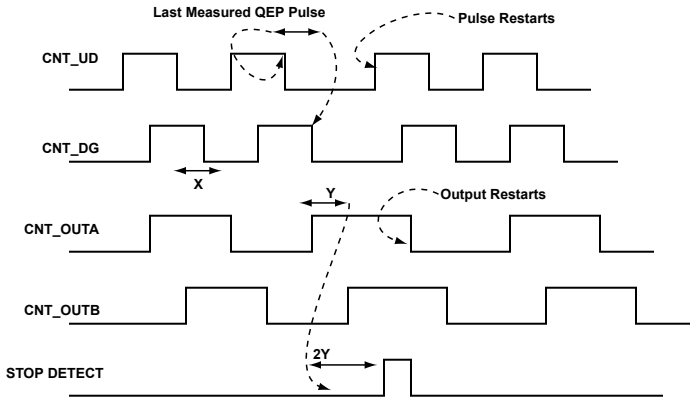


Figure 22-4: M/N Stop Detection

NOTE: The figure depicts an N/M ratio of 3/2.

$$Y = X \times N/M$$

If there are no QEP pulses seen during the period 2Y, then the QEP dividers assume that inputs have stopped. The QEP output does not toggle until a new input pulse is identified. Even if there are errors such as trembles or illegal Gray code that occur during this period, the stop condition is still detected. The error cases are not treated as a valid QEP pulse.

If the input width varies from one QEP pulse to the other more by more than 2Y, then a false stop is detected. Therefore, the maximum supported deceleration rate is from X to 2Y. This rate translates to a width-change of X to 2X × N/M between two QEP pulses. Using a worst case of N/M = 1, the maximum a pulse width can change is twice the size of the previous pulse. So, the allowable width of a QEP pulse following a 20 µs pulse is <40 µs (with N/M=1). Similarly, if there is a 1 ms QEP, the pulse that follows is within 2 ms. If not, a stop condition is detected at N/M=1.

The stop detection interrupt or status is also raised when a QEP pulse is wide and caused the internal counter to overflow. The internal counter-overflow occurs if the pulse is larger than 4,026,531,840 × M (or hex (0xF000_0000) × M) system clock cycles.

NOTE: If new QEP pulses are identified with a different direction after a STOP is detected, no direction-change interrupt occurs. (The dividers assume that the pulse is new start after the STOP detection.) However, the QEP dividers start providing the output with the new direction.

M/N Error Condition

If the CNT_UD/CNT_DG does not follow the quadrature encoder mode, then the outputs continue to run in quadrature encoder mode using the QEP input width measured before the error condition. (The error is ignored.) Although the output continues to run, the counter raises an illegal Gray code interrupt when the inputs are not in quadrature encode mode with CNT_CFG.CNTMODE set to QUAD_ENC mode.

To stop the QEP dividers on an error condition, disable the CNT_CFG.DIVEN bit to three-state the CNT_OUTA/CNT_OUTB divided outputs immediately.

M/N Restrictions

The following restrictions apply to the programmable M and N values.

a) M, N values cannot be changed on-the-fly (when the CNT_CFG.DIVEN bit =1).

b) $M<f_{SYSCLK}/f_{QIN(max)}$, where f_{QIN} is input frequency, f_{SYSCLK} is the frequency of the system clock.

c) N is greater than or equal to M ($n \geq M$)

d) N is greater than 1 (N>1)

e) M is greater than 1 (M>1)

If $M>f_{SYSCLK}/f_{QIN(max)}$, then an error status bit (CNT_STAT.MERR) is set and an interrupt is generated (if the interrupt enable bit (CNT_IMSK.MERR) is set). This sequence of events occurs each time the QEP input pulse frequency does not meet the requirement after the end of each QEP input pulse.

The ***Max M Values*** table shows the maximum M values for f_{SYSCLK} = 100 MHz.

Table 22-7: Max M Values

f_{QIN}(KHz)	M
6666.66	15
2000.0	50
333.33	300
166.67	600

GP Counter Programming Model

The following sections provide information for programming the interface.

GP Counter General Programming Flow

The following are general guidelines for configuring and enabling the GP counter.

1. Initialize (but do not enable) the GP counter for the desired mode and settings through the CNT_CFG register.

2. Usually, events of interest are processed using interrupts rather than by polling status bits. In this case, clear all status bits and activate the interrupt generation requests with the CNT_IMSK register.
3. Configure interrupts at the system level to insure desired interrupt signaling to the system.
4. If timing information is required, set up the relevant GP Timer in width capture mode.
5. Finally, enable interrupt requests and the GP Counter itself using the CNT_IMSK and CNT_CFG registers, respectively.

M/N Scaling Programming Guidelines

To use the QEP dividers, set both the CNT_CFG.EN and CNT_CFG.DIVEN bits. Note that the counter must be enabled at least two system clock (SCLK) cycles before enabling the rotary dividers (set CNT_CFG.EN at least 2 SCLK cycles before setting the CNT_CFG.DIVEN bit. To detect an illegal gray code error condition, program the CNT_CFG.CNTMODE for quadrature encode mode (QUAD_ENC).

The polarity bits (CNT_CFG.CDGINV and CNT_CFG.CUDINV) do not invert the QEP divider outputs, the QEP divider output polarity is the same as the input polarity. However the polarity bits cannot be changed on the fly when the rotary dividers are running (the polarity bits should not be changed when the CNT_CFG.DIVEN bit =1). If this occurs, it may be treated as an error or a direction change by the QEP dividers. Do not change the CNT_CFG bits on the fly when QEP dividers are enabled.

GP Counter Mode Configuration

The GP counter can use the CNT_ZM input pin to sense the zero marker output of a rotary device or to detect the pressing of a push button. There are four programming schemes, which are functional in all counter modes:

- Push-button mode
- Zero-marker-zeros-counter mode
- Zero-marker-error mode
- Zero-once mode

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation:

1. Set CNT_IMSK.CZM to enable (unmask) the zero marker interrupt.
2. Select the active edge polarity through the CNT_CFG.CZMINV bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

An active edge at the CNT_ZM input sets the CNT_IMSK.CZM bit.

Configuring Zero-Marker-Zeros-Counter Mode

The following provides information on configuring zero-marker-zeros-counter mode for the GP counter.

1. Set CNT_IMSK.CZMZ to enable CNT_CNTR. The zero marker interrupt zeroes the counter.
2. Set CNT_CFG.ZMZC to enable ZMZC mode.
3. Select the active edge polarity through the CNT_CFG.CZMINV bit.
4. Proceed with any other desired configuration steps and enable the peripheral.

This configuration causes an active level at the CNT_ZM pin to clear the CNT_CNTR register and keep it cleared until the CNT_ZM pin is deactivated. In addition, the CNT_STAT.CZMZ bit is set.

Configuring Zero-Marker-Error Mode

The GP counter uses this mode to detect discrepancies between counter-value and the zero marker output of certain rotary encoder devices.

1. Set the CNT_STAT.CZME bit to enable this mode.
2. Select the active edge of the CNT_ZM pin through the CNT_CFG.CZMINV bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

When the GP counter detects an active edge at the CNT_ZM input pin, it compares the four LSBs of the CNT_CNTR register to zero. If they are not zero, the GP counter uses CNT_STAT.CZME bit to signal a mismatch.

Configuring Zero-Once Mode

The GP counter uses this mode to perform an initial reset of the counter-value when it detects an active zero marker. After that, the zero marker is ignored (the counter is no longer reset).

1. Set the CNT_CMD.W1ZMONCE bit to enable this mode.
2. Select the active edge of the CNT_ZM pin through the CNT_CFG.CZMINV bit.
3. Ensure that at least one of the following bits is enabled: CNT_IMSK.CZM, CNT_IMSK.CZME, CNT_IMSK.CZMZ.
4. Proceed with any other desired configuration steps and enable the peripheral.

The CNT_CNTR register and the CNT_CMD.W1ZMONCE bit are cleared on the next active edge of the CNT_ZM pin. Now the CNT_CMD.W1ZMONCE bit can be read to check whether the event has already occurred.

Configuring Boundary Auto-Extend Mode

In this mode, hardware modifies the boundary registers (CNT_MIN and CNT_MAX) whenever the CNT_CNTR value reaches either of them. The GP counter uses this mode to monitor the widest angle a thumb wheel even if the software did not generate interrupts.

1. Initialize CNT_CNTR with the desired value.
2. Set both CNT_MIN and CNT_MAX to this same value.

3. Configure the CNT_CFG.BNDMODE field for auto extend mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

The CNT_MAX register is loaded with the current CNT_CNTR value when the latter increments beyond the CNT_MAX value. Similarly, the CNT_MIN register is loaded with the CNT_CNTR value when the latter decrements below the CNT_MIN value. The CNT_STAT.MAXC and CNT_STAT.MINC status bits are set when the CNT_CNTR value matches the respective boundary register value.

Configuring Boundary Capture Mode

In this mode, the CNT_CNTR value is latched into the CNT_MIN register at one detected edge of the CNT_ZM input pin, and latched into the CNT_MAX boundary register at the opposite edge.

1. To capture the CNT_ZM pin rising edge into CNT_MIN and the falling edge into CNT_MAX, program CNT_CFG.CZMINV for active high polarity. Conversely, to capture the CNT_ZM pin falling edge into CNT_MIN and the rising edge into CNT_MAX, program CNT_CFG.CZMINV for active low polarity.
2. Program the CNT_IMSK.MAXC and CNT_IMSK.MINC interrupt mask bits according to interrupt generation requirements.
3. Configure the CNT_CFG.BNDMODE field for boundary capture mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

The CNT_STAT.MAXC and CNT_STAT.MINC status bits report the capture event, depending on how interrupt masks are configured.

Configuring Boundary Compare and Boundary Zero Modes

In these modes, the two boundary registers (CNT_MAX and CNT_MIN) are compared to the value in the CNT_CNTR register.

1. Program CNT_MAX and CNT_MIN registers with appropriate upper and lower range values.
2. Program the CNT_IMSK.MAXC and CNT_IMSK.MINC interrupt mask bits according to interrupt generation requirements.
3. Configure the CNT_CFG.BNDMODE field for boundary compare mode.
4. Proceed with any other desired configuration steps and enable the peripheral.

If after incrementing, CNT_CNTR = CNT_MAX, then the CNT_STAT.MAXC bit is set. Similarly if after decrementing, CNT_CNTR = CNT_MIN, then the CNT_STAT.MINC bit is set.

Additionally, for boundary zero mode, the counter-value in CNT_CNTR is set to zero. The CNT_STAT.MAXC and CNT_STAT.MINC bits are not set when software updates the CNT_MAX or CNT_MIN registers.

Configuring GP Counter Push-Button Operation

Use the following procedure to configure push-button operation:

1. Set CNT_IMSK.CZM to enable (unmask) the zero marker interrupt.
2. Select the active edge polarity through the CNT_CFG.CZMINV bit.
3. Proceed with any other desired configuration steps and enable the peripheral.

An active edge at the CNT_ZM input sets the CNT_IMSK.CZM bit.

GP Counter Programming Concepts

Using the features, operating modes, and event control for the GP counter to their greatest potential requires an understanding of some GP counter-related concepts. Some key aspects to consider are input noise filtering and capturing timing information.

CNT Input Noise Filtering

In all modes, the three input pins can be filtered to present clean signals to the GP counter logic. The GP counter uses the CNT_CFG.DEBEN bit to enable or disable this filtering. The ***Programmable Noise Filtering*** figure shows the filtering operation for the CNT_UD pin.

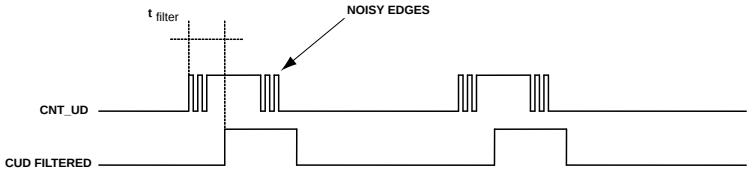


Figure 22-5: Programmable Noise Filtering

The CNT module implements the filtering mechanism using counters for each GP counter pin, where each counter is initialized from the CNT_DEBNCE.DPRESCALE field. When a transition is detected on a pin, the corresponding counter starts counting up to the programmed number of SCLK cycles. The state of the pin is latched after time t_{filter} and passed on to the GP counter logic.

The following formula determines the time t_{filter}, given SCLK and the CNT_DEBNCE.DPRESCALE value, where lower values of CNT_DEBNCE.DPRESCALE result in shorter debounce delays:

$$t_{filter} = 128 \times (2^{DPRESCALE} \times SCLK)$$

Capturing Counter Interval and CNT_CNTR Read Timing

When the count speed is low, it is often useful to capture the time elapsed since the last count event. Program the TIMER_TMR[n]_CFG register of the associated GP timer in a width capture mode with the following bit settings.

- TIMER_TMR[n]_CFG.PULSEHI = 0
- TIMER_TMR[n]_CFG.TMODE = b#1011
- TIMER_TMR[n]_CFG.TINSEL = 1

The ***Capture Intervals*** figure shows and the following list describe the operation of the GP counter and the GP timer in this mode.

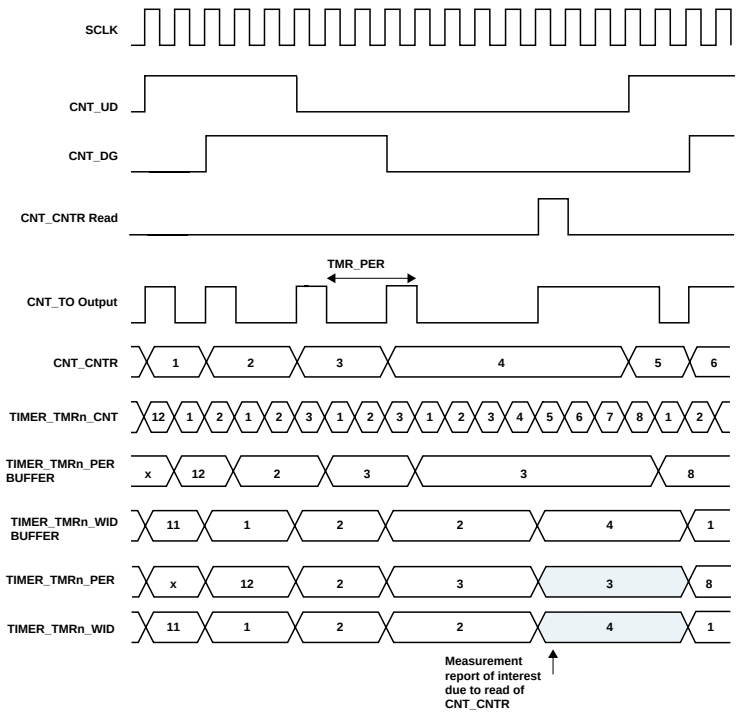


Figure 22-6: Capture Intervals

NOTE: SCLK in the ***Capture Intervals*** figure is SCLK.

1. The CNT_TO signal generates a pulse every time a count event occurs. In addition, when the processor reads the CNT_CNTR register, the CNT_TO signal presents a pulse which is extended (high) until the next count event.
2. The GP timer updates its TIMER_TMR[n]_PER register with the period (measured from falling edge to falling edge, because TIMER_TMR[n]_CFG.PULSEHI = 0) of the CNT_TO signal.
3. The TIMER_TMR[n]_WID register is updated with the pulse width (the portion where CNT_TO is low, again because TIMER_TMR[n]_CFG.PULSEHI = 0).
4. Both registers are updated at every rising edge of the CNT_TO signal (because TIMER_TMR[n]_CFG.TMODE= b#011).

The TIMER_TMR[n]_PER register contains the period between the last two count events. The TIMER_TMR[n]_WID register contains the time since the last count event and the read of the CNT_CNTR register, both measured in SCLK cycles.

Read the CNT_CNTR register to latch the two time measurements, providing a coherent triplet of information to calculate speed and position.

NOTE: Speed restrictions apply to the use of the CNT_TO signal. Therefore, programs must not operate at count event rates that are high. For instance, if CNT_CNTR is incremented or decremented every SCLK cycle (timed direction mode), the CNT_TO signal is not valid.

Capturing Time Interval Between Successive Counter Events

When the required timing information is the interval between successive count events, program the associated timer in a width capture mode. Set the TIMER_TMR[n]_CFG bit of TIMER_TMR[n]_CFG.PULSEHI = 1, TIMER_TMR[n]_CFG.TMODE = b#1010 and TIMER_TMR[n]_CFG.TINSEL = 1. Typically, this information is sufficient if the speed of GP counter events does not to reach low values.

The ***Period Register Timing*** figure shows the operation of the GP counter and the GP timer in this mode.

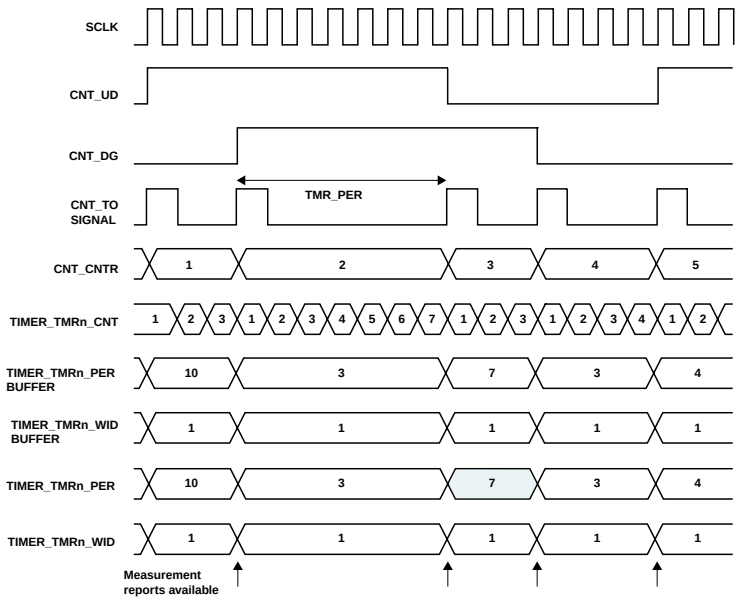


Figure 22-7: Period Register Timing

The CNT_TO signal generates a pulse every time a count event occurs. The GP timer updates its TIMER_TMR[n]_PER register with the period (measured from rising edge to rising edge) of the CNT_TO signal. The TIMER_TMR[n]_PER register is updated at every rising edge of the CNT_TO signal and contains the number of SCLK cycles that have elapsed since the previous rising edge.

Incidentally, the TIMER_TMR[n]_WID register is also updated at the same time, but is generally of no interest in this mode of operation. If no reads of the CNT_CNTR register occur between counter events, the TIMER_TMR[n]_WID register only contains the width of the CNT_TO pulse. If a read of CNT_CNTR has occurred between events, the TIMER_TMR[n]_WID register contains the time between the read of CNT_CNTR and the next event.

This mode can also be used with TIMER_TMR[n]_CFG.PULSEHI = 0. In this case, the period of CNT_TO is measured between falling edges. It results in the same values as in the previous case, only the latching occurs one SCLK cycle later.

GP Counter Event Control

Eleven events can be signaled to the processor using status information and optional interrupt requests. The GP counter uses the respective bits in the CNT_IMSK register to enable the interrupt requests. It uses dedicated bits in the CNT_STAT register to report events. When an interrupt request from the GP counter is serviced, the application software is responsible for correct interpretation of the events. It is recommended to logically AND the content of the CNT_IMSK and CNT_STAT registers to identify pending interrupt requests.

Perform a write-one-to-clear (W1C) operation to the CNT_STAT register to clear the interrupt requests. Hardware does not clear the status bits automatically, unless the counter module is disabled.

The following sections describe the events associated with the GP counter.

Illegal Gray and Binary Code Events

When illegal transitions occur in quadrature encoder or binary encoder modes, the CNT_STAT.IC bit is set. If enabled by the CNT_STAT.IC bit, the counter module generates an interrupt request. Set the CNT_STAT.IC bit only in the quadrature encoder or binary encoder modes.

Up/Down Count Events

The GP counter uses the CNT_STAT.UC bit to indicate whether the counter has been incremented. Similarly, the CNT_STAT.DC bit reports decrements. The two events are independent. For instance, if the counter increments by one and then decrements by two, both bits remain set, even though the resulting counter-value shows a decrement by one.

In up/down counter mode, hardware can detect simultaneous active edges on the CNT_UD and CNT_DG inputs. In that case, the CNT_CNTR remains unchanged, but both the CNT_STAT.UC and CNT_STAT.DC bits are set. Interrupt requests for these events can be enabled through the CNT_IMSK.UC and CNT_IMSK.DC bits. Use this feature carefully when the counter is clocked at high rates. This suggestion is especially critical when the counter operates in DIR_TMR mode, as interrupts are generated every SCLK cycle.

These events can also be used for more push buttons, when GP counter features are unnecessary. When up/down counter mode is enabled, the GP counter can use these count events to report interrupts from push buttons that connect to the CNT_UD and CNT_DG inputs.

Zero-Count Events

The CNT_STAT.CZERO status bit indicates that the CNT_CNTR has reached a value equal to 0x0000 0000 after an increment or decrement. This bit is not set when the counter value is set to zero by a write to CNT_CNTR or by setting the CNT_CMD.W1LCNTZERO bit. If enabled by the CNT_IMSK.CZERO bit, the GP counter module generates an interrupt request.

Overflow Events

There are two status bits that indicate whether the signed counter-register has overflowed from a positive to a negative value or conversely. The CNT_STAT.COV31 bit reports that the 32-bit CNT_CNTR register has either incremented from 0x7FFF FFFF to 0x8000 0000, or decremented from 0x8000 0000 to 0x7FFF FFFF.

If enabled by the CNT_IMSK.COV31 bit, an interrupt request is generated. Similarly, in applications where only the lower 16 bits of the counter are of interest, the CNT_STAT.COV15 status bit reports counter transitions from 0xXXXX 7FFF to 0xXXXX 8000, or from 0xXXXX 8000 to 0xXXXX 7FFF. If enabled by the CNT_IMSK.COV15 bit, an interrupt request is generated.

Boundary Match Events

The CNT_STAT.MINC and CNT_STAT.MAXC status bits report boundary events as described in Configuring Boundary Capture Mode. These bits are not set if the software updates the CNT_CNTR, CNT_MAX, or CNT_MIN registers or writes to the CNT_CMD register. The CNT_IMSK.MINC and CNT_IMSK.MAXC bits enable interrupt request generation on boundary events.

Zero Marker Events

The CNT_STAT.CZM, CNT_STAT.CZME, and CNT_STAT.CZMZ bits are associated with zero marker events, as described in Configuring GP Counter Push-Button Operation. Each of these events can optionally generate an interrupt request, when enabled by the corresponding CNT_IMSK.CZM, CNT_IMSK.CZME and CNT_IMSK.CZMZ bits.

CM41X_M4 CNT Register Descriptions

CNT (CNT) contains the following registers.

Table 22-8: CM41X_M4 CNT Register List

Name	Description
CNT_CFG	Configuration Register
CNT_CMD	Command Register
CNT_CNTR	Counter Register
CNT_DEBNCE	Debounce Register
CNT_IMSK	Interrupt Mask Register
CNT_MAX	Maximum Count Register
CNT_MDIV	M Value for Divider
CNT_MIN	Minimum Count Register
CNT_NDIV	N Value for Divider
CNT_STAT	Status Register

Configuration Register

The CNT_CFG register configures counter modes, configures input pins, and enables the CNT.

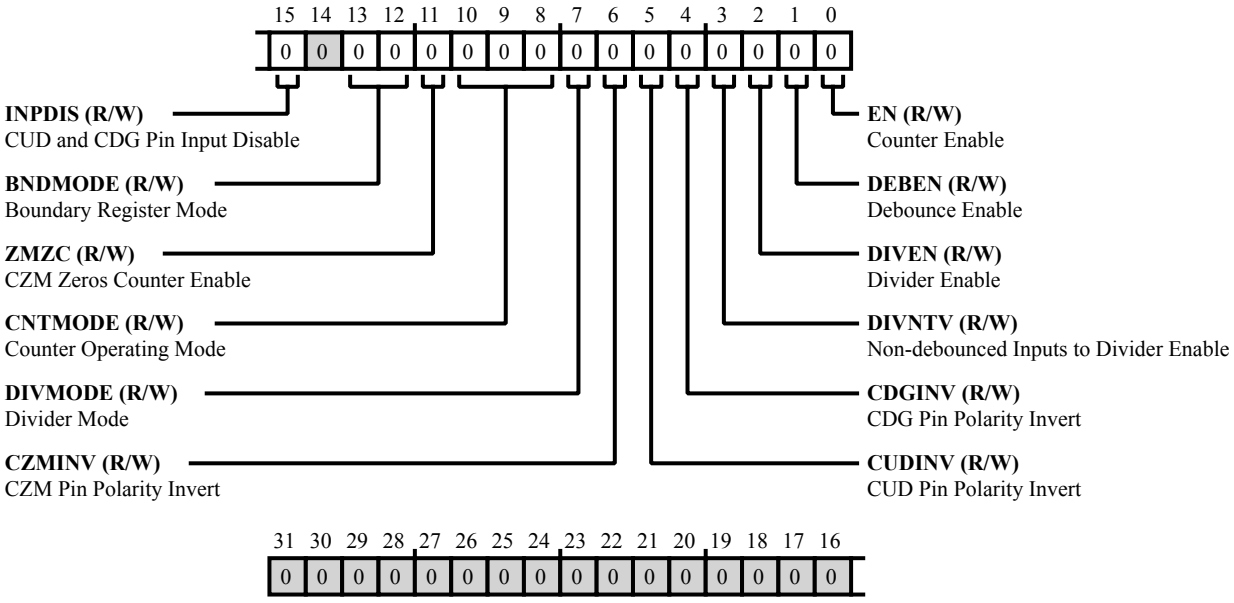


Figure 22-8: CNT_CFG Register Diagram

Table 22-9: CNT_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	INPDIS	CUD and CDG Pin Input Disable. The CNT_CFG.INPDIS disables or enables the CNT_UD input pin and the CNT_DG pin.	
		0	Enable
		1	Pin Input Disable
13:12 (R/W)	BNDMODE	Boundary Register Mode. The CNT_CFG.BNDMODE bit field selects the mode for the CNT_MIN and CNT_MAX boundary registers.	
		0	BND_COMP. Boundary Compare Mode
		1	BND_ZERO. Boundary Zero Mode
		2	BND_CAPT. Boundary Capture Mode
		3	BND_AEXT. Boundary Auto-extend Mode

Table 22-9: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	ZMZC	CZM Zeros Counter Enable. The CNT_CFG.ZMZC bit enables or disables level sensitive - active CNT_ZM pin operation to zero the CNT_CNTR register.	
		0	Disable
		1	Enable
10:8 (R/W)	CNTMODE	Counter Operating Mode. The CNT_CFG.CNTMODE bit field selects the operating mode for the CNT_UD input pin and the CNT_DG pin.	
		0	QUAD_ENC. Quadrature Encoder Mode
		1	BIN_ENC. Binary Encoder Mode
		2	UD_CNT. Rotary Counter Mode
		4	DIR_CNT. Direction Counter Mode
		5	DIR_TMR. Direction Timer Mode
7 (R/W)	DIVMODE	Divider Mode. The CNT_CFG.DIVMODE bit selects between weighted and non weighted averaging. This bit is used in M/N Scaling mode.	
		0	Weighted Average Division
		1	Non Weighted Division
6 (R/W)	CZMINV	CZM Pin Polarity Invert. The CNT_CFG.CZMINV bit selects the polarity for the CNT_ZM pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge
5 (R/W)	CUDINV	CUD Pin Polarity Invert. The CNT_CFG.CUDINV bit selects the polarity for the CNT_UD pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge

Table 22-9: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	CDGINV	CDG Pin Polarity Invert. The CNT_CFG.CDGINV bit selects the polarity for the CNT_DG pin. This polarity must be configured before the counter is enabled. It must not change on-the-fly while the counter is enabled.	
		0	Active High, Rising Edge
		1	Active Low, Falling Edge
3 (R/W)	DIVNTV	Non-debounced Inputs to Divider Enable. The CNT_CFG.DIVNTV bit enables non-debounced inputs to the divider.	
		0	Disable
		1	Enable
2 (R/W)	DIVEN	Divider Enable. The CNT_CFG.DIVEN bit enables the divider.	
		0	Disable
		1	Enable
1 (R/W)	DEBEN	Debounce Enable. The CNT_CFG.DEBEN bit enables or disables CNT input debounce filtering operation selected with the CNT_DEBNCE register.	
		0	Disable
		1	Enable
0 (R/W)	EN	Counter Enable. The CNT_CFG.EN bit enables or disables CNT operation.	
		0	Counter Disable
		1	Counter Enable

Command Register

The CNT_CMD register configures the CNT, enabling operations such as zeroing a counter register and copying or swapping boundary registers. These actions are taken by setting the appropriate bit.

Read operations from this register do not return meaningful values, with the exception of the CNT_CMD.W1ZMONCE bit, where a set bit indicates that the bit has been set by software before, but a zero marker event has not yet been detected on the CNT_ZM pin yet. For more information, see the CNT functional description.

The CNT_CNTR, CNT_MIN, and CNT_MAX registers can be initialized to zero by setting the CNT_CMD.W1LCNTZERO, CNT_CMD.W1LMINZERO, and CNT_CMD.W1LMAXZERO bits. In addition to clearing registers, the CNT_CMD register permits modifying the CNT_MIN and CNT_MAX boundary registers in a number of ways. The current counter value in the CNT_CNTR register can be captured and loaded into either of the two boundary registers to create new boundary limits. This operation is performed by setting the CNT_CMD.W1LMAXCNT and CNT_CMD.W1LMINCNT bits. Alternatively, the counter can be loaded from CNT_MAX or CNT_MIN using the CNT_CMD.W1LCNTMAX and CNT_CMD.W1LCNTMIN bits. It is also possible to transfer the current CNT_MAX value into CNT_MIN (or conversely) through the CNT_CMD.W1LMINMAX and CNT_CMD.W1LMAXMIN bits.

Another counter operation is the ability to only have the zero marker clear the CNT_CNTR register once. For more information, see the CNT functional description.

It is possible for multiple actions to be performed simultaneously by setting multiple bits in the CNT_CMD register. However, there are restrictions. The bits associated with each command have been grouped together such that all bits that involve a write to the CNT_CNTR, CNT_MAX, or CNT_MIN registers are located within bits 4-bit groups of the CNT_CMD register.

Note that a maximum of three commands can be issued at any one time, excluding the CNT_CMD.W1ZMONCE command. Also, note that CNT_CMD.W1LCNTMIN, CNT_CMD.W1LCNTMAX, and CNT_CMD.W1LCNTZERO bits have to be used exclusively. Never set more than one of them at the same time.

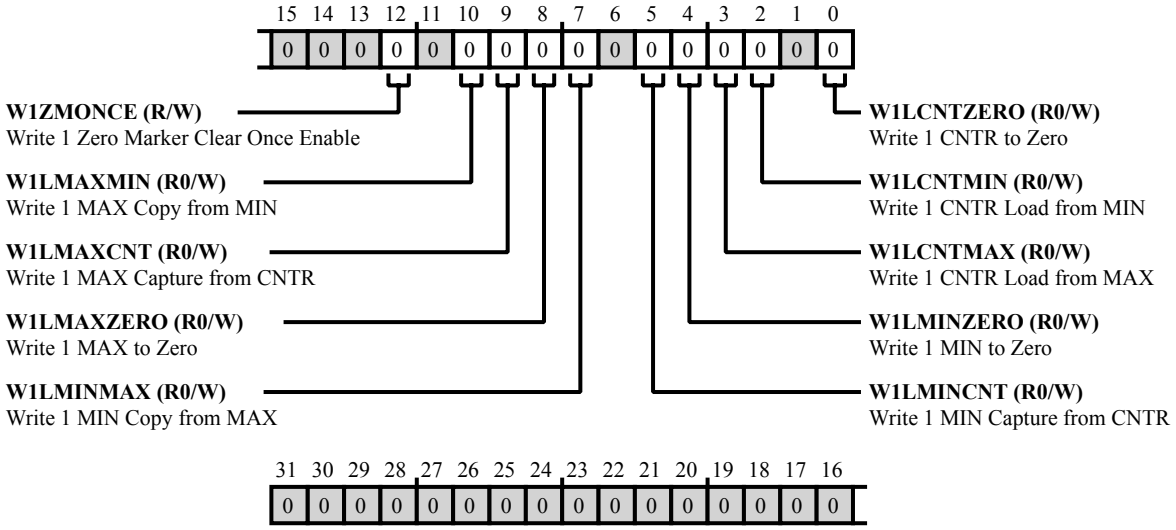


Figure 22-9: CNT_CMD Register Diagram

Table 22-10: CNT_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	W1ZMONCE	Write 1 Zero Marker Clear Once Enable. The CNT_CMD.W1ZMONCE enables a single zero marker clear of the CNT_CNTR register. Reading a 1 in this bit indicates that the bit has been set by software before, but no zero marker event has been detected on the CNT_ZM pin yet.
10 (R0/W)	W1LMAXMIN	Write 1 MAX Copy from MIN. The CNT_CMD.W1LMAXMIN bit transfers the current CNT_MIN register value into CNT_MAX register.
9 (R0/W)	W1LMAXCNT	Write 1 MAX Capture from CNTR. The CNT_CMD.W1LMAXCNT bit loads the current value in the CNT_CNTR register into the CNT_MAX register to create a new boundary limit.
8 (R0/W)	W1LMAXZERO	Write 1 MAX to Zero. Writing a 1 to the CNT_CMD.W1LMAXZERO bit clears the CNT_MAX register.
7 (R0/W)	W1LMINMAX	Write 1 MIN Copy from MAX. The CNT_CMD.W1LMINMAX bit transfers the current CNT_MAX register value into CNT_MIN register.
5 (R0/W)	W1LMINCNT	Write 1 MIN Capture from CNTR. The CNT_CMD.W1LMINCNT bit loads the current value in the CNT_CNTR register into the CNT_MIN register to create a new boundary limit.
4 (R0/W)	W1LMINZERO	Write 1 MIN to Zero. Writing a 1 to the CNT_CMD.W1LMINZERO bit clears the CNT_MIN register.

Table 22-10: CNT_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R0/W)	W1LCNTMAX	Write 1 CNTR Load from MAX. The CNT_CMD.W1LCNTMAX bit loads the current value in the CNT_MAX register into the CNT_CNTR register to create a new boundary limit.
2 (R0/W)	W1LCNTMIN	Write 1 CNTR Load from MIN. The CNT_CMD.W1LCNTMIN bit loads the current value in the CNT_MIN register into the CNT_CNTR register to create a new boundary limit.
0 (R0/W)	W1LCNTZERO	Write 1 CNTR to Zero. Writing a 1 to the CNT_CMD.W1LCNTZERO bit clears the CNT_CNTR register.

Counter Register

The CNT_CNTR register holds the 32-bit, two's-complement count value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows use of the CNT as a 16-bit counter if sufficient for the application.

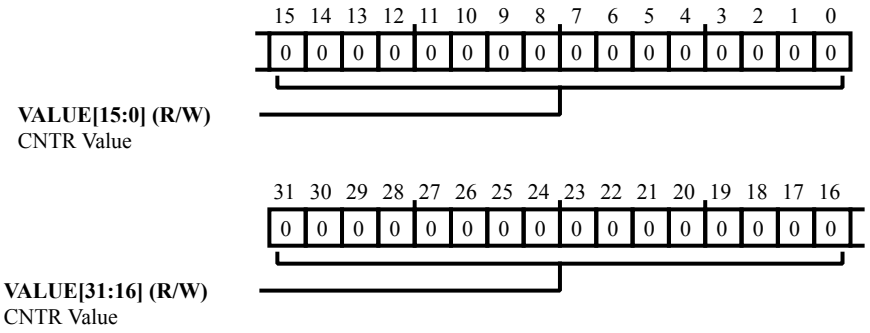


Figure 22-10: CNT_CNTR Register Diagram

Table 22-11: CNT_CNTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	CNTR Value. The CNT_CNTR.VALUE bit field holds the 32-bit, two's-complement count value.

Debounce Register

The CNT_DEBNCE register selects the noise filtering characteristic of the three input pins according to the formula:

t_{filter} = 128 x ($2^{DPRESCALE}$/ SCLK)

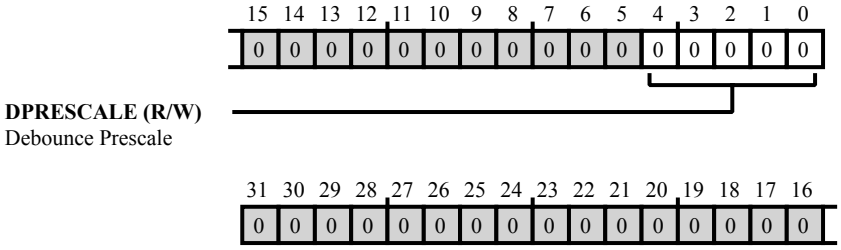


Figure 22-11: CNT_DEBNCE Register Diagram

Table 22-12: CNT_DEBNCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4:0 (R/W)	DPRESCALE	Debounce Prescale. The CNT_DEBNCE.DPRESCALE selects the desired number of input filtering cycles (and resulting input debounce time) in multiples of SCLK.	
		0	1x cycles = 128 SCLK cycles
		1	2x cycles
		2	4x cycles
		3	8x cycles
		4	16x cycles
		5	32x cycles
		6	64x cycles
		7	128x cycles
		8	256x cycles
		9	512x cycles
		10	1024x cycles
		11	2048x cycles
		12	4096x cycles
		13	8192x cycles
		14	16384x cycles
		15	32768x cycles
		16	65536x cycles

Table 22-12: CNT_DEBNCE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		17	131072x cycles
		18	Reserved from this value. The values 10010 - 11111 are reserved.
		31	Reserved until this value

Interrupt Mask Register

The CNT_IMSK register supports enabling (unmasking) interrupt request generation from each of the CNT events.

All bits in CNT_IMSK either disable/mask an interrupt request (if bit cleared) or enable/unmask an interrupt request (if bit set).

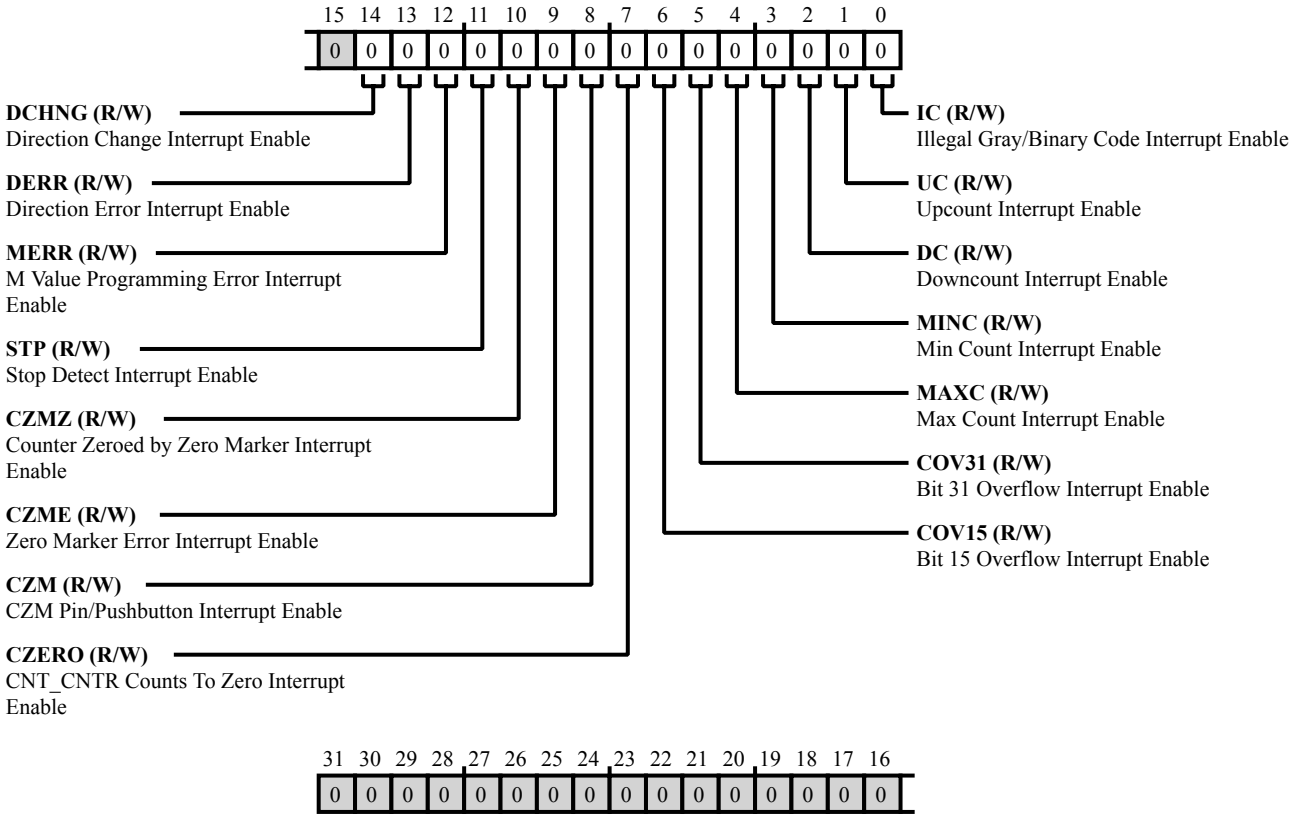


Figure 22-12: CNT_IMSK Register Diagram

Table 22-13: CNT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	DCHNG	Direction Change Interrupt Enable. The CNT_IMSK.DCHNG bit enables (unmasks) the direction change interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
13 (R/W)	DERR	Direction Error Interrupt Enable.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 22-13: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	MERR	M Value Programming Error Interrupt Enable. The CNT_IMSK.MERR bit enables (unmasks) the M value programming error interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
11 (R/W)	STP	Stop Detect Interrupt Enable. The CNT_IMSK.STP bit enables (unmasks) the stop detect interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
10 (R/W)	CZMZ	Counter Zeroed by Zero Marker Interrupt Enable. The CNT_IMSK.CZMZ bit enables (unmasks) the counter zeroed by zero marker interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
9 (R/W)	CZME	Zero Marker Error Interrupt Enable. The CNT_IMSK.CZME bit enables (unmasks) the zero marker error interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
8 (R/W)	CZM	CZM Pin/Pushbutton Interrupt Enable. The CNT_IMSK.CZM bit enables (unmasks) the CZM pin/pushbutton interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
7 (R/W)	CZERO	CNT_CNTR Counts To Zero Interrupt Enable. The CNT_IMSK.CZERO bit enables (unmasks) the counts to zero interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
6 (R/W)	COV15	Bit 15 Overflow Interrupt Enable. The CNT_IMSK.COV15 bit enables (unmasks) the bit 15 overflow interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 22-13: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	COV31	Bit 31 Overflow Interrupt Enable. The CNT_IMSK.COV31 bit enables (unmasks) the bit 31 overflow interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
4 (R/W)	MAXC	Max Count Interrupt Enable. The CNT_IMSK.MAXC bit enables (unmasks) the max count interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
3 (R/W)	MINC	Min Count Interrupt Enable. The CNT_IMSK.MINC bit enables (unmasks) the min count interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
2 (R/W)	DC	Downcount Interrupt Enable. The CNT_IMSK.DC bit enables (unmasks) the down count interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
1 (R/W)	UC	Upcount Interrupt Enable. The CNT_IMSK.UC bit enables (unmasks) the up count interrupt request.	
		0	Mask Interrupt
		1	Unmask Interrupt
0 (R/W)	IC	Illegal Gray/Binary Code Interrupt Enable. The CNT_IMSK.IC bit enables (unmasks) the illegal Gray/Binary Code interrupt request and should only be used in these modes.	
		0	Mask Interrupt
		1	Unmask Interrupt

Maximum Count Register

The CNT_MAX register holds the 32-bit, two's-complement, higher boundary value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows for using the CNT as a 16-bit counter if sufficient for the application.

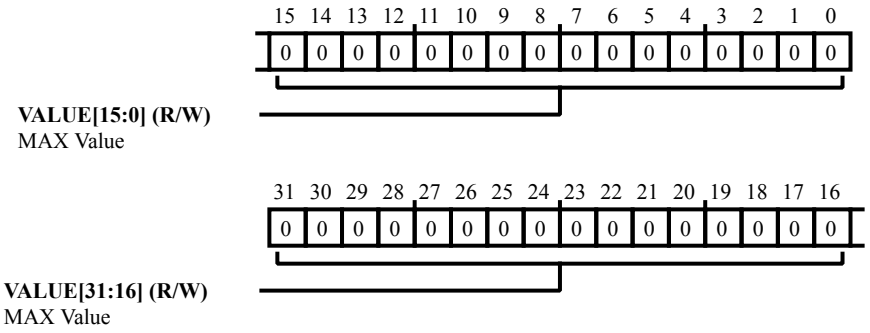


Figure 22-13: CNT_MAX Register Diagram

Table 22-14: CNT_MAX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MAX Value. The CNT_MAX.VALUE bit field holds the 32-bit, two's-complement, higher boundary value.

M Value for Divider

The CNT_MDIV register determines the M value for the M/N division of the QEP inputs that are provided on the CNT_UD/CNT_DG pins.

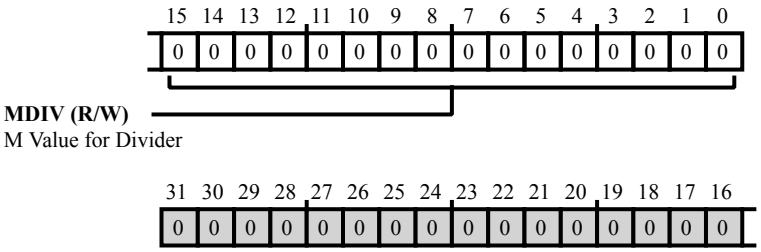


Figure 22-14: CNT_MDIV Register Diagram

Table 22-15: CNT_MDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MDIV	M Value for Divider. The CNT_MDIV.MDIV bit field determines the M value for the M/N division of the QEP inputs that are provided on the CNT_UD/CNT_DG pins.

Minimum Count Register

The CNT_MIN register holds the 32-bit, two's-complement, lower boundary value. It can be read and written at any time. Hardware ensures that reads and write are atomic, by providing respective shadow registers. This register can be accessed with either 32-bit or 16-bit operations. This allows for using the CNT as a 16-bit counter if sufficient for the application.

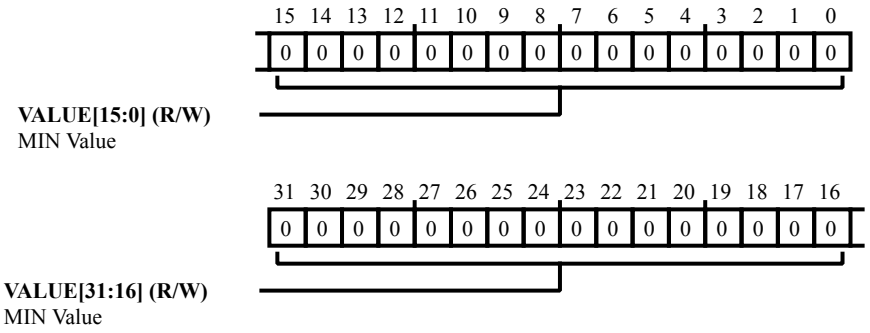


Figure 22-15: CNT_MIN Register Diagram

Table 22-16: CNT_MIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	MIN Value. The CNT_MIN.VALUE bit field holds the 32-bit, two's-complement, lower boundary value.

N Value for Divider

The CNT_NDIV register determines the N value for the M/N division of the QEP inputs that are provided on the CNT_UD/CNT_DG pins.

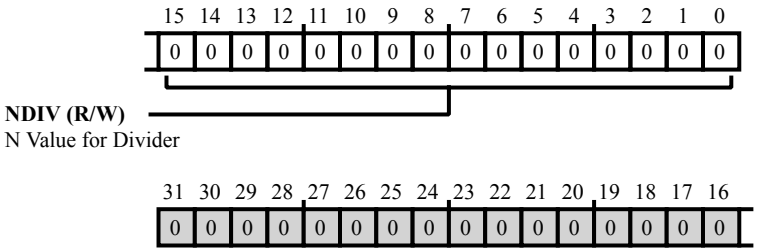


Figure 22-16: CNT_NDIV Register Diagram

Table 22-17: CNT_NDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	NDIV	N Value for Divider. The CNT_NDIV.NDIV bit field determines the N value for the M/N division of the QEP inputs that are provided on the CNT_UD/CNT_DG pins.

Status Register

The CNT_STAT register provides status information for each of the CNT events as configured in the CNT_IMSK register. When a CNT event is detected, the corresponding bit in this register is set. It remains set until either software writes a 1 to the bit (write-1-to-clear) or the CNT is disabled.

All bits in the CNT_STAT register indicate either no interrupt request pending (if bit cleared) or an interrupt request pending (if bit set).

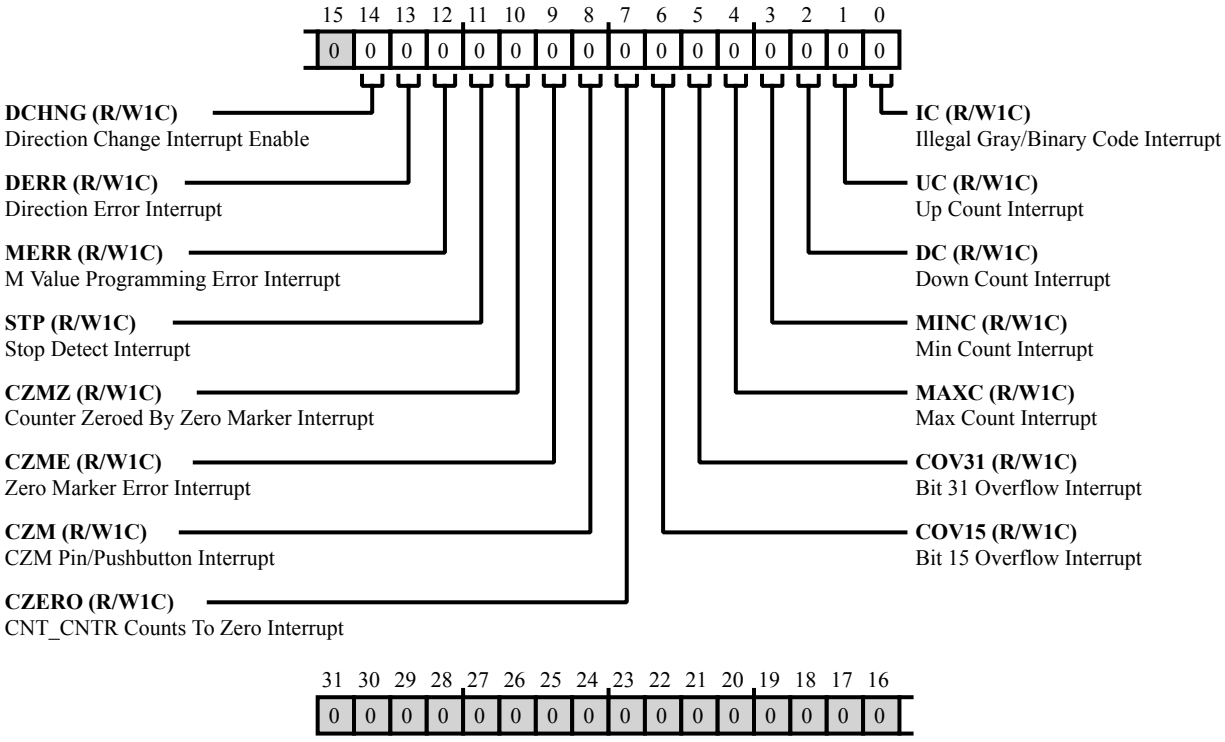


Figure 22-17: CNT_STAT Register Diagram

Table 22-18: CNT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/W1C)	DCHNG	Direction Change Interrupt Enable. The CNT_STAT.DCHNG bit is set if the direction change is valid. Direction change status and interrupt request are generated only if QEP dividers are enabled.
13 (R/W1C)	DERR	Direction Error Interrupt.

Table 22-18: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	MERR	M Value Programming Error Interrupt. The CNT_STAT.MERR bit indicates a M value programming error. This interrupt request is generated when a M value greater than SYSCLK frequency/Q_{IN} frequency is programmed.	
11 (R/W1C)	STP	Stop Detect Interrupt. The CNT_STAT.STP bit indicates a stop detect error. This interrupt request is generated if the QEP pulse is wider than 2 x X x N/M. The internal counter overflow also causes STOP condition and it occurs if pulse was larger than 4,026,531,840 (0xF000_0000) x M) system clock cycles.	
10 (R/W1C)	CZMZ	Counter Zeroed By Zero Marker Interrupt. The CNT_STAT.CZMZ bit indicates a zero marker error. If the CNT_CFG.ZMZC bit =1, this interrupt request is generated when the CZMII latch reports a significant edge on the CZM input. Once cleared by software the CNT_STAT.CZM bit is not set again when the CZM input remains active without pulsing.	
		0	No error
		1	Error occurred
9 (R/W1C)	CZME	Zero Marker Error Interrupt. The CNT_STAT.CZME bit behaves similarly to the CNT_STAT.CZM bit, with the exception that CNT_STAT.CZME is not set on the CZM edge when the lower four bits of the CNT_CNTR are not zero. In many applications this indicates an error condition, as the zero marker might be out of sync with the counter.	
		0	No error
		1	Error occurred
8 (R/W1C)	CZM	CZM Pin/Pushbutton Interrupt. The CNT_STAT.CZM bit indicates a CZM pin/pushbutton error. This interrupt request is generated when a significant edge is seen on the CZM pin, regardless what mode the counter is operating in. This is often used to sense push buttons (especially with the debouncing circuit enabled).	
		0	No error
		1	Error occurred

Table 22-18: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	CZERO	CNT_CNTR Counts To Zero Interrupt. The CNT_STAT.CZERO bit indicates a counts to zero error. This error is generated when the CNT_CNTR register has incremented or decremented toward 0x0000.0000. The latch is not set when software writes to the CNT_CNTR register directly or when the counter is zeroed by writes to the CNT_CMD register.	
		0	No error
		1	Error occurred
6 (R/W1C)	COV15	Bit 15 Overflow Interrupt. The CNT_STAT.COV15 bit indicates a bit 15 overflow error. This error is generated when the 16-bit twos-complement CNT_CNTR register has incremented from 0xxxxx.7FFF to 0xxxxx.8000 or decremented from 0xxxxx.8000 to 0xxxxx.7FFF.	
		0	No error
		1	Error occurred
5 (R/W1C)	COV31	Bit 31 Overflow Interrupt. The CNT_STAT.COV31 bit indicates a bit 31 overflow error. This error is generated when the 32-bit twos-complement CNT_CNTR register has incremented from 0x7FFF.FFFF to 0x8000.0000 or decremented from 0x8000.0000 to 0x7FFF.FFFF.	
		0	No error
		1	Error occurred
4 (R/W1C)	MAXC	Max Count Interrupt. The CNT_STAT.MAXC bit indicates a max count error. This interrupt is used in boundary compare (BND_COMP) mode. If after incrementing the CNT_CNTR register equals CNT_MAX, the CNT_STAT.MAXC bit is set.	
		0	No error
		1	Error occurred
3 (R/W1C)	MINC	Min Count Interrupt. The CNT_STAT.MINC bit indicates a minimum count error. This interrupt is used in boundary compare (BND_COMP) mode. If, after decrementing, the CNT_CNTR register equals CNT_MIN, the CNT_STAT.MINC bit is set.	
		0	No error
		1	Error occurred

Table 22-18: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	DC	Down Count Interrupt. The CNT_STAT.DC bit indicates a down count error. This interrupt is generated when the CNT_CNTR register decrements.	
		0	No error
		1	Error occurred
1 (R/W1C)	UC	Up Count Interrupt. The CNT_STAT.UC bit indicates an up count interrupt. This interrupt is generated when the CNT_CNTR register increments.	
0 (R/W1C)	IC	Illegal Gray/Binary Code Interrupt. The CNT_STAT.IC bit indicates a illegal Gray/Binary Code interrupt and should only be used in these modes. In normal operation those codes can increment or decrement the CNT_CNTR register by one at a time. If the sensed inputs instruct the counter to increment or decrement by two, the CNT_STAT.IC bit is set. Hardware sets the CNT_STAT.IC bit in QUAD_ENC and BIN_ENC encoder modes only.	
		0	No error
		1	Error occurred

23 Debounce Filter (DBC)

The Debounce block accepts a noisy input (which may have multiple transitions) and synchronizes and digitally filters the input to create a clean single transition on the output. Filter parameters control the response of the digital filter. For de-bouncing of relays/switches the parameters may be set for response times on the order of milliseconds. When used as a deglitching filter the parameters may be set for response times on the order of 100 ns.

NOTE: The register descriptions for the Debounce Filter are located in the System Block (SYSBLK) and PADS chapter.

DBC Features

The DBC module has the following features:

- Digital filter implemented with 4-bit counter/accumulator clocked by a sample clock
- Sample clock frequency can be adjusted to meet different signal/noise bandwidth requirements. For example, debouncing may require a low frequency and deglitching may be high.
- Two filter modes are available depending upon specific response desired
  - Counter Filter mode toggles output after N successive samples of new state
  - Accumulator Filter mode toggles output after N accumulated samples of new state
- Digital pre-filter is clocked at the system clock rate. This feature limits aliasing into lower sample frequencies. In strict mode, this filter requires all values within a sample clock period to be in the same state. In non-strict mode only the two values before the sample clock edge are considered.
- Debounce filter defaults to bypass mode. In this mode the input signal is passed directly to the output and no filtering is applied.

DBC Functional Description

The DBC unit consists of a set of identical channel filter blocks, each of which filters one input channel signal DBC_IN[n] and generates a corresponding channel output signal DBC_OUT[n].

The inputs are asynchronous signals and are connected at the SoC top level directly to asynchronous pad inputs.

The outputs may be connected at the SoC top level to a variety of signal destinations, such as GPIO PORT blocks, PWM TRIP input signals, and so on. The outputs of the filter block may also be asynchronous (if the channel's filter block is bypassed with `PADS_DBC[n]_CTL.EN` =1) or may be synchronous to SCLK (`PADS_DBC[n]_CTL.EN` =0).

The four filter blocks associated with the DBC module are shown in the ***Debounce Filter Connections*** table.

Table 23-1: Debounce Filter Connections

Channel	DBC_IN Connection	DBC_OUT Connection
0	(~PWM_TRIPAb) pin	PWM_TRIP matrix
1	(~PWM_TRIPBb) pin	PWM_TRIP matrix
2	(~PWM_TRIPCb) pin	PWM_TRIP matrix
3	COMP_OUT_IRQ	AFE_FOCP_IN

Block Diagram

The ***Debounce Filter Block Diagram*** shows the DBC module.

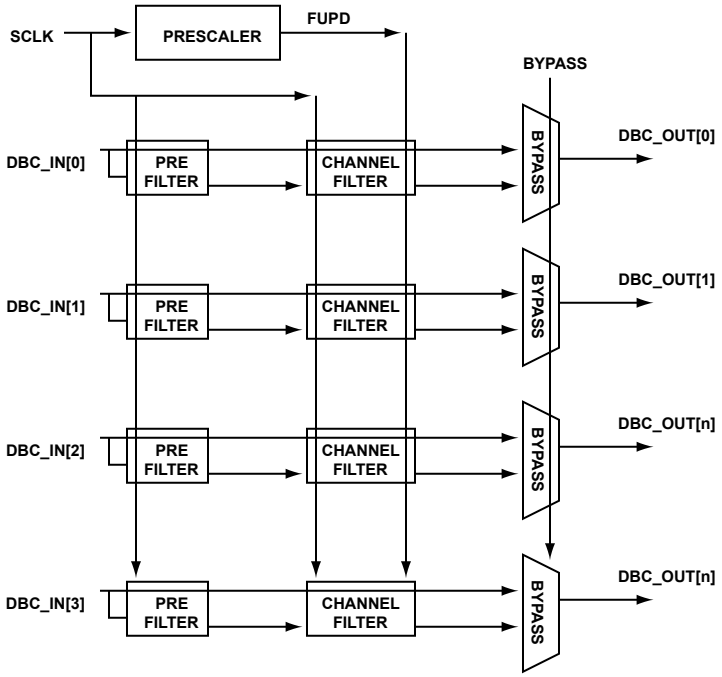


Figure 23-1: Debounce Filter Block Diagram

The Debounce Filter consists of a set of channel filter blocks, each with a synchronization pre-filter, and a global clock prescaler block that is shared between all channels.

- The prescaler divides the system clock by a programmable amount. The output of the prescaler is provided to each de-bounce filter block as a filter update enable signal (FUPD) for the counter and accumulator.

NOTE: The filter update (FUPD) clock period is generated by the prescaler and is limited to integer multiples of the system clock period. This period can range from 1 SCLK period to ~65,536.

- The anti-aliasing pre-filter always samples the channel input at the system clock rate. This filter also acts as a synchronizer to ensure that meta-stability issues are addressed.
- In Counter Filter mode, a filter block output remains in its current state until the new (opposite) state is recognized for N successive samples in a row. If a sample is detected which agrees with the current state, the counter is reset to zero.
- In Accumulator Filter mode, a filter block output remains in its current state until an averaging threshold is reached. Then the new (opposite) state is recognized for N samples more than the current state. The accumulator increments if the new state is recognized, and decrements if the current state is recognized. The accumulator is initially reset to 0 and is also set to 0 after any toggle, therefor accumulator must increment to a threshold of N before another toggle occurs.

DBC Architectural Concepts

The Debounce digital filter creates a clean signal from a very noisy one. Signals tend to be stable for fairly long periods of time. During the transition period of a signal change, noise is introduced. When the transition stops, the signal goes back to a stable state for a long period of time. The duration of the noisy transition region is assumed to be less than the stable period.

A key consideration in making a clean signal is response time. The signal can be sampled very slowly (much slower than the length of the transition region) and clean edges can be achieved. However, the response may be slower than desired. For example, slow response time may lead to missed events.

Mechanical Switch Debounce Filtering

The debounce filter takes advantage of this large ratio of stable periods to switching periods.

A mechanical switch may bounce intermittently between two logic levels when pressed or released. This bounce period may be on the order of 10 milliseconds duration. To debounce this signal, the program creates a sampling window larger than the bounce period and then checks that the signal remains stable for more than the worst case specified bounce period. For example, if the bounce period is specified at a maximum of 8 milliseconds, a program samples the signal at 2 milliseconds intervals for 16 milliseconds before changing to the new state.

Glitch Filtering

Some signals may have slow edge rates and be susceptible to noise from adjacent wires, transmission line effects or power supply noise. These signals may “glitch” a few times during a short noisy period. An example of this is a two-wire interface signal which has edge rates on the order of 100 ns. In this case, glitches of 50 ns must be filtered and the response time must be fast enough to transmit data every 2.5 microseconds (400 kHz rate).

Dynamic Range and Prescaling

Response time cannot be faster than the width of any glitch which is desired to be rejected. A key feature of the DBC is that the sampling clock used for the filter covers a fairly wide range to handle different types of signals and minimize response times.

Each channel's filter has an 8-bit state accumulator and a corresponding 8-bit programmable filter threshold. This allows each channel to filter out (exclude) transitions from 1 to 255 samples in length. This defines the dynamic range of the DBC unit.

Some electrical and mechanical processes act on longer timescales than SCLK/256. A global prescaler is provided which divides SCLK to create the filter update pseudo-clock FUPD. The prescaler has a range of 1 to 2^{16}. This allows the DBC unit to filter input transitions on timescales up to 2^{24} SCLK periods in length, or about 160 milliseconds at a 100 MHz SCLK rate.

The range of input signal pulse durations that can be filtered at various Prescale and Threshold settings is illustrated in the examples in the ***Range of Input Signal Pulse Duration Examples*** table.

Table 23-2: Range of Input Signal Pulse Duration Examples

Prescale Value	Threshold=4 (SCLK cycles)	Time Assuming 100 MHz SCLK	Threshold=255 (SCLK cycles)	Time Assuming 100 MHz SCLK
1 (prescale disabled)	4	40 ns	255	2.55 µs
16	64	640 ns	4,080	4.1 µs
256	1,024	1 µs	65,280	65.2 µs
1,024	4,096	4.1 µs	261,120	261 µs
8,192	32,768	32.8 µs	2,088,960	2.1 ms
65,535	212,140	212.1 µs	16,711,425	167.1 ms

DBC Operating Modes

The following sections provide details on DBC operation.

Bypass Mode (Disabled)

In Bypass mode (`PADS_DBC[n]_CTL.EN` =0), the debounce filter is disabled and the signal path is completely bypassed; the input signal DBC_IN[n] is combinatorially passed to the output DBC_OUT[n] with no clocked latency. This replicates the behavior that would be observed if the DBC unit were not present at all.

This mode is provided for use cases which require no input debouncing and which may need to operate in the absence of the system clock (for example, PWM_TRIP). Note that if the SCLK is sometimes stopped, the debounce filter units may need to be disabled (bypassed) to retain a connection path to the destination unit (for example, for a safety-related signal).

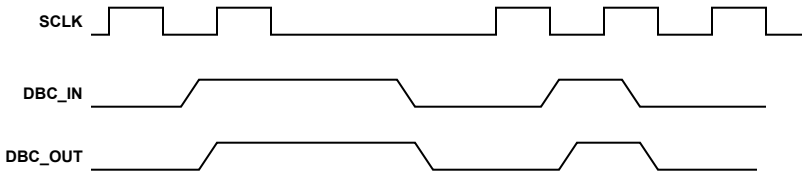


Figure 23-2: Disabled Mode

When the debounce filter is not bypassed, the asynchronous input DBC_IN[n] is synchronized to SCLK by the pre-filter to produce a DBC_IN_sync signal for the filter block.

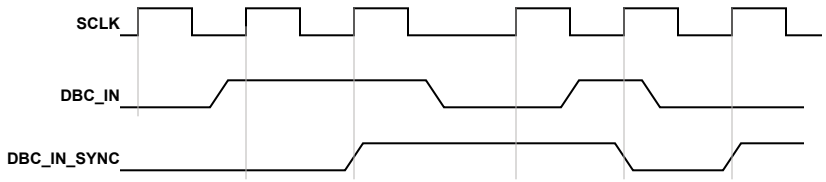


Figure 23-3: Filtering (Non-Bypass mode)

When the debounce filter is enabled by setting the `PADS_DBC[n]_CTL.EN` bit =1 (not in bypass mode), the input signal DBC_IN is synchronized by the pre-filter, which adds one SCLK of latency in addition to the latency of the debounce filter.

Counter Filter Mode

In counter mode, successive samples for a channel are counted and compared to a threshold (T) programmed in the `PADS_DBC[n]_CTL.THRESH` bit field. The programmed value of T is `PADS_DBC[n]_CTL.THRESH` + 1. The output remains at its current state until T successive samples of the opposite state are recorded. Then, the output toggles, the counter is reset, and the filter begins monitoring for the opposite input polarity.

The ***Counter Filter Mode, Threshold = 5*** timing diagram shows the counter filter mode for a threshold value T = 5. In the portion of the diagram with a white background, the filter is monitoring for 0-to-1 transitions; in the portion with the shaded background, the filter is monitoring for 1-to-0 transitions.

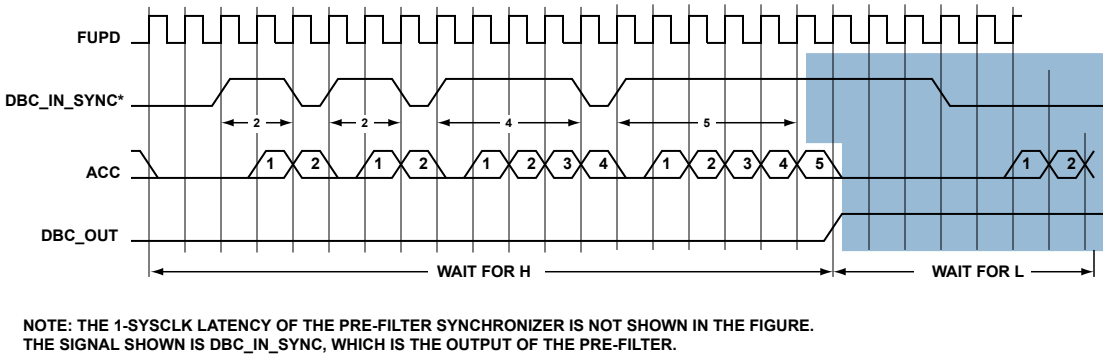


Figure 23-4: Counter Filter Mode, Threshold =5

A boundary value of 0 in the `PADS_DBC[n]_CTL.THRESH` bit field is legal and corresponds to an identity filter —the DBC_IN_SYNC signal is replicated one prescaled FUPD later on the DBC_OUT signal.

Accumulator Filter Mode

In Accumulator mode, the channel's signal is averaged until it crosses a positive or negative threshold (T). The accumulator operates as follows.

1. The input is compared on each cycle to the current output.
    a. If the polarity is different, the accumulator increments.
    b. If the polarity is the same, the accumulator decrements (but does not decrement below zero).
2. If the accumulator exceeds the programmed threshold (reaches T):
    a. The output toggles to the new value.
    b. The accumulator is reset.
    c. The filter begins monitoring for the opposite input polarity.

The ***Accumulator Filter T = 5*** timing diagram shows accumulator filter mode for a threshold value T = 5. In the portion of the diagram with a white background, the filter is monitoring for 0-to-1 transitions; in the portion with the shaded background, the filter is monitoring for 1-to-0 transitions.

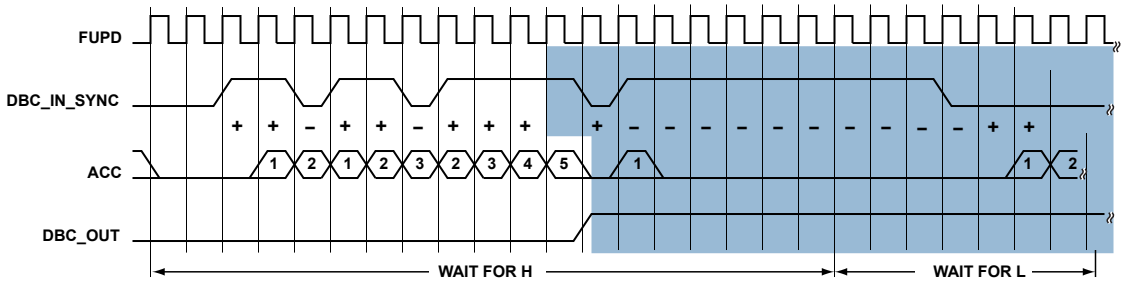


Figure 23-5: Accumulator Filter, Threshold = 5

A `PADS_DBC[n]_CTL.THRESH` boundary value of 0 is legal and corresponds to an identity filter; the DBC_IN_SYNC signal is replicated one prescaled FUPD clock later on DBC_OUT.

Input Prescaling Modes

The rate at which channel data is sampled is controlled by the channel's prescaling mode. The programmed value of prescale is `PADS_DBC_PRESCALE` + 1. The prescaler is common to all DBCs.

In System Clock mode, (`PADS_DBC[n]_CTL.PRECLKSEL` =0), the filter accepts a new data sample on each SCLK.

In Prescaled Clock mode (`PADS_DBC[n]_CTL.PRECLKSEL` =1), the filter updates its state based on the prescaled filter clock FUPD. The prescaler's period is selected by the `PADS_DBC_PRESCALE` register and has a range of 1:1 to 1:(2^{16}).

- A nonzero value of N in the `PADS_DBC_PRESCALE` register selects a SCLK:FUPD frequency divisor of N + 1:1.

- A zero value in the PADS_DBC_PRESCALE register selects a SCLK:FUPD frequency divisor of 1:1. In this case FUPD = SCLK and the filter units act exactly as if prescaling was not selected (updating the state on each SCLK cycle.)

Prescaled Decimated Mode

When the Prescaled clock is used, the method of taking samples at the prescaled rate is controlled by the PADS_DBC[n]_CTL.DEC bit field. In Decimate mode (where PADS_DBC[n]_CTL.PRECLKSEL =1 and PADS_DBC[n]_CTL.DEC =1), the data sample is taken only from the value of the input when the prescaled clock FUPD is asserted, and the values of the input signal between FUPD pulses are discarded. The current phase of the filter output does not influence the prescaled input sampling.

In Counter mode, a pulse wider than ((threshold × update clock period) + system clock period) is never filtered. A pulse smaller than ((threshold × update clock period) – update clock period) – system clock period) is always filtered.

```
Prescaled=4 (Program DBC_PRESCALE to 3), Decimated Mode, Threshold=4 (Program
THRESH to 3)
```

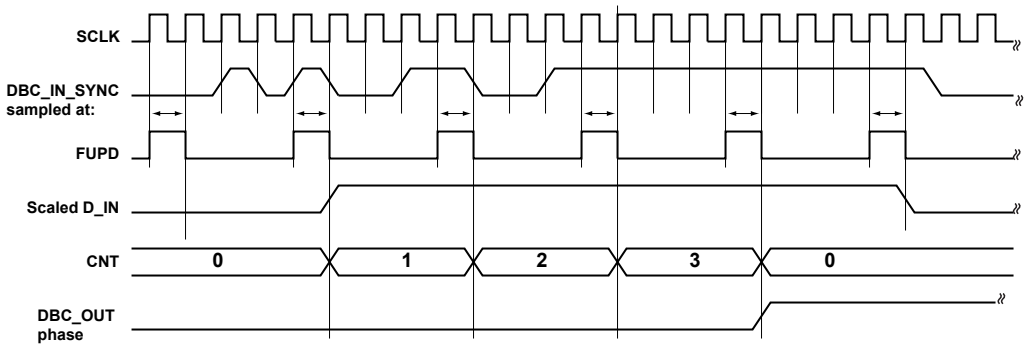


Figure 23-6: Prescaled Decimated Mode Timing

Prescaled Continuous Mode

In Continuous mode (PADS_DBC[n]_CTL.PRECLKSEL =1, PADS_DBC[n]_CTL.DEC =0), the input signal is observed continuously, including the interval between FUPD falling edges (more precisely, starting with the first SCLK cycle after an FUPD pulse, and continuing until the cycle of the next FUPD pulse). The observation considers input states relative to the current filter phase (the current state of the filter output.)

- If and only if ALL of the input states are in opposite phase (the polarity OPPOSITE to the filter output), then the prescaled input sample is the OPPOSITE polarity to the filter output.
- If SOME or ALL of the input states are in phase (the SAME as the polarity of the filter output), then the prescaled input sample is the SAME polarity as the filter output.
- In Counter mode, a pulse wider than ((Threshold period × update clock period) + Update clock period + System Clock period) always gets through the filter but a pulse smaller than ((Threshold period × update clock period) - System Clock period) always gets filtered.

In the ***Prescaled Continous Mode DBC Low Timing*** figure, the prescaler operation is shown for the case where the filter is in the Low phase (DBC_OUT is low).

```
Prescaled=4 (Program PADS_DBC_PRESCALE to 3), Continuous Mode (output phase low)
```

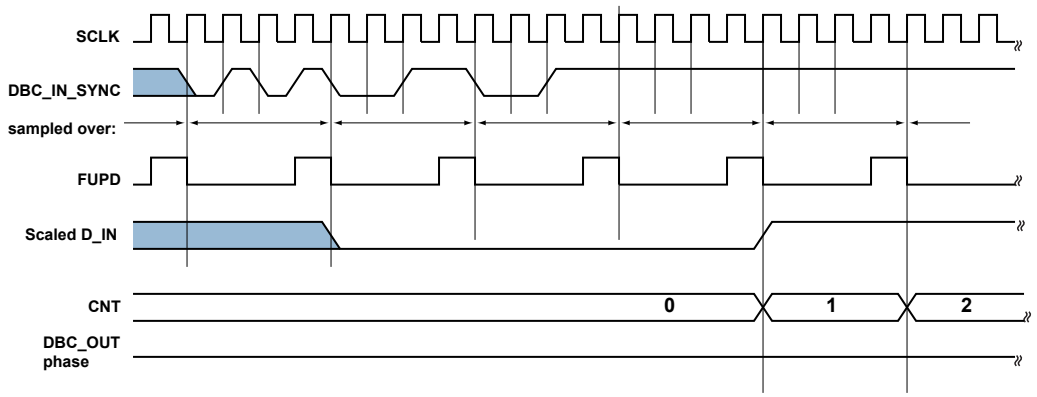


Figure 23-7: Prescaled Continous Mode DBC Low Timing

In the ***Prescaled Continous Mode DBC High Timing*** figure, the prescaler operation is shown for the case where the filter is in the high phase (DBC_OUT is high).

```
Prescaled=4 (Program DBC_PRESCALE to 3), Continuous Mode (output phase high)
```

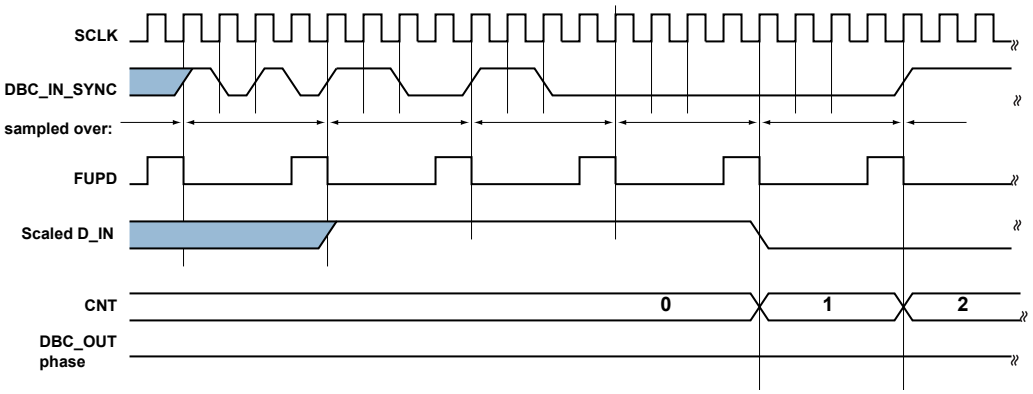


Figure 23-8: Prescaled Continous Mode DBC High Timing

Programming Model

The prescaler (PRESCALE) may only be changed when all of the filter units are bypassed (disabled).

NOTE: Note that the pinmux connections to the DBC unit should be configured prior to enabling the DBC, as spurious logic transitions from changes in the pinmux selection may be interpreted by the DBC as signal edges.

To configure the debounce unit:

1. Configure the pinmux.

2. Write the desired prescaler value.
3. Configure each channel in turn by writing to the `PADS_DBC[n]_CTL.EN` bit.

Register Descriptions

The Debounce Filter does not have a MMR interface. Control signals are provided by generic system registers. For the See the System Block (SYSBLK) and PADS chapter for details on Debounce Filter control registers.

24 Pulse-Width Modulator (PWM)

The pulse-width modulator (PWM) module is a flexible and programmable waveform generator. With minimal CPU intervention, the PWM peripheral can generate complex waveforms for:

- Motor control
- Pulse-coded modulation (PCM)
- Digital-to-analog conversion (DAC)
- Power switching
- Power conversion

The PWM module has four PWM pairs capable of three-phase PWM generation for source inverters for AC induction and DC brushless motors.

PWM Features

Each PWM generation unit features:

- 16-bit center-based PWM generation unit
- Programmable PWM pulse width
- Single or double update modes
- Programmable dead time and switching frequency
- Twos-complement implementation which permits smooth transition to full-on and full-off states
- Dedicated asynchronous PWM shutdown signal
- Debounce filter option on trip inputs that allow the system programmer to filter out short transients

Functional Description

The following sections provide details on the functionality of the PWM.

- Architectural Concepts

- Timer Units
- Channel Timing Control Unit
- Output Disable and Cross-Over Modes
- Sync Operation Modes

CM41X_M4 PWM Register List

The Pulse-Width Modulator unit (PWM) includes multiple timers (providing period flexibility) and channels (providing mode, interrupt, and pulse shape flexibility), permitting a wide variety of PWM output options for motor control and other applications. A set of registers governs PWM operations. For more information on PWM functionality, see the PWM register descriptions.

Table 24-1: CM41X_M4 PWM Register List

Name	Description
PWM_ACTL	Channel A Control Register
PWM_AH0	Channel A-High Duty-0 Register
PWM_AH0_HP	Channel A-High Heightened-Precision Duty-0 Register
PWM_AH1	Channel A-High Duty-1 Register
PWM_AH1_HP	Channel A-High Heightened-Precision Duty-1 Register
PWM_AH_DUTY0	Channel A-High Full Duty0 Register
PWM_AH_DUTY1	Channel A-High Full Duty1 Register
PWM_AL0	Channel A-Low Duty-0 Register
PWM_AL0_HP	Channel A-Low Heightened-Precision Duty-0 Register
PWM_AL1	Channel A-Low Duty-1 Register
PWM_AL1_HP	Channel A-Low Heightened-Precision Duty-1 Register
PWM_AL_DUTY0	Channel A-Low Full Duty0 Register
PWM_AL_DUTY1	Channel A-Low Full Duty1 Register
PWM_BCTL	Channel B Control Register
PWM_BH0	Channel B-High Duty-0 Register
PWM_BH0_HP	Channel B-High Heightened-Precision Duty-0 Register
PWM_BH1	Channel B-High Duty-1 Register
PWM_BH1_HP	Channel B-High Heightened-Precision Duty-1 Register
PWM_BH_DUTY0	Channel B-High Full Duty0 Register
PWM_BH_DUTY1	Channel B-High Full Duty1 Register
PWM_BL0	Channel B-Low Duty-0 Register

Table 24-1: CM41X_M4 PWM Register List (Continued)

Name	Description
PWM_BL0_HP	Channel B-Low Heightened-Precision Duty-0 Register
PWM_BL1	Channel B-Low Duty-1 Register
PWM_BL1_HP	Channel B-Low Heightened-Precision Duty-1 Register
PWM_BL_DUTY0	Channel B-Low Full Duty0 Register
PWM_BL_DUTY1	Channel B-Low Full Duty1 Register
PWM_CCTL	Channel C Control Register
PWM_CH0	Channel C-High Pulse Duty Register 0
PWM_CH0_HP	Channel C-High Pulse Heightened-Precision Duty Register 0
PWM_CH1	Channel C-High Pulse Duty Register 1
PWM_CH1_HP	Channel C-High Pulse Heightened-Precision Duty Register 1
PWM_CHANCFG	Channel Configuration Register
PWM_CHA_DT	Channel A Dead-time Register
PWM_CHB_DT	Channel B Dead-time Register
PWM_CHC_DT	Channel C Dead-time Register
PWM_CHD_DT	Channel D Dead-time Register
PWM_CHOPCFG	Chop Configuration Register
PWM_CH_DUTY0	Channel C-High Full Duty0 Register
PWM_CH_DUTY1	Channel C-High Full Duty1 Register
PWM_CL0	Channel C-Low Pulse Duty Register 0
PWM_CL0_HP	Channel C-Low Pulse Duty Register 1
PWM_CL1	Channel C-Low Duty-1 Register
PWM_CL1_HP	Channel C-Low Heightened-Precision Duty-1 Register
PWM_CL_DUTY0	Channel C-Low Full Duty0 Register
PWM_CL_DUTY1	Channel C-Low Full Duty1 Register
PWM_CTL	Control Register
PWM_DCTL	Channel D Control Register
PWM_DH0	Channel D-High Duty-0 Register
PWM_DH0_HP	Channel D-High Pulse Heightened-Precision Duty Register 0
PWM_DH1	Channel D-High Pulse Duty Register 1
PWM_DH1_HP	Channel D High Pulse Heightened-Precision Duty Register 1
PWM_DH_DUTY0	Channel D-High Full Duty0 Register

Table 24-1: CM41X_M4 PWM Register List (Continued)

Name	Description
PWM_DH_DUTY1	Channel D-High Full Duty1 Register
PWM_DL0	Channel D-Low Pulse Duty Register 0
PWM_DL0_HP	Channel D-Low Heightened-Precision Duty-0 Register
PWM_DL1	Channel D-Low Pulse Duty Register 1
PWM_DL1_HP	Channel D-Low Heightened-Precision Duty-1 Register
PWM_DLYA	Channel A Delay Register
PWM_DLYB	Channel B Delay Register
PWM_DLYC	Channel C Delay Register
PWM_DLYD	Channel D Delay Register
PWM_DL_DUTY0	Channel D-Low Full Duty0 Register
PWM_DL_DUTY1	Channel D-Low Full Duty1 Register
PWM_ILAT	Interrupt Latch Register
PWM_IMSK	Interrupt Mask Register
PWM_STAT	Status Register
PWM_SWTRIP	Software Trip Register
PWM_SYNC_WID	Sync Pulse Width Register
PWM_TM0	Timer 0 Period Register
PWM_TM1	Timer 1 Period Register
PWM_TM2	Timer 2 Period Register
PWM_TM3	Timer 3 Period Register
PWM_TM4	Timer 4 Period Register
PWM_TRIPCFG	Trip Configuration Register
PWM_TRIP_POL	Trip Polarity Register

Additional PWM Registers

The PWM module contains an additional configuration register, SYSBLK_PWM_SYS_CFG that provides additional functionality. See the System Block (SYSBLK) and PADS chapter.

CM41X_M4 PWM Interrupt List

Table 24-2: CM41X_M4 PWM Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
39	PWM0_TRIP	PWM0 Trip	Level	
40	PWM1_TRIP	PWM1 Trip	Level	
41	PWM2_TRIP	PWM2 Trip	Level	
79	PWM0_SYNC	PWM0 PWMTMR Grouped	Edge	
80	PWM1_SYNC	PWM1 PWMTMR Grouped	Edge	
81	PWM2_SYNC	PWM2 PWMTMR Grouped	Edge	

CM41X_M4 PWM Trigger List

Table 24-3: CM41X_M4 PWM Trigger List Masters

Trigger ID	Name	Description	Sensitivity
71	PWM0_SYNC	PWM0 PWMTMR Grouped	Edge
72	PWM1_SYNC	PWM1 PWMTMR Grouped	Edge
73	PWM2_SYNC	PWM2 PWMTMR Grouped	Edge

Table 24-4: CM41X_M4 PWM Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
73	PWM0_SYNC	PWM0 PWMTMR Grouped	Pulse
74	PWM1_SYNC	PWM1 PWMTMR Grouped	Pulse
75	PWM2_SYNC	PWM2 PWMTMR Grouped	Pulse

PWM Definitions

The following definitions are helpful when using the PWM module.

Chopping

Used to simplify the design of isolated gate drive circuits for PWM inverters. If using a transformer coupled power device gate drive amplifier, then the active PWM signal must be chopped at a high frequency.

Dead-Time

A short delay introduced between turning off one PWM signal (for example, AH) and turning on the complementary signal (for example, AL). This short time delay permits turning off a power switch (AH in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the dc link capacitor of a typical voltage source inverter.

Duty Cycle

The proportion of on time to the regular interval or period of time (expressed in percent, 100% being fully on). A low duty cycle corresponds to low power, because the power is off for most of the time.

Switching Frequency

The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load.

Architectural Concepts

A clock, whose period is t_{SCLK}, drives the PWM controller. The PWM generator produces four pairs (four high-side and four low-side) of PWM signals on the eight PWM output pins. Each high and low pair signal constitutes a channel. For example, the PWM_AL and PWM_AH signals make up channel A, and the PWM_BL and PWM_BH signals make up channel B, and so on.

Each pair of channel outputs references either a main timer or an independent timer. These timers operate on a switching frequency determined by the PWM_TM0 through PWM_TM4 registers. There are two duty registers for every PWM output. The registers enable generation of symmetrical or asymmetrical waveforms. The waveforms produce lower harmonic distortion in three-phase PWM inverters, with minimal CPU intervention.

Block Diagram

The ***PWM Block Diagram*** figure shows a block diagram that represents the main functional blocks of the PWM controller.

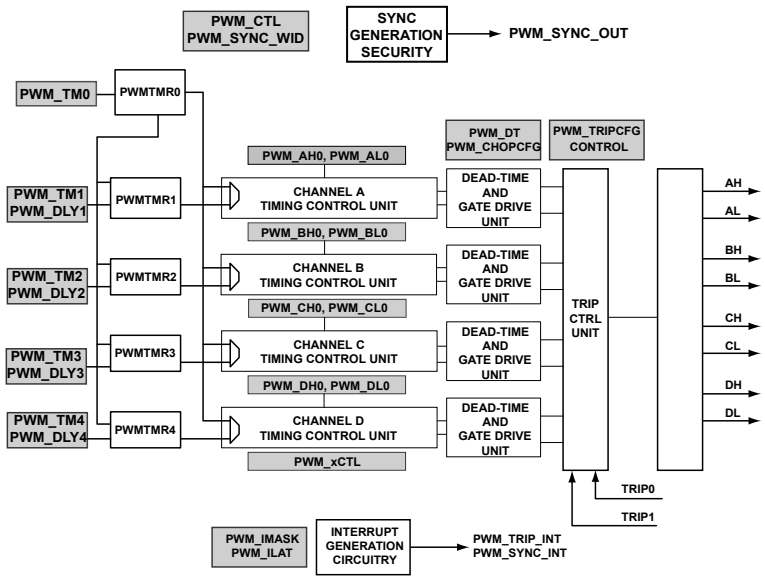


Figure 24-1: PWM Block Diagram

The following list describes the primary blocks.

- Each pair of PWM signals references either the main timer or the independent timer.
- PWMTMR0 is the main timer and can trigger the delayed start of the other timers.
- Timing control units, one for each channel, together form the core of the PWM. The unit generates the required complex waveforms on the high-side and low-side outputs for the respective channel.
- Dead-time insertion occurs after the ideal PWM output pair is generated.
- The gate drive unit generates the high-frequency chopping signal and then mixes it with the requisite PWM output signals.
- The PWM shutdown and interrupt controller manage the various PWM shutdown modes for the timing unit and generate the requisite interrupt signals.
- The PWM sync pulse control unit generates the internal PWM_SYNC pulse and also controls whether the external PWM_SYNC input pulse is used.

Timer Units

Five timers make up the time base for the PWM module. The main timer, PWMTMR0 operates at a switching frequency determined by the period register PWM_TM0. The four remaining timers (PWMTMR1 through PWMTMR4) can operate at independent switching frequencies determined by their respective registers.

The respective time registers (PWM_TM1 through PWM_TM4) can be programmed to work at a multiple of the main timer frequency. In this case, the PWM_DLYA through PWM_DLYD registers control the lead-lag phase of a given timer based on the main timer PWMTMR0.

NOTE: The delayed operation of a timer requires one of the following:

- The register value of the timer must be equal to the PWM_TM0 register value.
- The PWM_TM0 value must be an integer multiple of each register of the timer. Non-integer multiples are not allowed.

PWM Timer Period (PWM_TM) Registers

The 16-bit read/write PWM period registers (PWM_TM0 through PWM_TM4) control the PWM switching frequency. The fundamental timing unit of the PWM controller is t_{SCLK}. Therefore, the time increment (t_{SCLK}) is 10 ns for a 100-MHz system clock (SCLK) frequency, f_{SCLK}. The value written to the register of a timer is effectively the number of t_{SCLK} clock increments in one half of a PWM period. The following equation describes the required timer register value as a function of the desired PWM switching frequency (f_{PWM}):

$$PWM\_TM = f_{SCLK}/2 \times f_{PWM}$$

Therefore, the PWM switching period (T_s) is:

$$T_s = 2 \times PWM\_TM \times t_{SCLK}$$

For example, for an f_{SCLK} of 100 MHz and a desired PWM switching frequency (f_{PWM}) of 10 kHz (T_s = 100 ms), the correct value to load into the timer register is:

$PWM\_TM = 100 \times 10^6 \div 2 \times 10 \times 10^3 = 5000$

The largest value that can be written to the 16-bit timer register is 0xFFFF = 65,535. For an f_{SCLK} of 100 MHz, this value corresponds to a minimum PWM switching frequency of:

$f_{PWM(min)} = 100 \times 10^6 \div 2 \text{ x } 65535 = 762 \text{ Hz}$

NOTE: Timer register values of 0 and 1 are not defined. Do not use these values when the PWM outputs or PWM is enabled.

Timer Unit Operation

The PWM timers are up-down counters, and they operate on the peripheral clock with a period of t_{CK}. The period of the PWM timer is divided into two halves. In the first half, the timer roughly counts down from PWM_TMx/2 to –PWM_TMx/2. During this half, the `PWM_STAT.TMR0PHASE` through `PWM_STAT.TMR4PHASE` bits are held at 0. In the second half of the period, the timer roughly counts up from –PWM_TMx/2 to PWM_TMx/2. The `PWM_STAT.TMR0PHASE` through `PWM_STAT.TMR4PHASE` bits indicates a 1 during this half.

The actual partition of the periods varies slightly between odd and even values of the half-period, in the `PWM_TM[n]` registers.

If a timer register value is odd, for example 11, then that timer loads +5 at the beginning of the period. The timer counts down from +5 to –5 in the first half, reloads –5 at the midpoint and counts up from –5 to +5 in the second half. The reload values at the period and mid-period boundaries are the same as the previous count. The timer counts 2 × 11 half-periods = 22 total counts in the entire period as shown in the ***Operation of Timer for Odd Value of PWM_TM*** figure.

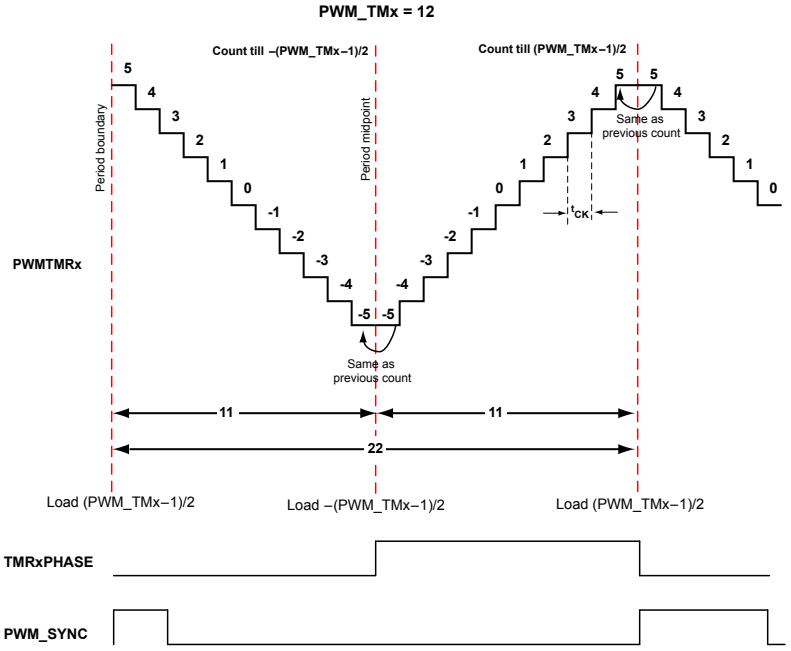


Figure 24-2: Operation of Timer for Odd Value of PWM_TM

When the timer register value is even, for example 12, then that timer loads +5 at the beginning of the period. The timer counts from +5 to –6 in the first half, reloads –5 at the midpoint and counts up from –5 to +6 in the second half. The reload values at the period and mid-period boundaries are different from the previous count. It counts 2 × 12 half-periods = 24 total counts in the entire period as shown in the ***Operation of Timer for Even Value of PWM_TM*** figure.

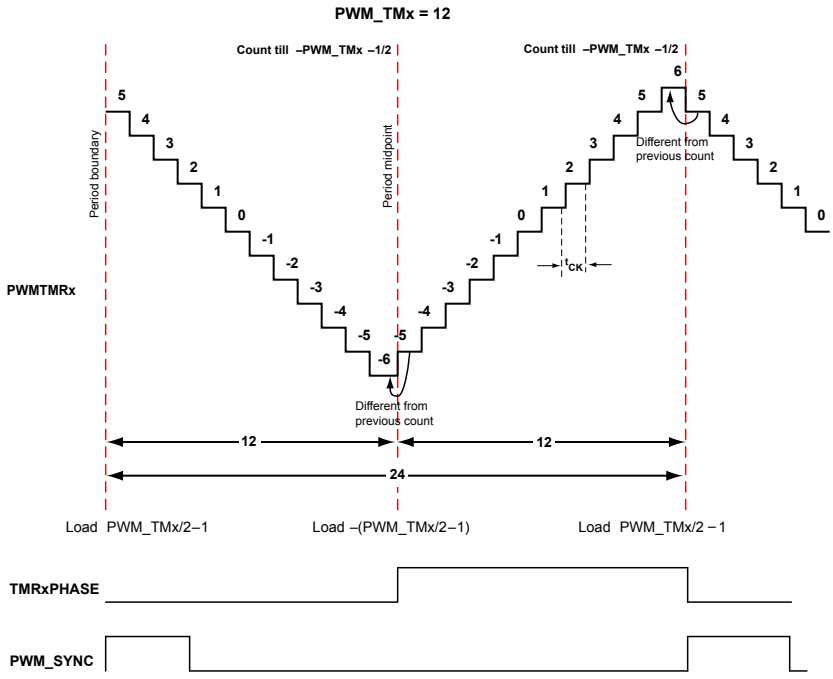


Figure 24-3: Operation of Timer for Even Value of PWM_TM

NOTE: In the operation discussed in this section, double-buffering of all channel registers and the timer registers takes place at the period boundary of the respective timers.

Phase Offset Control

The PWM timers (PWMTMR1 through PWMTMR4) can operate with a programmable phase lag relative to the main timer, PWMTMR0. To implement phase offset for a channel, use the counter-registers for channel delay (PWM_DLYA - PWM_DLYD) with the PWMTMR0 and set the PWM_CTL.DLYAEN bit to l.

Phase offset works as follows.

1. If phase lag is used for channel A (and channel A uses PWMTMR1 to generate a duty cycle), when PWMTMR0 reaches its period boundary, it triggers the PWM_DLYA register. The register counts out the number SCLK cycles that are equal to the value programmed in the PWM_DLYA register.
2. At the end of this count, the PWM_DLYA register sends out a trigger to PWMTMR1. It receives a synchronization pulse in every period of PWMTMR0 at a point delayed from its period boundary by the value in the PWM_DLYA register.

For more information on how channels can reference different timers for their outputs, see Channel Timing Control Unit.

NOTE: Satisfy the following conditions when using this feature on timer y for channel Y relative to PWMTMRx.

- Program the PWM_DLY[n] register to a value less than 2 × PWM_TM[n].
- PWM_TM0 = N × PWM_TM[n], where N is an integer.

The function of PWM_TM[n] (PWMTMR1 in the example) differs in cases where PWM_TM0 = PWM_TM1 (Case 1) to cases where PWM_TM0 = N × PWM_TM1 (Case 2). The following examples describe both cases.

Case 1: PWM_TM0 = PWM_TMy

When PWM_TM0 = PWM_TMy, PWMTMRy restarts its period after receiving the synchronization pulse from the channel delay register (PWM_DLY[n]). If the trigger from the PWM_DLY[n] register is late, PWMTMRy holds its count until the trigger occurs. If the trigger is a bit early, PWMTMRy reloads without regard to whether it has completed its current period. As a result, PWMTMRy resyncs with PWMTMR0 with the phase lag programmed in the PWM_DLYA register in every one of its periods.

In this case, the expiration of the delay registers (PWM_DLY[n]) is the period boundary of PWMTMRy. Now, all the double buffered registers related to the given channel update (except the delay registers which are double buffered at the period boundary of PWMTMR0).

The *Phase Offset Control Using DELAY* figure shows an example where:

- PWM_TM0, PWM_TM1, and PWM_TM2 are programmed with the same value.
- PWM_DLYA and PWM_DLYB are programmed with values DELAY1 and DELAY2 respectively, such that DELAY2 > DELAY1.

- The outputs of Channel A are referenced to PWMTMR1. The outputs of channel B are referenced to PWMTMR2.

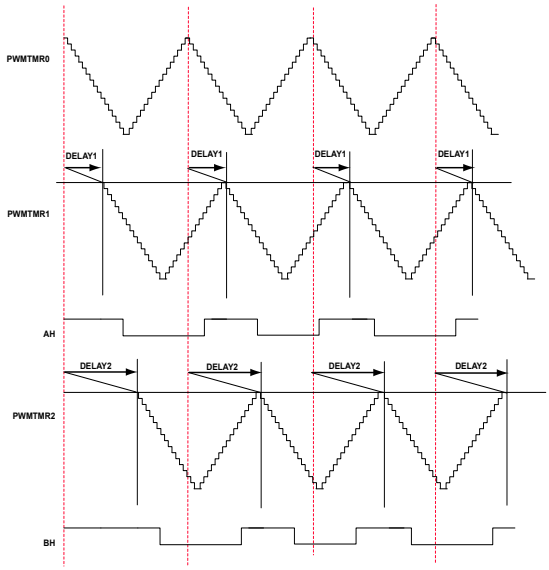


Figure 24-4: Phase Offset Control Using DELAY

The delay registers are double buffered and the new value of DELAY reloads at the period boundary of PWMTMR0. The two options exist when the new value is different from the older one. The behavior of PWMTMRy in both these cases is discussed. The ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows the behavior in the two cases. It is assumed that channel B references its outputs to PWMTMR0 and channel A references its outputs to PWMTMR1.

1. The new delay value is higher than the previous value. Here the corresponding PWMTMRy allows more than one time period between consecutive triggers from the channel delay (`PWM_DLYA` - `PWM_DLYD`) registers. In this case, after reaching its period boundary, PWMTMRy holds its count at the period boundary and waits for the trigger from the channel delay register. The ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows case A functionality.
2. The next delay value programmed is smaller than the previous value. Here, the corresponding PWMTMRy allows only less than one time period between consecutive triggers from the channel delay register. Though the trigger comes earlier in this case, before PWMTMRy has counted out one full period, it reloads and starts its period again. The ***Impact of New DELAY Value on Timer Count for Equal Timer Periods*** figure shows case B functionality.

Therefore, PWMTMR1 waits and obeys a synchronization pulse from the `PWM_DLYA` register in every one of its periods.

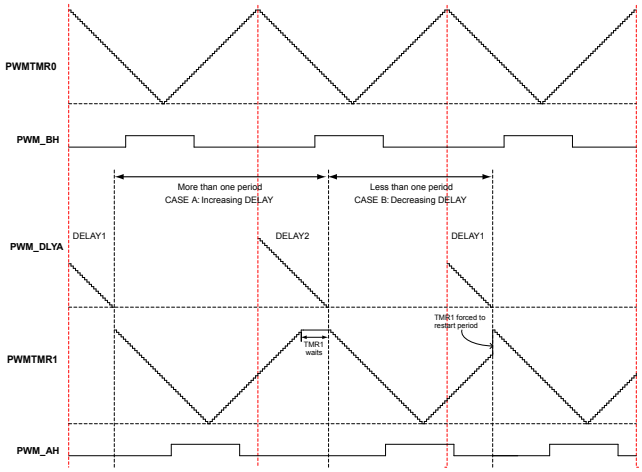


Figure 24-5: Impact of New DELAY Value on Timer Count for Equal Timer Periods

Case 2: PWM_TM0 = N x PWM_TMy

In this case, within a single period of PWMTMR0 a program can fit multiple periods (N) of PWMTMRy. Additionally, the channel delay register is triggered only once every N periods of PWMTMRy.

The operation is as follows: Every Nth period of PWMTMRy, PWMTMRy expects a synchronization pulse from the `PWM_DLY[n]` register. When this register counts out that period and the trigger has not yet arrived, PWMTMRy waits at the end of the period for the trigger. PWMTMRy starts counting down once the trigger arrives. If the trigger comes earlier, PWMTMRy restarts immediately without waiting to complete the period count.

In the intervening periods, PWMTMRy operates independently. As the period ends, PWMTMRy reloads and starts the next period without intervention from the channel delay register.

The ***Impact of DELAY Value Change for the Multiple Timer Periods*** shows an example with N = 2. PWMTMRy syncs up with PWMTMR0 every second period, and is free running across every odd period boundary.

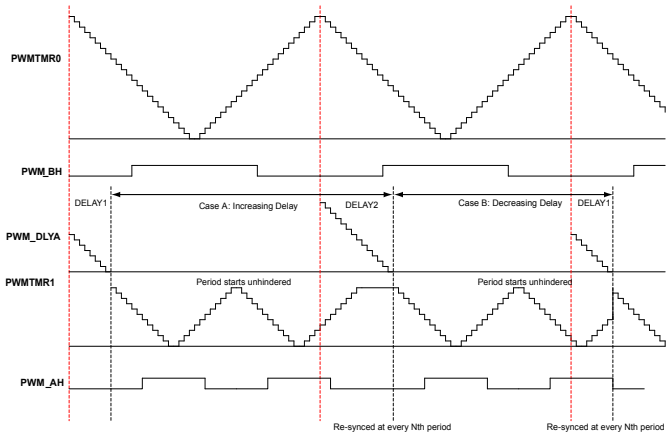


Figure 24-6: Impact of DELAY Value Change for Multiple Timer Periods

Channel Timing Control Unit

The channel timing control unit is the core of the PWM. There are four separate channels, each channel controlling a pair of output signals – the high-side output and the low-side output.

Channel Control

The PWM_CHANCFG register controls the static configuration of all the channels and is initialized once before the beginning of a PWM operation.

NOTE: The PWM_CHANCFG register is not double buffered. Do not change the contents of the register once the PWM is enabled.

Each channel works with a reference timer base. The time base can be either the main timer PWMTMR0 or the appropriate PWMTMRx. Configure the time base with the PWM_CHANCFG.REFTMRA bit field as follows.

- Channel A works with PWMTMR0 or PWMTMR1
- Channel B works with PWMTMR0 or PWMTMR2
- Channel C works with PWMTMR0 or PWMTMR3
- Channel D works with PWMTMR0 or PWMTMR4

The double-buffered channel control registers (PWM_ACTL through PWM_DCTL) contain bits that control the dynamic pulse behavior of the channel outputs. These registers have bits that enable or disable outputs and select the pulse position of outputs (explained in the following section).

Pulse Positioning and Duty Cycle Registers

The PWM uses the PULSEMODEHI and PULSEMODELO bit fields of the PWM_[n]CTL registers to define the region within the timer period where the output pulses are positioned.

- When the PWM_CHANCFG.MODELSC bit is 0, the PWM uses the PULSEMODEHI field to specify the pulse positioning for both the high-side and low-side outputs of the channel.
- When the bit is 1, the PWM uses PWM_ACTL.PULSEMODELO to define the pulse positioning for the low-side output of the channel. It uses the PWM_ACTL.PULSEMODEHI to define the pulse positioning for the high-side output of the channel.

Each channel output has two duty-cycle registers: PWM_AH0 and PWM_AH1 for the high-side output, and PWM_AL0 and PWM_AL1 for the low-side output. These registers determine the width of the output pulses. When the PWM_CHANCFG.MODELSC bit is 0, the high-side duty-cycle registers are used to determine the width of the output pulse for the low side. The duty cycle range that can be programmed into these registers is between –PWM_TM[n]/2 and +PWM_TM[n]/2 , when ignoring dead time.

When including dead time for channel A, for PULSEMODEs 00 and 01, the programmed duty cycle is modified. The range is limited between the values [–PWM_TM[n]/2 + PWM_CHA_DT] and [+PWM_TM[n]/2 + PWM_CHA_DT] considering the high-side output. For PULSEMODEs 10 and 11, the high-side duty cycle registers range is limited between values [PWM_TM[n]/2 + PWM_CHA_DT] and [–PWM_TM[n]/2 – PWM_CHA_DT].

The following section explains dead time in detail.

- Switching Dead Time (PWM_DT) Register

NOTE: Values programmed into these registers that fall outside these limits result in over or under modulation.

Duty Cycle and Pulse Positioning Control

The PWM_ACTL.PULSEMODEHI and PWM_ACTL.PULSEMODELO fields control how the duty cycle registers modify the waveform of the high and low-side outputs. (The PWM_ACTL.PULSEMODEHI and PWM_ACTL.PULSEMODELO fields are referred to as *pulse mode* in the subsequent discussion.)

- Pulse mode = 00 – Produce a symmetrical pulse waveform around the center of the PWM period. In this mode, PWM uses only one of the duty cycle registers for an output. For example, for the AH output, PWM uses only the PWM_AH0 register. In this mode, the values in the duty cycle registers are scaled such that a value of 0 produces a 50% duty cycle.
- Pulse mode = 01 – Produce an asymmetrical pulse waveform around the center of the PWM period. In this mode, PWM uses both duty cycle registers. For example, for the PWM_AH output, PWM uses the PWM_AH0 and PWM_AH1 registers. In this mode, if the PWM_AH1 register is programmed with the same value as the PWM_AH0 register, the output is identical to the output when pulse mode =00.
- Pulse mode = 10 or 11 – Produce pulse waveforms either on the first half or the second half of the PWM period respectively. PWM uses both PWM_AH0 and PWM_AH1 registers.

  Pulse mode = 10. If the low side works from the low-side duty-cycle registers, strictly adhere to the condition PWM_AL0> PWM_AL1.

  In pulse mode = 11. If the low side works from the low-side duty-cycle registers, strictly adhere to the condition PWM_AL0< PWM_AL1.

The ***Pulse Positioning Modes*** figure shows the pulse positioning modes as previously described for PWM_AH. In the figure, DUTY0 is the value in the PWM_AL0 register and DUTY1 is the value in the PWM_AH1 register. The step signal, count, indicates the output of the timer for channel A. In the example, the signal is configured as active high and dead time is zero.

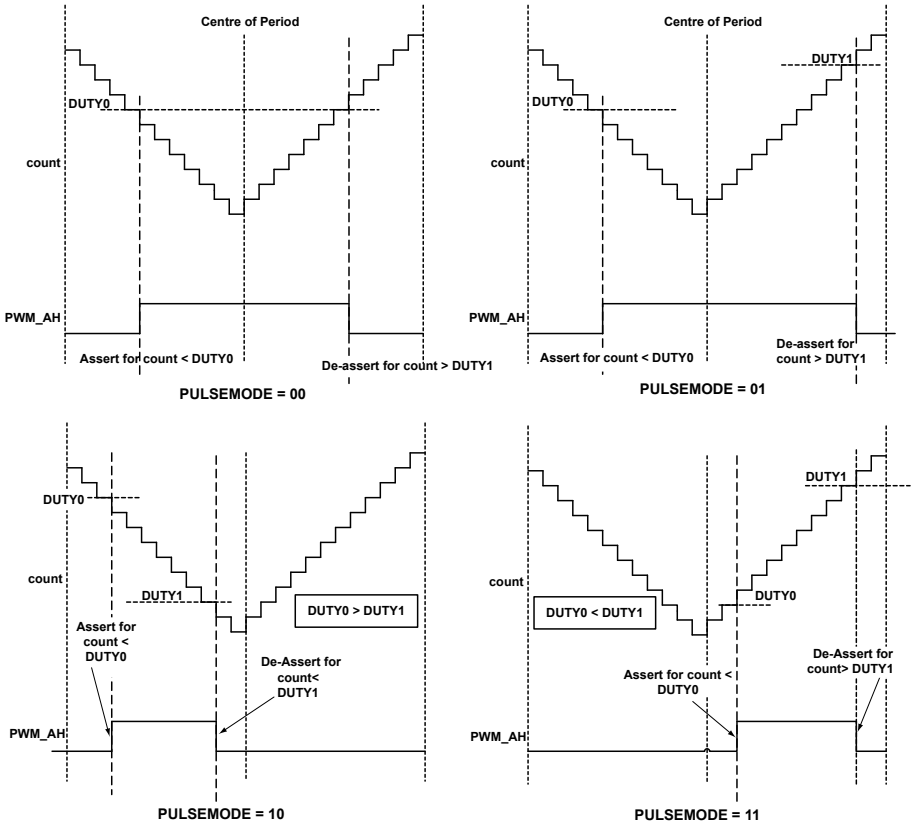


Figure 24-7: Pulse Positioning Modes

Channel Low Side Output Dependent Operation Mode and Dead Time

The low-side output waveform can be programmed to depend on the waveform of the high-side output or to be independent. The PWM uses the `PWM_CHANCFG.MODELSC` bit to control this functionality.

For example, channel A produces the high-side output PWM_AH and the low-side output PWM_AL. When the `PWM_CHANCFG.MODELSC` bit =0, the low-side output is also generated using the high-side duty-cycle registers for pulse width, the `PWM_ACTL.PULSEMODEHI` bits for pulse positioning and the `PWM_CHANCFG.POLAH` bit for polarity. If the `PWM_CH[x]_DT` register is 0, the low-side output is an inverted version of the high-side output.

When the `PWM_CH[x]_DT` register is programmed with a non-zero value, both the high-side and low-side outputs are scaled symmetrically about the points of transition in the zero dead time case. The PWM scales the output by the value programmed in the `PWM_CH[x]_DT` register.

The ***Channel Outputs in Dependent Mode for Pulse Mode*** figures show the high and low-side outputs for the case with zero and non-zero dead time for `PWM_ACTL.PULSEMODEHI` =00 and 01. DUTY0 is the value programmed into the `PWM_AH0` register. DUTY1 is the value programmed into the `PWM_AH1` register. The `PWM_CHANCFG.POLAH` bit =1, indicates that both signals are active high. The `PWM_CHA_DT` register holds the value DT.

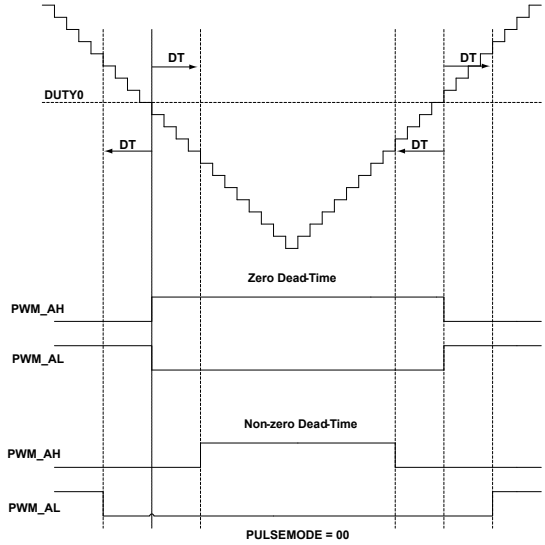


Figure 24-8: Channel Outputs in Dependent Mode for Pulse Mode = 00

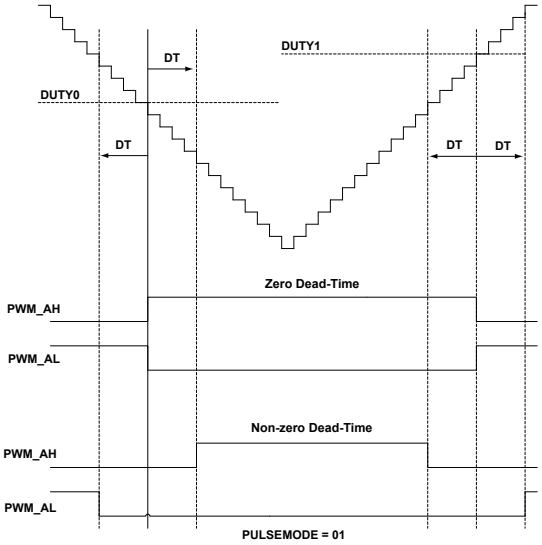


Figure 24-9: Channel Outputs in Dependent Mode for Pulse Mode = 01

The following pair of figures shows the high and low-side outputs for the case with zero and non-zero dead-time for `PWM_ACTL.PULSEMODEHI` =10 and 11. In the figures, DUTY0 is the value programmed into `PWM_AH0` register and DUTY1 is the value programmed into the `PWM_AH1` register. `PWM_CHANCFG.POLAH` is 1 indicating that both signals are active high. The channel dead-time registers hold the value DT.

NOTE: Using dead time, the guidelines for programming the duty-cycle registers in pulse modes 10 and 11 given in Duty Cycle and Pulse Positioning Control are modified as follows:

Pulse mode 10: PWM_xH0 – DT > PWM_xH1 + DT

Pulse mode 11: PWM_xH0 + DT < PWM_xH1 – DT

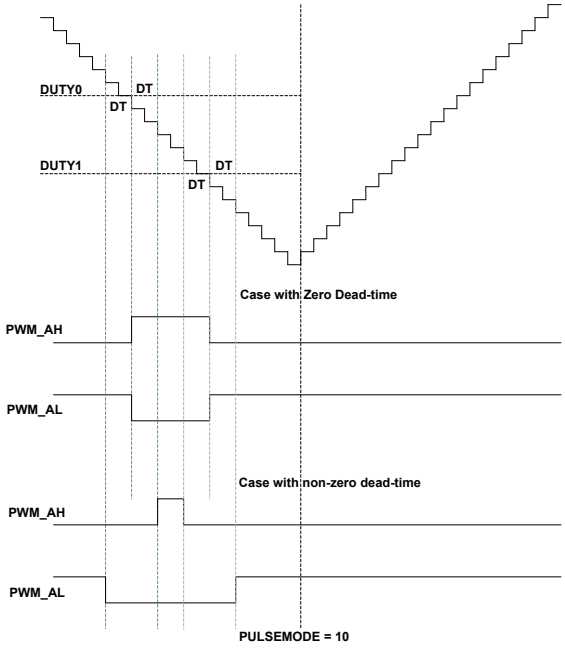


Figure 24-10: Channel Outputs in Dependent Mode for Pulse Mode = 10

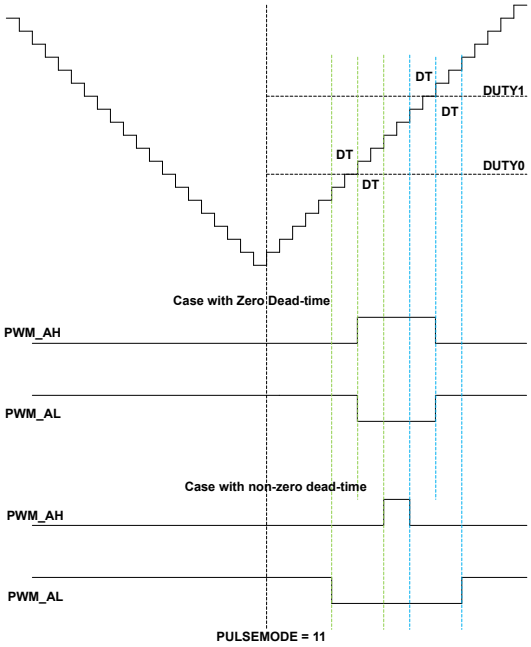


Figure 24-11: Channel Outputs in Dependent Mode for Pulse Mode = 11

Channel High Side and Low Side Outputs, Independent Operation Mode

Independent control of the `PWM_AH0` and `PWM_AL0` channel outputs is possible by setting the `PWM_CHANCFG.MODELSA` bit to 1. In this case, the PWM module:

- Generates PWM_AH using the `PWM_AH0` register
- Uses the `PWM_AH1` register to configure pulse width

- Uses the PWM_ACTL.PULSEMODEHI bit to configure pulse position
- Uses the PWM_CHANCFG.POLAH bit to configure polarity
- Generates PWM_AL using PWM_AL0
- Uses the PWM_AL1 register to configure pulse width
- Uses the PWM_ACTL.PULSEMODELO bit to configure pulse position
- Uses the PWM_CHANCFG.POLAL bit to configure polarity

NOTE: In independent mode, the dead-time insertion is not applicable. The hardware forces dead time to zero.

The ***PWM_AH and PWM_AL in Independent Operation Mode*** figure shows an example of the independent mode of operation where PWM_AH and PWM_AL work from different register bits.

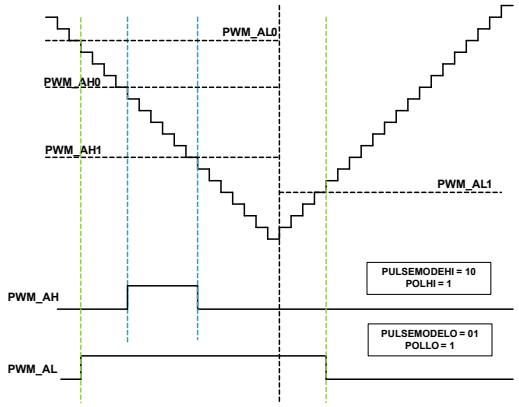


Figure 24-12: PWM_AH and PWM_AL in Independent Operation Mode

PWM_AH and PWM_AL can be positioned in the timer period with a given phase difference between them. Program the PWM_ACTL.PULSEMODEHI and PWM_ACTL.PULSEMODELO bits to different values to achieve this positioning as shown in the ***Channel Outputs Controlled Independently*** figure.

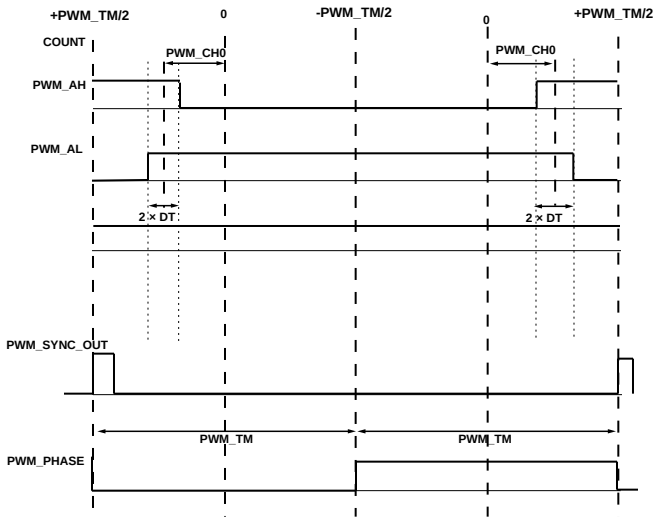


Figure 24-13: Channel Outputs Controlled Independently

Switched Reluctance Motors Application

In typical power converter configurations for switched or variable reluctance motors, motor winding is connected between the two power switches of a given inverter leg. To allow for a complete circuit in the motor winding, turn on both switches at the same time.

PWM uses switched reluctance motors in the following configurations: hard chop, alternate chop, soft chop—bottom on, and soft chop— top on.

The ***Four SR Mode Types, Active High PWM Output Signals*** figure shows the four SR mode types as active high PWM output signals.

Hard chop mode contains independently programmed rising edges of the high and low signals of a channel in the same PWM half cycle. Both signals contain independently programmed falling edges in the next PWM half cycle. The `PWM_CHANCFG.POLAH` and `PWM_CHANCFG.POLAL` bits are programmed to same values.

Alternate chop mode is similar to normal PWM operation except that the PWM channel high and low signal edges are always opposite and are independently programmed. The `PWM_CHANCFG.POLAH` and `PWM_CHANCFG.POLAL` bits are programmed to opposite values. The low-side invert is the only difference between hard chop mode and alternate chop mode.

Soft chop - bottom on uses a 100% duty on the low side of the channel. Soft chop - top on uses a 100% duty on the high side of the channel. Similar to hard chop mode, PWM uses the `PWM_AH0` duty register for the high channel and the `PWM_AL0` duty register for the low channel.

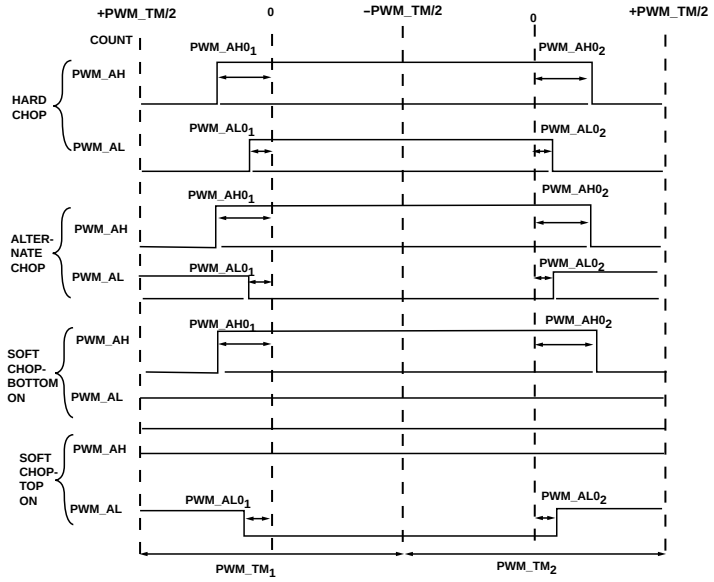


Figure 24-14: Four SR Mode Types, Active High PWM Output Signals

Switching Dead Time (PWM_DT) Register

The second important parameter that must be set up in the initial configuration of the PWM controller is the switching dead time. Dead time is a short delay introduced between turning off one PWM signal (for example, AH) and turning on the complementary signal (for example, AL). This short time delay permits turning off a power switch (AH in this case) to completely recover its blocking capability before the complementary switch is turned on. This time delay prevents a potentially destructive short-circuit condition from developing across the DC link capacitor of a typical voltage source inverter.

The 10-bit, read/write channel A through channel D dead-time registers (`PWM_CHA_DT` through `PWM_CHD_DT`) control the dead time for channel-x. If the value carried by any dead-time register is PWMDT, the dead time, Td, for that channel is:

T_d = `PWM_CH[x]_DT` × 2 × t_{SCLK}

Therefore, a dead-time value of 0x00A introduces a 200-ns delay (for an SCLK of 100 MHz). The delay occurs between turning off any PWM signal (for example, AH) and then turning on its complementary signal (for example, AL). The length of dead time can be programmed in increments of 2 × t_{SCLK} (or 20 ns for an SCLK of 100 MHz). The channel A through channel D dead-time registers have a maximum value of 0x3FF (1023 decimal) and correspond to a maximum programmed dead time of:

$T_{d(max)} = 1023 \times 2 \times t_{SCLK} = 1023 \times 2 \times 10 \times 10^{-9} = 20.5$ µs for an f_{SCLK} rate of 100 MHz.

Write 0 to the `PWM_CHA_DT` through `PWM_CHD_DT` registers to program the dead time.

Duty Cycle with Dead Time Control: Calculations for PULSEMODE 00

The duty cycle registers are scaled so that a value of 0 represents a 50% PWM duty cycle. The switching signals produced are also adjusted to incorporate the programmed dead-time value using the channel dead-time registers

(PWM_CHA_DT through PWM_CHD_DT). The unit in this case produces active low signals so that a low level corresponds to a command to turn-on the associated power device.

The ***Dead Time Between Outputs in Dependent Mode*** figure shows a typical pair of PWM outputs, PWM_AH and PWM_AL. The time values in the figure indicate the integer value in the associated register and can be converted to time by multiplying by the fundamental time increment, t_{CK}. In the example, channel A is working from PWM_TM0.

In the example, the pulse mode is set to 00 so that the switching patterns are perfectly symmetrical about the midpoint of the switching period. The dead time is incorporated by moving the switching instants of both PWM signals away from the instant set by the PWM_AH0 register. Both switching edges are moved by an equal amount (DT × t_{CK}) to preserve the symmetrical output patterns. Also shown is the PWM_SYNC output pulse whose rising edge denotes the beginning of the switching period, and the PWM_STAT.TMR0PHASE bit.

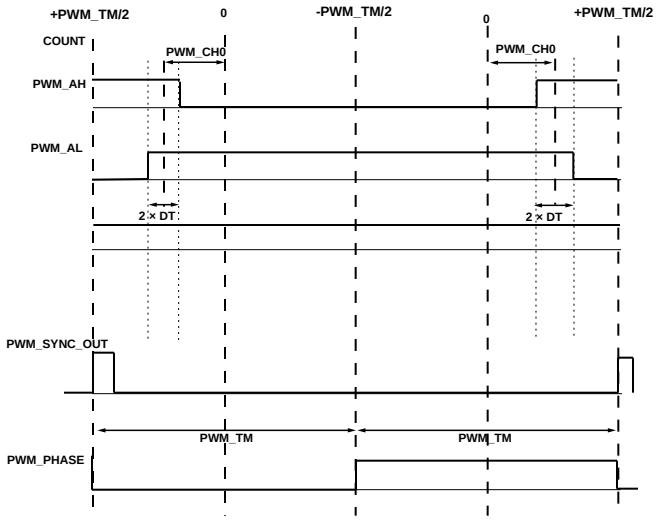


Figure 24-15: Dead Time Between Outputs in Dependent Mode

The PWM timing unit produces the resulting on-times (active low) of the PWM signals over the full PWM period (two half-periods). The figure illustrates this timing. The timing can be written per the following equation.

T_{AH} = (PWM_TM0 + 2 × (PWM_AH0 – PWM_CH[x]_DT)) × t_{CK};

Range of T_{AH} is [0:2 × PWM_TM0 × t_{CK}]

T_{AL} = (PWM_TM0 – 2 × (PWM_AH0 + PWM_CH[x]_DT)) × t_{CK};

Range of T_{AL} is [0:2 × PWM_TM0 × t_{CK}]

$$d_{AH} = \frac{T_{AH}}{Ts} = \frac{1}{2} + \frac{PWM\_AH0 - DT}{PWM\_TM}$$

$$d_{AL} = \frac{T_{AL}}{Ts} = \frac{1}{2} - \frac{PWM\_AH0 + DT}{PWM\_TM0}$$

The negative values of T_{AH} and T_{AL} are not permitted and the minimum permissible value is zero, corresponding to a 0% duty cycle. In a similar fashion, the maximum value is T_s, the PWM switching period, corresponding to a 100% duty cycle. Calculation of duty for other pulse modes can be similarly executed.

Special Consideration for PWM Operation in Over-Modulation

The PWM timing unit can produce PWM signals with variable duty cycle values at the PWM output pins. In pulse modes 00 and 01, at the extremities of the modulation process, duty cycles of 0% and 100% occur. In pulse modes 01 and 10, at the extremities of the modulation process, duty cycles of 0% and 50% occur. The modulation is called *full off* when the lower extremity of modulation is set for any PWM timer period for the corresponding channel. The modulation is called *full on* when the higher extremity of modulation is set for any PWM timer period for the corresponding channel. In between, for other duty cycle values, the operation is termed *normal modulation*.

Full On Modulation

In pulse modes 00 and 01, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the period of the PWM timer that channel is referencing. The conditions for full on modulation are:

- PWM_xH0 - DT > PWM_TMy/2 for pulse mode 00
- PWM_xH1 - DT > PWM_TMy/2 for pulse modes 00 and 01

In pulse mode 10, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the first half period of the PWM timer that the channel is referencing. The conditions for full on modulation are:

- PWM_xH0 - DT > PWM_TMy/2 for pulse mode 10
- PWM_xH1 + DT < PWM_TMy/2 for pulse mode 10

In pulse mode 11, a PWM channel is in full on modulation if the high-side output of that channel is asserted. The output is asserted for the whole duration of the second half period of the PWM timer that the channel is referencing. The conditions for full on modulation are:

- PWM_xH0 + DT < PWM_TMy/2 for pulse mode 11
- PWM_xH1 - DT > PWM_TMy/2 for pulse mode 11

Full Off Modulation

In pulse modes 00 and 01, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the period of the PWM timer that channel is referencing. The conditions for full off modulation are:

- PWM_xH0 - DT < PWM_TMy/2 for pulse mode 00
- PWM_xH1 - DT < PWM_TMy/2 for pulse modes 00 and 01

In pulse mode 10, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the first half period of the PWM timer that the channel is referencing. In the second half-period, it is deasserted anyway. The conditions for full off modulation are:

- PWM_xH0 - DT < PWM_TMy/2 for pulse mode 10
- PWM_xH1 + DT < PWM_xH0 DT for pulse mode 10

In pulse mode 11, a PWM channel is in full off modulation if the high-side output of that channel is deasserted. The output is deasserted for the whole duration of the second half period of the PWM timer that the channel is referencing. In the first half of the period, it is deasserted anyway. The conditions for full off modulation are:

- PWM_xH0 + DT > PWM_TMy/2 for pulse mode 11
- PWM_xH1 - DT > PWM_xH0 + DT for pulse mode 11

Normal Modulation

All other cases of modulation fall under this category.

Emergency Dead-Time Delays

Sometimes, during modulation transition, it is necessary to insert more emergency dead-time delays to prevent potential shoot through conditions in the inverter. (For example, when the PWM transitions into or out of full on or full off modulation.) Disabling and enabling usage (related to the `PWM_ACTL.DISHI` and `PWM_ACTL.DISLO` bits) also can potentially cause outputs to violate shoot-through condition criteria. Another case is when large values vary the phase delay of a PWM timer. These transitions are detected automatically. If appropriate for safety, an emergency dead-time is inserted to prevent shoot through conditions.

There is another atypical case for the insertion of the additional emergency dead time. It occurs when both PWM signals do not toggle within a dead time of each other. In this case, insert more emergency dead time into one of the PWM signals of a given pair during these transitions. The dead-time delay is inserted into the PWM signal that is toggling into the on-state. In effect, an amount ($2 \times DT \times t_{CK}$) from the rising edge of the opposite output delays the turn-on of this signal. After this delay, the PWM signal is allowed to turn-on provided the desired output is still scheduled to be in the on-state after the emergency dead-time delay.

The ***Over Modulation Transition Example*** figure illustrates two examples of such a transition. In the figure, `PWM_ACTL.PULSEMODEHI` is kept at 1. The `PWM_AH` signal has been in full on modulation for some time and, during the current period, its pulse mode is changed to 10, keeping the full on condition. At the half-period boundary, `PWM_AH` is forced to transition to a deasserted state because pulse mode is 10. An emergency dead-time is inserted on the low-side output.

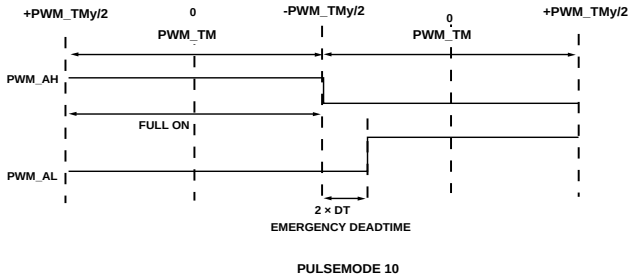


Figure 24-16: Over Modulation Transition Example

Gate Drive Unit

The gate drive unit of the PWM adds features that simplify the design of isolated gate drive circuits for PWM inverters. When using a transformer coupled power device gate drive amplifier, the active PWM signal must be chopped at a high frequency. The `PWM_CHOPCFG` register allows the programming of this chopping mode for high frequency. The chopped active PWM signals can be required for the high-side drivers only, for the low-side drivers only, or for both the high-side and low-side switches. Therefore, independent control of this mode for both high and low-side switches is included with two separate control bits in the `PWM_CHANCFG` register.

The ***High-Side and Low-Side Outputs With Gate Chop Enabled*** figure shows the typical PWM output signals with high-frequency chopping enabled on both high-side and low-side signals. Chopping of the PWM outputs is enabled by setting bits in `PWM_CHANCFG` register. The 8-bit `PWM_CHOPCFG.VALUE` value controls the high frequency chopping. The following equation gives the period of this high frequency carrier.

$T_{chop} = [4 \times (CHOPDIV + 1)] \times t_{CK}$

and the chopping frequency is therefore an integral subdivision of the peripheral clock frequency:

$f_{chop} = f_{CK}/[4 \times (CHOPDIV + 1)]$

The `PWM_CHOPCFG.VALUE` value can range from 0 to 255, corresponding to a programmable chopping frequency rate from 122 kHz to 31.25 MHz for a 125 MHz, f_{CK} rate. Program the gate drive features before enabling the PWM controller. Do not change the gate drive features during normal operation of the controller. Following a reset, clear all bits of the `PWM_CHANCFG` register so that high frequency chopping is disabled, by default.

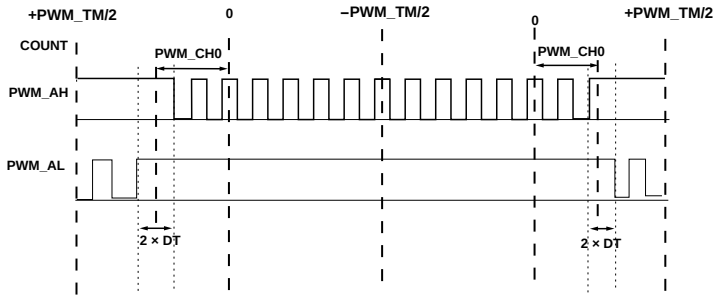


Figure 24-17: High-Side and Low-Side Outputs With Gate Chop Enabled

Output Control Feature Precedence

The order of applying output control features to the PWM signal is important and significant. Use the following order for applying the signal features to the PWM output signal.

1. Duty generation
2. Cross-over
3. High-side or low-side disable
4. Emergency dead-time insertion
5. HPPWM correction for wrong programming
6. Gate-drive chopping
7. Polarity
8. Heightened-Precision PWM (HPPWM) edge placement

NOTE: When HPPWM operation is enabled, the ***cross-over*** feature and the ***gate-drive chopping*** feature must be disabled.

Operating Modes

The PWM generator is capable of operating in the following modes:

- Sync Operation Modes
- Output Disable and Cross-Over Modes
- Heightened-Precision Edge Placement
- Emulation Mode

Sync Operation Modes

The `PWMx_SYNC` pins can operate as internally generated or externally generated. If its internally generated, PWMs can drive the signal to synchronize other PWMs and other devices. If externally generated, the PWMs can be synchronized externally.

Synchronizing Other Devices

The PWM may be programmed to generate a SYNC pulse of any width, which can be driven via the pinmux onto a GPIO. This operation allows synchronization with other off-chip slave devices and is controlled by the pinmux, and the `PWM_CTL.EXTSYNC` and PWM SYNC out secondary enable bits (`SYSBLK_PWM_SYS_CFG.PWM0SYNCOE` through `SYSBLK_PWM_SYS_CFG.PWM2SYNCOE`). To perform this operation:

- The pinmux must select FER =1 and set MUX to select the PWM function, AND

- either the PWM_CTL.EXTSYNC =0 (internal PWM sync), OR
- The SYSBLK_PWM_SYS_CFG.PWM0SYNCOE through SYSBLK_PWM_SYS_CFG.PWM2SYNCOE bits =1 (for PWM accepting a SYNC trigger input, but also generating a SYNC output GPIO pulse of programmable width).

Internal PWM Sync Generation

The PWM controller produces an output PWM synchronization pulse at a rate equal to period of a selected PWM timer. Programming the PWM_CTL.INTSYNCREF field controls this selection.

The PWM_SYNC pulses are generated at their respective period boundaries if:

- the other timers are running with a non-zero delay offset in relation to PWMTMR0, *and*
- the PWM_SYNC pulse is referenced to any of these timers.

The boundaries have a lag-lead offset compared to PWMTMR0.

This pulse is available for external use at the PWM_SYNC_OUT pin. The width of the PWM_SYNC pulse is programmable by the 10-bit read/write PWM_SYNC_WID register. The following equation gives the width of the PWM_SYNC pulse:

$t_{PWMSYNC} = t_{SCLK} \times (PWMSYNCWT + 1)$

The width of the pulse is programmable from t_{CK} to 1024 t_{CK} (corresponding to 8 ns to 8.19 µs for a f_{CK} rate of 125 MHz). Following a reset, the PWM_SYNC_WID register contains 0x3FF (1023 decimal) so that the default PWM_SYNC width is 8.19 µs, for a 125 MHz f_{CK}.

External (Triggered) PWM Sync Generation

By setting the PWM_CTL.EXTSYNC bit, the PWM is set up in a mode to expect an external PWM_SYNC signal on the PWM_SYNC_IN pin through the TRU. The trigger source can be any TRU1 trigger master. Multiple PWM units can be precisely synchronized by selecting the same trigger master as the sources for each PWM sync trigger slave. Examples of useful trigger masters include:

- A PWM_SYNC GPIO master. This option allows synchronizing the PWMs to an off-chip timing source. (The PWM_CTL.EXTSYNC bit must be programmed to 1 to enable the GPIO input.) The PWM_SYNC GPIO pads are connected through the pin mux to configurable trigger masters using PWM master IDs TRGM_SYS_PWMn_SYNC_IN. Using the TRU, these masters can be connected to the PWM_SYNC trigger slaves in any desired combination.
- A TTU trigger output. This option can include a member of a TTU trigger group, which can also control the timing of other devices such as ADCs, SINC filter inputs, or GP timers. The TTU supports relative trigger delays so that timing offsets can be applied to manage system latencies with precision.
- A general-purpose timer trigger master
- A software trigger master. This option allows starting one or more PWM units simultaneously by an MMR write.

The external PWM_SYNC signal only determines the operation of the main timer PWMTMR0.

Synchronize the external sync by setting the PWM_CTL.EXTSYNCSEL bit to 0 (assumes the external PWM_SYNC selected is asynchronous).

The external PWM_SYNC period is expected to be an integer multiple of the value of the PWM_TM0 period register. When the rising edge of the external PWM_SYNC is detected, the PWMTMR0 timer is restarted at the beginning of its period. If the external PWM_SYNC period is not exactly an integer multiple of the internal PWM_SYNC, the behavior of the PWM channel outputs which are referenced to PWMTMR0 are clipped.

CAUTION: Do not change the value of the PWM_CTL.EXTSYNC bit while the PWM is enabled (PWM_CTL.GLOBEN =1).

Output Disable and Cross-Over Modes

Each PWM_ACTL channel control register contains separate enable bits for the high and low-side signals. The PWM module uses the PWM_ACTL.DISHI and PWM_ACTL.DISLO bits in the channel A control register to enable or disable the PWM_AH and PWM_AL outputs respectively. If the disable bit is set (=1), then the corresponding PWM output is disabled, irrespective of the value of the corresponding duty cycle register. This PWM output signal remains in the OFF state as long as the corresponding enable or disable bit is set.

The cross-over bit (PWM_ACTL.XOVR) allows programs to send the low-side output through the high-side output pin and the high-side output through the low-side output pin.

One example uses the following configuration:

- The PWM_AH0 register =0
- The PWM_CHANCFG.MODELSC bit =0
- The PWM_ACTL.DISLO bit =1
- The PWM_ACTL.XOVR bit =1

The low-side output remains off, as in the case without crossover. The difference in cross-over is that the high-side output changes character and becomes like the low-side. What actually occurs is that the low-side duty cycle is sent to the high-side output pins, and the high-side duty cycle is sent to the low side pins. Because the PWM_ACTL.DISLO bit =1, the low-side pin remains off (see Output Control Feature Precedence).

The ***XOVR and DISHI/DISLO Functionality*** figure shows this example. In case 1, PWM_ACTL.XOVR =0; and in case 2, PWM_ACTL.XOVR =1.

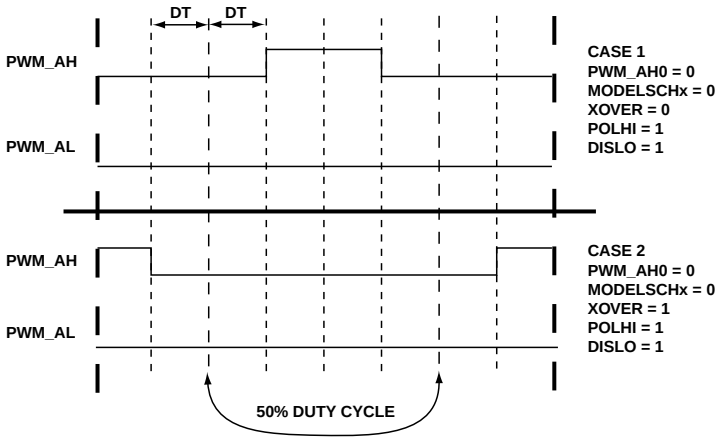


Figure 24-18: XOVR and DISHI/DISLO Functionality

Brushless DC Motor (Electronically Commutated Motor) Control

In the control of an electronically commutated motor (ECM), only two inverter legs are switched at any time. Often, the high-side device in one leg must be switched on at the same time as the low-side driver in a second leg. It is possible to turn on the high-side switch of phase A and the low-side switch of phase B at the same time by:

- Programming identical values for the duty cycles for two PWM channels (for example, `PWM_CH0` = `PWM_CH1`), and
- Setting the `PWM_BCTL.XOVR` bit to crossover the BH and BL pair of PWM signals

To control ECM, normally the third inverter leg (phase C in this example) is disabled for a number of PWM cycles. To implement this function, both the `PWM_CH` and `PWM_CL` outputs are disabled by setting the `PWM_CCTL.DISHI` and `PWM_CCTL.DISLO` bits.

In normal ECM operation, each inverter leg is disabled for certain time periods so that the PWM channel registers change based on the position of the rotor shaft (motor commutation).

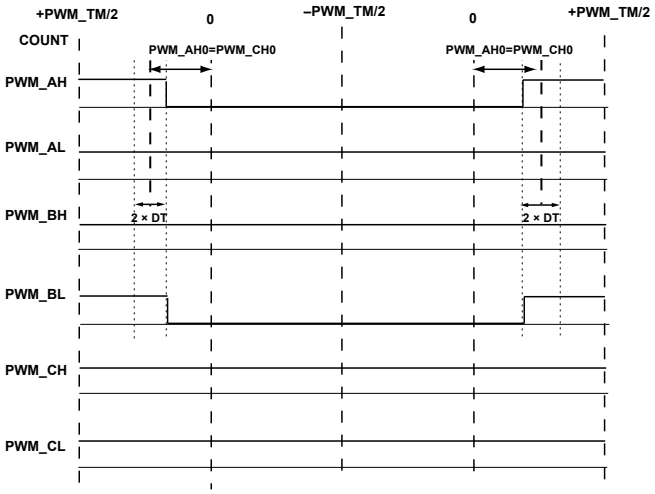


Figure 24-19: ECM Control

Heightened-Precision Edge Placement

Heightened-precision edge placement allows a fine-grained edge placement within the system clock period. The ***Heightened-Precision Steps in a Single SCLK Period*** figure shows how the SCLK aligned edge is moved to finer resolution.

The heightened-precision mode is enabled by setting bits that correspond to the high or low side output option for the channel in the PWM_CHANCFG register. bit. The PWM_AH0_HP and PWM_AH1_HP registers work alongside the PWM_AH0 and PWM_AH1 registers to provide the overall resolution. The following example explains how signed decimal programming is implied for the heightened-precision duty values.

For the PWM_AH output, the duty-cycle register-pair PWM_AH0 and PWM_AH0_HP work together in a Q15.8 signed two's complement fixed-point format as shown in the ***Duty Cycle Notation for Heightened-Precision Edge Placement*** figure. The weight of bit position at k is 2k.

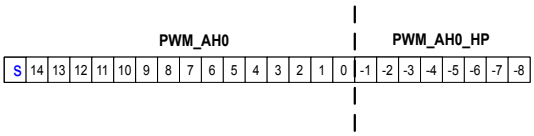


Figure 24-20: Duty Cycle Notation for Heightened-Precision Edge Placement

In the normal modes of operation (not involving heightened-precision edge placement), only the PWM_AH0 register value is programmed. The duty value programmed is a two's complement integer value. If a value of -1 is desired, the PWM_AH0 register is programmed with the number 0xFFFF (which is the two's complement of 1 in 16 bits).

In the heightened-precision mode, if a duty value corresponding to 0.75 is required, the equivalent two's complement value of 0.75 in the Q15.8 format is computed: 1111_1111_1111_1111.0100_0000 = 0xFFFF.40. In this case, the PWM_AH0 register is programmed to 0xFFFF and the PWM_AH0_HP register is programmed to 0x40.

Heightened-Precision Edge Placement Example

The following is an example of heightened precision edge placement.

On the *positive side* of the fractional of the duty cycle, at 2 t_{CK} + 0.25 t_{CK}, the values for the PWM_AH0 and PWM_AH0_HP registers are calculated as follows.

The PWM_AH0 =0x0002 and PWM_AH0_HP =0x40 (bits 7:6).

The PWM_AH_DUTY0 register contains the bit fields from both the PWM_AH0 and PWM_AH0_HP registers. Bits [15:14] represent the decimal part or heightened-precision value and bits [31:16] represent the coarse duty cycle. The value for the combined registers is PWM_AH_DUTY0 =0x00024000.

On the *negative side* of the fractional of the duty cycle, at –2 t_{CK} – 0.25 t_{CK}, the values for the PWM_AH0 and PWM_AH0_HP registers are calculated as follows:

The coarse register represents the next count of the coarse value for negative values so that –2 becomes –3. These values are the two's complement of the positive offset (value = 3) PWM_AH0 = 0xFFFD and PWM_AH0_HP =0xC0 (bits 7:6).

To derive the correct format for a negative duty-cycle value, for example, –2.25, use: coarse value + 1 =3 for the coarse value and 1 for 0.25. Write out the absolute value as a 32-bit number first:

0000 0000 0000 0011 (.) 0100 0000 0000 0000

Then take the two's complement of the entire 32-bit number:

1111 1111 1111 1101 (.) 1100 0000 0000 0000

NOTE: This value is also written into the full duty register (PWM_AH_DUTY0). The correct value for the combined registers written in the PWM_AH_DUTY0 register is 0xFFFDC000.

NOTE: The above examples only consider 2 bits of precision, while the ADSP-CM41xF supports up to 4 bits (refer to the product datasheet for maximum heightened precision resolution in time). That means bits [7:4] of the PWM_AH_DUTY0 register can be used to define the higher precision. Bits [15:12] of represent the decimal part or heightened-precision value and bits [31:16] represent the coarse duty cycle.

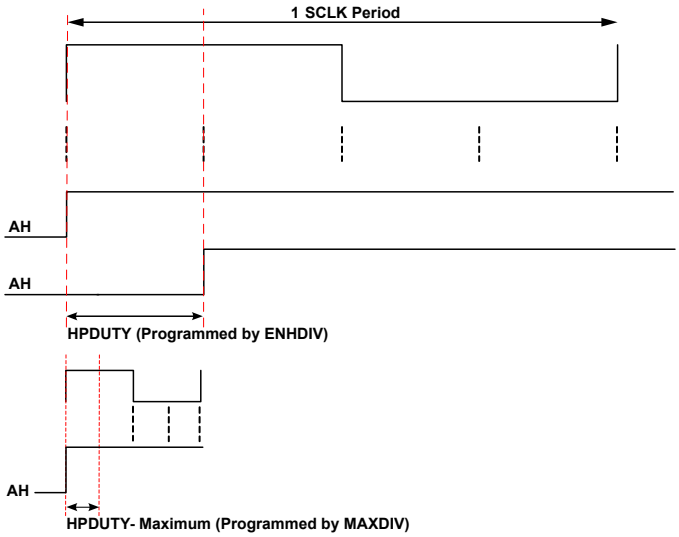


Figure 24-21: Heightened-Precision Steps in a Single SCLK Period

Sample Waveforms for High- and Low-Side with Precision Placement

When the PWM module uses heightened-precision in the dependent mode of operation, both high and low-side outputs shift in the same direction. This operation can result in pulse-expansion of the high-side and pulse-contraction of the low-side or conversely.

The ***Output Shift in The Same Direction*** figure shows an example of a case with DT =1, and pulse expansion occurs on the high-side and pulse-contraction on the low-side. It juxtaposes a case where the PWM module does not use heightened-precision and a case where PWM does use it.

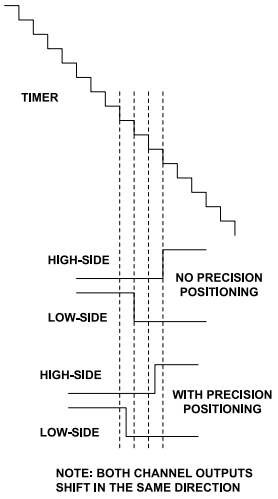


Figure 24-22: Output Shift in The Same Direction

The following ***Precision Placement: PULSEMODE*** figures illustrate the cases for pulse modes 1, 2, and 3 (pulse mode 00 is a trivial case of pulse mode 01). The pulse modes are configured in the `PWM_ACTL` and `PWM_BCTL` channel control registers. The figures show what happens to the edges as a decimal part is added to a programmed positive duty. In each case, assume that the original `PWM_AH0.DUTY` register value, which is the coarse duty value, changes to the `PWM_AH1.DUTY` value after programming the enhanced-resolution (`PWM_AH0_HP`, `PWM_AH1_HP`) registers. For example, changing 14 to 14.25 and 10 to 10.75. The figures are not drawn to these numbers. For negative duty values, the shifts are in the opposite direction.

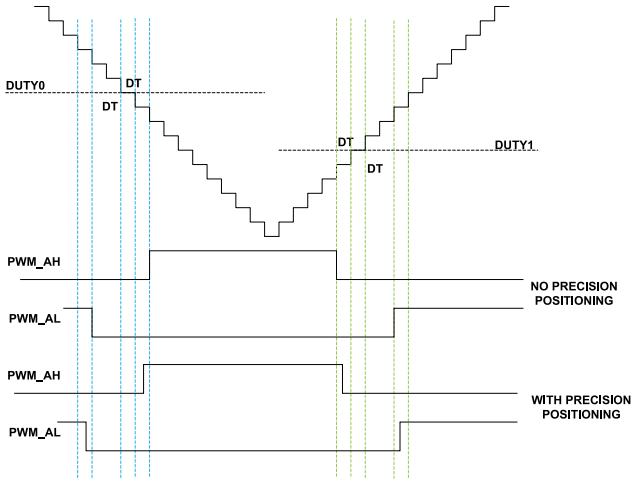


Figure 24-23: Precision Placement: PULSEMODE=01

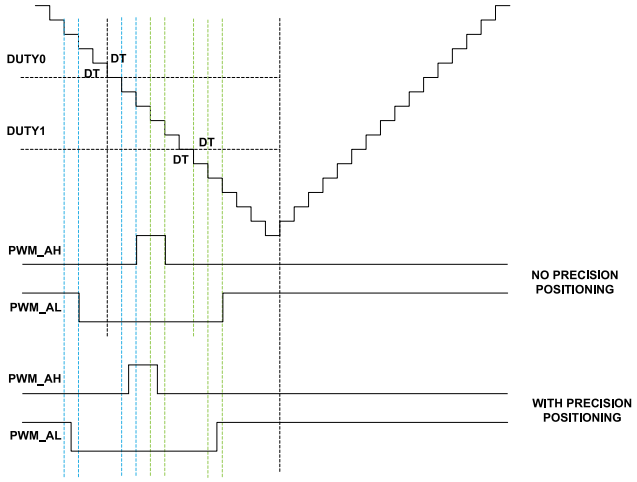


Figure 24-24: Precision Placement: PULSEMODE=10

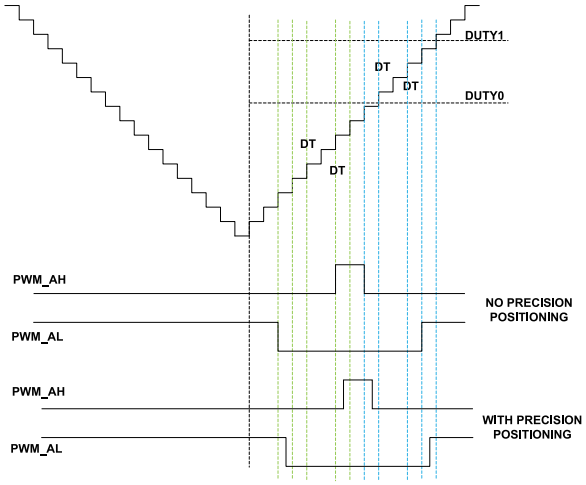


Figure 24-25: Precision Placement: PULSEMODE=11

Emulation Mode

Depending upon system configuration, a debug halt may be presented to the PWM unit. The PWM has local control of its response to an emulator halt based on the `PWM_CTL.EMURUN` bit setting.

- `PWM_CTL.EMURUN` =1. When the processor is halted in emulation mode, the outputs continue to toggle and be driven out of the PWM block. The counters and status register bits are set or reset according to the PWM timer count or period settings.
- `PWM_CTL.EMURUN` =0. When the processor is halted in emulation mode, the outputs are shut down (enter their inactive state based on polarity), and all counters that affect status register bits are paused.

  The `PWM_STAT.EMU` bit is set.
- At restart, the PWM counters resume from their paused value. The outputs are still held in their inactive state.

  To reactivate the outputs, clear the `PWM_STAT.EMU` bit with a W1C operation.

NOTE: The PWM_STAT.EMU bit is not cleared by disabling the PWM or writing 0 to the PWM_CTL.EMURUN bit.

Emergency dead time is not ensured on re-enabling the outputs by a W1C operation to the PWM_STAT.EMU bit.

Sub SCLK heightened-precision edge placements can be off on the first output edge for every channel on clearing the PWM_STAT.EMU bit.

Event Control

The PWM uses bits in the PWM_IMSK and PWM_ILAT registers for event control. These registers allow masking and show masked interrupt request status bits, respectively. The interrupt status bits are latched and held on the interrupt event. The software must write a 1 to clear the interrupt status bit, usually during the interrupt service routine.

The timer period (TMRxPER) interrupt requests are configured using the PWM_ILAT.TMR0PER - PWM_ILAT.TMR4PER bits. The PWM uses the interrupts to execute an interrupt service routine (ISR) periodically. The ISR updates the PWM channel control and duty registers (according to a control algorithm based on expected operation and sampled existing operation). The TMRxPER interrupts also can trigger an ADC to sample data for use during the ISR.

The PWM uses the PWM_CTL.INTSYNCREF bit field to control the PWM_SYNC interrupt request. The PWM uses the bit field to assign the interrupt request to a system user interrupt of the core. The PWM_SYNC can be configured to be either internal or externally driven using the PWM_CTL.EXTSYNC bit. When configured as an external sync, the signal can be further configured as synchronous or asynchronous using the PWM_CTL.EXTSYNCSEL bit.

As an example, when the PWM_SYNC interrupt is serviced:

- The ADC samples data
- The data is algorithmically interpreted
- New PWM channel duties are calculated and written to the PWM

More sophisticated implementations include different startup, run time, and shutdown algorithms to determine PWM channel duties based on expected behavior and further features.

During the PWM_SYNC interrupt driven control loop, only the channel delay registers, the duty registers, and the channel C high pulse duty register values are typically updated. To see programming limitations on the PWM registers, see the "Register Descriptions".

Status information about the PWM is available in the PWM_STAT register, which stores all status bits, including raw interrupt status bits. In particular, the period boundary of each timer is available, as well as status bits. The PWM uses the status bits to indicate whether the operation is in the first half or the second half of the timer. Additionally, the TRIP status is also available. For more information on TRIP interrupts, see Trip Control Unit .

Trip Control Unit

Each PWM unit features two TRIP input ports: TRIP0 and TRIP1. The TRIP0 port can be connected to any of five input signals as noted below: three active-low pins PWM_TRIPA, PWM_TRIPB, and PWM_TRIPC and two pins from the analog front end: AFE_OK and COMP_OUT_IRQ. The ***PWM Trip Signal Mapping*** table details the control bits which enable the signal connection. Note that the SYSBLK_PWM_SYS_CFG register allows programming any combination of trigger sources independently for TRIP0 port of each PWM unit. The ***PWM Trip Input Ports*** figure shows how the package pins and control bits are used as inputs to the PWM trip unit port.

Table 24-5: PWM Trip Signal Mapping

Enable bit in **SYSBLK_PWM_SYS_CFG**	Destination PWM Unit	PWM Trip Unit Port
SYSBLK_PWM_SYS_CFG.PWM0TRIPMX[0]	PWM0	TRIP0
SYSBLK_PWM_SYS_CFG.PWM0TRIPMX[1]		
SYSBLK_PWM_SYS_CFG.PWM0TRIPMX[2]		
SYSBLK_PWM_SYS_CFG.FOCPTRIPEN0		
SYSBLK_PWM_SYS_CFG.AFEOKTRIPEN0		
SYSBLK_PWM_SYS_CFG.PWM1TRIPMX[0]	PWM1	TRIP0
SYSBLK_PWM_SYS_CFG.PWM0TRIPMX[1]		
SYSBLK_PWM_SYS_CFG.PWM0TRIPMX[2]		
SYSBLK_PWM_SYS_CFG.FOCPTRIPEN1		
SYSBLK_PWM_SYS_CFG.AFEOKTRIPEN0		
SYSBLK_PWM_SYS_CFG.PWM2TRIPMX[0]	PWM2	TRIP0
SYSBLK_PWM_SYS_CFG.PWM2TRIPMX[1]		
SYSBLK_PWM_SYS_CFG.PWM2TRIPMX[2]		

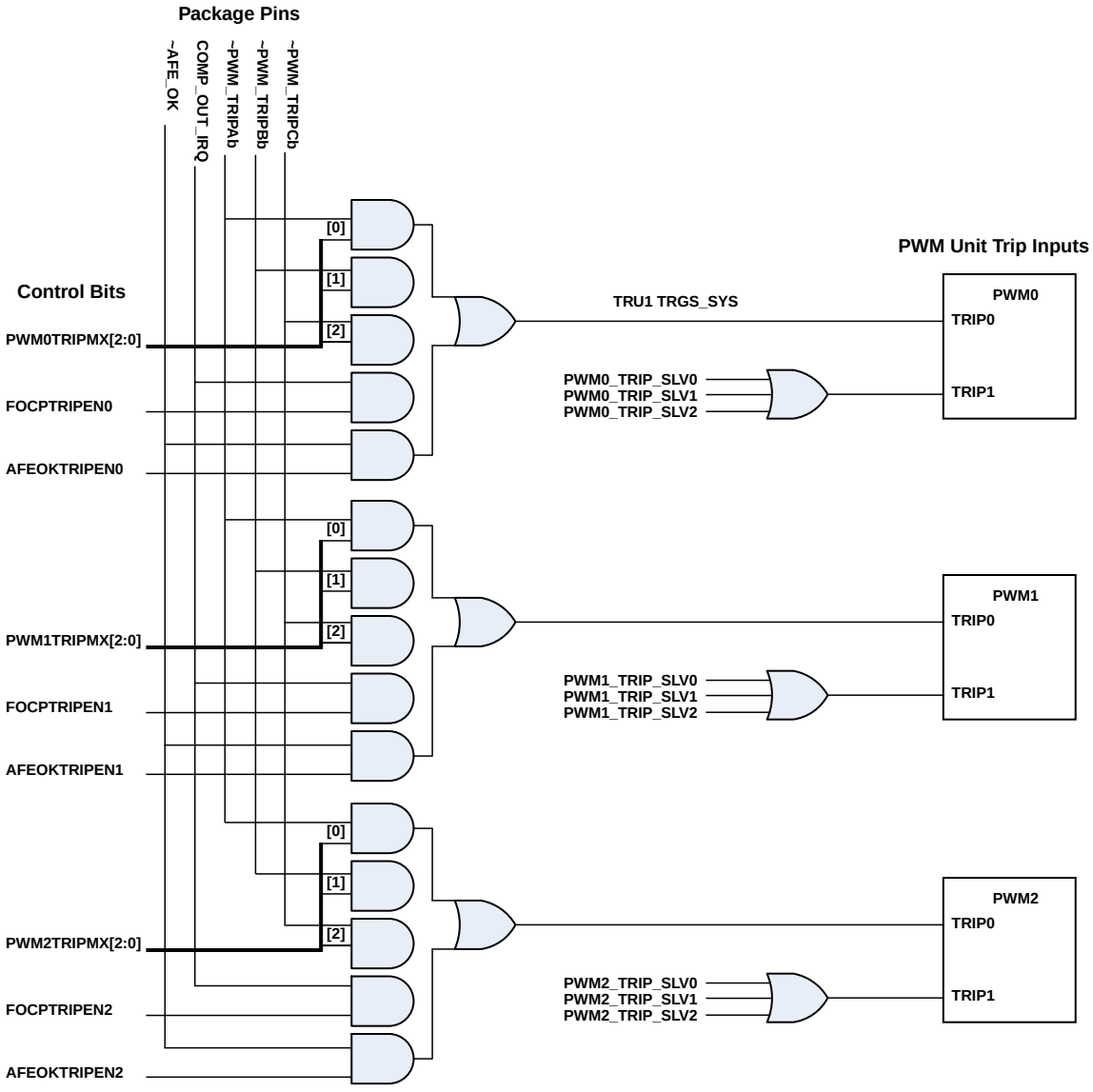


Figure 24-26: PWM Trip Input Ports

The PWM output signals can be shut-off in a number of different ways. The trip inputs ($\overline{\texttt{PWM\_TRIP[n]}}$) can be mapped to provide either a temporary or permanent shutdown on any channel outputs (hi/lo/pair). This shutdown mechanism is asynchronous so that the associated PWM output disable circuitry does not go through any clocked logic. This functionality ensures correct PWM shutdown even in the event of a loss of the processor clock. In addition to the hardware shutdown features, the PWM system can be shut down in software with the `PWM_CTL.SWTRIP` bit.

The external trip signals $\overline{\texttt{PWM\_TRIP0}}$, and $\overline{\texttt{PWM\_TRIP1}}$ that come from GPIO generate the TRIP0 input whereas the trigger slaves PWM0_TRIP_TRIGn, PWM1_TRIP_TRIGn and PWM2_TRIP_TRIGn generate TRIP1 input.

The external trip signals $\overline{\texttt{PWM\_TRIPA}}$, $\overline{\texttt{PWM\_TRIPB}}$, and $\overline{\texttt{PWM\_TRIPC}}$ that come from GPIO generate the TRIP0 input whereas the trigger slaves PWM0_TRIP_SLVn, PWM1_TRIP_SLVn and PWM2_TRIP_SLVn generate TRIP1 input. In addition, the internal signals SYSBLK_SYSSTAT.AFE_LIMIT (indicating a detection on the AFE fast overcurrent protection comparators) and SYSBLK_SYSSTAT.AFE_OK (indicating a fault condition on the AFE) can also generate the TRIPO0 input. Each input connection of the GPIO or AFE to INPUT0 of each PWM block can be individually enabled. See the SYSBLK_PWM_SYS_CFG register for details.

During any external trip event (if not disabled), the PWM outputs are turned off. When a PWM output is turned off, it means that the output level is held at a polarity opposite that given in the PWM_CHANCFG.POLAH through PWM_CHANCFG.POLDH (channel high side polarity) bits. The PWM sync pulse continues to operate, when it is already enabled. A PWMTRIP interrupt occurs if unmasked, to notify the software of this event. In dependent mode of operation, both high and low-side outputs refer to the channel high side polarity bits.

Even if the clock to the PWM is damaged, an external trip event turns off the PWM outputs. In this case, the PWMTRIP interrupt request may not occur.

An additional level of protection for PWM outputs is available in the PORTs (GPIO) units, called pin safe state. This feature allows a pre-programmed safe state (0,1, or tristate) to be individually selected for each GPIO. The state is asserted asynchronously upon selected conditions, and with a selected microsecond-scale delay even if the clocks or one of the power supplies is out of specification. See the PORT chapter for details.

The PWM trip unit processes hardware or software fault conditions and shuts down the PWM channel outputs immediately on the occurrence of these conditions. The PWM can enable the shutdown mechanism separately for each channel. The design also allows for a self-restart mechanism to be enabled on a channel. Self-restart reenables the channel outputs following the fault condition (allowed only on hardware trips) when the PWMTMRy that the channel is using reaches its period boundary.

1. $\overline{\texttt{PWM\_TRIPA}}$ input pin
2. $\overline{\texttt{PWM\_TRIPB}}$ input pin
3. $\overline{\texttt{PWM\_TRIPC}}$ input pin
4. SYSBLK_SYSSTAT.AFE_OK signal
5. SYSBLK_SYSSTAT.AFE_LIMIT signal

The $\overline{\texttt{PWM\_TRIPA}}$, $\overline{\texttt{PWM\_TRIPB}}$, and $\overline{\texttt{PWM\_TRIPC}}$ and SYSBLK_SYSSTAT.AFE_OK signals are active low for the trip mechanism, where a falling edge indicates a fault condition. The SYSBLK_SYSSTAT.AFE_LIMIT is active-high, where a rising edge indicates a fault condition.

The trip unit can shut down an output of a particular channel in response to the fault event on any of these signal sources. To enable this functionality, program the connection of the signal source to the TRIP0 input of the particular PWM unit using the SYSBLK1.PWM_TRIP_CFG register. Then, program the PWM_TRIPCFG.EN0A bit corresponding to the channel.

Program the PWM_TRIPCFG.MODE0A bits to specify the restart mechanism for a channel that has been tripped.

1. If the PWM_TRIPCFG.MODE0A bit =0, once tripped, a trip condition is registered on this channel in the PWM_STAT.FLTTRIPA bit and the outputs of that channel are immediately shut down. This condition is called a *fault trip* condition. To resume channel output when a fault trip occurs, write a 1 to clear the PWM_STAT.FLTTRIPA bit. A processor write cannot clear the bit when the trip condition is still active. The raw trip status is available for both pins in the PWM_STAT.RAWTRIP0 register bits.
2. If the PWM_TRIPCFG.MODE0A bit =1, once tripped, a trip condition is registered on this channel in the PWM_STAT.SRTRIPA bit and the outputs of that channel are immediately shut down. This condition is called a *self-restart trip* condition. If the trip condition is not active at the next period boundary of the PWMTMRy that the channel is using, the status register bit is cleared. The outputs are restored.

The trip input pins have an internal pull-up resistor on the chip pin.

In addition to the hardware trip conditions, a global software trip bit in the PWM_CTL register allows for a software-forced fault trip condition. When the global software trip bit is set to 1, irrespective of the values in the PWM_TRIPCFG register, it sets all the PWM_STAT.FLTTRIPA bits and also gates the channel outputs. To remove the trip condition from the channel, perform a W1C on the PWM_STAT.FLTTRIPA bit of the particular channel.

If the PWM_TRIPCFG.EN0A bit is set to 1 to, for any channel, then the occurrence of a fault condition on the PWMTRIPy bit is logged in the PWM_STAT.FLTTRIPA register bit. If the corresponding PWM_IMSK.TRIP0 bit = 1, then an interrupt request is generated. Tripping a channel output does not interfere with PWM_SYNC generation.

The ***Operation Under Hardware Fault Conditions*** figure shows an example where PWMTRIP0 is enabled on channel A as self-restart trip. Channel A works with the PWM_CHANCFG.POLAH bit =1. In period 2, the PWM_AH signal is full on modulated, and tries to rise at the period boundary where the self-restart occurs for the channel. However, since the low-side output of the channel was only recently removed due to a trip, the rise edge on PWM_AH is delayed until the emergency dead-time period is over. PWMTRIP1 is enabled on channel B as a fault trip. Channel B works with the PWM_CHANCFG.POLAH bit =0. PWMTRIP1 stays low for an extended time period. The first processor write to reenable the channel output fails. The second processor write passes since the fault condition has gone away.

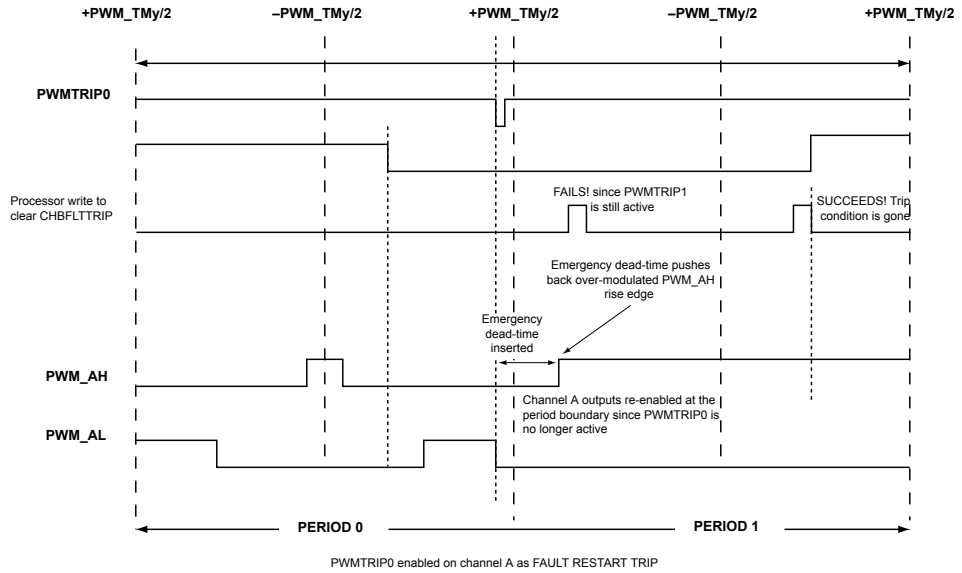


Figure 24-27: PWM Trip Input Ports, Restart

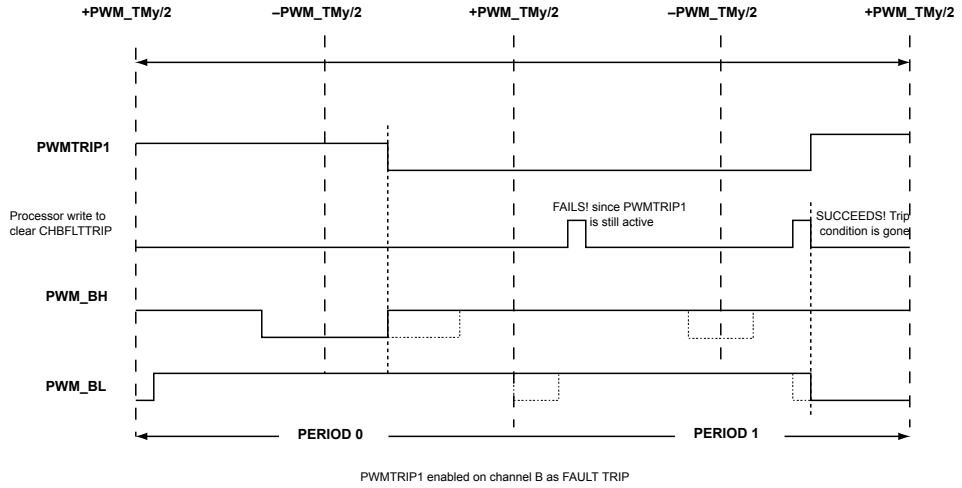


Figure 24-28: PWM Trip Input Ports, Fault

NOTE: Dead time is ensured on re-enabling the channel outputs after trip.

NOTE: Programs must not allow changes in the configuration or enable bits of `PWM_TRIPCFG` register within ± 10 clock cycles of when the external trip pulse toggles. (The configuration or enable bits of `PWM_TRIPCFG` register select between trip enable and disable). If this time frame is not followed, then unexpected behavior occurs.

Trip Mechanisms

The PWM controller has various mechanisms that send the PWM output into the trip state.

Software Trip - With a single write to the `PWM_CTL.SWTRIP` register, any combination of the eight PWM outputs can be forced into the trip state. (Each PWM unit has a separate such register). This mechanism a synchronous.

The processor and clocks must be functioning, and separate MMR writes trip the PWM outputs in different PWM units.

TRU Hardware Trip - Events routed (synchronously) through the TRU1 can cause any desired combination of individual trip outputs to go to their trip states. Each output can be individually programmed to be sensitive to this trip source (TRIP1) (for example, FOCP over-current protection comparator limit detection from the AFE).

Pin Input Hardware Trip - Events detected outside the processor (for example, from external gate drivers), can cause trips by connecting such signals to the `PWM_TRIPA`/`PWM_TRIPB`/`PWM_TRIPC` package input pins (assignable GPIOs).

Output Pin Safe State

For each PWM output, there are several mechanisms available for switching the output waveform to its predefined safe state The mechanisms can be individually programmed in either the `PWM_TRIPCFG` register within the PWM unit, or within the `PADS_PORT[n]_TRIPST`m registers for GPIO control.

Using the GPIO pin safe state functions in the PORT blocks, each pin assigned to PWM outputs can be programmed to (asynchronously) respond to serious system faults, including VDD supply out-of-range and system clocks out-of-frequency range.

Any pin safe state response to arbitrary events detected by M4-side or M0-side resources can be enabled using TRU1 or TRU0 trigger routing to the TRU1 TRGS_SYS_FRC_PIN_SAFE_SLVn slaves. For example, a pin-input hardware trip can be debounced in the DBC unit and routed through the TRGM_SYS_PWM_DBC_TRIPA/B/C_IN trigger master to the trigger slaves.

Although the GPIO pin safe state control and PWM trip configuration provide overall control, there are a few differences to note:

- The GPIO pin safe state mechanism has higher priority than the mechanism of the PWM unit.
    - If both trip mechanisms are activated at the same time, the state programmed in the GPIO pin safe state logic is applied.
    - The GPIO pin safe state can be programmed for a trip delay, measured in microseconds. This option facilitates programming an early trip state in the PWM unit registers, and programming an independent late trip state in the GPIO registers. Both states are unrelated to the output level generated by the PWM timer.
    - The GPIO pin safe states are stored in asynchronous 3.3v logic, while the PWM unit is embodied in synchronous 1.2v logic. In the event of loss of clock or of loss of VDDINT, the GPIO pin safe state mechanism responds as programmed.
- In the PWM unit registers, a state of High_Z selects disabling the output driver, but the weak pull-up resistor continues to drive (assuming the GPIO pin safe state for that output is not activated or selected).
- In the GPIO pin safe state registers, a state of High-Z selects disabling both the output driver and the weak pull-up resistor, for a true high-impedance state.

Programming Model

The following sections provide general (and some application-specific) programming steps for configuring and using the PWM module.

- Programming Model for Three-Phase AC Motor Control

Programming Model for Three-Phase AC Motor Control

The ***PWM Module and Interaction with System*** figure shows how the PWM unit (green) interfaces to both software (blue) and hardware (yellow). The software configures the unit, calculates duty cycles (Duty A, Duty B, Duty C), and services the interrupts generated by the module (PWM SYNC IRQ, TRIP IRQ). The hardware applies the gate signals (AH, AL, BH, BL, CH, CL) to the inverter and provides an over-current trip signal back to the unit (TRIP0).

The typical three-phase AC motor configuration shown in the ***PWM Module and Interaction with System*** figure applies for both permanent magnet and induction motor types.

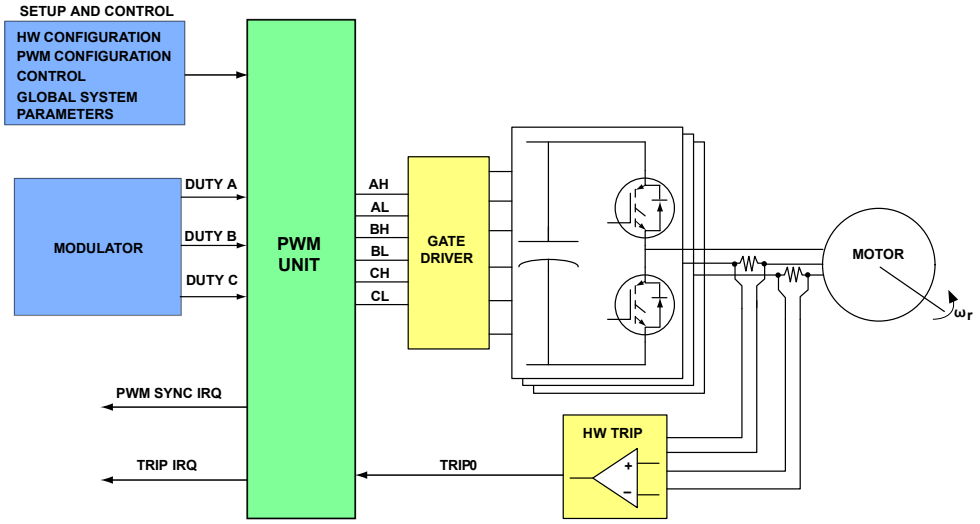


Figure 24-29: PWM Module and Interaction with System

System Parameters

The following system parameters (characteristics) influence the module configuration for this application. This example system features:

- One three-phase AC machine
- B6 inverter
- SVPWM, including both linear- and over-modulation
- Switching frequency of 20 kHz
- Dead time of 1us

- Trip signal generated by hardware
- Active high-level gate drive
- Core frequency of 200 MHz
- Peripheral clock of 100 MHz

System State Sequencing

Managing the system state and sequence of states is critically important when programming the PWM module. The ***PWM System States*** figure provides an overview of these states.

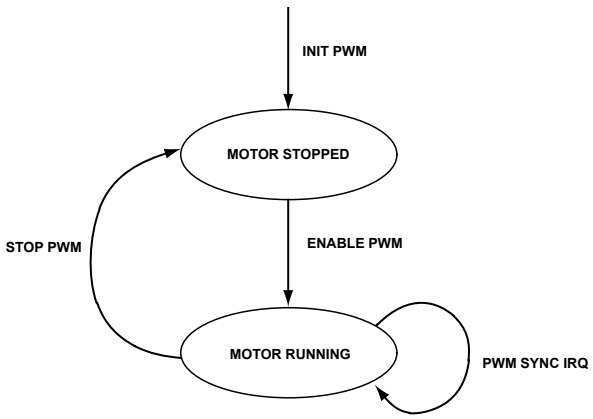


Figure 24-30: PWM System States

As shown in the state diagram, the module configuration is updated on state transitions (indicated by the arrows). The transitions are initialization, motor start, PWM sync interrupt request (on each), and motor stop. The following sections discuss the transitions in detail.

- PWM Initialization for Motor Control
- PWM Enable for Motor Control
- PWM Response to Sync Interrupt for Motor Control
- PWM Disable (and Stop the Motor) for Motor Control

PWM Initialization for Motor Control

The processor must program the PWM at power-up as follows and repeat this programming to bring the PWM and the system into a known (safe) state.

1. Place the PWM module in a safe state and set up synchronization of the module using the following bitwise operations on the `PWM_CTL` and `PWM_CHANCFG` registers:

```
PWM_CTL &= 0xFFE0FF08
PWM_CTL |= 0x20000
```

```
PWM_CHANCFG &= 0x80808080
PWM_CHANCFG |= 0x24242424
```

These operations result in the following bit settings:

- Disable PWM (PWM_CTL.GLOBEN =0)
- Disable delay for channels A, B, C, D (PWM_CTL.DLYAEN through PWM_CTL.DLYDEN =0). All phases must run with same phase.
- Use internal synchronization by timer TMR0 (PWM_CTL.EXTSYNC =0, PWM_CTL.EXTSYNCSEL =1)
- The same timer, TMR0 (PWM_CTL.INTSYNCREF = b#000) synchronizes all phases.
- Low-side is always the inverse of high-side (PWM_CHANCFG.POLAL through PWM_CHANCFG.POLDL = 1).
- System uses active high gate driver (PWM_CHANCFG.ENCHOPAH through PWM_CHANCFG.ENCHOPDH =1).
- Disable gate chopping (PWM_CHANCFG.ENCHOPAL through PWM_CHANCFG.ENCHOPDL =0). PWM does not use the pulse transformer.

2. Set up the trip and associated interrupt requests using the following bitwise operations on the PWM_TRIPCFG and PWM_ILAT registers:

```
PWM_TRIPCFG &= 0xF0F0F0F0
PWM_TRIPCFG |= 0x1010101
PWM_ILAT &= 0xFFE0FFFC
PWM_ILAT |= 0x1
```

These operations result in the following bit settings:

- All phases must shut down simultaneously in case of fault: (PWM_TRIPCFG.EN0A through PWM_TRIPCFG.EN0D =0, PWM_TRIPCFG.MODE0A through PWM_TRIPCFG.MODE0D =0, PWM_TRIPCFG.EN1A through PWM_TRIPCFG.EN1D =0, PWM_TRIPCFG.MODE1A through PWM_TRIPCFG.MODE1D =0)
- Enable TRIP0 as fault trigger for all channels (PWM_TRIPCFG.EN0A through PWM_TRIPCFG.MODE1D =1).
- For thermal control and synchronization, SW intervention is needed at trip. Do not use automatic restart of any channels.
- Generate an interrupt at trip on TRIP0 (PWM_ILAT.TMR0PER = 1).

3. Configure the PWM channels using the following bitwise operations on the PWM_TRIPCFG and PWM_ILAT registers:

```
PWM_CHA_DT=0x32 PWM_CHB_DT=0x32 PWM_CHC_DT=0x32
PWM_TM0 = 0x9C4
PWM_ACTL = 0xFFFFF0000
PWM_BCTL = 0xFFFFF0000
PWM_CCTL = 0xFFFFF0000
PWM_AH0 = 0x0
PWM_BH0 = 0x0
PWM_CH0 = 0x0
```

These operations result in the following bit settings:

- Configure a dead time of 1 µs (DT =0x32 =0x32)
- Configure a PWM frequency of 20 kHz (assuming a system clock frequency of 100Mhz, so a clock divisor of 5000:1) (PWM_TM0 =0x9C4)
- Disable all outputs (PWM_ACTL.DISHI through PWM_CCTL.DISHI =0, PWM_ACTL.DISLO through PWM_CCTL.DISLO =0)
- Use conventional PWM, disable crossover (PWM_ACTL.XOVR through PWM_CCTL.XOVR =0)
- Use symmetrical pulse position on all outputs (PWM_ACTL.PULSEMODEHI through PWM_CCTL.PULSEMODEHI =0, PWM_ACTL.PULSEMODELO through PWM_CCTL.PULSEMODELO =0)
- Set an initial duty-cycle of 50% (PWM_AH0 throughPWM_CH0 =0x0)

PWM Enable for Motor Control

The processor must do the following programming to enable the PWM before starting the motor.

1. Start the PWM timer TMR0 using the PWM_CTL |=0x1 bitwise operation on the PWM_CTL register.

   ADDITIONAL INFORMATION: This operation has the same effect as setting the PWM_CTL.GLOBEN bit =1.

2. Enable six PWM outputs using the following bitwise operations on the PWM_ACTL through PWM_CCTL registers.

   ```
   PWM_ACTL|=0x3
   PWM_BCTL|=0x3
   PWM_CCTL|=0x3
   ```

   ADDITIONAL INFORMATION: These operations have the same effect as enabling high and low-side channel outputs by setting the PWM_ACTL.DISHI through PWM_CCTL.DISHI bits =1 and the PWM_ACTL.DISLO through PWM_CCTL.DISLO bits =1.

3. Enable the PWM TRIP0 interrupt using the PWM_ILAT |= 0x1 bitwise operation on the PWM_ILAT register.

   ADDITIONAL INFORMATION: This operation has the same effect as setting the PWM_ILAT.TRIP0 bit =1

PWM Response to Sync Interrupt for Motor Control

When the PWM sync interrupt occurs, the processor could need to update to the PWM duty cycle with a value calculated by the motor control algorithm. This application uses symmetric pulses position and uses dependent high and low-side output. So, the PWM updates only one register for each phase.

1. Write the new duty cycle value (calculated by motor control algorithm) to the timer when the sync interrupt occurs.

   The following bitwise operations on the PWM_AH0 through PWM_CH0 registers accomplish this task:

```
PWM_AH0 = Duty_A_mc_algorithm_current_value
PWM_BH0 = Duty_B_mc_algorithm_current_value
PWM_CH0 = Duty_C_mc_algorithm_current_value
```

PWM Disable (and Stop the Motor) for Motor Control

The processor must program the PWM as follows to stop the motor, disable the PWM, and disable PWM interrupts. These actions place the PWM and system in a safe, passive state.

1. Disable the PWM timer using the PWM_CTL &= 0xFFFFFFFE bitwise operation on the PWM_CTL register.

   ADDITIONAL INFORMATION: This operation has the same effect as clearing the PWM_CTL.GLOBEN bit =0.

2. Disable all PWM outputs using the following bitwise operations on the PWM_ACTL through PWM_CCTL registers.

```
PWM_ACTL &= 0xFFFFFFFC
PWM_BCTL &= 0xFFFFFFFC
PWM_CCTL &= 0xFFFFFFFC
```

   These operations disable PWM outputs where the PWM_ACTL.DISHI through PWM_CCTL.DISHI bits =1 and the PWM_ACTL.DISLO through PWM_CCTL.DISLO bits =0)

3. Set the PWM duty-cycle to 50% using the following bitwise operations on the PWM_AH0 through PWM_CH0 registers.

```
PWM_AH0 =0x0
PWM_BH0 =0x0
PWM_CH0 =0x0
```

   These operations have the same effect as clearing the PWM_AH0.DUTY through PWM_CH0.DUTY bit =0.

4. Disable the PWM TRIP0 interrupt request using the `PWM_ILAT &= 0xFFFFFFFE` bitwise operation on the `PWM_ILAT` register.

   ADDITIONAL INFORMATION: This operation has the same effect as clearing the PWM TRIP0 interrupt request `PWM_ILAT.TRIP0` =0.

CM41X_M4 PWM Register Descriptions

Pulse-Width Modulator (PWM) contains the following registers.

Table 24-6: CM41X_M4 PWM Register List

Name	Description
PWM_ACTL	Channel A Control Register
PWM_AH0	Channel A-High Duty-0 Register
PWM_AH0_HP	Channel A-High Heightened-Precision Duty-0 Register
PWM_AH1	Channel A-High Duty-1 Register
PWM_AH1_HP	Channel A-High Heightened-Precision Duty-1 Register
PWM_AH_DUTY0	Channel A-High Full Duty0 Register
PWM_AH_DUTY1	Channel A-High Full Duty1 Register
PWM_AL0	Channel A-Low Duty-0 Register
PWM_AL0_HP	Channel A-Low Heightened-Precision Duty-0 Register
PWM_AL1	Channel A-Low Duty-1 Register
PWM_AL1_HP	Channel A-Low Heightened-Precision Duty-1 Register
PWM_AL_DUTY0	Channel A-Low Full Duty0 Register
PWM_AL_DUTY1	Channel A-Low Full Duty1 Register
PWM_BCTL	Channel B Control Register
PWM_BH0	Channel B-High Duty-0 Register
PWM_BH0_HP	Channel B-High Heightened-Precision Duty-0 Register
PWM_BH1	Channel B-High Duty-1 Register
PWM_BH1_HP	Channel B-High Heightened-Precision Duty-1 Register
PWM_BH_DUTY0	Channel B-High Full Duty0 Register
PWM_BH_DUTY1	Channel B-High Full Duty1 Register
PWM_BL0	Channel B-Low Duty-0 Register
PWM_BL0_HP	Channel B-Low Heightened-Precision Duty-0 Register
PWM_BL1	Channel B-Low Duty-1 Register
PWM_BL1_HP	Channel B-Low Heightened-Precision Duty-1 Register

Table 24-6: CM41X_M4 PWM Register List (Continued)

Name	Description
PWM_BL_DUTY0	Channel B-Low Full Duty0 Register
PWM_BL_DUTY1	Channel B-Low Full Duty1 Register
PWM_CCTL	Channel C Control Register
PWM_CH0	Channel C-High Pulse Duty Register 0
PWM_CH0_HP	Channel C-High Pulse Heightened-Precision Duty Register 0
PWM_CH1	Channel C-High Pulse Duty Register 1
PWM_CH1_HP	Channel C-High Pulse Heightened-Precision Duty Register 1
PWM_CHANCFG	Channel Configuration Register
PWM_CHA_DT	Channel A Dead-time Register
PWM_CHB_DT	Channel B Dead-time Register
PWM_CHC_DT	Channel C Dead-time Register
PWM_CHD_DT	Channel D Dead-time Register
PWM_CHOPCFG	Chop Configuration Register
PWM_CH_DUTY0	Channel C-High Full Duty0 Register
PWM_CH_DUTY1	Channel C-High Full Duty1 Register
PWM_CL0	Channel C-Low Pulse Duty Register 0
PWM_CL0_HP	Channel C-Low Pulse Duty Register 1
PWM_CL1	Channel C-Low Duty-1 Register
PWM_CL1_HP	Channel C-Low Heightened-Precision Duty-1 Register
PWM_CL_DUTY0	Channel C-Low Full Duty0 Register
PWM_CL_DUTY1	Channel C-Low Full Duty1 Register
PWM_CTL	Control Register
PWM_DCTL	Channel D Control Register
PWM_DH0	Channel D-High Duty-0 Register
PWM_DH0_HP	Channel D-High Pulse Heightened-Precision Duty Register 0
PWM_DH1	Channel D-High Pulse Duty Register 1
PWM_DH1_HP	Channel D High Pulse Heightened-Precision Duty Register 1
PWM_DH_DUTY0	Channel D-High Full Duty0 Register
PWM_DH_DUTY1	Channel D-High Full Duty1 Register
PWM_DL0	Channel D-Low Pulse Duty Register 0
PWM_DL0_HP	Channel D-Low Heightened-Precision Duty-0 Register

Table 24-6: CM41X_M4 PWM Register List (Continued)

Name	Description
PWM_DL1	Channel D-Low Pulse Duty Register 1
PWM_DL1_HP	Channel D-Low Heightened-Precision Duty-1 Register
PWM_DLYA	Channel A Delay Register
PWM_DLYB	Channel B Delay Register
PWM_DLYC	Channel C Delay Register
PWM_DLYD	Channel D Delay Register
PWM_DL_DUTY0	Channel D-Low Full Duty0 Register
PWM_DL_DUTY1	Channel D-Low Full Duty1 Register
PWM_ILAT	Interrupt Latch Register
PWM_IMSK	Interrupt Mask Register
PWM_STAT	Status Register
PWM_SWTRIP	Software Trip Register
PWM_SYNC_WID	Sync Pulse Width Register
PWM_TM0	Timer 0 Period Register
PWM_TM1	Timer 1 Period Register
PWM_TM2	Timer 2 Period Register
PWM_TM3	Timer 3 Period Register
PWM_TM4	Timer 4 Period Register
PWM_TRIPCFG	Trip Configuration Register
PWM_TRIP_POL	Trip Polarity Register

Channel A Control Register

The PWM_ACTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

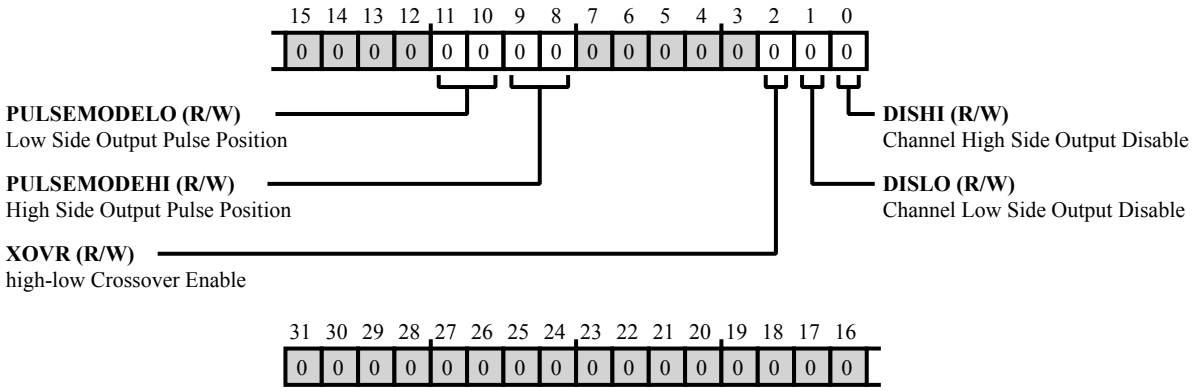


Figure 24-31: PWM_ACTL Register Diagram

Table 24-7: PWM_ACTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_ACTL.PULSEMODELO bits select the pulse position for Channel A low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_AL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_AL0 and PWM_AL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_AL0 and PWM_AL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 24-7: PWM_ACTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_ACTL.PULSEMODEHI bits select the pulse position for Channel A high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_AH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_AH0 and PWM_AH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_AH0 and PWM_AH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_ACTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_ACTL.DISLO bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_ACTL.DISHI bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel A-High Duty-0 Register

The PWM_AH0 and PWM_AH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the channel A duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_ACTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_AH0 register to determine the assertion and deassertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel A high pulse output for a count less than PWM_AH0 and deasserts this output for a count greater than PWM_AH1.

The value range for the PWM_AH0 and PWM_AH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHA_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_AH0 and PWM_AH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHA_DT (two's complement) and +PWM_TM0/2 + PWM_CHA_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHA_DT (two's complement) and -PWM_TM0/2 - PWM_CHA_DT .

Note that using values in the PWM_AH0 or PWM_AH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

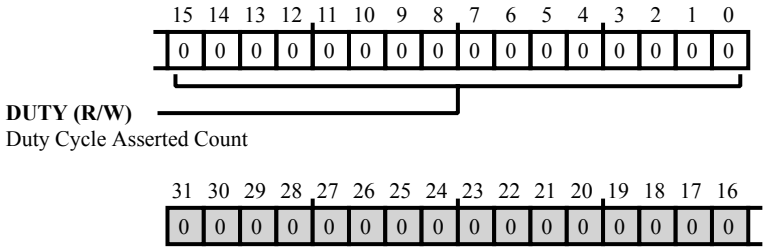


Figure 24-32: PWM_AH0 Register Diagram

Table 24-8: PWM_AH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_AH0.DUTY bits select the duty cycle asserted count for Channel A high side output.

Channel A-High Heightened-Precision Duty-0 Register

The PWM_AH0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_AH0 register, allows programs to specify fractional duty cycles.The PWM_AH0_HP register and the PWM_AH0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_AH0_HP and the PWM_AH0 registers are also present in the single full duty register (if available).

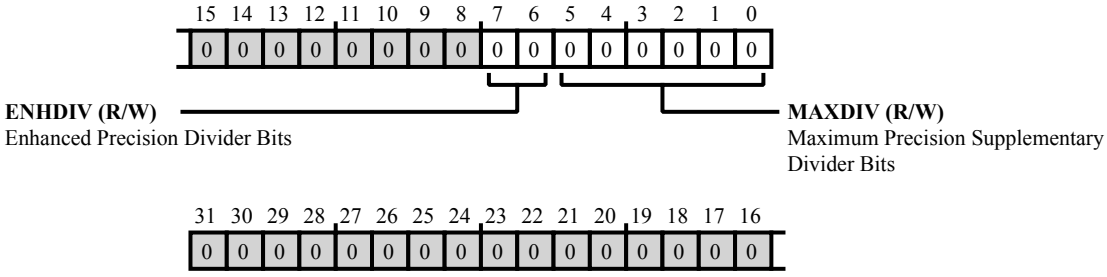


Figure 24-33: PWM_AH0_HP Register Diagram

Table 24-9: PWM_AH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits. The PWM_AH0_HP.MAXDIV bits provide fractional duty cycles for Channel A high side output.

Channel A-High Duty-1 Register

The PWM_AH0 and PWM_AH1 registers determine the width for the high side output pulses. For more information, see the PWM_AH0 register description.

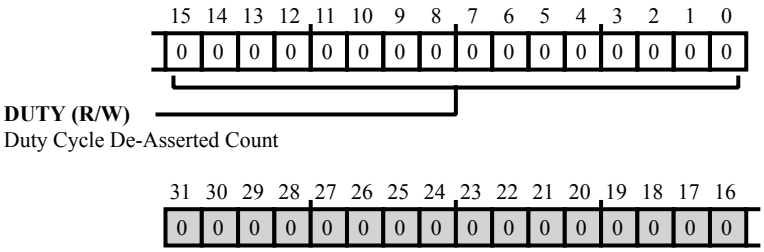


Figure 24-34: PWM_AH1 Register Diagram

Table 24-10: PWM_AH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_AH1.DUTY bits select the duty cycle de-asserted count for Channel A high side output.

Channel A-High Heightened-Precision Duty-1 Register

The PWM_AH1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_AH0 register, allows programs to specify fractional duty cycles.The PWM_AH1_HP register and the PWM_AH1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_AH1_HP and the PWM_AH1 registers are also present in the single full duty register (if available).

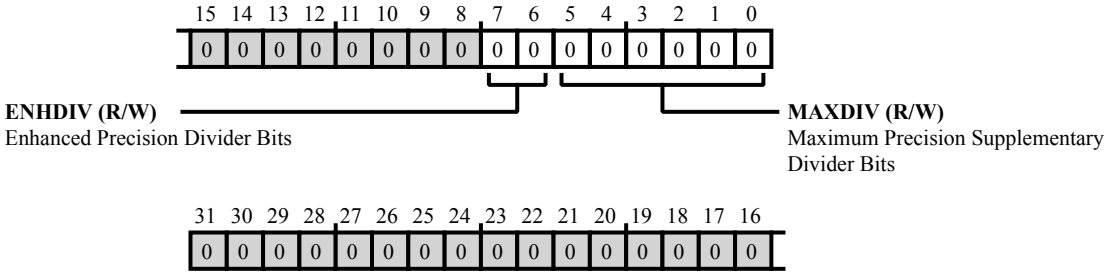


Figure 24-35: PWM_AH1_HP Register Diagram

Table 24-11: PWM_AH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AH1_HP.ENHDIV bits provide fractional duty cycles for Channel A high side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel A-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_AH_DUTY0 register contains the PWM_AH_DUTY0.DUTY bit field from the PWM_AH0 register and the PWM_AH_DUTY0.ENHDIV bit field from the PWM_AH0_HP register.

Note that the PWM_AH_DUTY0 register reads the PWM_AH0 and the PWM_AH0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

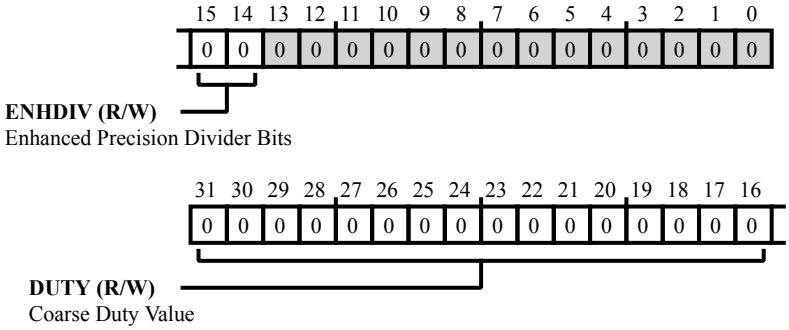


Figure 24-36: PWM_AH_DUTY0 Register Diagram

Table 24-12: PWM_AH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_AH_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_AH_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AH_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_AH_DUTY1 register contains the PWM_AH_DUTY1.DUTY bit field from the PWM_AH1 register and the PWM_AH_DUTY1.ENHDIV bit field from the PWM_AH1_HP register.

Note that the PWM_AH_DUTY1 register reads the PWM_AH1 and the PWM_AH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

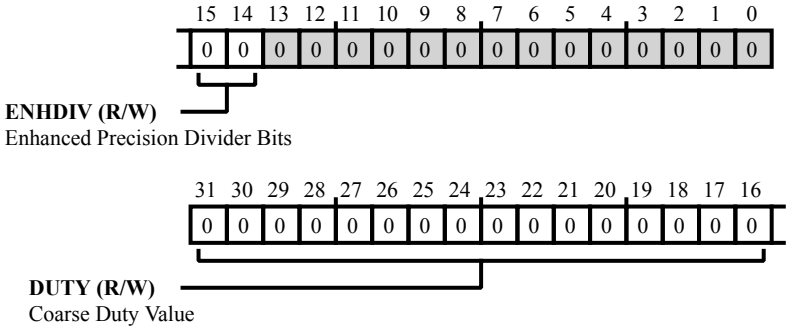


Figure 24-37: PWM_AH_DUTY1 Register Diagram

Table 24-13: PWM_AH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_AH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_AH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-Low Duty-0 Register

The PWM_AL0 and PWM_AL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the channel A duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_ACTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_AL0 register to determine the assertion and deassertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel A low pulse output for count less than PWM_AL0 and deasserts this output for count greater than PWM_AL1.

The value range for the PWM_AL0 and PWM_AL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHA_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_AL0 and PWM_AL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHA_DT (two's complement) and +PWM_TM0/2 + PWM_CHA_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHA_DT (two's complement) and -PWM_TM0/2 - PWM_CHA_DT.

Note that using values in the PWM_AL0 or PWM_AL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

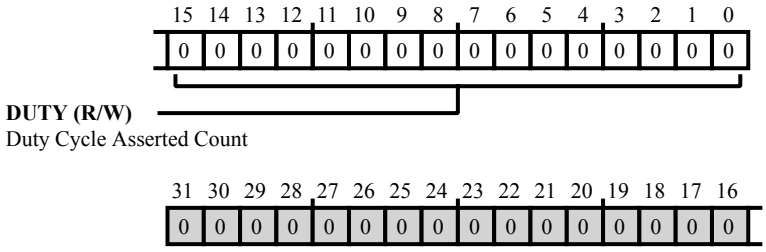


Figure 24-38: PWM_AL0 Register Diagram

Table 24-14: PWM_AL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_AL0.DUTY bits select the duty cycle asserted count for Channel A low side output.

Channel A-Low Heightened-Precision Duty-0 Register

The PWM_AL0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_AL0 register, allows programs to specify fractional duty cycles.The PWM_AL0_HP register and the PWM_AL0 register work together in a Q15.8 signed two's complement fixed-point format. Note that the bit fields in the PWM_AL0_HP and the PWM_AL0 registers are also present in the single full duty register (if available).

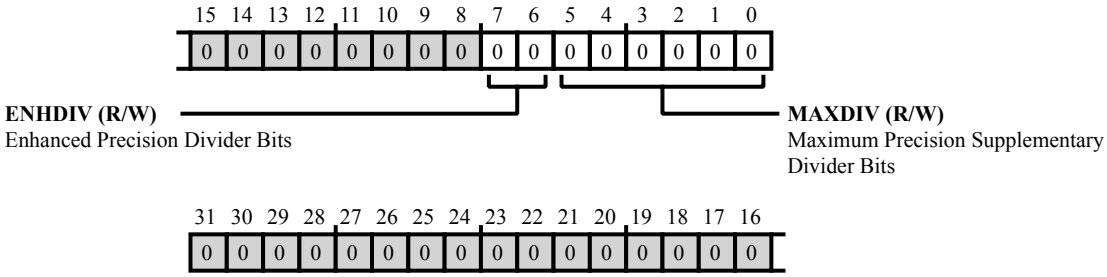


Figure 24-39: PWM_AL0_HP Register Diagram

Table 24-15: PWM_AL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AL0_HP.ENHDIV bits provide fractional duty cycles for Channel A low side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel A-Low Duty-1 Register

The PWM_AL0 and PWM_AL1 registers determine the width for the low side output pulses. For more information, see the PWM_AL0 register description.

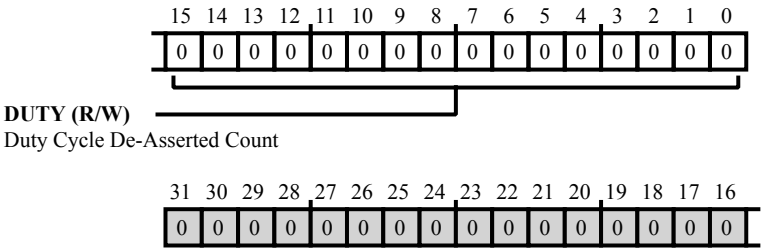


Figure 24-40: PWM_AL1 Register Diagram

Table 24-16: PWM_AL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_AL1.DUTY bits select the duty cycle de-asserted count for Channel A low side output.

Channel A-Low Heightened-Precision Duty-1 Register

The PWM_AL1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_AL1 register, allows programs to specify fractional duty cycles.The PWM_AL1_HP register and the PWM_AL1 register work together in a Q15.8 signed two's complement fixed-point format. Note that the bit fields in the PWM_AL1_HP and the PWM_AL1 registers are also present in the single full duty register (if available).

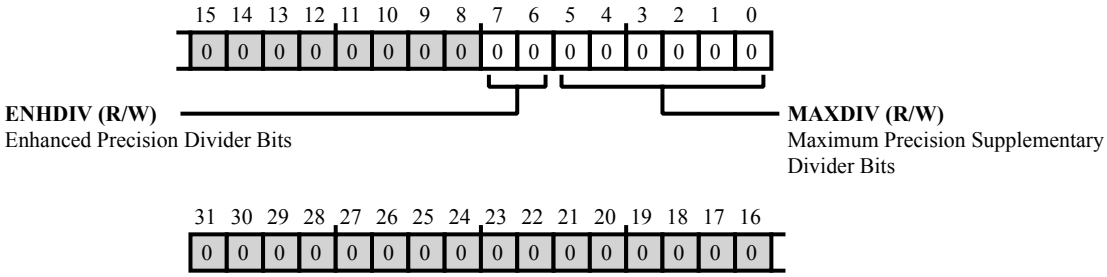


Figure 24-41: PWM_AL1_HP Register Diagram

Table 24-17: PWM_AL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AL1_HP.ENHDIV bits provide fractional duty cycles for Channel A low side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel A-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_AL_DUTY0 register contains the PWM_AL_DUTY0.DUTY bit field from the PWM_AL0 register and the PWM_AL_DUTY0.ENHDIV bit field from the PWM_AL0_HP register.

Note that the PWM_AL_DUTY0 register reads the PWM_AL0 and the PWM_AL0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

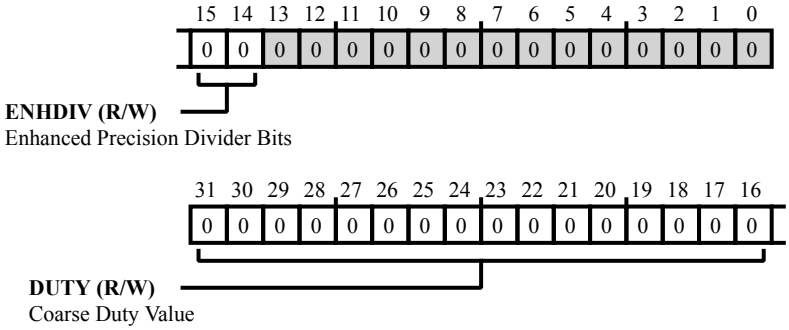


Figure 24-42: PWM_AL_DUTY0 Register Diagram

Table 24-18: PWM_AL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_AL_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_AL_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AL_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel A-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_AL_DUTY1 register contains the PWM_AL_DUTY1.DUTY bit field from the PWM_AL1 register and the PWM_AL_DUTY1.ENHDIV bit field from the PWM_AH0_HP register.

Note that the PWM_AL_DUTY1 register reads the PWM_AL1 and the PWM_AL1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

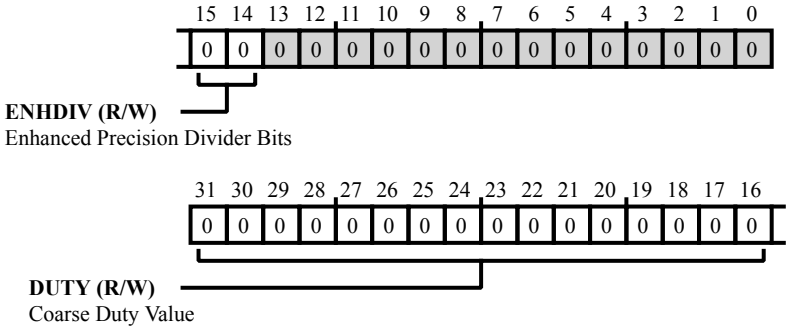


Figure 24-43: PWM_AL_DUTY1 Register Diagram

Table 24-19: PWM_AL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_AL_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_AL_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_AL_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B Control Register

The PWM_BCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

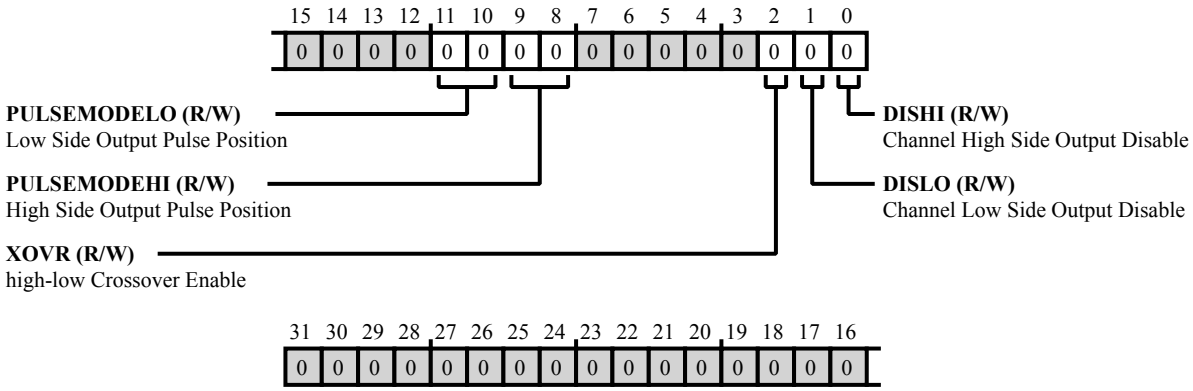


Figure 24-44: PWM_BCTL Register Diagram

Table 24-20: PWM_BCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_BCTL.PULSEMODELO bits select the pulse position for Channel B low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_BL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_BL0 and PWM_BL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_BL0 and PWM_BL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 24-20: PWM_BCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_BCTL.PULSEMODEHI bits select the pulse position for Channel B high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_BH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_BH0 and PWM_BH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_BH0 and PWM_BH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_BCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_BCTL.DISLO bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_BCTL.DISHI bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel B-High Duty-0 Register

The PWM_BH0 and PWM_BH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the channel B duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_BCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_BH0 register to determine the assertion and deassertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel B high pulse output for count less than PWM_BH0 and deasserts this output for count greater than PWM_BH1.

The value range for the PWM_BH0 and PWM_BH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHB_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_BH0 and PWM_BH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHB_DT (two's complement) and +PWM_TM0/2 + PWM_CHB_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHB_DT (two's complement) and -PWM_TM0/2 - PWM_CHB_DT.

Note that using values in the PWM_BH0 or PWM_BH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

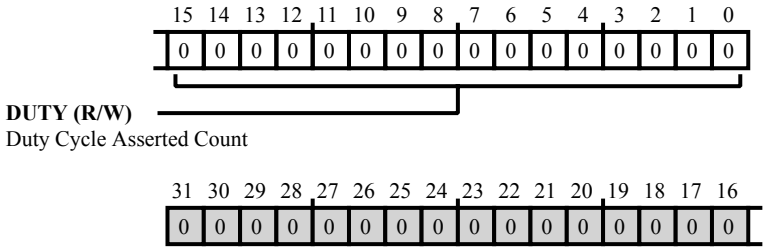


Figure 24-45: PWM_BH0 Register Diagram

Table 24-21: PWM_BH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count.

Channel B-High Heightened-Precision Duty-0 Register

The PWM_BH0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_BH0 register, allows programs to specify fractional duty cycles. The PWM_BH0_HP register and the PWM_BH0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_BH0_HP and the PWM_BH0 registers are also present in the single full duty register (if available).

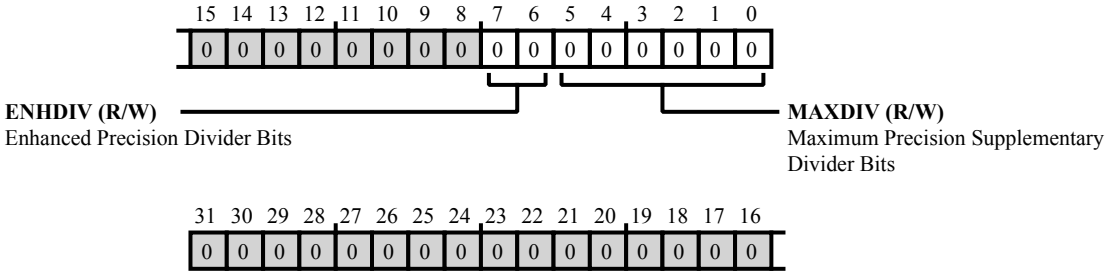


Figure 24-46: PWM_BH0_HP Register Diagram

Table 24-22: PWM_BH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BH0_HP.ENHDIV bits provide fractional duty cycles for Channel B high side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel B-High Duty-1 Register

The PWM_BH0 and PWM_BH1 registers determine the width for the high side output pulses. For more information, see the PWM_BH0 register description.

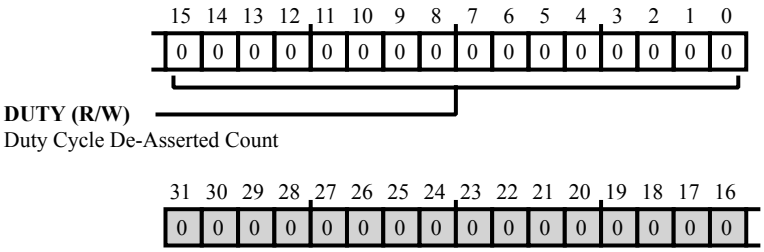


Figure 24-47: PWM_BH1 Register Diagram

Table 24-23: PWM_BH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count.

Channel B-High Heightened-Precision Duty-1 Register

The PWM_BH1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_BH1 register, allows programs to specify fractional duty cycles. The PWM_BH1_HP register and the PWM_BH1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_BH1_HP and the PWM_BH1 registers are also present in the single full duty register (if available).

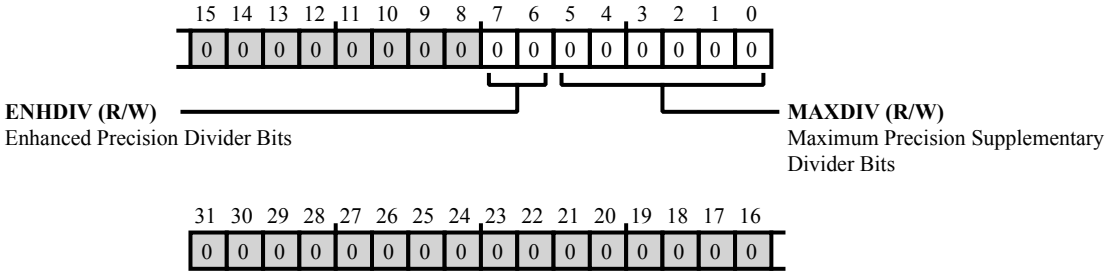


Figure 24-48: PWM_BH1_HP Register Diagram

Table 24-24: PWM_BH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BH1_HP.ENHDIV bits provide fractional duty cycles for Channel B high side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel B-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_BH_DUTY0 register contains the PWM_BH_DUTY0.DUTY bit field from the PWM_BH0 register and the PWM_BH_DUTY0.ENHDIV bit field from the PWM_BH0_HP register.

Note that the PWM_BH_DUTY0 register reads the PWM_BH0 and the PWM_BH0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

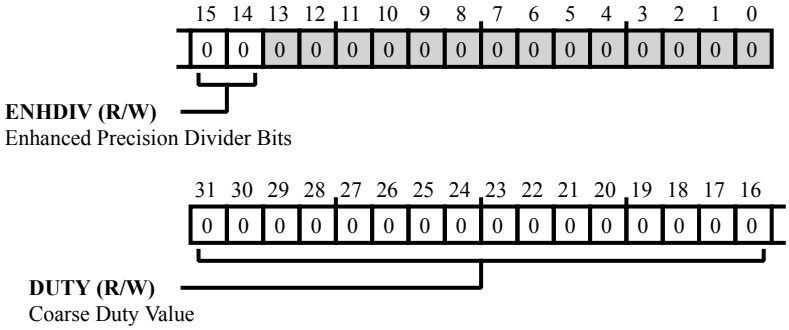


Figure 24-49: PWM_BH_DUTY0 Register Diagram

Table 24-25: PWM_BH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_BH_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_BH_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BH_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_BH_DUTY1 register contains the PWM_BH_DUTY1.DUTY bit field from the PWM_BH1 register and the PWM_BH_DUTY1.ENHDIV bit field from the PWM_BH1_HP register.

Note that the PWM_BH_DUTY1 register reads the PWM_BH1 and the PWM_BH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

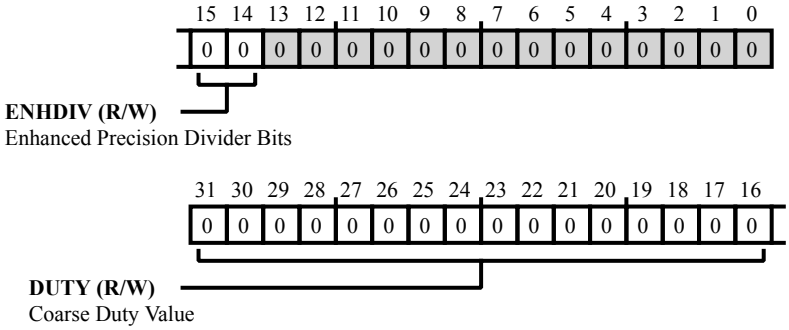


Figure 24-50: PWM_BH_DUTY1 Register Diagram

Table 24-26: PWM_BH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_BH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_BH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-Low Duty-0 Register

The PWM_BL0 and PWM_BL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the channel B duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_BCTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_BL0 register to determine the assertion and deassertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel B low pulse output for count less than PWM_BL0 and deasserts this output for count greater than PWM_BL1.

The value range for the PWM_BL0 and PWM_BL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHB_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_BL0 and PWM_BL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHB_DT (two's complement) and +PWM_TM0/2 + PWM_CHB_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHB_DT (two's complement) and -PWM_TM0/2 - PWM_CHB_DT.

Note that using values in the PWM_BL0 or PWM_BL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

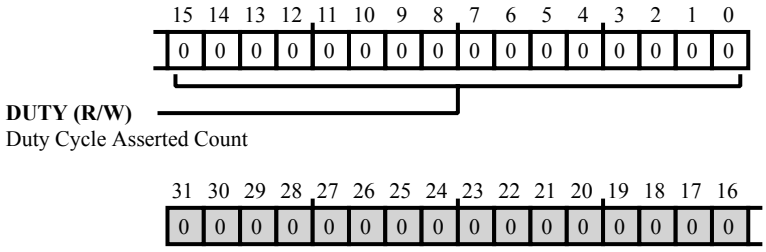


Figure 24-51: PWM_BL0 Register Diagram

Table 24-27: PWM_BL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_BL0.DUTY bits select the duty cycle asserted count for Channel B low side output.

Channel B-Low Heightened-Precision Duty-0 Register

The PWM_BL0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_BL0 register, allows programs to specify fractional duty cycles. The PWM_BL0_HP register and the PWM_BL0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_BL0_HP and the PWM_BL0 registers are also present in the single full duty register (PWM_BL_DUTY0).

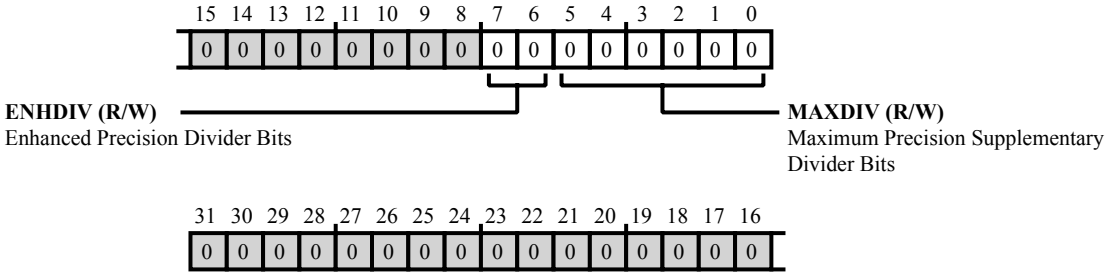


Figure 24-52: PWM_BL0_HP Register Diagram

Table 24-28: PWM_BL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BL0_HP.ENHDIV bits provide fractional duty cycles for Channel B low side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel B-Low Duty-1 Register

The PWM_BL0 and PWM_BL1 registers determine the width for the low side output pulses. For more information, see the PWM_BL0 register description.

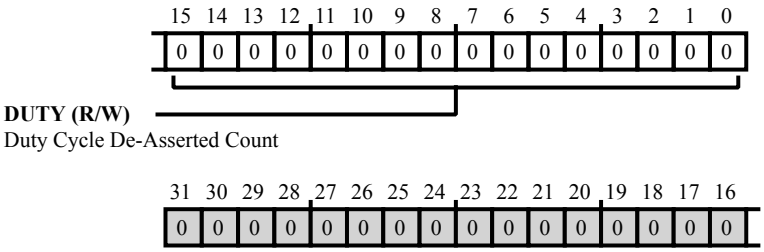


Figure 24-53: PWM_BL1 Register Diagram

Table 24-29: PWM_BL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_BL1.DUTY bits select the duty cycle de-asserted count for Channel B low side output.

Channel B-Low Heightened-Precision Duty-1 Register

The PWM_BL1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_BL1 register, allows programs to specify fractional duty cycles. The PWM_BL1_HP register and the PWM_BL1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_BL1_HP and the PWM_BL1 registers are also present in the single full duty register (if available).

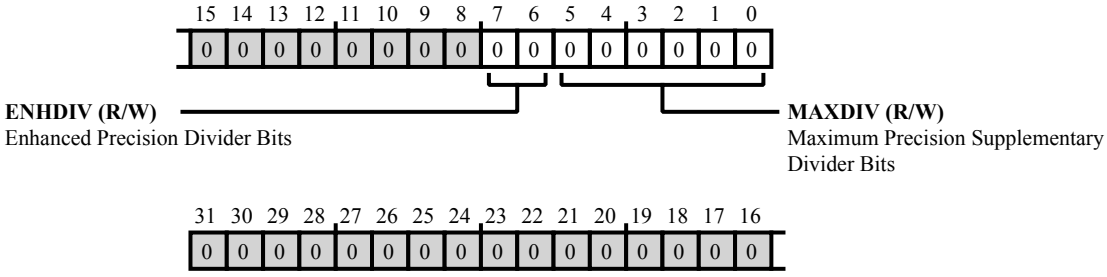


Figure 24-54: PWM_BL1_HP Register Diagram

Table 24-30: PWM_BL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BL1_HP.ENHDIV bits provide fractional duty cycles for Channel B low side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel B-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_BL_DUTY0 register contains the PWM_BL_DUTY0.DUTY bit field from the PWM_BL0 register and the PWM_BL_DUTY0.ENHDIV bit field from the PWM_BL0_HP register.

Note that the PWM_BL_DUTY0 register reads the PWM_BL0 and the PWM_BL0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

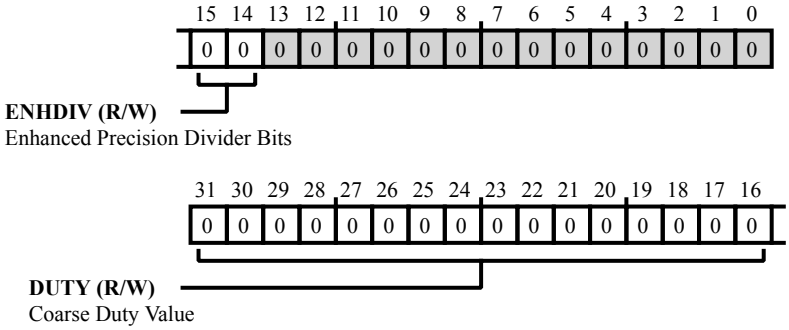


Figure 24-55: PWM_BL_DUTY0 Register Diagram

Table 24-31: PWM_BL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_BL_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_BL_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BL_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel B-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_BL_DUTY1 register contains the PWM_BL_DUTY1.DUTY bit field from the PWM_BL1 register and the PWM_BL_DUTY1.ENHDIV bit field from the PWM_BL1_HP register.

Note that the PWM_BL_DUTY1 register reads the PWM_BL1 and the PWM_BL1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

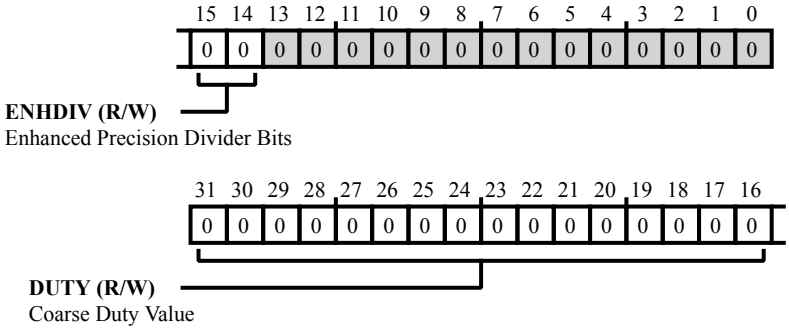


Figure 24-56: PWM_BL_DUTY1 Register Diagram

Table 24-32: PWM_BL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_BL_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_BL_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_BL_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C Control Register

The PWM_CCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

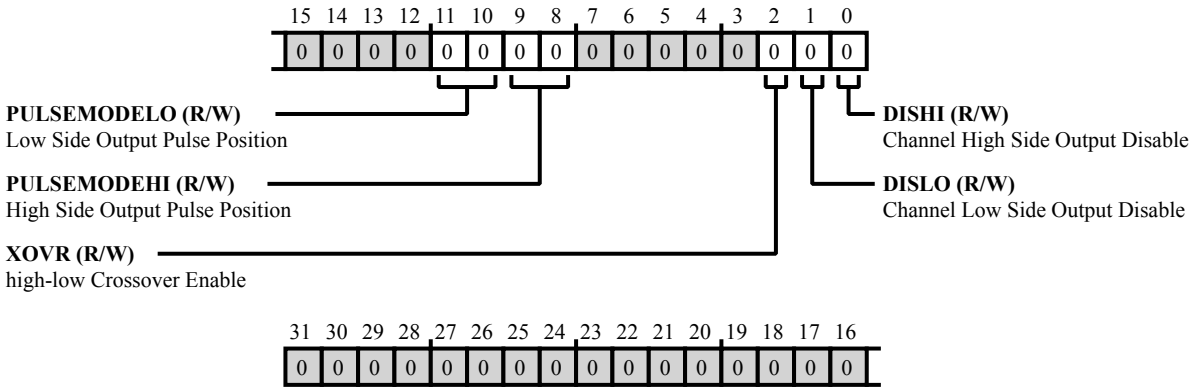


Figure 24-57: PWM_CCTL Register Diagram

Table 24-33: PWM_CCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_CCTL.PULSEMODELO bits select the pulse position for Channel C low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_CL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_CL0 and PWM_CL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_CL0 and PWM_CL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 24-33: PWM_CCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_CCTL.PULSEMODEHI bits select the pulse position for Channel C high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_CH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_CH0 and PWM_CH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_CH0 and PWM_CH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_CCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_CCTL.DISLO bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_CCTL.DISHI bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel C-High Pulse Duty Register 0

The PWM_CH0 and PWM_CH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the channel C duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_CCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_CH0 register to determine the assertion and deassertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel C high pulse output for count less than PWM_CH0 and deasserts this output for count greater than PWM_CH1.

The value range for the PWM_CH0 and PWM_CH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHC_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_CH0 and PWM_CH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHC_DT (two's complement) and +PWM_TM0/2 + PWM_CHC_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHC_DT (two's complement) and -PWM_TM0/2 - PWM_CHC_DT.

Note that using values in the PWM_CH0 or PWM_CH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

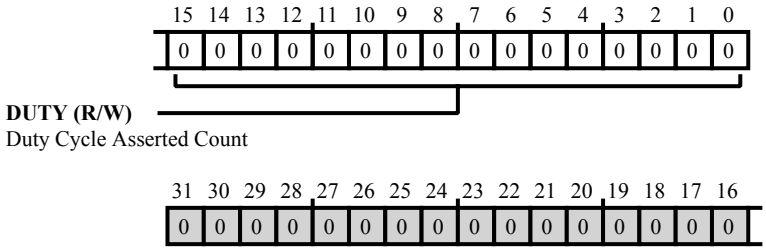


Figure 24-58: PWM_CH0 Register Diagram

Table 24-34: PWM_CH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_CH0.DUTY bits select the duty cycle asserted count for Channel C high side output.

Channel C-High Pulse Heightened-Precision Duty Register 0

The PWM_CH0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_CH0 register, allows programs to specify fractional duty cycles. The PWM_CH0_HP register and the PWM_CH0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_CH0_HP and the PWM_CH0 registers are also present in the single full duty register (if available).

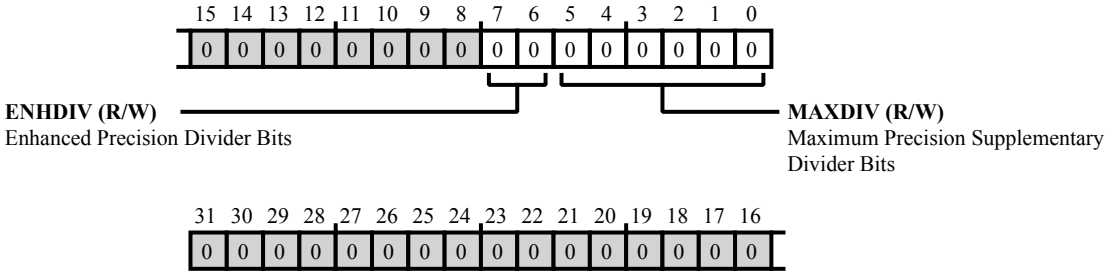


Figure 24-59: PWM_CH0_HP Register Diagram

Table 24-35: PWM_CH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH0_HP.ENHDIV bits provide fractional duty cycles for Channel C high side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel C-High Pulse Duty Register 1

The PWM_CH0 and PWM_CH1 registers determine the width for the high side output pulses. For more information, see the PWM_CH0 register description.

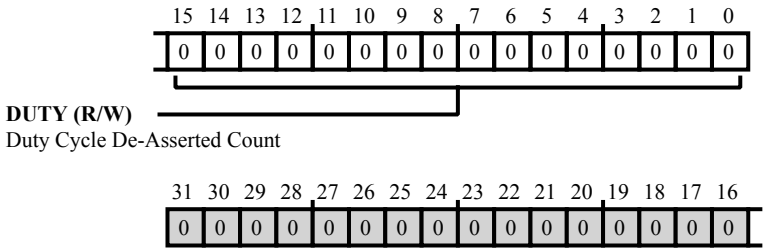


Figure 24-60: PWM_CH1 Register Diagram

Table 24-36: PWM_CH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_CH1.DUTY bits select the duty cycle de-asserted count for Channel C high side output.

Channel C-High Pulse Heightened-Precision Duty Register 1

The PWM_CH1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_CH1 register, allows programs to specify fractional duty cycles. The PWM_CH1_HP register and the PWM_CH1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_CH1_HP and the PWM_CH1 registers are also present in the single full duty register (if available).

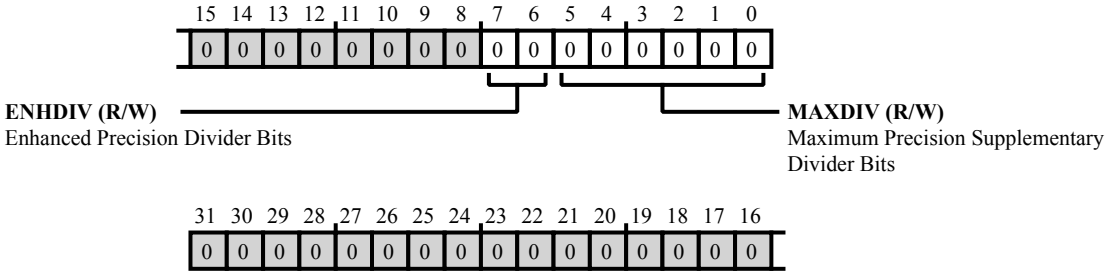


Figure 24-61: PWM_CH1_HP Register Diagram

Table 24-37: PWM_CH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH1_HP.ENHDIV bits provide fractional duty cycles for Channel C high side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel Configuration Register

The PWM_CHANCFG register configures Channel A, B, C, and D reference timer selection, high and low side output features, and enables high frequency chopping operation. Do not change the value of any bits in the PWM register while the PWM is enabled (PWM_CTL.GLOBEN =1).

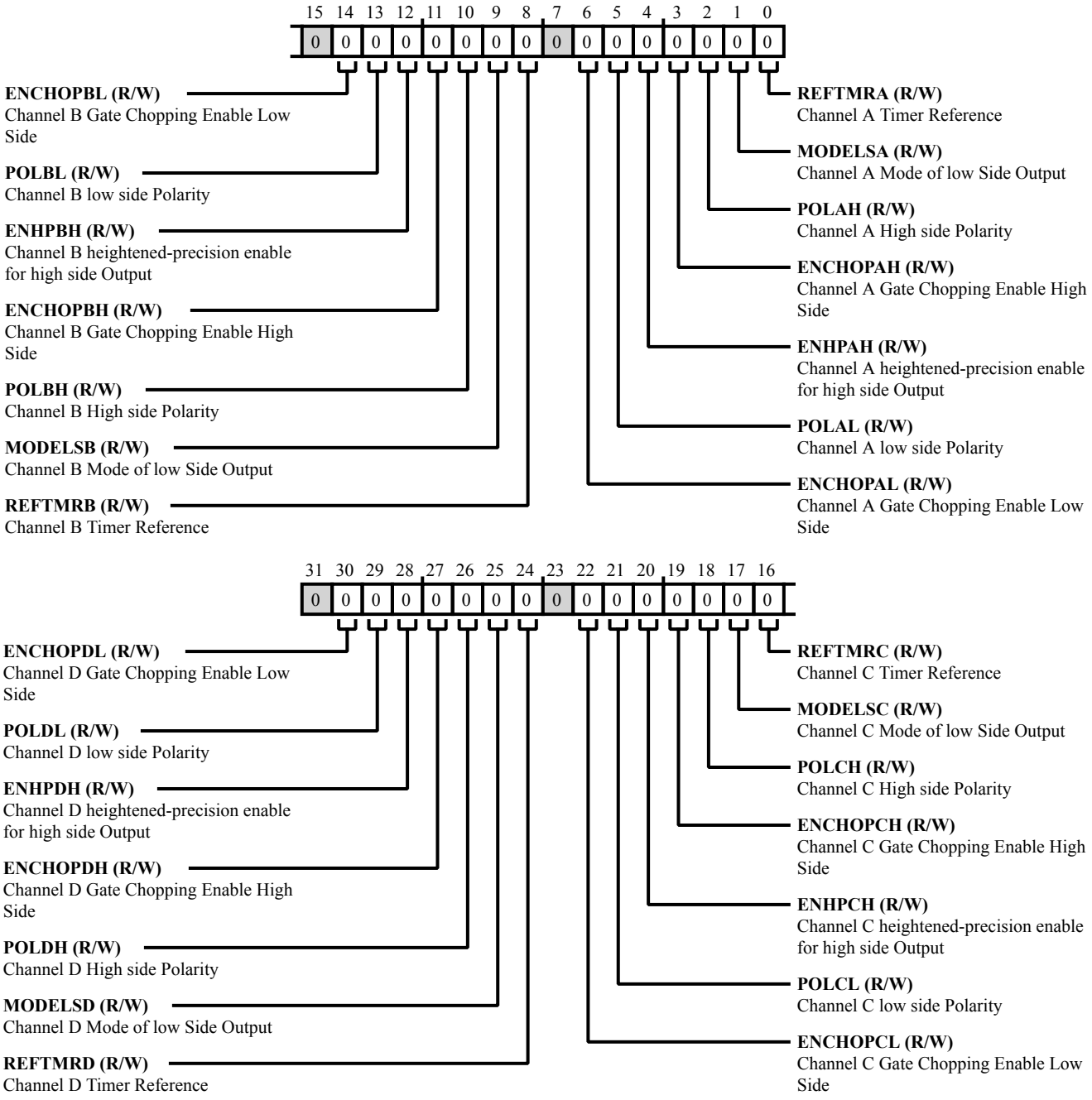


Figure 24-62: PWM_CHANCFG Register Diagram

Table 24-38: PWM_CHANCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	ENCHOPDL	Channel D Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPDL bit enables mixing of the Channel D low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel D Low Side
		1	Enable Chopping Channel D Low Side
29 (R/W)	POLDL	Channel D low side Polarity. The PWM_CHANCFG.POLDL bit selects the Channel D low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
28 (R/W)	ENHPDH	Channel D heightened-precision enable for high side Output. The PWM_CHANCFG.ENHPDH bit enables heightened-precision Channel D high side output.	
		0	Disable HP Output Channel D High
		1	Enable HP Output Channel D High
27 (R/W)	ENCHOPDH	Channel D Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPDH bit enables mixing of the Channel D high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel D High Side
		1	Enable Chopping Channel D High Side
26 (R/W)	POLDH	Channel D High side Polarity. The PWM_CHANCFG.POLDH bit selects the Channel D high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High

Table 24-38: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	MODELSD	Channel D Mode of low Side Output. The PWM_CHANCFG.MODELSD bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSD =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control
24 (R/W)	REFTMRD	Channel D Timer Reference. The PWM_CHANCFG.REFTMRD bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel D operation.	
		0	PWMTMR0 is Channel D reference
		1	PWMTMR4 is Channel D reference
22 (R/W)	ENCHOPCL	Channel C Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPCL bit enables mixing of the Channel C low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel C Low Side
		1	Enable Chopping Channel C Low Side
21 (R/W)	POLCL	Channel C low side Polarity. The PWM_CHANCFG.POLCL bit selects the Channel C low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
20 (R/W)	ENHPCH	Channel C heightened-precision enable for high side Output. The PWM_CHANCFG.ENHPCH bit enables heightened-precision Channel C high side output.	
		0	Disable HP Output Channel C High
		1	Enable HP Output Channel C High

Table 24-38: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	ENCHOPCH	Channel C Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPCH bit enables mixing of the Channel C high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel C High Side
		1	Enable Chopping Channel C High Side
18 (R/W)	POLCH	Channel C High side Polarity. The PWM_CHANCFG.POLCH bit selects the Channel C high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
17 (R/W)	MODELSC	Channel C Mode of low Side Output. The PWM_CHANCFG.MODELSC bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSC =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control
16 (R/W)	REFTMRC	Channel C Timer Reference. The PWM_CHANCFG.REFTMRC bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel C operation.	
		0	PWMTMR0 is Channel C reference
		1	PWMTMR3 is Channel C reference
14 (R/W)	ENCHOPBL	Channel B Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPBL bit enables mixing of the Channel B low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel B Low Side
		1	Enable Chopping Channel B Low Side

Table 24-38: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	POLBL	Channel B low side Polarity. The PWM_CHANCFG.POLBL bit selects the Channel B low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
12 (R/W)	ENHPBH	Channel B heightened-precision enable for high side Output. The PWM_CHANCFG.ENHPBH bit enables heightened-precision Channel B high side output.	
		0	Disable HP Output Channel B High
		1	Enable HP Output Channel B High
11 (R/W)	ENCHOPBH	Channel B Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPBH bit enables mixing of the Channel B high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel B High Side
		1	Enable Chopping Channel B High Side
10 (R/W)	POLBH	Channel B High side Polarity. The PWM_CHANCFG.POLBH bit selects the Channel B high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
9 (R/W)	MODELSB	Channel B Mode of low Side Output. The PWM_CHANCFG.MODELSB bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSB =0, the low side output is an inverted form of the high side output, which is generated using the PWM_BH0 and PWM_BH1 registers for pulse width, using the PWM_BCTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLBH bits for polarity.	
		0	Invert of high output
		1	Independent control
8 (R/W)	REFTMRB	Channel B Timer Reference. The PWM_CHANCFG.REFTMRB bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel B operation.	
		0	PWMTMR0 is Channel B reference
		1	PWMTMR2 is Channel B reference

Table 24-38: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	ENCHOPAL	Channel A Gate Chopping Enable Low Side. The PWM_CHANCFG.ENCHOPAL bit enables mixing of the Channel A low side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel A Low Side
		1	Enable Chopping Channel A Low Side
5 (R/W)	POLAL	Channel A low side Polarity. The PWM_CHANCFG.POLAL bit selects the Channel A low side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High
4 (R/W)	ENHPAH	Channel A heightened-precision enable for high side Output. The PWM_CHANCFG.ENHPAH bit enables heightened-precision Channel A high side output.	
		0	Disable HP Output Channel A High
		1	Enable HP Output Channel A High
3 (R/W)	ENCHOPAH	Channel A Gate Chopping Enable High Side. The PWM_CHANCFG.ENCHOPAH bit enables mixing of the Channel A high side output signals with a high-frequency chopping signal, which is configured with the PWM_CHOPCFG register.	
		0	Disable Chopping Channel A High Side
		1	Enable Chopping Channel A High Side
2 (R/W)	POLAH	Channel A High side Polarity. The PWM_CHANCFG.POLAH bit selects the Channel A high side output polarity (active-high or active-low).	
		0	Active Low
		1	Active High

Table 24-38: PWM_CHANCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MODELSA	Channel A Mode of low Side Output. The PWM_CHANCFG.MODELSA bit selects whether the low side output waveform is based on independent controls or whether the low side output depends on the high side output controls. When PWM_CHANCFG.MODELSA =0, the low side output is an inverted form of the high side output, which is generated using the PWM_AH0 and PWM_AH1 registers for pulse width, using the PWM_ACTL.PULSEMODEHI bits for pulse positioning, and PWM_CHANCFG.POLAH bits for polarity.	
		0	Invert of high output
		1	Independent control
0 (R/W)	REFTMRA	Channel A Timer Reference. The PWM_CHANCFG.REFTMRA bit selects whether the PWM uses PWMTMR1 or PWMTMR0 as the reference timer for Channel A operation.	
		0	PWMTMR0 is Channel A reference
		1	PWMTMR1 is Channel A reference

Channel A Dead-time Register

The PWM_CHA_DT register controls the value of dead-time for channel A independently.

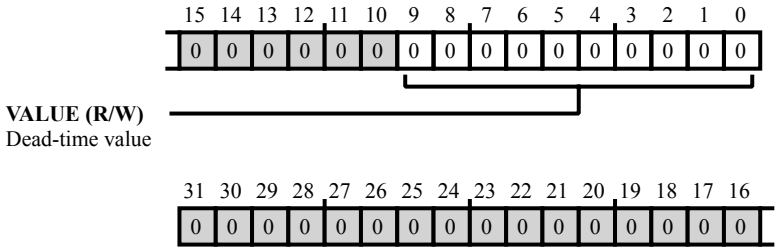


Figure 24-63: PWM_CHA_DT Register Diagram

Table 24-39: PWM_CHA_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead-time value.

Channel B Dead-time Register

The PWM_CHB_DT register controls the value of dead-time for channel B independently.

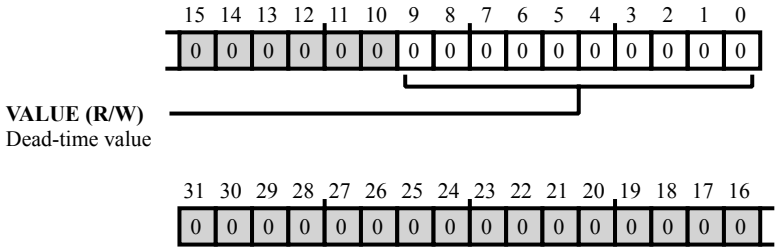


Figure 24-64: PWM_CHB_DT Register Diagram

Table 24-40: PWM_CHB_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead-time value. The PWM_CHB_DT.VALUE bit field contains the dead-time value.

Channel C Dead-time Register

The PWM_CHC_DT register controls the value of dead-time for channel C independently.

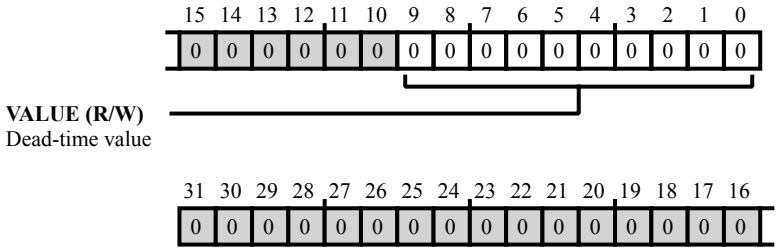


Figure 24-65: PWM_CHC_DT Register Diagram

Table 24-41: PWM_CHC_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead-time value. The PWM_CHC_DT.VALUE bit field contains the dead-time value.

Channel D Dead-time Register

The PWM_CHD_DT register controls the value of dead-time for channel D independently.

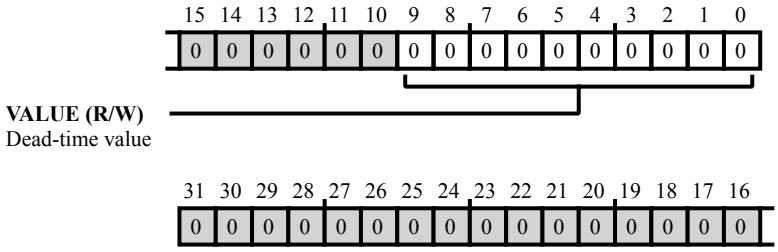


Figure 24-66: PWM_CHD_DT Register Diagram

Table 24-42: PWM_CHD_DT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Dead-time value. The PWM_CHD_DT.VALUE bit field contains the dead-time value.

Chop Configuration Register

The PWM_CHOPCFG register holds a divisor value that controls the chopping frequency. The PWM permits a mixing of the output signals with a high-frequency chopping signal to aid with interfacing to pulse transformers. Also note that high-frequency chopping may be independently enabled for each channel's high-side and the low-side outputs using channel control bits. (For example, control chopping for Channel A with the PWM_CHANCFG.ENCHOPAH and PWM_CHANCFG.ENCHOPAH bits.)

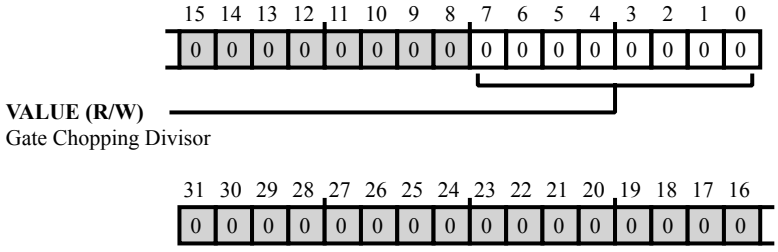


Figure 24-67: PWM_CHOPCFG Register Diagram

Table 24-43: PWM_CHOPCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Gate Chopping Divisor. The PWM_CHOPCFG.VALUE bits provide the high frequency chopping divisor. When the divisor value is changed, the new period takes effect from the next edge of the chopping signal. The PWM_CHOPCFG.VALUE value may be calculated using either of the following formulas: CHOPDIV = $[(T_{CHOP}/T_{CK}) / 4] - 1$ CHOPDIV = $[(f_{CK} / f_{CHOP}) / 4] - 1$

Channel C-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CH_DUTY0 register contains the PWM_CH_DUTY0.DUTY bit field from the PWM_CH0 register and the PWM_CH_DUTY0.ENHDIV bit field from the PWM_CH0_HP register.

Note that the PWM_CH_DUTY0 register reads the PWM_CH0 and the PWM_CH0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

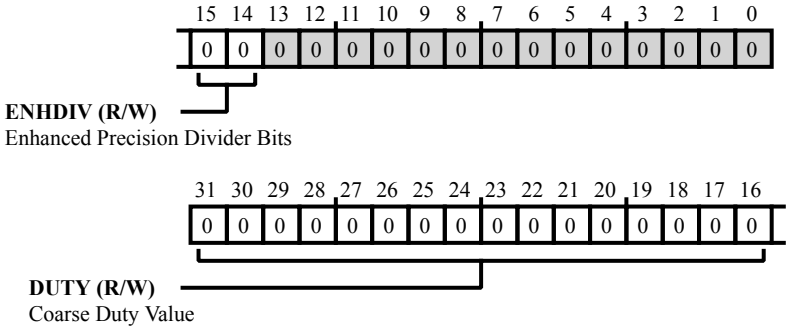


Figure 24-68: PWM_CH_DUTY0 Register Diagram

Table 24-44: PWM_CH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CH_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CH_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CH_DUTY1 register contains the PWM_CH_DUTY1.DUTY bit field from the PWM_CH1 register and the PWM_CH_DUTY1.ENHDIV bit field from the PWM_CH1_HP register.

Note that the PWM_CH_DUTY1 register reads the PWM_CH1 and the PWM_CH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

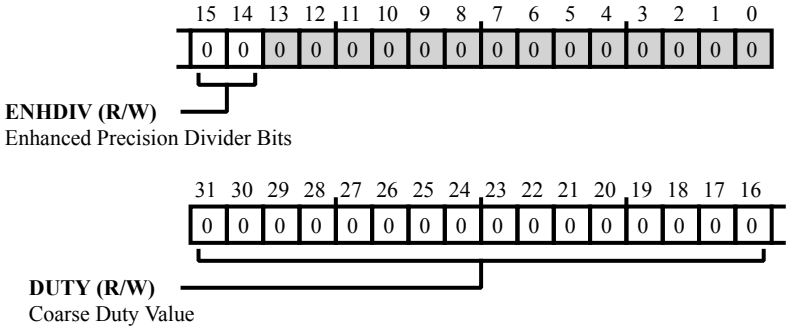


Figure 24-69: PWM_CH_DUTY1 Register Diagram

Table 24-45: PWM_CH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-Low Pulse Duty Register 0

The PWM_CL0 and PWM_CL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the channel C duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_CCTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_CL0 register to determine the assertion and deassertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel C low pulse output for count less than PWM_CL0 and deasserts this output for count greater than PWM_CL1.

The value range for the PWM_CL0 and PWM_CL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHC_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_CL0 and PWM_CL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHC_DT (two's complement) and +PWM_TM0/2 + PWM_CHC_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHC_DT (two's complement) to -PWM_TM0/2 - PWM_CHC_DT.

Note that using values in the PWM_CL0 or PWM_CL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

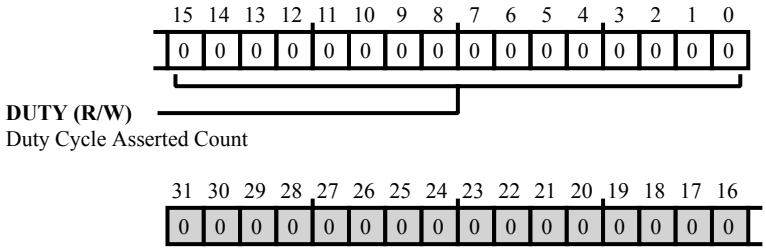


Figure 24-70: PWM_CL0 Register Diagram

Table 24-46: PWM_CL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_CL0.DUTY bits select the duty cycle asserted count for Channel C low side output.

Channel C-Low Pulse Duty Register 1

The PWM_CL0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_CL0 register, allows programs to specify fractional duty cycles. The PWM_CL0_HP register and the PWM_CL0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_CL0_HP and the PWM_CL0 registers are also present in the single full duty register (if available).

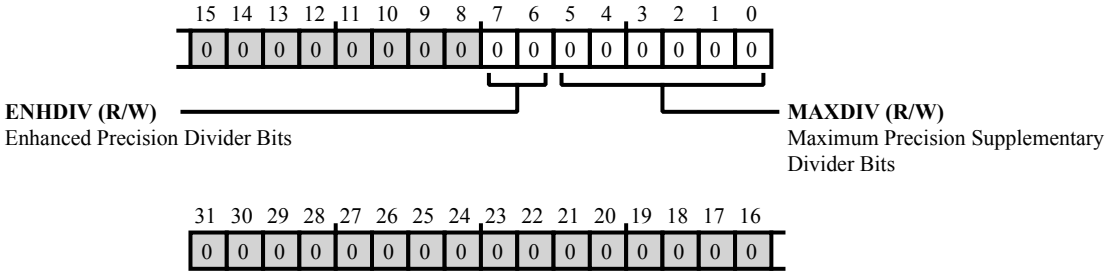


Figure 24-71: PWM_CL0_HP Register Diagram

Table 24-47: PWM_CL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CL0_HP.ENHDIV bits provide fractional duty cycles for Channel C low side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel C-Low Duty-1 Register

The PWM_CL0 and PWM_CL1 registers determine the width for the low side output pulses. For more information, see the PWM_CL0 register description.

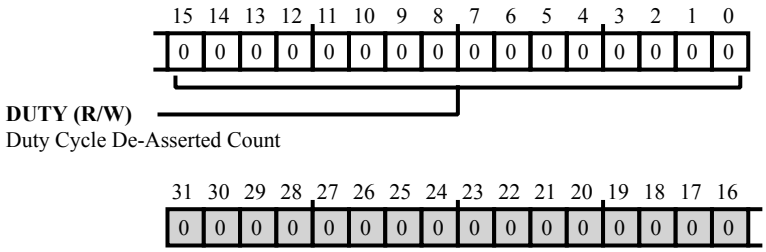


Figure 24-72: PWM_CL1 Register Diagram

Table 24-48: PWM_CL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count.

Channel C-Low Heightened-Precision Duty-1 Register

The PWM_CL1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_CL1 register, allows programs to specify fractional duty cycles. The PWM_CL1_HP register and the PWM_CL1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_CL1_HP and the PWM_CL1 registers are also present in the single full duty register (if available).

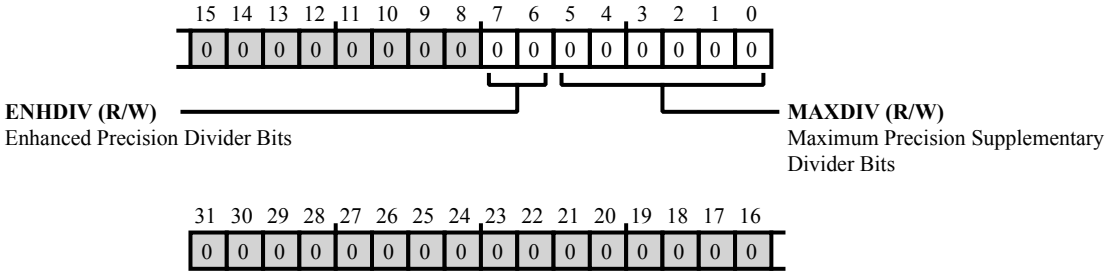


Figure 24-73: PWM_CL1_HP Register Diagram

Table 24-49: PWM_CL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CL1_HP.ENHDIV bits provide fractional duty cycles for Channel C low side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel C-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CL_DUTY0 register contains the PWM_CL_DUTY0.DUTY bit field from the PWM_CL0 register and the PWM_CL_DUTY0.ENHDIV bit field from the PWM_CL0_HP register.

Note that the PWM_CL_DUTY0 register reads the PWM_CL0 and the PWM_CL0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

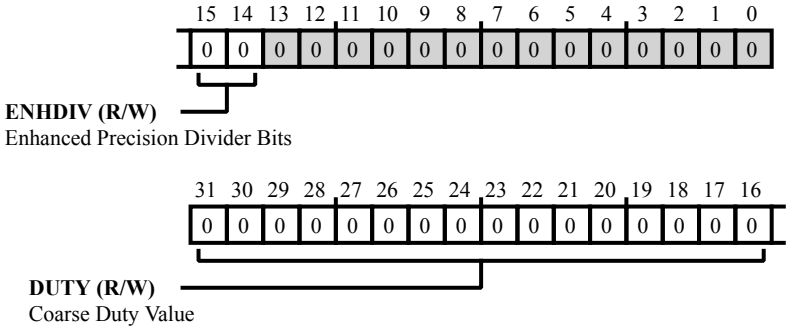


Figure 24-74: PWM_CL_DUTY0 Register Diagram

Table 24-50: PWM_CL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CL_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CL_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CL_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel C-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_CL_DUTY1 register contains the PWM_CL_DUTY1.DUTY bit field from the PWM_CL1 register and the PWM_CL_DUTY1.ENHDIV bit field from the PWM_CL1_HP register.

Note that the PWM_CL_DUTY1 register reads the PWM_CL1 and the PWM_CL1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

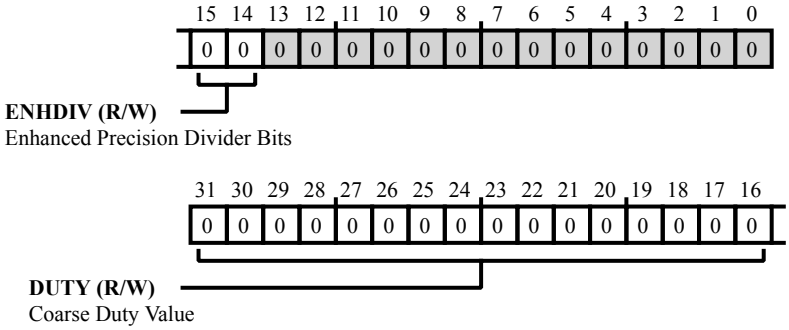


Figure 24-75: PWM_CL_DUTY1 Register Diagram

Table 24-51: PWM_CL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_CL_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_CL_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_CL_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Control Register

The PWM_CTL register enables the PWM, enables delay counters for the channels, and configures sync features. This register also provides support for tripping a PWM fault condition through software.

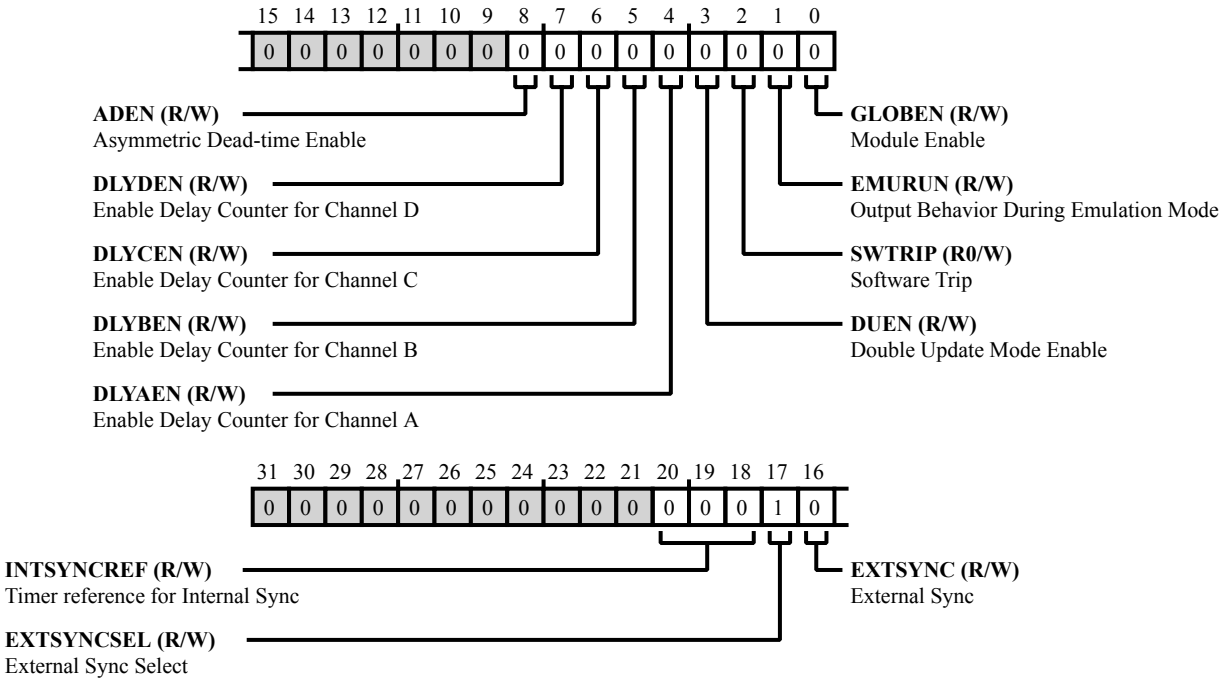


Figure 24-76: PWM_CTL Register Diagram

Table 24-52: PWM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20:18 (R/W)	INTSYNCREF	Timer reference for Internal Sync. The PWM_CTL.INTSYNCREF bits select the timer reference for the internal sync. Note that all other combinations reserved.	
		0	PWMTMR0 provides sync reference
		1	PWMTMR1 provides sync reference
		2	PWMTMR2 provides sync reference
		3	PWMTMR3 provides sync reference
		4	PWMTMR4 provides sync reference

Table 24-52: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	EXTSYNCSEL	External Sync Select. The PWM_CTL.EXTSYNCSEL bit selects whether the external sync signal is synchronous or asynchronous. Note that latency in PWM sync response differs between asynchronous and synchronous external sync modes. For more information, see the PWM functional description.	
		0	Asynchronous External Sync
		1	Synchronous External Sync
16 (R/W)	EXTSYNC	External Sync. The PWM_CTL.EXTSYNC bit selects whether the PWM uses an external or internal sync signal for the main timer (PWMTMR0). Do not change the value of the PWM_CTL.EXTSYNC bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Internal sync used
		1	External sync used
8 (R/W)	ADEN	Asymmetric Dead-time Enable. When symmetric dead-time is enabled (PWM_CTL.ADEN = 0), in the dependent mode, both the high-side and low-side outputs are reduced by DT cycles on both the assertion and de-assertion edges. When symmetric dead-time is disabled (PWM_CTL.ADEN = 1), the falling edges of both high and low-side outputs occur at the programmed duty value, but their rise-times are delayed by 2 x DT.	
		0	Dead-time is symmetric
		1	Dead-time is asymmetric
7 (R/W)	DLYDEN	Enable Delay Counter for Channel D. The PWM_CTL.DLYDEN bit enables the Channel D delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYDEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
6 (R/W)	DLYCEN	Enable Delay Counter for Channel C. The PWM_CTL.DLYCEN bit enables the Channel C delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYCEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable

Table 24-52: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	DLYBEN	Enable Delay Counter for Channel B. The PWM_CTL.DLYBEN bit enables the Channel B delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYBEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
4 (R/W)	DLYAEN	Enable Delay Counter for Channel A. The PWM_CTL.DLYAEN bit enables the Channel A delay counter, supporting phase-offset control. Do not change the value of the PWM_CTL.DLYAEN bit while the PWM is enabled (PWM_CTL.GLOBEN =1).	
		0	Disable
		1	Enable
3 (R/W)	DUEN	Double Update Mode Enable. In Single Update Mode, double buffering of all registers happens at the period boundary of the PWM timer. In Double Update Mode, double buffering of all registers (except delay counter registers) happens at the middle of the period as well as the beginning of the period.	
		0	Single Update Mode
		1	Double Update Mode
2 (R0/W)	SWTRIP	Software Trip. The PWM_CTL.SWTRIP bit permits tripping a fault condition through software, shutting down PWM output. This bit always read as 0. If the PWM_CTL.SWTRIP bit and PWM_CTL.GLOBEN bit are set in the same write, the write does not trip the fault condition.	
		0	Don't force fault
		1	Force a Fault Trip Condition
1 (R/W)	EMURUN	Output Behavior During Emulation Mode. The PWM_CTL.EMURUN bit selects PWM output behavior during emulation mode.	
		0	Disable Outputs
		1	Enable Outputs

Table 24-52: PWM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	GLOBEN	Module Enable. The PWM_CTL.GLOBEN bit enables the PWM, enabling all timers and outputs. While this bit is enabled, processor code should not change the value of the PWM_CTL.DLYAEN bit, PWM_CTL.DLYBEN bit, PWM_CTL.DLYCEN bit, PWM_CTL.DLYDEN bit, PWM_CTL.EXTSYNCSEL bit, or any bits in the PWM_CHANCFG register. Note that there is a latency between PWM disable and the cessation of output waveforms. There is also a latency between PWM enable and start of output waveforms. For the latency description, see the PWM functional description.	
		0	Disable
		1	Enable

Channel D Control Register

The PWM_DCTL register selects the low and high side output pulse mode, enables low and high side output, and enables low/high side output crossover.

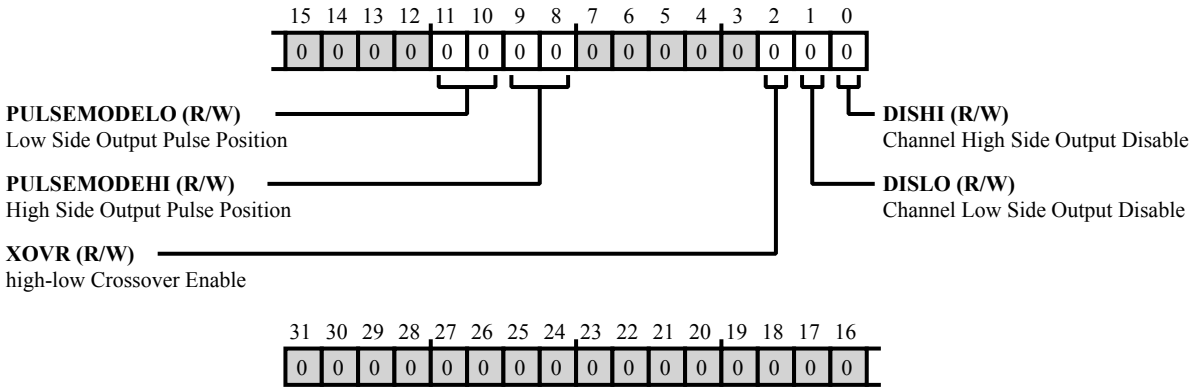


Figure 24-77: PWM_DCTL Register Diagram

Table 24-53: PWM_DCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:10 (R/W)	PULSEMODELO	Low Side Output Pulse Position. The PWM_DCTL.PULSEMODELO bits select the pulse position for Channel D low side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_DL0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_DL0 and PWM_DL1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_DL0 and PWM_DL1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half

Table 24-53: PWM_DCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	PULSEMODEHI	High Side Output Pulse Position. The PWM_DCTL.PULSEMODEHI bits select the pulse position for Channel D high side output. In symmetrical mode, the channel forms a symmetrical pulse waveform around the center of the PWM period. Only one of the duty cycle registers is used for an output in symmetrical mode. Note that in this mode, the values in the PWM_DH0 register is scaled, such that a value of 0 produces 50% duty. In asymmetrical mode, the channel forms an asymmetrical pulse waveform around the center of the PWM period. This mode uses both the duty cycle registers (PWM_DH0 and PWM_DH1). In left half or right half mode, the channel forms the pulse waveforms on either the first half (left) or the second half (right) of the PWM period. This mode uses both the duty cycle registers (PWM_DH0 and PWM_DH1).	
		0	Symmetrical
		1	Asymmetrical
		2	Left Half
		3	Right Half
2 (R/W)	XOVR	high-low Crossover Enable. The PWM_DCTL.XOVR bit enables crossover between the channels high and low side outputs. When enabled, this bit directs the PWM to send the low-side output through the high-side output pin and the high-side output through the low side output pin.	
		0	Disable Crossover
		1	Enable Crossover
1 (R/W)	DISLO	Channel Low Side Output Disable. The PWM_DCTL.DISLO bit enables the channels low side output.	
		0	Enable Low Side Output
		1	Disable Low Side Output
0 (R/W)	DISHI	Channel High Side Output Disable. The PWM_DCTL.DISHI bit enables the channels high side output.	
		0	Enable High Side Output
		1	Disable High Side Output

Channel D-High Duty-0 Register

The PWM_DH0 and PWM_DH1 registers determine the width for the high side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the high side output pulses for the channel D duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_DCTL.PULSEMODEHI bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_DH0 register to determine the assertion and deassertion count for the high side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel D high pulse output for count less than PWM_DH0 and deasserts this output for count greater than PWM_DH1.

The value range for the PWM_DH0 and PWM_DH1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHD_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_DH0 and PWM_DH1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHD_DT (two's complement) to +PWM_TM0/2 + PWM_CHD_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHD_DT (two's complement) to -PWM_TM0/2 - PWM_CHD_DT.

Note that using values in the PWM_DH0 or PWM_DH1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

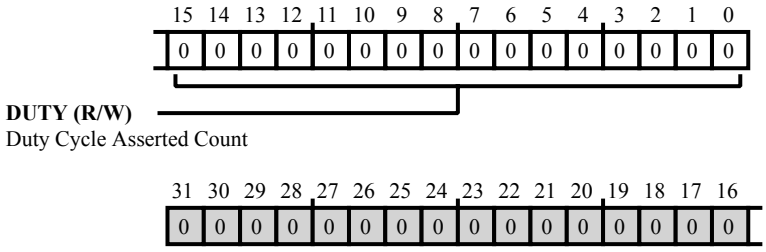


Figure 24-78: PWM_DH0 Register Diagram

Table 24-54: PWM_DH0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_DH0.DUTY bits select the duty cycle asserted count for Channel D high side output.

Channel D-High Pulse Heightened-Precision Duty Register 0

The PWM_DH0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_DH0 register, allows programs to specify fractional duty cycles. The PWM_DH0_HP register and the PWM_DH0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_DH0_HP and the PWM_DH0 registers are also present in the single full duty register (if available).

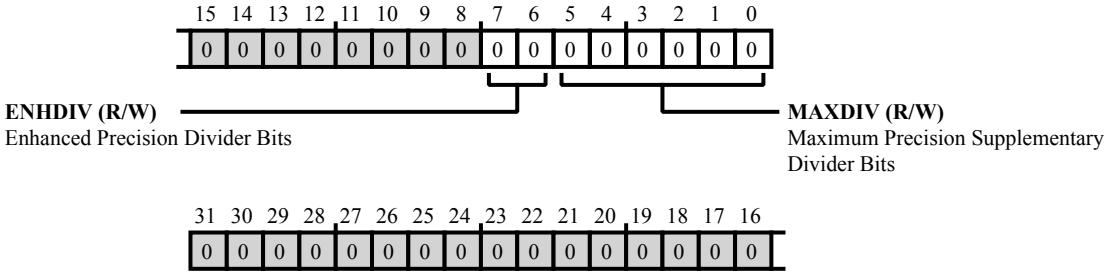


Figure 24-79: PWM_DH0_HP Register Diagram

Table 24-55: PWM_DH0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DH0_HP.ENHDIV bits provide fractional duty cycles for Channel D high side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel D-High Pulse Duty Register 1

The PWM_DH0 and PWM_DH1 registers determine the width for the high side output pulses. For more information, see the PWM_DH0 register description.

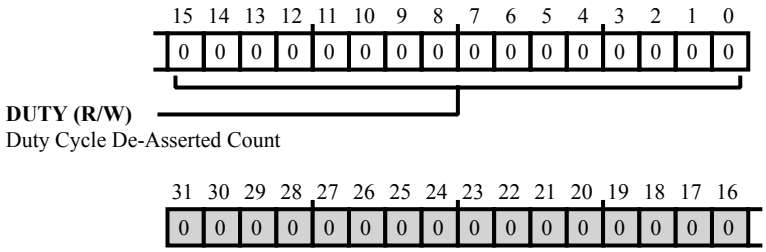


Figure 24-80: PWM_DH1 Register Diagram

Table 24-56: PWM_DH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_DH1.DUTY bits select the duty cycle de-asserted count for Channel D high side output.

Channel D High Pulse Heightened-Precision Duty Register 1

The PWM_DH1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_DH1 register, allows programs to specify fractional duty cycles. The PWM_DH1_HP register and the PWM_DH1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_DH1_HP and the PWM_DH1 registers are also present in the single full duty register (if available).

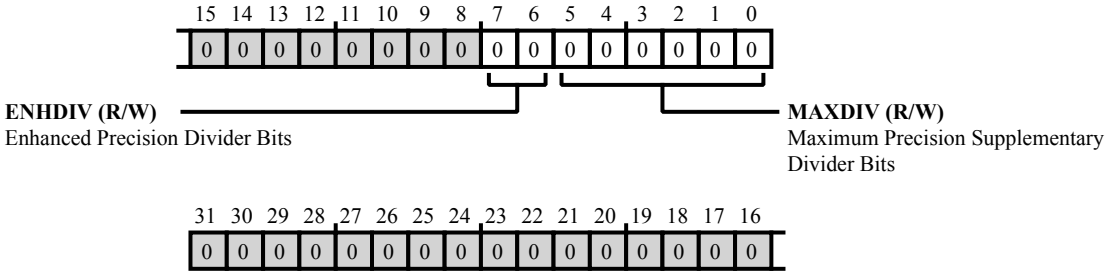


Figure 24-81: PWM_DH1_HP Register Diagram

Table 24-57: PWM_DH1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DH1_HP.ENHDIV bits provide fractional duty cycles for Channel D high side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel D-High Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_DH_DUTY0 register contains the PWM_DH_DUTY0.DUTY bit field from the PWM_DH0 register and the PWM_DH_DUTY0.ENHDIV bit field from the PWM_DH0_HP register.

Note that the PWM_DH_DUTY0 register reads the PWM_DH0 and the PWM_DH0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

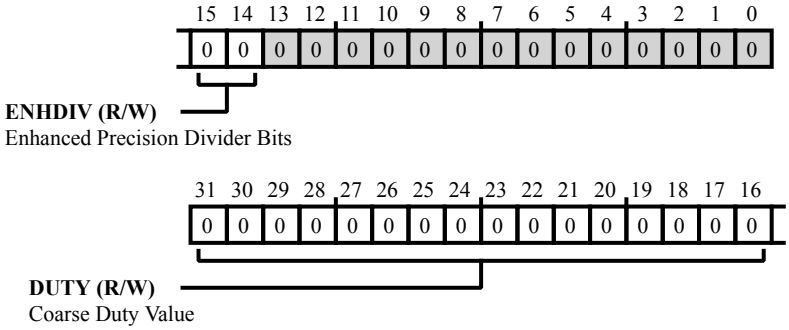


Figure 24-82: PWM_DH_DUTY0 Register Diagram

Table 24-58: PWM_DH_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_DH_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_DH_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DH_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-High Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_DH_DUTY1 register contains the PWM_DH_DUTY1.DUTY bit field from the PWM_DH1 register and the PWM_DH_DUTY1.ENHDIV bit field from the PWM_DH1_HP register.

Note that the PWM_DH_DUTY1 register reads the PWM_DH1 and the PWM_DH1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

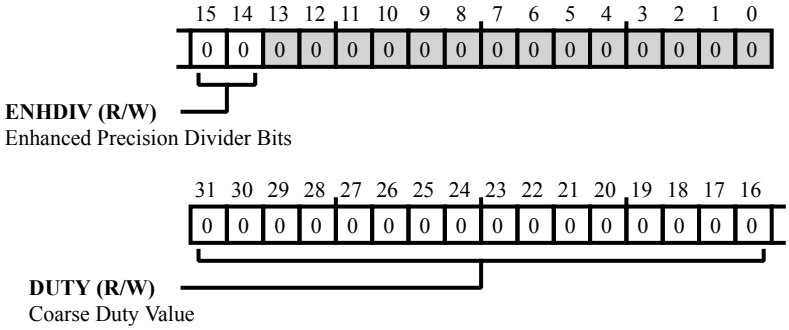


Figure 24-83: PWM_DH_DUTY1 Register Diagram

Table 24-59: PWM_DH_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_DH_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_DH_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DH_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-Low Pulse Duty Register 0

The PWM_DL0 and PWM_DL1 registers determine the width for the low side output pulses. The values in these registers select the assertion count (in terms of t_{CK}) of the low side output pulses for the channel D duty cycle.

The operation of the duty-cycle registers varies, depending on the pulse mode selected with the PWM_DCTL.PULSEMODELO bits. When the pulse mode is symmetrical, the PWM uses the value in the PWM_DL0 register to determine the assertion and deassertion count for the low side output pulses. When the pulse mode is asymmetrical, left half, or right half, the PWM asserts channel D low pulse output for count less than PWM_DL0 and deasserts this output for count greater than PWM_DL1.

The value range for the PWM_DL0 and PWM_DL1 registers depends on the period of the timer being used by the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 (two's complement) and +PWM_TM0/2, when dead time (PWM_CHD_DT) is not considered.

When dead time is considered for symmetrical and asymmetrical pulse modes, the value range for PWM_DL0 and PWM_DL1 depends on the period of the time being used by the channel and the amount of dead time applied to the channel. For example, if PWM_TM0 is used, the duty cycle values may be between -PWM_TM0/2 + PWM_CHD_DT (two's complement) and +PWM_TM0/2 + PWM_CHD_DT.

When dead time is considered for left half or right half pulse modes, if PWM_TM0 is used, the duty cycle values may be between PWM_TM0/2 + PWM_CHD_DT (two's complement) and -PWM_TM0/2 - PWM_CHD_DT.

Note that using values in the PWM_DL0 or PWM_DL1 registers that fall outside these limits causes PWM over or under modulation.

For more information about pulse modes and duty cycle selection, see the Functional Description section.

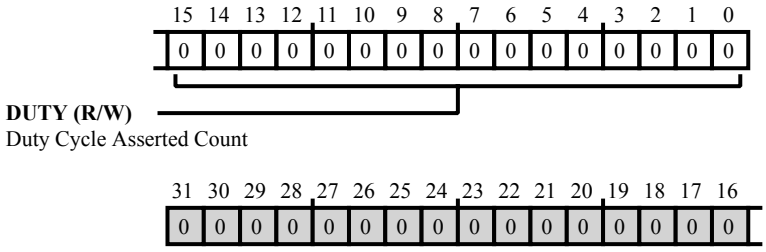


Figure 24-84: PWM_DL0 Register Diagram

Table 24-60: PWM_DL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle Asserted Count. The PWM_DL0.DUTY bits select the duty cycle asserted count for Channel D low side output.

Channel D-Low Heightened-Precision Duty-0 Register

The PWM_DL0_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_DL0 register, allows programs to specify fractional duty cycles. The PWM_DL0_HP register and the PWM_DL0 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_DL0_HP and the PWM_DL0 registers are also present in the single full duty register (if available).

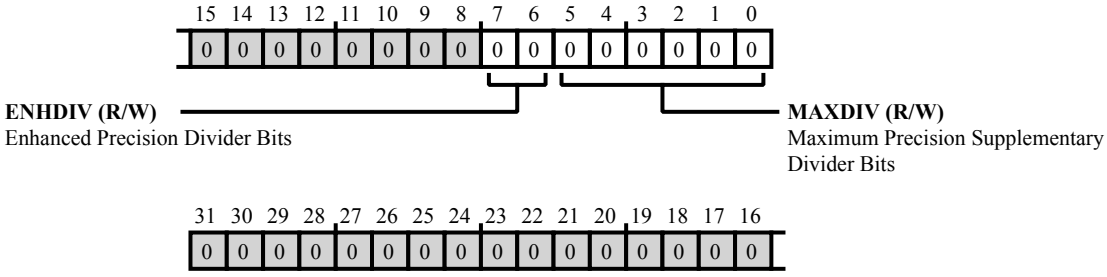


Figure 24-85: PWM_DL0_HP Register Diagram

Table 24-61: PWM_DL0_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DL0_HP.ENHDIV bits provide fractional duty cycles for Channel D low side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel D-Low Pulse Duty Register 1

The PWM_DL0 and PWM_DL1 registers determine the width for the low side output pulses. For more information, see the PWM_DL0 register description.

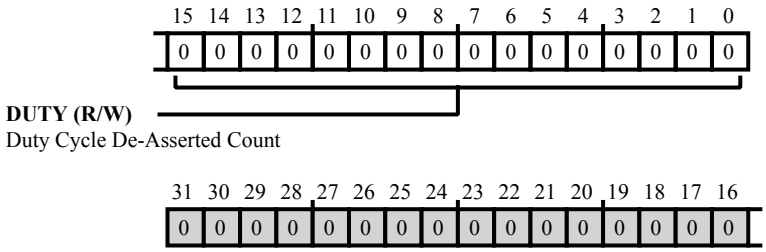


Figure 24-86: PWM_DL1 Register Diagram

Table 24-62: PWM_DL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DUTY	Duty Cycle De-Asserted Count. The PWM_DL1.DUTY bits select the duty cycle de-asserted count for Channel D low side output.

Channel D-Low Heightened-Precision Duty-1 Register

The PWM_DL1_HP register provides a fine-grained edge placement within the system clock period. This register, in conjunction with the PWM_DL1 register, allows programs to specify fractional duty cycles. The PWM_DL1_HP register and the PWM_DL1 register work together in a Q15.8 signed two's complement fixed-point format.

Note that the bit fields in the PWM_DL1_HP and the PWM_DL1 registers are also present in the single full duty register (if available).

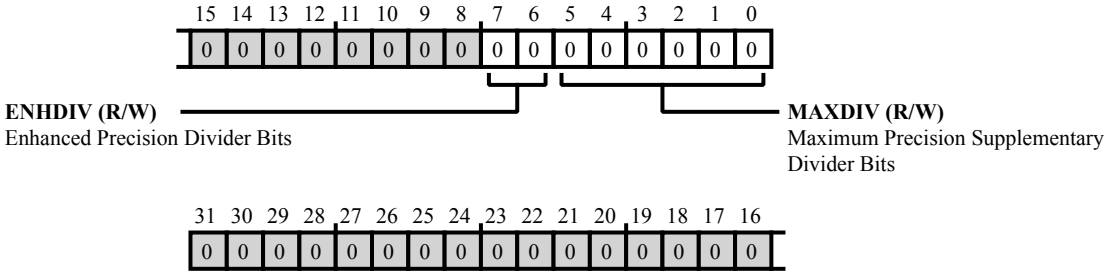


Figure 24-87: PWM_DL1_HP Register Diagram

Table 24-63: PWM_DL1_HP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:6 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DL1_HP.ENHDIV bits provide fractional duty cycles for Channel D low side output.
5:0 (R/W)	MAXDIV	Maximum Precision Supplementary Divider Bits.

Channel A Delay Register

The PWM_DLYA register controls a delay for the Channel A timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYAEN =1). For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYA delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYA must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

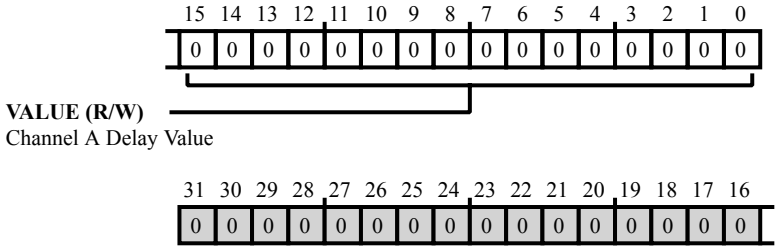


Figure 24-88: PWM_DLYA Register Diagram

Table 24-64: PWM_DLYA Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel A Delay Value. The PWM_DLYA.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel A.

Channel B Delay Register

The PWM_DLYB register controls a delay for the Channel B timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYBEN =1). For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYB delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYB must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

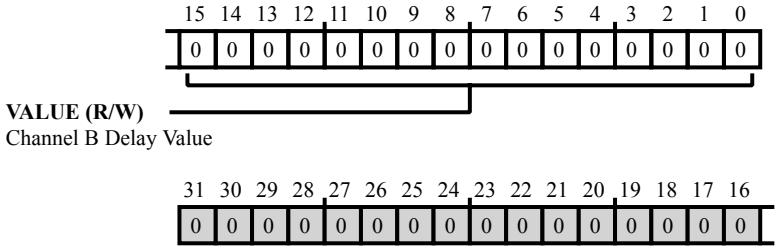


Figure 24-89: PWM_DLYB Register Diagram

Table 24-65: PWM_DLYB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel B Delay Value. The PWM_DLYB.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel B.

Channel C Delay Register

The PWM_DLYC register controls a delay for the Channel C timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYCEN =1). For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYC delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYC must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

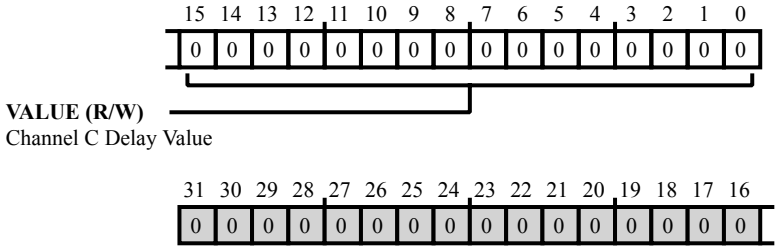


Figure 24-90: PWM_DLYC Register Diagram

Table 24-66: PWM_DLYC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel C Delay Value. The PWM_DLYC.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel C.

Channel D Delay Register

The PWM_DLYD register controls a delay for the Channel D timer (only PWMTMR1, PWMTMR2, PWMTMR3 or PWMTMR4) with reference to the main timer (PWMTMR0). To use apply this delay, the delay must be enabled (PWM_CTL.DLYDEN =1). For more information about applying the delay, see the PWM Functional Description section. Note that the PWM_DLYD delay value must be less that less that twice the period value of the timer being used for the channel (for example, if PWMTMR1 is used, PWM_DLYD must be less than 2xPWMTMR1). Also, note that the period of the main timer must be an integer multiple of the timer being used for the channel (for example, if PWMTMR1 is used, PWMTMR0 = NxPWMTMR1, where N is an integer).

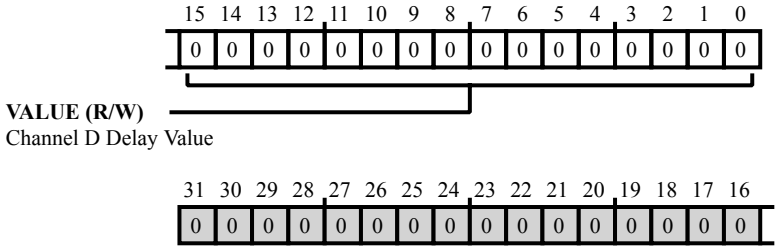


Figure 24-91: PWM_DLYD Register Diagram

Table 24-67: PWM_DLYD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Channel D Delay Value. The PWM_DLYD.VALUE bits select the phase delay between the main timer (PWMTMR0) and the timer used for Channel D.

Channel D-Low Full Duty0 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_DL_DUTY0 register contains the PWM_DL_DUTY0.DUTY bit field from the PWM_DL0 register and the PWM_DL_DUTY0.ENHDIV bit field from the PWM_DL0_HP register.

Note that the PWM_DL_DUTY0 register reads the PWM_DL0 and the PWM_DL0_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

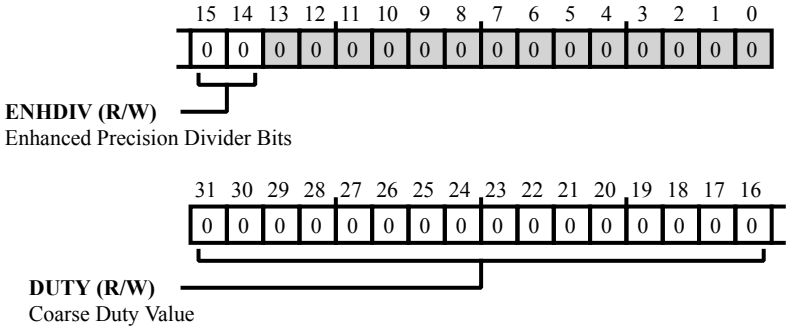


Figure 24-92: PWM_DL_DUTY0 Register Diagram

Table 24-68: PWM_DL_DUTY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_DL_DUTY0.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_DL_DUTY0.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DL_DUTY0.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Channel D-Low Full Duty1 Register

The full duty registers can be used instead of the combined duty and heightened-precision duty registers.The PWM_DL_DUTY1 register contains the PWM_DL_DUTY1.DUTY bit field from the PWM_DL1 register and the PWM_DL_DUTY1.ENHDIV bit field from the PWM_DL1_HP register.

Note that the PWM_DL_DUTY1 register reads the PWM_DL1 and the PWM_DL1_HP register values and visa-versa.

When heightened-precision edge placement is enabled, bits [15:8] of these registers form the decimal part of a non-integer, fixed-point duty cycle value in Q15.8 format. The lowest bits are ignored.

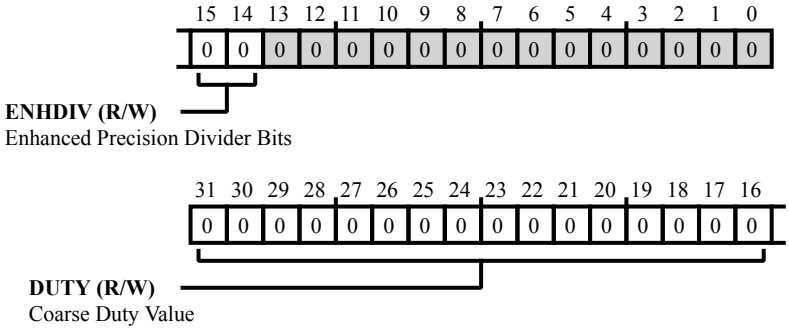


Figure 24-93: PWM_DL_DUTY1 Register Diagram

Table 24-69: PWM_DL_DUTY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	DUTY	Coarse Duty Value. The PWM_DL_DUTY1.DUTY bits determine the output pulse-widths in the normal PWM operation. When heightened-precision edge placement is enabled, the PWM_DL_DUTY1.DUTY bit field forms the integer part of a non-integer, fixed-point duty cycle value in Q15.8 format.
15:14 (R/W)	ENHDIV	Enhanced Precision Divider Bits. The PWM_DL_DUTY1.ENHDIV bits form the decimal part of a non-integer, fixed-point duty cycle value when heightened-precision edge placement is enabled in Q15.8 format.

Interrupt Latch Register

The PWM_ILAT register latches the occurrence of unmasked (enabled) PWM interrupt requests. These interrupt requests are unmasked or masked with the PWM_IMSK register.

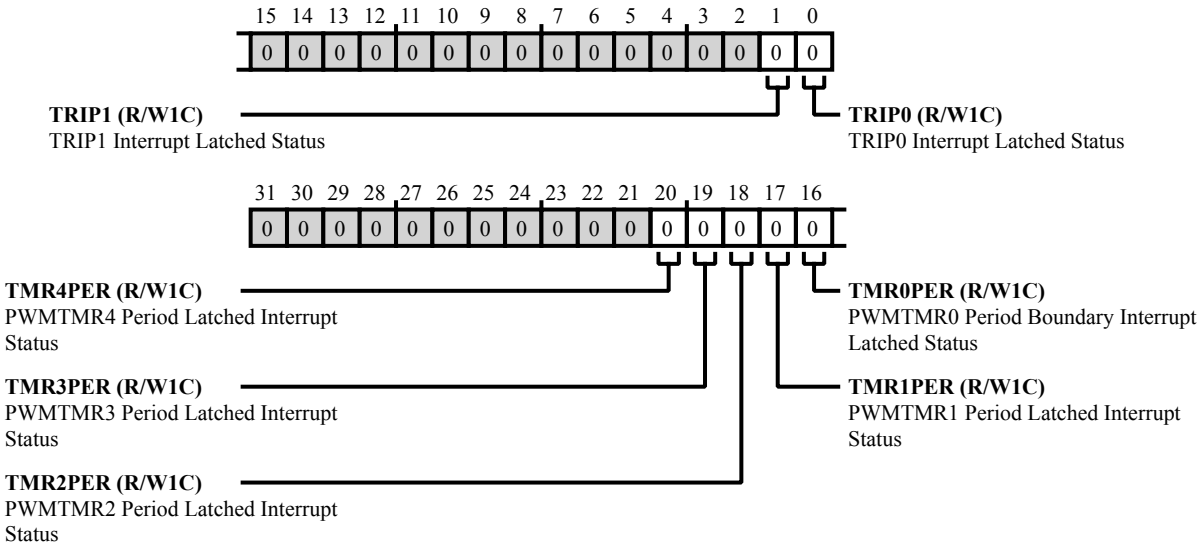


Figure 24-94: PWM_ILAT Register Diagram

Table 24-70: PWM_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	TMR4PER	PWMTMR4 Period Latched Interrupt Status. The PWM_ILAT.TMR4PER bit indicates the latched status of the PWMTMR4 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
19 (R/W1C)	TMR3PER	PWMTMR3 Period Latched Interrupt Status. The PWM_ILAT.TMR3PER bit indicates the latched status of the PWMTMR3 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
18 (R/W1C)	TMR2PER	PWMTMR2 Period Latched Interrupt Status. The PWM_ILAT.TMR2PER bit indicates the latched status of the PWMTMR2 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched

Table 24-70: PWM_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	TMR1PER	PWMTMR1 Period Latched Interrupt Status. The `PWM_ILAT.TMR1PER` bit indicates the latched status of the PWMTMR1 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
16 (R/W1C)	TMR0PER	PWMTMR0 Period Boundary Interrupt Latched Status. The `PWM_ILAT.TMR0PER` bit indicates the latched status of the PWMTMR0 period boundary interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
1 (R/W1C)	TRIP1	TRIP1 Interrupt Latched Status. The `PWM_ILAT.TRIP1` bit indicates the latched status of the TRIP1 interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched
0 (R/W1C)	TRIP0	TRIP0 Interrupt Latched Status. The `PWM_ILAT.TRIP0` bit indicates the latched status of the TRIP0 interrupt request.	
		0	No Interrupt Latched
		1	Interrupt Latched

Interrupt Mask Register

The PWM_IMSK register masks (disables) or unmasks (enables) PWM interrupt requests. When an unmasked interrupt occurs, the PWM latches the interrupt status in the PWM_ILAT register.

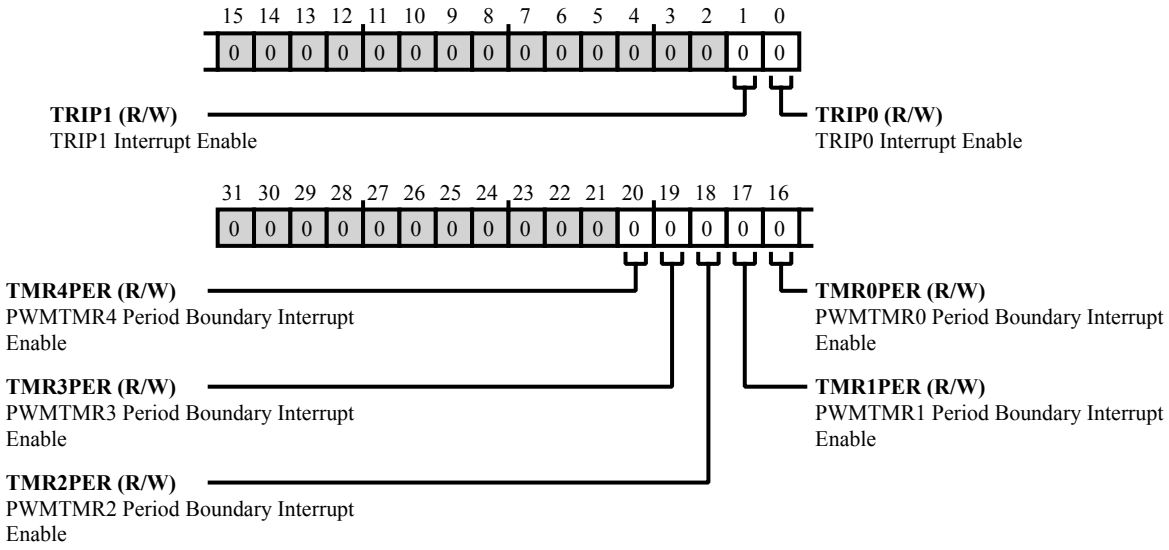


Figure 24-95: PWM_IMSK Register Diagram

Table 24-71: PWM_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W)	TMR4PER	PWMTMR4 Period Boundary Interrupt Enable. The PWM_IMSK.TMR4PER bit enables (unmasks) the PWMTMR4 period boundary interrupt request. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR4PER =1).	
		0	Mask PWMTMR4 Period Interrupt
		1	Unmask PWMTMR4 Period Interrupt
19 (R/W)	TMR3PER	PWMTMR3 Period Boundary Interrupt Enable. The PWM_IMSK.TMR3PER bit enables (unmasks) the PWMTMR3 period boundary interrupt request. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR3PER =1).	
		0	Mask PWMTMR3 Period Interrupt
		1	Unmask PWMTMR3 Period Interrupt

Table 24-71: PWM_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	TMR2PER	PWMTMR2 Period Boundary Interrupt Enable. The PWM_IMSK.TMR2PER bit enables (unmasks) the PWMTMR2 period boundary interrupt request. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR2PER =1).	
		0	Mask PWMTMR2 Period Interrupt
		1	Unmask PWMTMR2 Period Interrupt
17 (R/W)	TMR1PER	PWMTMR1 Period Boundary Interrupt Enable. The PWM_IMSK.TMR1PER bit enables (unmasks) the PWMTMR1 period boundary interrupt request. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR1PER =1).	
		0	Mask PWMTMR1 Period Interrupt
		1	Unmask PWMTMR1 Period Interrupt
16 (R/W)	TMR0PER	PWMTMR0 Period Boundary Interrupt Enable. The PWM_IMSK.TMR0PER bit enables (unmasks) the PWMTMR0 period boundary interrupt request. This condition occurs when the timers period boundary is reached (PWM_STAT.TMR0PER =1).	
		0	Mask PWMTMR0 Period Interrupt
		1	Unmask PWMTMR0 Period Interrupt
1 (R/W)	TRIP1	TRIP1 Interrupt Enable. The PWM_IMSK.TRIP1 bit enables (unmasks) the TRIP1 interrupt request. This condition occurs when fault input is tripped (PWM_STAT.TRIP1 =1).	
		0	Mask TRIP1 Interrupt
		1	Unmask TRIP1 Interrupt
0 (R/W)	TRIP0	TRIP0 Interrupt Enable. The PWM_IMSK.TRIP0 bit enables (unmasks) the TRIP0 interrupt request. This condition occurs when fault input is tripped (PWM_STAT.TRIP0 =1).	
		0	Mask TRIP0 Interrupt
		1	Unmask TRIP0 Interrupt

Status Register

The PWM_STAT register indicates the PWM PWMTRIP1-0 fault and input level status, indicates the Channel A-D fault and self-restart status, and indicates the PWMTMR4-0 phase.

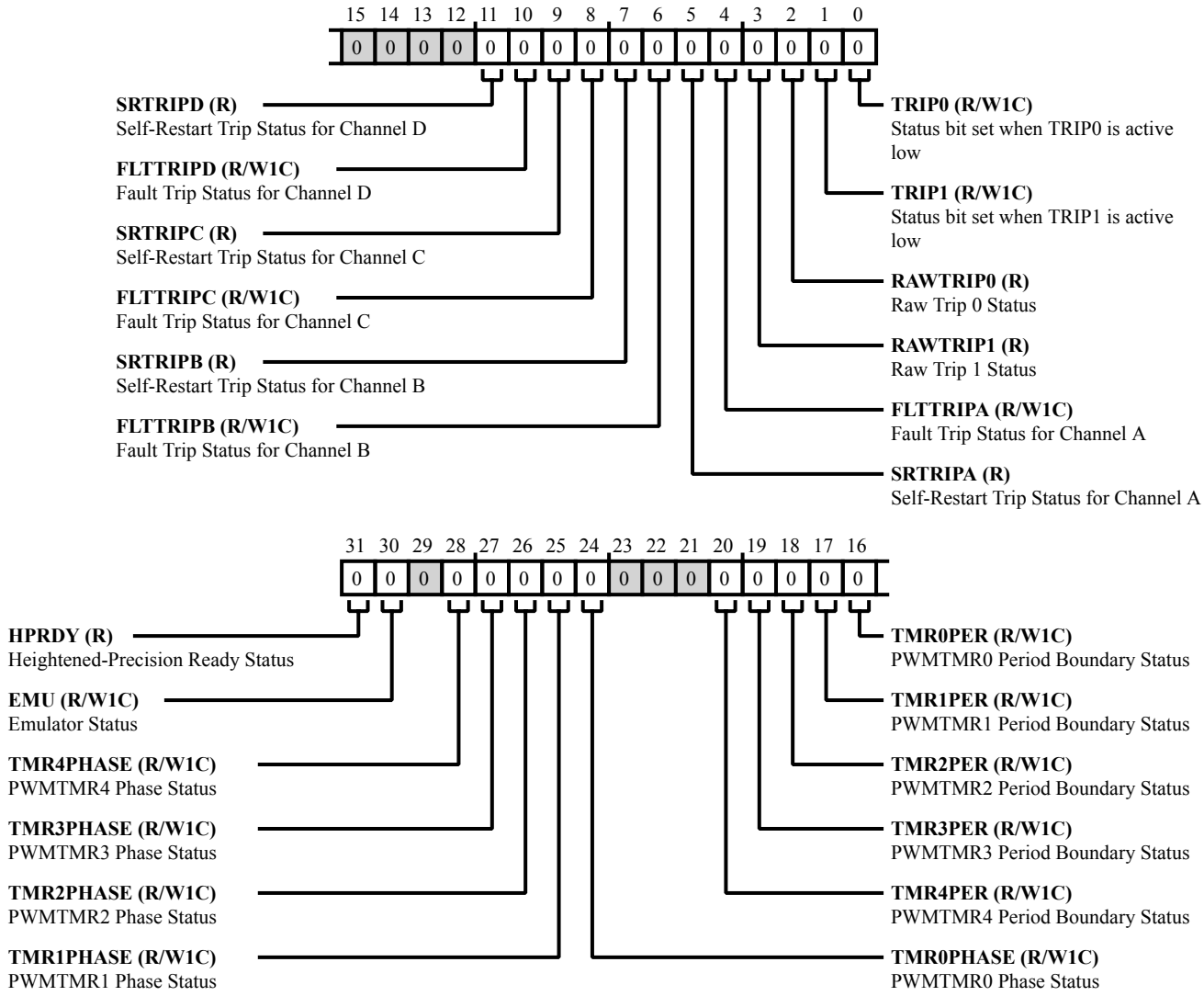


Figure 24-96: PWM_STAT Register Diagram

Table 24-72: PWM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/NW)	HPRDY	Heightened-Precision Ready Status. The PWM_STAT.HPRDY bit indicates whether or not the PWM is ready for heightened-precision operation.	
		0	HPPWM Not Ready For Operation
		1	HPPWM Ready For Operation
30 (R/W1C)	EMU	Emulator Status.	
28 (R/W1C)	TMR4PHASE	PWMTMR4 Phase Status. The PWM_STAT.TMR4PHASE bit indicates the current phase for the PWMTMR4 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
27 (R/W1C)	TMR3PHASE	PWMTMR3 Phase Status. The PWM_STAT.TMR3PHASE bit indicates the current phase for the PWMTMR3 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
26 (R/W1C)	TMR2PHASE	PWMTMR2 Phase Status. The PWM_STAT.TMR2PHASE bit indicates the current phase for the PWMTMR2 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
25 (R/W1C)	TMR1PHASE	PWMTMR1 Phase Status. The PWM_STAT.TMR1PHASE bit indicates the current phase for the PWMTMR1 waveform.	
		0	1st Half Phase
		1	2nd Half Phase
24 (R/W1C)	TMR0PHASE	PWMTMR0 Phase Status. The PWM_STAT.TMR0PHASE bit indicates the current phase for the PWMTMR0 waveform.	
		0	1st Half Phase
		1	2nd Half Phase

Table 24-72: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/W1C)	TMR4PER	PWMTMR4 Period Boundary Status. The PWM_STAT.TMR4PER bit indicates whether or not the PWMTMR4 period boundary has been reached.	
		0	PWMTMR4 period boundary not reached
		1	PWMTMR4 period boundary reached
19 (R/W1C)	TMR3PER	PWMTMR3 Period Boundary Status. The PWM_STAT.TMR3PER bit indicates whether or not the PWMTMR3 period boundary has been reached.	
		0	PWMTMR3 period boundary not reached
		1	PWMTMR3 period boundary reached
18 (R/W1C)	TMR2PER	PWMTMR2 Period Boundary Status. The PWM_STAT.TMR2PER bit indicates whether or not the PWMTMR2 period boundary has been reached.	
		0	PWMTMR2 period boundary not reached
		1	PWMTMR2 period boundary reached
17 (R/W1C)	TMR1PER	PWMTMR1 Period Boundary Status. The PWM_STAT.TMR1PER bit indicates whether or not the PWMTMR1 period boundary has been reached.	
		0	PWMTMR1 period boundary not reached
		1	PWMTMR1 period boundary reached
16 (R/W1C)	TMR0PER	PWMTMR0 Period Boundary Status. The PWM_STAT.TMR0PER bit indicates whether or not the PWMTMR0 period boundary has been reached.	
		0	PWMTMR0 period boundary not reached
		1	PWMTMR0 period boundary reached
11 (R/NW)	SRTRIPD	Self-Restart Trip Status for Channel D. The PWM_STAT.SRTRIPD bit indicates whether the PWM Channel D self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel D Self-Restart Trip Status is "not tripped"
		1	Channel D Self-Restart Trip Status is "tripped"

Table 24-72: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	FLTTRIPD	Fault Trip Status for Channel D. The PWM_STAT.FLTTRIPD bit indicates whether the PWM Channel D fault has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel D Fault Trip Status is "not tripped"
		1	Channel D Fault Trip Status is "tripped"
9 (R/NW)	SRTRIPC	Self-Restart Trip Status for Channel C. The PWM_STAT.SRTRIPC bit indicates whether the PWM Channel C self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel C Self-Restart Trip Status is "not tripped"
		1	Channel C Self-Restart Trip Status is "tripped"
8 (R/W1C)	FLTTRIPC	Fault Trip Status for Channel C. The PWM_STAT.FLTTRIPC bit indicates whether the PWM Channel C fault has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel C Fault Trip Status is "not tripped"
		1	Channel C Fault Trip Status is "tripped"
7 (R/NW)	SRTRIPB	Self-Restart Trip Status for Channel B. The PWM_STAT.SRTRIPB bit indicates whether the PWM Channel B self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel B Self-Restart Trip Status is "not tripped"
		1	Channel B Self-Restart Trip Status is "tripped"
6 (R/W1C)	FLTTRIPB	Fault Trip Status for Channel B. The PWM_STAT.FLTTRIPB bit indicates whether the PWM Channel B fault has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel B Fault Trip Status is "not tripped"
		1	Channel A Fault Trip Status is "tripped"

Table 24-72: PWM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	SRTRIPA	Self-Restart Trip Status for Channel A. The PWM_STAT.SRTRIPA bit indicates whether the PWM Channel A self-restart has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel A Self-Restart Trip Status is "not tripped"
		1	Channel A Self-Restart Trip Status is "tripped"
4 (R/W1C)	FLTTRIPA	Fault Trip Status for Channel A. The PWM_STAT.FLTTRIPA bit indicates whether the PWM Channel A fault has been tripped. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Channel A Fault Trip Status is "not tripped"
		1	Channel A Fault Trip Status is "tripped"
3 (R/NW)	RAWTRIP1	Raw Trip 1 Status. The PWM_STAT.RAWTRIP1 bit indicates the raw input level for the PWM TRIP1 input.	
		0	TRIP1 Level is Low
		1	TRIP1 Level is High
2 (R/NW)	RAWTRIP0	Raw Trip 0 Status. The PWM_STAT.RAWTRIP0 bit indicates the raw input level for the PWM TRIP0 input.	
		0	TRIP0 Level is Low
		1	TRIP0 Level is High
1 (R/W1C)	TRIP1	Status bit set when TRIP1 is active low. The PWM_STAT.TRIP1 bit indicates whether the PWM TRIP1 fault has been tripped with an active-low input.	
		0	TRIP1 status is "not tripped"
		1	TRIP1 status is "tripped" (active low)
0 (R/W1C)	TRIP0	Status bit set when TRIP0 is active low. The PWM_STAT.TRIP0 bit indicates whether the PWM TRIP0 fault has been tripped with an active-low input.	
		0	TRIP0 status is "not tripped"
		1	TRIP0 status is "tripped" (active low)

Software Trip Register

The PWM_SWTRIP register has individual, independent control for software trip of all PWM Channels. Unlike the Global SWTRIP bit, each channel (High and Low) has its own SWTRIP bit.

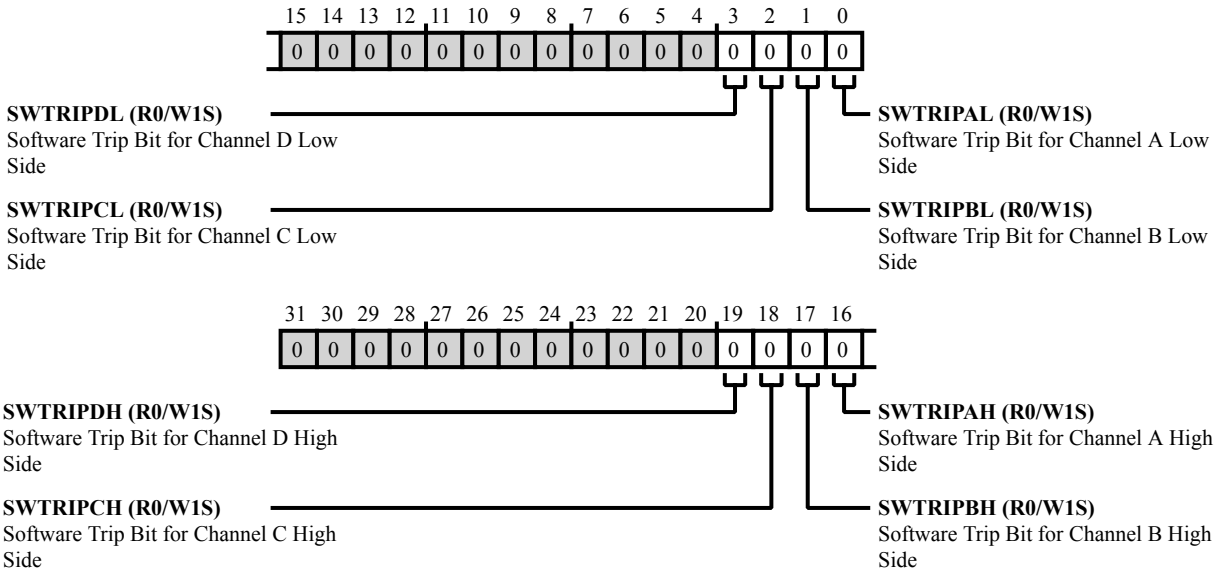


Figure 24-97: PWM_SWTRIP Register Diagram

Table 24-73: PWM_SWTRIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19 (R0/W1S)	SWTRIPDH	Software Trip Bit for Channel D High Side. Writing 1 forces Channel D High Side to trip.
18 (R0/W1S)	SWTRIPCH	Software Trip Bit for Channel C High Side. Writing 1 forces Channel C High Side to trip.
17 (R0/W1S)	SWTRIPBH	Software Trip Bit for Channel B High Side. Writing 1 forces Channel B High Side to trip.
16 (R0/W1S)	SWTRIPAH	Software Trip Bit for Channel A High Side. Writing 1 forces Channel A High Side to trip.
3 (R0/W1S)	SWTRIPDL	Software Trip Bit for Channel D Low Side. Writing 1 forces Channel D Low Side to trip.
2 (R0/W1S)	SWTRIPCL	Software Trip Bit for Channel C Low Side. Writing 1 forces Channel C Low Side to trip.
1 (R0/W1S)	SWTRIPBL	Software Trip Bit for Channel B Low Side. Writing 1 forces Channel B Low Side to trip.

Table 24-73: PWM_SWTRIP Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R0/W1S)	SWTRIPAL	Software Trip Bit for Channel A Low Side. Writing 1 forces Channel A Low Side to trip.

Sync Pulse Width Register

The PWM_SYNC_WID register selects the pulse width for the external sync pulse available on the PWM_SYNC pin. The relation between the PWM_SYNC_WID register value and the pulse width (T_{PWM_SYNC}) is give by the formula:

PWM_SYNC_WID = (T_{PWM_SYNC} / t_{CK}) -1

For more information about applying the sync pulse width, see the PWM Functional Description section. Note that if the pulse width is changed in between sync pulses, the PWM applies the changed width on the next internal sync pulse. If, while the sync pulse is active, the chosen timer reaches its period boundary, the changed pulse width takes effect on that period boundary.

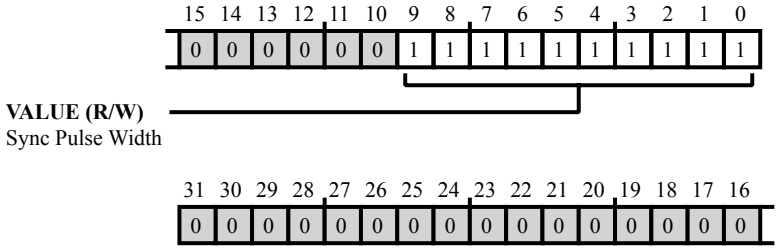


Figure 24-98: PWM_SYNC_WID Register Diagram

Table 24-74: PWM_SYNC_WID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	VALUE	Sync Pulse Width. The PWM_SYNC_WID.VALUE bits select the pulse width for the external sync pulse available on the PWM_SYNC pin.

Timer 0 Period Register

The PWM_TM0 register controls the switch period T_{SP} of the PWMTMR0 timer. The PWM_TM0 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM0= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM0 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

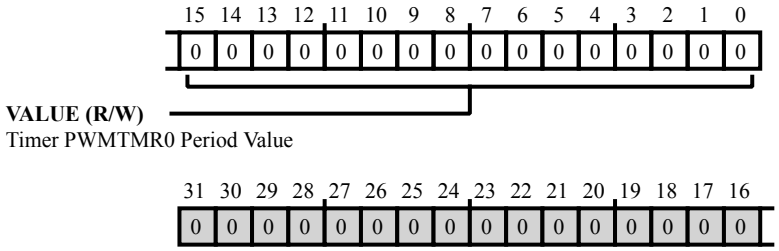


Figure 24-99: PWM_TM0 Register Diagram

Table 24-75: PWM_TM0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR0 Period Value. The PWM_TM0.VALUE bits select the period for the PWMTMR0 timer.

Timer 1 Period Register

The PWM_TM1 register controls the switch period (T_{SP} of the PWMTMR1 timer. The PWM_TM1 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM1= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM1 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

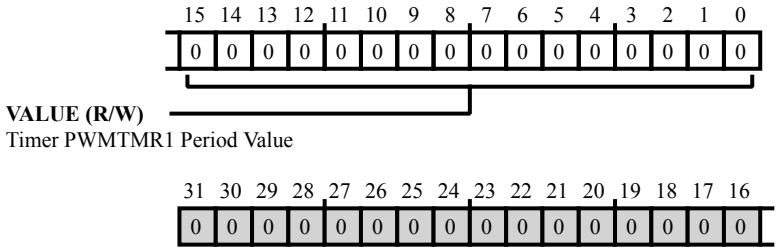


Figure 24-100: PWM_TM1 Register Diagram

Table 24-76: PWM_TM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR1 Period Value. The PWM_TM1.VALUE bits select the period for the PWMTMR1 timer.

Timer 2 Period Register

The PWM_TM2 register controls the switch period (T_{SP} of the PWMTMR2 timer. The PWM_TM2 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM1= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM2 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

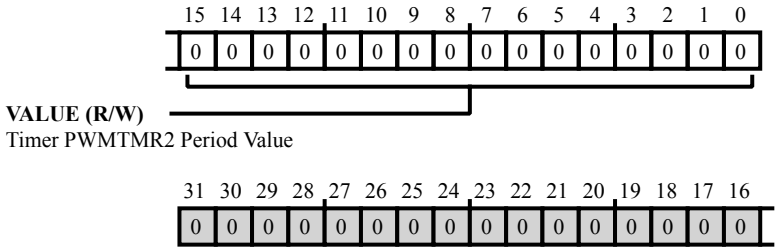


Figure 24-101: PWM_TM2 Register Diagram

Table 24-77: PWM_TM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR2 Period Value. The PWM_TM2.VALUE bits select the period for the PWMTMR2 timer.

Timer 3 Period Register

The PWM_TM3 register controls the switch period (T_{SP} of the PWMTMR3 timer. The PWM_TM3 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM3= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM3 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

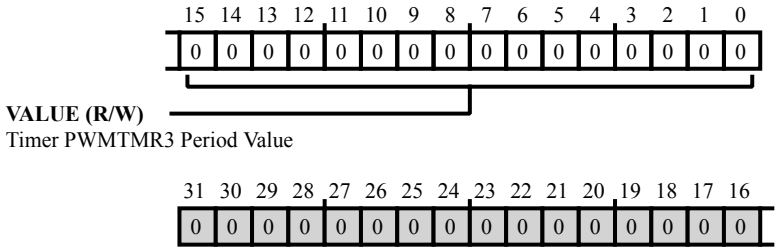


Figure 24-102: PWM_TM3 Register Diagram

Table 24-78: PWM_TM3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR3 Period Value. The PWM_TM3.VALUE bits select the period for the PWMTMR3 timer.

Timer 4 Period Register

The PWM_TM4 register controls the switch period (T_{SP} of the PWMTMR4 timer. The PWM_TM4 value is in units of t_{CK} (the period of the peripheral clock) and the and is given by the formula:

PWM_TM4= (T_{SP}) / 2 x t_{CK}

The value written to the register is effectively the number of t_{CK} clock increments in half the period of the respective timer. For more information about applying the switch period, see the PWM Functional Description section. Note that PWM_TM4 values of 0 and 1 are not defined and must not be used when the PWM is enabled.

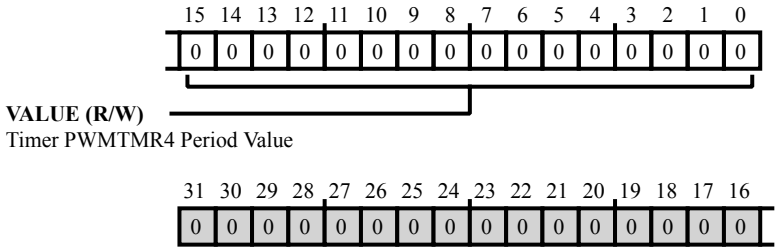


Figure 24-103: PWM_TM4 Register Diagram

Table 24-79: PWM_TM4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Timer PWMTMR4 Period Value. The PWM_TM4.VALUE bits select the period for the PWMTMR4 timer.

Trip Configuration Register

The PWM_TRIPCFG register configures Channel A, B, C, and D trip operation for trip inputs TRIP0 and TRIP1.

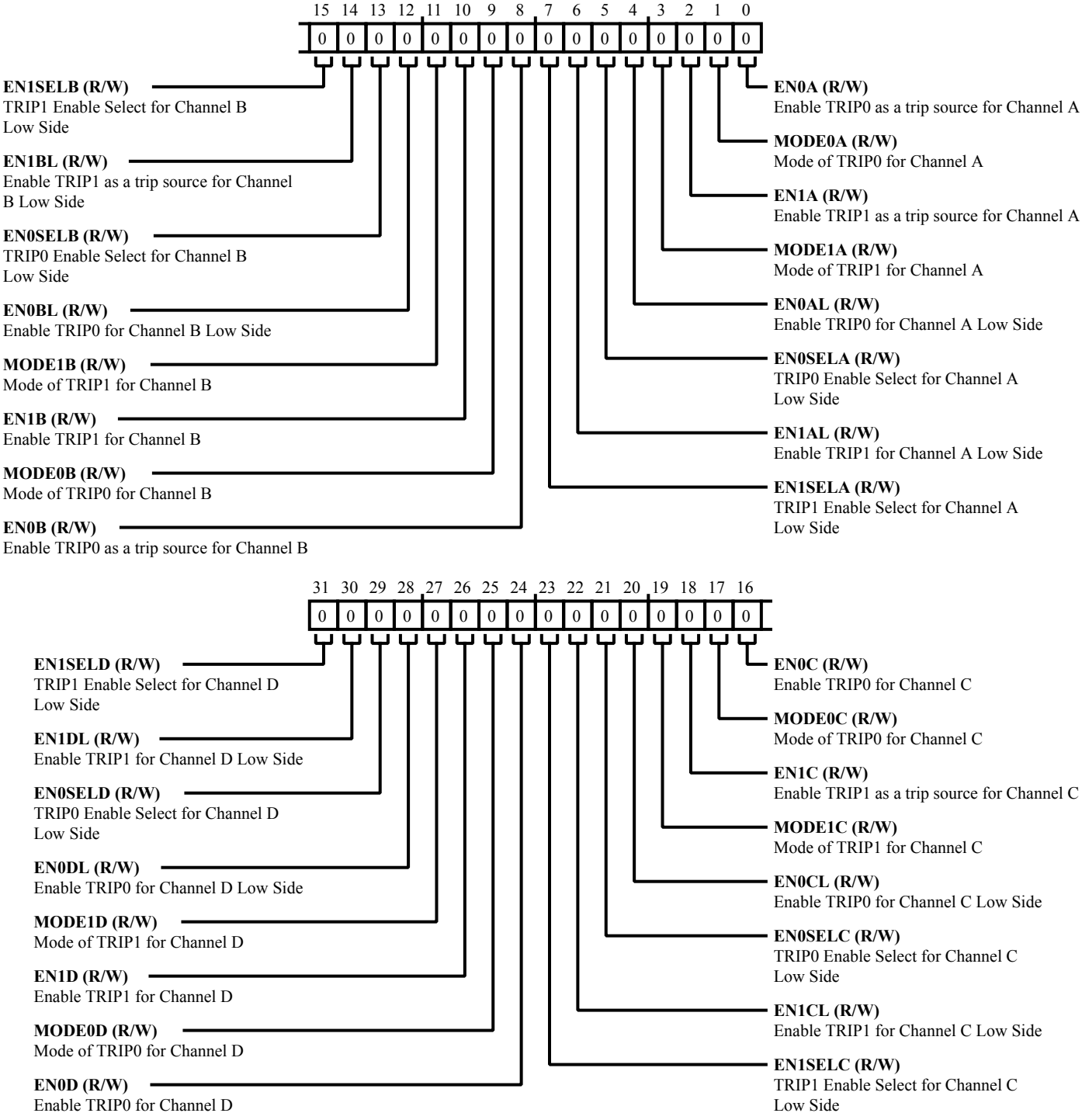


Figure 24-104: PWM_TRIPCFG Register Diagram

Table 24-80: PWM_TRIPCFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	EN1SELD	TRIP1 Enable Select for Channel D Low Side. The PWM_TRIPCFG.EN1SELD bit selects the TRIP1 Enable bit for Channel D Low Side. If cleared (=0), Channel D Low Side TRIP1 Enable depends on the PWM_TRIPCFG.EN0D bit setting. If set (=1), it depends on the PWM_TRIPCFG.EN0DL bit setting.	
		0	Select EN0D for Channel D Low Side Trip1 Enable
		1	Select EN0DL for Channel D Low Side Trip1 Enable
30 (R/W)	EN1DL	Enable TRIP1 for Channel D Low Side. The PWM_TRIPCFG.EN1DL bit independently enables TRIP1 as a source for Channel D Low Side.	
		0	Disable Trip1 for Channel D Low Side
		1	Enable Trip1 for Channel D Low Side
29 (R/W)	EN0SELD	TRIP0 Enable Select for Channel D Low Side. The PWM_TRIPCFG.EN0SELD bit selects the TRIP0 Enable bit for Channel D Low Side. If cleared (=0), Channel D Low Side TRIP0 Enable depends on the PWM_TRIPCFG.EN0D bit setting. If set (=1), it depends on the PWM_TRIPCFG.EN0DL bit setting.	
		0	Select EN0D for Channel D Low Side Trip0 Enable
		1	Select EN0DL for Channel D Low Side Trip0 Enable
28 (R/W)	EN0DL	Enable TRIP0 for Channel D Low Side. The PWM_TRIPCFG.EN0DL bit independently enables TRIP0 as a source for Channel D Low Side.	
		0	Disable Trip0 for Channel D Low Side
		1	Enable Trip0 for Channel D Low Side
27 (R/W)	MODE1D	Mode of TRIP1 for Channel D. The PWM_TRIPCFG.MODE1D bit selects the trip mode of TRIP1 for Channel D. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input

Table 24-80: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	EN1D	Enable TRIP1 for Channel D. The PWM_TRIPCFG.EN1D bit enables TRIP1 as a trip source for Channel D.	
		0	Disable TRIP1 for Channel D
		1	Enable TRIP1 for Channel D
25 (R/W)	MODE0D	Mode of TRIP0 for Channel D. The PWM_TRIPCFG.MODE0D bit selects the trip mode of TRIP0 for Channel D. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
24 (R/W)	EN0D	Enable TRIP0 for Channel D. The PWM_TRIPCFG.EN0D bit enables TRIP0 as a trip source for Channel D.	
		0	Disable TRIP0 for Channel D
		1	Enable TRIP0 for Channel D
23 (R/W)	EN1SELC	TRIP1 Enable Select for Channel C Low Side. The PWM_TRIPCFG.EN1SELC bit selects the TRIP1 Enable bit for Channel C Low Side. If cleared (=0), Channel C Low Side TRIP1 Enable depends on the PWM_TRIPCFG.EN0C bit setting. If set (=1), it depends on the PWM_TRIPCFG.EN0CL bit setting.	
		0	Select EN0C for Channel C Low Side Trip1 Enable
		1	Select EN0CL for Channel C Low Side Trip1 Enable
22 (R/W)	EN1CL	Enable TRIP1 for Channel C Low Side. The PWM_TRIPCFG.EN1CL bit independently enables TRIP0 as a source for Channel C Low Side.	
		0	Disable Trip1 for Channel C Low Side
		1	Enable Trip1 for Channel C Low Side

Table 24-80: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	EN0SELC	TRIP0 Enable Select for Channel C Low Side. The PWM_TRIPCFG.EN0SELC bit selects the TRIP0 Enable bit for Channel C Low Side. If cleared (=0), Channel C Low Side TRIP0 Enable depends on the PWM_TRIPCFG.EN0C bit setting. If set (=1), it depends on the PWM_TRIPCFG.EN0CL bit setting.	
		0	Select EN0C for Channel C Low Side Trip0 Enable
		1	Select EN0CL for Channel C Low Side Trip0 Enable
20 (R/W)	EN0CL	Enable TRIP0 for Channel C Low Side. The PWM_TRIPCFG.EN0CL bit independently enables TRIP0 as a source for Channel C Low Side.	
		0	Disable Trip0 for Channel C Low Side
		1	Enable Trip0 for Channel C Low Side
19 (R/W)	MODE1C	Mode of TRIP1 for Channel C. The PWM_TRIPCFG.MODE1C bit selects the trip mode of TRIP1 for Channel C. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
18 (R/W)	EN1C	Enable TRIP1 as a trip source for Channel C. The PWM_TRIPCFG.EN1C bit enables TRIP1 as a trip source for Channel C.	
		0	Disable TRIP1 for Channel C
		1	Enable TRIP1 for Channel C
17 (R/W)	MODE0C	Mode of TRIP0 for Channel C. The PWM_TRIPCFG.MODE0C bit selects the trip mode of TRIP0 for Channel C. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
16 (R/W)	EN0C	Enable TRIP0 for Channel C. The PWM_TRIPCFG.EN0C bit enables TRIP0 as a trip source for Channel C.	
		0	Disable TRIP0 for Channel C
		1	Enable TRIP0 for Channel C

Table 24-80: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	EN1SELB	TRIP1 Enable Select for Channel B Low Side. The PWM_TRIPCFG.EN1SELB bit selects the TRIP1 Enable bit for Channel B Low Side. If cleared (=0), Channel B Low Side TRIP1 Enable depends on the PWM_TRIPCFG.EN0B bit setting. If set (=1), it depends on the PWM_TRIPCFG.EN0BL bit setting.	
		0	Select EN0B for Channel B Low Side Trip1 Enable
		1	Select EN0BL for Channel B Low Side Trip1 Enable
14 (R/W)	EN1BL	Enable TRIP1 as a trip source for Channel B Low Side. The PWM_TRIPCFG.EN1BL bit enables TRIP1 as a source for Channel B Low Side independently	
		0	Disable Trip1 for Channel B Low Side
		1	Enable Trip1 for Channel B Low Side
13 (R/W)	EN0SELB	TRIP0 Enable Select for Channel B Low Side. The PWM_TRIPCFG.EN0SELB bit selects the TRIP0 Enable bit for Channel B Low Side. If cleared (=0), Channel B Low Side TRIP0 Enable depends on the PWM_TRIPCFG.EN0B bit setting. If set (=1), it depends on the PWM_TRIPCFG.EN0BL bit setting.	
		0	Select EN0B for Channel B Low Side Trip0 Enable
		1	Select EN0BL for Channel B Low Side Trip0 Enable
12 (R/W)	EN0BL	Enable TRIP0 for Channel B Low Side. The PWM_TRIPCFG.EN0BL bit independently enables TRIP0 as a source for Channel B Low Side.	
		0	Disable Trip0 for Channel B Low Side
		1	Enable Trip0 for Channel B Low Side
11 (R/W)	MODE1B	Mode of TRIP1 for Channel B. The PWM_TRIPCFG.MODE1B bit selects the trip mode of TRIP1 for Channel B. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input

Table 24-80: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	EN1B	Enable TRIP1 for Channel B. The PWM_TRIPCFG.EN1B bit enables TRIP1 as a trip source for Channel B.	
		0	Disable TRIP1 for Channel B
		1	Enable TRIP1 for Channel B
9 (R/W)	MODE0B	Mode of TRIP0 for Channel B. The PWM_TRIPCFG.MODE0B bit selects the trip mode of TRIP0 for Channel B. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
8 (R/W)	EN0B	Enable TRIP0 as a trip source for Channel B. The PWM_TRIPCFG.EN0B bit enables TRIP0 as a trip source for Channel B.	
		0	Disable TRIP0 for Channel B
		1	Enable TRIP0 for Channel B
7 (R/W)	EN1SELA	TRIP1 Enable Select for Channel A Low Side. The PWM_TRIPCFG.EN1SELA bit selects the TRIP1 Enable bit for Channel A Low Side. If cleared (=0), Channel A Low Side TRIP1 Enable depends on the PWM_TRIPCFG.EN0A bit setting. If set (=1), it depends on the PWM_TRIPCFG.EN0AL bit setting.	
		0	Select EN0A for Channel A Low Side Trip1 Enable
		1	Select EN0AL for Channel A Low Side Trip1 Enable
6 (R/W)	EN1AL	Enable TRIP1 for Channel A Low Side. The PWM_TRIPCFG.EN1AL bit independently enables TRIP1 as a source for Channel A Low Side.	
		0	Disable Trip1 for Channel A Low Side
		1	Enable Trip1 for Channel A Low Side

Table 24-80: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	EN0SELA	TRIP0 Enable Select for Channel A Low Side. The PWM_TRIPCFG.EN0SELA bit selects the TRIP0 Enable bit for Channel A Low Side. If cleared (=0), Channel A Low Side TRIP0 Enable depends on the PWM_TRIPCFG.EN0A bit setting. If set (=1), it depends on the PWM_TRIPCFG.EN0AL bit setting.	
		0	Select EN0A for Channel A Low Side Trip0 Enable
		1	Select EN0AL for Channel A Low Side Trip0 Enable
4 (R/W)	EN0AL	Enable TRIP0 for Channel A Low Side. The PWM_TRIPCFG.EN0AL bit independently enables TRIP0 as a source for Channel A Low Side.	
		0	Disable Trip0 for Channel A Low Side
		1	Enable Trip0 for Channel A Low Side
3 (R/W)	MODE1A	Mode of TRIP1 for Channel A. The PWM_TRIPCFG.MODE1A bit selects the trip mode of TRIP1 for Channel A. For more information, see the PWM_TRIPCFG.MODE0A bit description.	
		0	Fault Trip on TRIP1 Input
		1	Self Restart on TRIP1 Input
2 (R/W)	EN1A	Enable TRIP1 as a trip source for Channel A. The PWM_TRIPCFG.EN1A bit enables TRIP1 as a trip source for Channel A.	
		0	Disable TRIP1 for Channel A
		1	Enable TRIP1 for Channel A

Table 24-80: PWM_TRIPCFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MODE0A	Mode of TRIP0 for Channel A. The PWM_TRIPCFG.MODE0A bit selects the trip mode of TRIP0 for Channel A. In fault-trip mode (PWM_TRIPCFG.MODE0A =0), after the input is tripped, the trip status appears in the corresponding channels fault-trip status bit (for example, PWM_STAT.FLTTRIPA), and the PWM immediately shuts down outputs of that channel. After a fault trip occurs, when the trip condition is no longer active, the processor may cause channel outputs to resume by completing a write-1-to-clear the corresponding fault-trip status bit. The raw (input level) trip input state is available from the PWM_STAT.RAWTRIP0 and PWM_STAT.RAWTRIP0 bits. In self-restart mode (PWM_TRIPCFG.MODE0A =1), after the input is tripped, the trip status appears in the corresponding channels self-restart status bit (for example, PWM_STAT.SRTRIPA), and the PWM immediately shuts down outputs of that channel. On the next timer period boundary (of the PWMTMRx used by that channel), if the trip condition is not active, the PWM clears the status and restarts the channels output.	
		0	Fault Trip on TRIP0 Input
		1	Self Restart on TRIP0 Input
0 (R/W)	EN0A	Enable TRIP0 as a trip source for Channel A. The PWM_TRIPCFG.EN0A bit enables TRIP0 as a trip source for Channel A.	
		0	Disable TRIP0 for Channel A
		1	Enable TRIP0 for Channel A

Trip Polarity Register

The PWM_TRIP_POL register controls the Channel Polarity upon tripping for each PWM Channel independently.

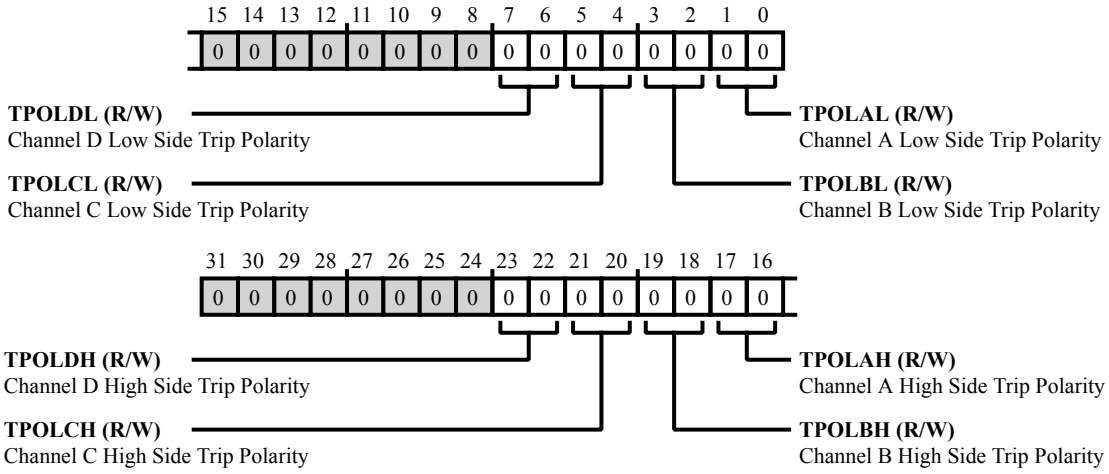


Figure 24-105: PWM_TRIP_POL Register Diagram

Table 24-81: PWM_TRIP_POL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:22 (R/W)	TPOLDH	Channel D High Side Trip Polarity.	
		0	Set Channel D High Side to Inactive Polarity upon Trip
		1	Set Channel D High Side to Active Polarity upon Trip
		2	Set Channel D High Side to High Impedance upon Trip
21:20 (R/W)	TPOLCH	Channel C High Side Trip Polarity.	
		0	Set Channel C High Side to Inactive Polarity upon Trip
		1	Set Channel C High Side to Active Polarity upon Trip
		2	Set Channel C High Side to High Impedance upon Trip
19:18 (R/W)	TPOLBH	Channel B High Side Trip Polarity.	
		0	Set Channel B High Side to Inactive Polarity upon Trip
		1	Set Channel B High Side to Active Polarity upon Trip
		2	Set Channel B High Side to High Impedance upon Trip
17:16 (R/W)	TPOLAH	Channel A High Side Trip Polarity.	
		0	Set Channel A High Side to Inactive Polarity upon Trip
		1	Set Channel A High Side to Active Polarity upon Trip
		2	Set Channel A High Side to High Impedance upon Trip

Table 24-81: PWM_TRIP_POL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	TPOLDL	Channel D Low Side Trip Polarity.	
		0	Set Channel D Low Side to Inactive Polarity upon Trip
		1	Set Channel D Low Side to Active Polarity upon Trip
		2	Set Channel D Low Side to High Impedance upon Trip
5:4 (R/W)	TPOLCL	Channel C Low Side Trip Polarity.	
		0	Set Channel C Low Side to Inactive Polarity upon Trip
		1	Set Channel C Low Side to Active Polarity upon Trip
		2	Set Channel C Low Side to High Impedance upon Trip
3:2 (R/W)	TPOLBL	Channel B Low Side Trip Polarity.	
		0	Set Channel B Low Side to Inactive Polarity upon Trip
		1	Set Channel B Low Side to Active Polarity upon Trip
		2	Set Channel B Low Side to High Impedance upon Trip
1:0 (R/W)	TPOLAL	Channel A Low Side Trip Polarity.	
		0	Set Channel A Low Side to Inactive Polarity upon Trip
		1	Set Channel A Low Side to Active Polarity upon Trip
		2	Set Channel A Low Side to High Impedance upon Trip

25 Universal Asynchronous Receiver/Transmitter (UART)

The UART module is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word lengths, stop bits, bit rates, and parity-generation options. Multiple events can generate interrupts.

The UART is logically compliant to EIA-232E, EIA-422, EIA-485 and LIN standards, but usually requires external transceiver devices to meet electrical requirements. In IrDA (Infrared Data Association) mode, the UART meets the half-duplex IrDA SIR (9.6/115.2 Kbps rate) protocol. In multi-drop bus mode, the UART meets the full-duplex MDB/ICP v2.0 protocol.

The UART module supports partial modem status and control functionality to allow for hardware flow control.

The UART is a DMA-capable peripheral with separate transmit and receive DMA master channels. The use of DMA requires minimal software intervention as the DMA engine moves the data. The UART can also use a programmed core mode of operation. The core mode requires software management of the data flow using either interrupts or polling.

The UART can use one of the peripheral timers for a hardware-assisted auto-baud detection mechanism. The timers are external to the UART.

UART Features

Each UART includes the following features.

- 5–8 data bits
- Programmable extra stop bit and programmable extra half-stop bit
- Even, odd, and sticky parity bit options
- Extra 8-stage receive FIFO with programmable threshold interrupt request
- Flexible transmit and receive interrupt request timing
- A status interrupt output

- Independent DMA operation for receive and transmit
- Programmable automatic request to send (RTS)/clear to send (CTS) hardware flow control
- False start bit detection
- SIR IrDA operation mode
- MDB/ICP v2.0 operation mode
- Internal loopback
- Improved bit rate granularity
- LIN break command/Inter-frame gap transmission support

Table 25-1: UART Specifications

Feature	Availability
Protocol	
Master-Capable	Yes
Slave-Capable	Yes
Transmission Simplex	Yes
Transmission Half-Duplex	Yes
Transmission Full-Duplex	Yes
Access Type	
Data Buffer	Yes
Core Data Access	Yes
DMA Data Access	Yes
DMA Channels	2 (per UART Port)
DMA Descriptor	Yes
Boot Capable	Yes (Slave Mode)
Local Memory	No
Clock Operation	SCLK/16

UART Functional Description

The following sections provide details on the UARTs functionality.

CM41X_M4 UART Register List

The Universal Asynchronous Receiver/Transmitter module (UART) is a full-duplex peripheral compatible with PC-style industry-standard UARTs. The UART converts data between serial and parallel formats. The serial communication follows an asynchronous protocol that supports various word length, stop bit, parity, and interrupt generation

options. A set of registers governs UART operations. For more information on UART functionality, see the UART register descriptions.

Table 25-2: CM41X_M4 UART Register List

Name	Description
UART_CLK	Clock Rate Register
UART_CTL	Control Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_RBR	Receive Buffer Register
UART_RSR	Receive Shift Register
UART_RXCNT	Receive Counter Register
UART_SCR	Scratch Register
UART_STAT	Status Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_THR	Transmit Hold Register
UART_TSR	Transmit Shift Register
UART_TXCNT	Transmit Counter Register

CM41X_M4 UART Interrupt List

Table 25-3: CM41X_M4 UART Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
139	UART1_STAT	UART1 Status	Level	
140	UART2_STAT	UART2 Status	Level	
141	UART3_STAT	UART3 Status	Level	
142	UART4_STAT	UART4 Status	Level	
143	UART0_STAT	UART0 Status	Level	

CM41X_M0 UART Interrupt List

Table 25-4: CM41X_M0 UART Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
26	UART0_STAT	UART0 Status	Level	

25.2.4 Trigger List

CM41X_M4 UART DMA Channel List

Table 25-5: CM41X_M4 UART DMA Channel List

DMA ID	DMA Channel Name	Description
DMA2	UART0_TXDMA	UART0 Transmit DMA
DMA3	UART0_RXDMA	UART0 Receive DMA
DMA4	UART1_TXDMA	UART1 Transmit DMA
DMA5	UART1_RXDMA	UART1 Receive DMA
DMA6	UART2_TXDMA	UART2 Transmit DMA
DMA7	UART2_RXDMA	UART2 Receive DMA
DMA8	UART3_TXDMA	UART3 Transmit DMA
DMA9	UART3_RXDMA	UART3 Receive DMA
DMA10	UART4_TXDMA	UART4 Transmit DMA
DMA11	UART4_RXDMA	UART4 Receive DMA

UART Block Diagram

The ***UART Block Diagram*** figure shows a simplified block diagram of one UART module and how it interconnects to the processor system.

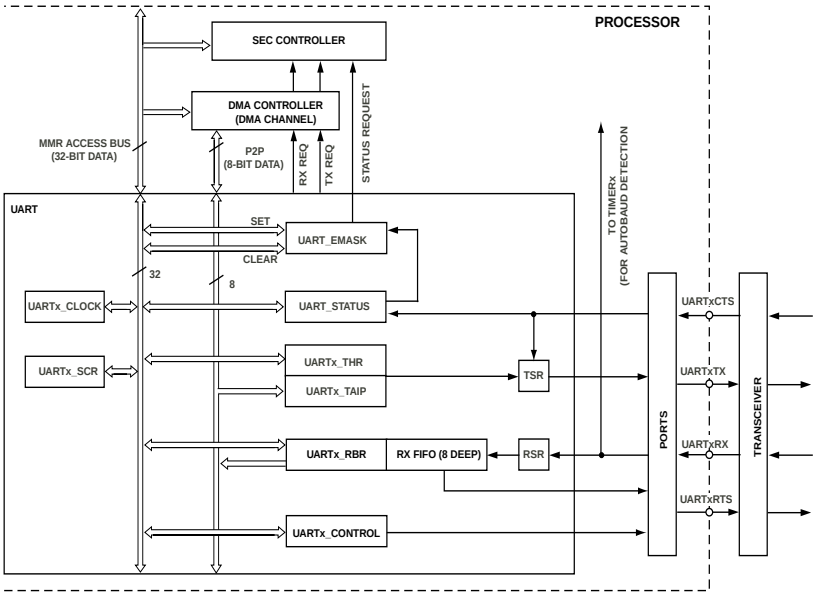


Figure 25-1: UART Block Diagram

UART Architectural Concepts

The following sections provide information about the UART architecture.

Internal Interface

The UART is a DMA-capable peripheral with support for separate transmit and receive DMA master channels. It operates in either DMA or programmed core modes. The core mode requires software management of the data flow using either interrupts or polling. The DMA method requires minimal software intervention, as the DMA engine itself moves the data. The `UART_RBR` and `UART_THR` registers also connect to one of the peripheral DMA buses.

All UART registers are 32 bits wide and the registers connect to the peripheral MMR bus. Not all MMRs can be used and unused bits are zero-filled. The UART has three interrupt outputs described as follows.

- The transmit and receive request outputs can function as DMA requests and connect to the DMA controller. Therefore, if the DMA is not enabled, the DMA controller simply forwards the request to the system event controller (SEC).
- The status interrupt output connects directly to one or more processor cores and event controllers. On many processors, the alternative capture input (`TIMER_ACI[n]`) of one of the GP timers also senses the `UART_RX` pin. When configured in capture mode, the processor can then use the GP timer to detect the bit rate of the received signal.

External Interface

Each UART features a `UART_RX` (receive) pin and a `UART_TX` (transmit) pin available through the general-purpose ports. These two pins usually connect to an external transceiver device that meets the electrical requirements of full-duplex or half-duplex standards. For example, EIA-232, EIA-422, 4-wire EIA-485 for full-duplex or 2-wire EIA-485, LIN for half-duplex. Additionally, the UART features a pair of clear-to-send, input pins (`UART_CTS`),

and request-to-send, output pins (UART_RTS) for hardware flow control. UART signals are multiplexed with other functions at the pin level.

Hardware Flow Control

To prevent the UART transmitter from sending data while the receiving counterpart is not ready, the UART features a UART_RTS/UART_CTS hardware flow control mechanism. The UART_RTS signal is an output that connects to the UART_CTS input of the communication partner. If data transfer is bidirectional, the figure shows the ***UART Hardware Flow Control*** handshake.

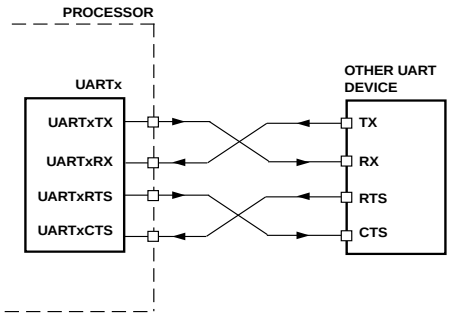


Figure 25-2: UART Hardware Flow Control

In both DMA and core mode, the receiver can deassert the UART_RTS signal to indicate that its receive buffer is almost full. Continued data transfers can cause an overrun error. The transmitter pauses when the UART_CTS input is in a deasserted state. In this state, the transmitter completes transmission of the data currently held in the transmit shift register (UART_TSR) but it does not continue with the data in the transmit hold register (UART_THR). If the UART_CTS pin is asserted again, the transmitter resumes and loads the content of UART_THR register into the UART_TSR register.

Bit Rate Generation

The peripheral clock (SCLK) and the 16-bit divisor in the UART_CLK register characterize the sample clock. The UART uses the UART_CTL.EN bit to enable the clock. By default, every serial bit is oversampled 16 times. The bit clock is 1/16th of the sample clock. If not in IrDA mode, the bit clock can equal the sample clock if the UART_CLK.EDBO bit is set, so that the following equation applies:

$$\text{Bit Rate} = \text{SCLK}/16^{(1-\text{EDBO})} \times \text{Divisor}$$

ADSP-CM41x Processor Example

The following table provides example divide factors required to support standard baud rates at a SCLK of 100 MHz.

Table 25-6: Uart Bit Rate Examples with 100 MHz SCLK

Bit Rate (bits/sec)	D Factor = 16			D Factor = 1		
	DL	Actual	% Error	DL	Actual	% Error
2400	2604	2400.15	0.006	41667	2399.98	0.001

Table 25-6: Uart Bit Rate Examples with 100 MHz SCLK (Continued)

Bit Rate (bits/ sec)	D Factor = 16			D Factor = 1		
	DL	Actual	% Error	DL	Actual	% Error
4800	1302	4800.31	0.006	20833	4800.08	0.002
9600	651	9600.61	0.006	10417	9599.69	0.003
19200	326	19171.78	0.147	5208	19201.23	0.006
38400	163	38343.56	0.147	2604	38402.46	0.006
57600	109	57339.45	0.452	1736	57603.69	0.006
115200	54	115740.74	0.469	868	115207.37	0.006
921600	7	892857.14	3.119	109	917431.19	0.452
1500000	4	1562500	4.167	67	1492537.31	0.498
3000000	2	3125000	4.167	33	3030303.03	1.01
6250000	1	6250000	0	16	6250000	0

NOTE: Careful selection of SCLK frequencies—that is, even multiples of desired bit rates— can result in lower error percentages. Setting the bit clock equal to the sample clock (`UART_CLK.EDBO` =1) improves bit rate granularity and enables the bit clock to more closely match the bit rate of the communication partner. The disadvantage to this configuration is that the power dissipation is higher and the sample points may not be as accurate. Therefore, it is recommended to use `UART_CLK.EDBO` =1 mode only when bit rate accuracy is not acceptable in `UART_CLK.EDBO` =0 mode. The `UART_CLK.EDBO` =1 mode is not intended to increase operation speed beyond the electrical limitations of the UART transfer protocol.

Autobaud Detection

At the chip level, the `UART_RX` pin is typically routed to an alternate capture input (`TIMER_ACI[n]`) of a general-purpose timer. When working in width capture mode, the processor uses this general-purpose timer to detect the bit rate applied to the `UART_RX` pin automatically by an external device. It often uses the capture capabilities of the timer to supervise the bit rate at run time. If the UART communicates with any device supplied by a weak clock oscillator that drifts over time, the processor can then readjust its UART bit rate dynamically, as required.

Often, the processor uses autobaud detection for initial bit rate negotiations where it is most likely a slave device waiting for the host to send a predefined autobaud character. This situation is common for UART booting. Do not enable the `UART_CTL.EN` bit while autobaud detection is in-process, to prevent the UART from starting a receive operation with incorrect bit rate matching. Alternatively, set the `UART_CTL.LOOP_EN` bit to disconnect the UART from its `UART_RX` pin.

A software routine can detect the pulse widths of serial stream bit cells. The sample base of the timer is synchronous with the UART operation (all derived from the same SCLK). The UART uses pulse widths to calculate the bit rate divider as follows:

Divisor = `TIMER_TMR[n]_WID`/$16^{(1-EDBO)}$ × Number of captured UART bits

To increase the number of timer counts and the resolution of the captured signal, do not measure just the pulse width of a single bit. Instead, enlarge the pulse of interest over more bits. Traditionally, a NULL character (ASCII 0x00) is used in autobaud detection, as shown in the ***Autobaud Detection*** figure.

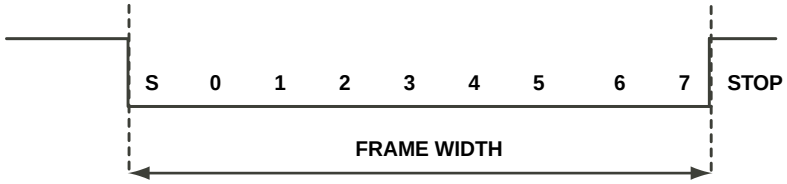


Figure 25-3: Autobaud Detection

Because the example frame encloses 8 data bits and 1 start bit, apply the following formula:

Divisor = `TIMER_TMR[n]_WID`/$16^{(1-EDBO)}$ × 9

NOTE: For processor-specific mapping of timer alternate capture inputs to the UARTs of the processor, see "Width Capture (WIDCAP) Mode" in the "*General-Purpose Timer (TIMER)*" chapter.

Real receive signals often have asymmetrical falling and rising edges, and the sampling logic level is not exactly in the middle of the signal voltage range. At higher bit rates, such pulse-width-based autobaud detection does not always return adequate results without extra conditioning of the analog signal. Measure signal periods to work around this issue.

For example, predefine the ASCII character “@” (0x40) as the autobaud detection character and measure the period between two subsequent falling edges. As shown in the ***Autobaud Detection Character 0x40*** figure, measure the period between the falling edge of the start bit and the falling edge after bit 6. Since this period encloses 8 bits, apply the following formula:

Divisor = `TIMER_TMR[n]_PER`/$16^{(1-EDBO)}$ × 8

or:

- Divisor = `TIMER_TMR[n]_PER`>> 7, if `UART_CLK.EDBO`=0
- Divisor = `TIMER_TMR[n]_PER`>> 3, if `UART_CLK.EDBO`=1

The ***Autobaud Detection Character 0x40*** figure shows the ASCII “@” (0x40) detection character.

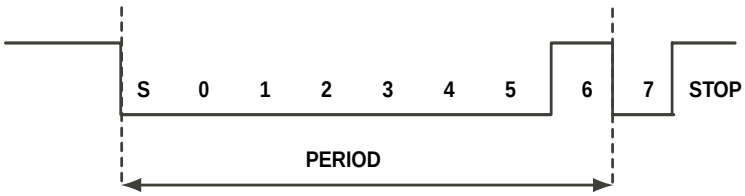


Figure 25-4: Autobaud Detection Character 0x40

For each UART unit, a general-purpose timer can be used to provide a hardware-assisted auto-baud detection mechanism for use with the UART. The ***Autobaud Timer Instances*** table shows these connections.

Table 25-7: Autobaud Timer Instances

UART Instance	TIMER Instance
UART0	TIMER0 TMRx
UART1	TIMER1 TMRx
UART2	TIMER1 TMRx
UART3	TIMER1 TMRx
UART4	TIMER1 TMRx

UART Debug Features

The UART can automatically calculate and transmit a parity bit. The ***UART Parity*** table summarizes parity behavior assuming 8-bit data words (`UART_CTL.WLS`=b#11).

Table 25-8: UART Parity

PEN	STP	EPS	Data (hex)	Data (binary, LSB first)	Parity
0	x	x	x	x	None
1	0	0	0x60	0000 0110	1
1	0	0	0x57	1110 1010	0
1	0	1	0x60	0000 0110	0
1	0	1	0x57	1110 1010	1
1	1	0	x	x	1
1	1	1	x	x	0

The two force error bits, `UART_CTL.FPE` and `UART_CTL.FFE`, are intended for test purposes. They are useful for debugging software, especially in loopback mode.

The UART can be set to internal loopback mode (`UART_CTL.LOOP_EN`=1). Loopback mode disconnects the input of the receiver from the receive pin and internally redirects the transmit output to the receiver. The transmit pin remains active and continues to transmit data externally as well. Loopback mode also forces the `UART_RTS` pin to deassert, disconnects the `UART_STAT.CTS` bit from the `UART_CTS` input pin, and connects the internal version of `UART_RTS` to the `UART_STAT.CTS` bit.

Additionally, the `UART_TX` pin can be forced to zero asynchronously using the `UART_CTL.SB` bit.

UART Operating Modes

The following sections describe the main operating modes of the UART.

- UART Mode
- IrDA SIR Mode

- Multi-Drop Bus Mode

UART Mode

The UART mode follows an asynchronous serial communication protocol with these options:

- 1 start bit
- 5–8 data bits
- Address bit (available in MDB mode only)
- None, even, odd or sticky parity
- 1, 1½, or 2 stop bits (1½ stop bits valid only in 5-bit word length)

The `UART_CTL` register controls the format of received and transmitted character frames. Data is always transmitted and received with the least significant bit (LSB) first.

The ***Bit Stream on a UART TX Pin Transmitting an "S" Character (0x53)*** figure shows a typical physical bit stream measured on a `UART_TX` pin.

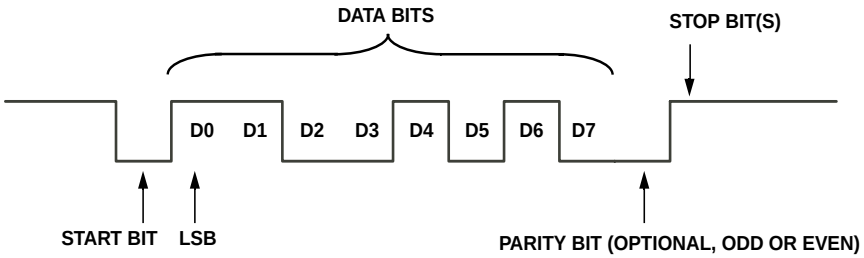


Figure 25-5: Bit Stream on a UART TX Pin Transmitting an "S" Character (0x53)

IrDA SIR Mode

The UART also supports serial data communication by way of infrared signals, according to the recommendations of the Infrared Data Association (IrDA). The physical layer known as IrDA SIR (9.6/115.2 Kbps rate) is based on return-to-zero-inverted (RZI) modulation. The UART does not support pulse position modulation.

Using the 16x data rate clock, RZI modulation is achieved by inverting and modulating the non-return-to-zero (NRZ) code normally transmitted by the UART. On the receive side, the UART uses a 16x clock to determine an IrDA pulse sample window, from which it recovers the RZI modulated NRZ code.

NOTE: The `UART_CLK.EDBO` bit is not valid in IrDA mode. Clear (=0) this bit in this mode.

Multi-Drop Bus Mode

The UART uses a protocol for point-to-point connections as well as in networks where the EIA-485 standard is representative of UART-based bus systems. The EIA-232 standard defines point-to-point connections. In such networks, node addressing is important.

In a multidrop bus (MDB) network, for example, an address bit enhances the UART frame. The address bit is inserted between the data bits and the optional parity bit. To configure the UART for MDB mode, set the mode of operation bits (UART_CTL.MOD [5:4]) to 01.

By convention, the address bit is transmitted low for regular data bytes. When set, it marks special address bytes that require the attention of all nodes on the network.

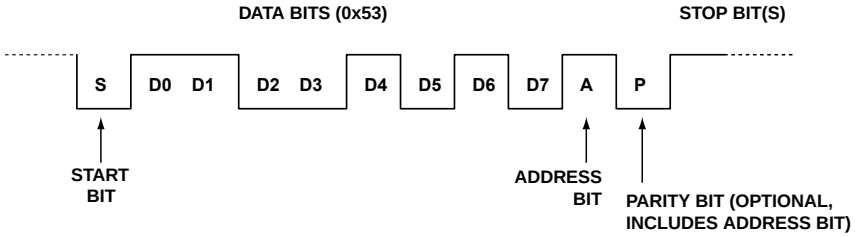


Figure 25-6: UART Frame with Address Bit

All transmit operations are processed through the transmit buffer register (UART_THR), so all DMA data transmissions clear the address bit. If data is written to the transmit address or insert pulse register (UART_TAIP) instead, the same transmit operation is initiated with the only exception that the address bit is sent high.

The UART uses the UART_STAT.ADDR bit of the receiver to signal whether the previously received frame had the address bit set or not. Hardware updates it every time a new frame is received. When the enable address word interrupt bit (UART_IMSK.EAWI) is set, the reception of an address byte triggers a special status interrupt request.

The address sticky bit (UART_STAT.ASTKY) is the sticky version of the UART_STAT.ADDR bit. Hardware sets it whenever the UART_STAT.ADDR bit is set. Software can clear the UART_STAT.ASTKY bit with a W1C operation.

In MDB mode, only address bytes progress to the receive FIFO by default. Data bytes are gated unless the UART_STAT.ASTKY bit is set. The receiver ignores all traffic on the UART bus. This way, the processor can go into low-power mode and interrupt activity does not load the processor every time a frame is transmitted on the UART bus. If, however, an address frame is transmitted, the receiver immediately samples all further traffic. A software routine can analyze the received data, decide whether it was of relevance for the local network node, and W1C the UART_STAT.ASTKY bit if it was not.

Software can overrule of the hardware address frame detection by setting the UART_STAT.ADDR bit and (indirectly) the UART_STAT.ASTKY bit with a W1S operation.

The MDB mode follows an asynchronous serial communication protocol with the following options.

- 1 start bit
- 5–8 data bits
- Address bit
- None, even, odd or sticky parity
- 1, 1½, or 2 stop bits (1½ stop bits are valid only in 5-bit word length)

NOTE: If the address bit and parity bit are both enabled, the parity check and generation includes the address bit in its parity calculation.

UART Data Transfer Modes

The UART can transfer data using both the core and DMA. Receive and transmit paths operate independently except that the bit rate and the frame format are identical for both transfer directions. Transmit and receive channels are both buffered. The UART_THR register buffers the transmit shift register (UART_TSR) and the UART_RBR register buffers the receive shift register (UART_RSR).

UART Mode Transmit Operation (Core)

In core mode, the processor core moves data to and from the UART. A write to the UART_THR register initiates the transmit operation. If no former operation is pending, the UART_THR register passes the data immediately to the UART_TSR register. There, it is shifted out at the bit rate characterized by the UART_CLK register, with start, stop, and parity bits appended as defined by the UART_CTL register.

The UART_THR register and the UART_TSR register can be modeled as a two-stage transmit buffer. The least significant bit (LSB) always transmits first. This bit is bit 0 of the value written to the UART_THR register.

UART Mode LIN Break Command

Some UART-based protocols demand synchronization methods that are not native to standard UART implementations. For example, the Local Interconnect Network (LIN) protocol requires a low-pulse of well-defined transmit length as a prologue to every multi-byte message. Its length must be at least 13 bit-times.

With previous UARTs, there were two options to implement this protocol:

- A null byte is transmitted with a temporarily lowered bit rate, or
- A software counter generates the period and the asynchronous set break (SB) mechanisms pull the transmit pin low

Since both methods have their disadvantages, the newer UART introduces a new inter-frame gap technique.

The feature is not available in MDB or IrDA operating modes. However, in standard UART mode (bits UART_CTL.MOD [5:4] =00), a write to the UART_TAIP register initiates the transmission of an inter-frame pulse. If the transmit buffer is not empty, the UART first transmits all bytes in the queue. It only initiates with pulse generation after the last stop bit of the last byte has been shifted out.

The value written into the UART_TAIP register defines the nature and the duration of the transmitted pulse. Bits [6:0] control the duration in bit-times and bit [7] controls the value (duration = UART_TAIP[6:0] / UART_CLK[15:0]). If UART_TAIP[7] is set, and an active high pulse is issued, the number of stop bits is extended. If UART_TAIP[7] is cleared, a low pulse is generated. Invert the polarity using the UART_CTL.FCPOL bit. Writing a value of 13 into the UART_TAIP register generates the break command as required by the LIN protocol.

NOTE: If the UART_CTL.TPOLC bit is enabled, an inverted most-significant bit can be transmitted.

NOTE: If another transmission is pending (in the UART_TSR register), the UART_TAIP initiated pulse is queued until after all pending operations have finished and all stop bits are transmitted.

The transmission of break command/inter-frame gap precedes transmission of the number of stop bits as set in the UART_CTL.STB and UART_CTL.STBH bit fields.

The UART receiver can detect break commands through the break indicator (UART_STAT.BI) flag. This flag reports that an entire UART frame has been received in low state. It does not report whether the duration of the received low pulse was exact or at least 13 bit-times as LIN masters transmit. Typically, the break indicator meets LIN requirements. The processor can use GP timers to determine the pulse width more precisely, if necessary.

Each UART_RX pin is also routed to a GP timer through its alternate capture input (TACI). This functionality is not only useful for bit rate detection (*autobaud*) but also helps to measure the pulse widths precisely on the UART_RX input. Additionally, the GP timers can issue an interrupt request or a fault condition when the received pulse width is shorter than a bit time or longer than the worst-case break condition. The windowed watchdog width mode of the GP timers controls this functionality.

UART Mode Receive Operation (Core)

The receive operation uses the same data format as the transmit configuration except that one valid stop bit is always sufficient. The UART_CTL.STB and UART_CTL.STBH bits have no impact on the receiver.

The UART receiver senses the falling edges of the receive input. When it detects an edge, the receiver starts sampling the input according to settings in the UART_CLK register. The receiver samples the start bit (majority sampling) close to its midpoint. If sampled low, it assumes a valid start condition. Otherwise, it discards the detected falling edge.

After detection of the start bit, the received word is shifted into the UART_RSR register.

After the corresponding stop bit is received, the content of the UART_RSR register is transferred to the 8-deep receive FIFO and is accessible by reading the UART_RBR register.

The receive FIFOs and the UART_RBR register act as a 9-stage receive buffer. If the stop bit of the ninth word is received before software reads the UART_RBR register, an overrun error is reported. Overruns protect data in the UART_RBR register and the receive FIFO from being overwritten by further data until the software clears the UART_STAT.OE bit. However, the data in the UART_RSR register is immediately destroyed as soon as the overrun occurs.

The sampling clock is 16 times faster than the bit clock. The receiver oversamples every bit 16 times and makes a majority-decision based on the middle three samples. This functionality improves immunity against noise and hazards on the line. The receiver disregards spurious pulses of less than two times the sampling clock period.

Normally, the receiver samples every incoming bit at exactly the 7th, 8th and 9th sample clock. If, however, the UART_CLK.EDBO bit is set to 1, the receiver samples bits roughly at 7/16th, 8/16th, and 9/16th of their period. This configuration achieves better bit rate granularity and accuracy as required at high operation speeds. Hardware design must ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

Reception starts when the UART receiver detects a falling edge on the UART_RX input pin. The receiver attempts to see a start bit. The data is shifted into the UART_RSR register. After the ninth sample of the first, the receiver processes the stop bit and copies the received data to the 8-stage receive FIFO. The UART_RSR recovers for further data reception.

The receiver samples data bits close to their midpoint. Because the receiver clock is typically asynchronous to the data rate of the transmitter, the sampling point can drift relative to the center of the data bits. The sampling point is synchronized again with each start bit, so the error accumulates only over the length of a single word. The polarity of received data is selectable, using the UART_CTL.RPOLC bit.

NOTE: The receiver checks for only a single stop bit. After the third sample of the first stop bit has been received (at time 9/16th of the stop bit duration), the receiver immediately acts (status update). It then prepares for new falling edge detection (start detection).

IrDA Transmit Operation

To generate the IrDA pulse transmitted by the UART, the normal NRZ output of the transmitter is first inverted if the UART_CTL.TPOLC bit is configured for active-low operation. In this configuration, a zero is transmitted as a high pulse of 16 UART clock periods and a one is transmitted as a low pulse for 16 UART clock periods. Then, six UART clock periods delay the leading edge of the pulse. Similarly, eight UART clock periods truncate the trailing edge of the pulse. For a 16-cycle UART clock period, this operation results in the final representation of the original zero as a high pulse of only 3/16 clock periods. The ***IrDA Transmit Pulse*** figure shows how the pulse is centered around the middle of the bit time. The final IrDA pulse is fed to the off-chip infrared driver.

This modulation approach ensures a pulse width output from the UART of three cycles high out of every 16 UART clock cycles. As shown in the ***IrDA Transmit Pulse*** figure, the error terms associated with the bit rate generator are small and well within the tolerance of most infrared transceiver specifications.

NOTE: In IrDA mode, writes to the UART_TAIP register are equivalent to writes to the UART_THR register.

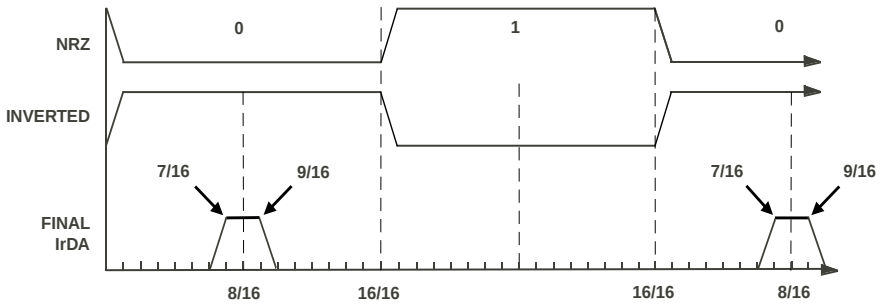


Figure 25-7: IrDA Transmit Pulse

IrDA Receive Operation

The IrDA receiver function is more complex than the transmit function. The receiver must discriminate the IrDA pulse and reject noise. The receiver looks for the IrDA pulse in a narrow window centered around the middle of the expected pulse.

Glitch filtering is accomplished by counting 16 system clocks from the time the receiver detects an initial pulse. If the pulse is absent when the counter expires, the receiver interprets it as a glitch. Otherwise, the receiver interprets it as a zero. This assessment is acceptable because glitches originating from on-chip capacitive cross-coupling typically do not last for more than a fraction of the system clock (SCLK) period. Appropriate shielding avoids sources outside of the chip and not part of the transmitter. The only other source of a glitch is the transmitter itself. The processor relies on the transmitter to perform within specification. If the transmitter violates the specification, unpredictable results can occur. The 4-bit counter adds an extra level of protection at a minimal cost.

NOTE: Because SCLK can change across systems, the longest glitch tolerated is inversely proportional to the SCLK frequency.

A counter that is clocked at the 16x bit-time sample clock determines the receive sampling window. The sampling window is resynchronized with each start bit by centering the sampling window around the start bit.

The polarity of receive data is selectable, using the `UART_CTL.RPOLC` bit. The ***IrDA Receiver Pulse Detection*** figure provides examples of each polarity type.

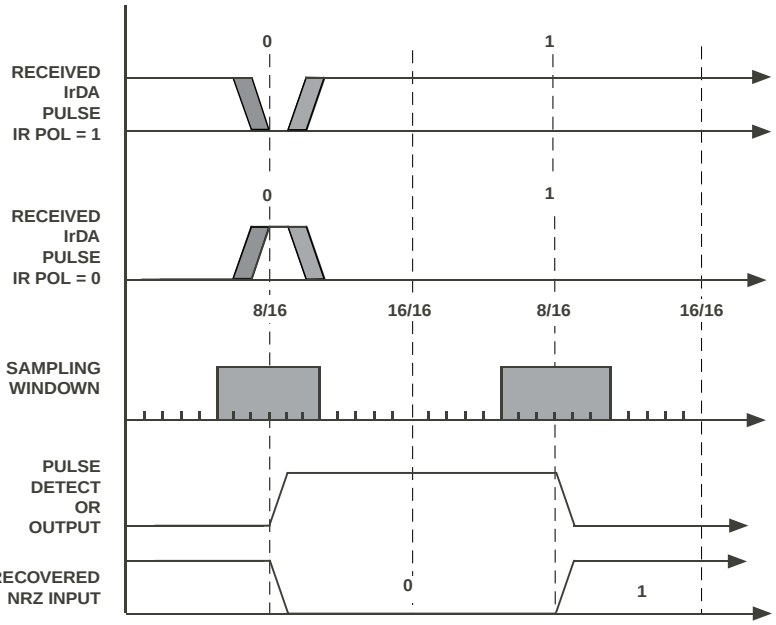


Figure 25-8: IrDA Receiver Pulse Detection

MDB Transmit Operation

In MDB mode, receive and transmit paths operate independently from each other, except for sharing bit rate and frame formats for both transfer directions.

Transmit operation is initiated by writing the `UART_THR` or `UART_TAIP` registers. A write to the `UART_THR` register transmits the written word with the appending address bit set low. A write to the `UART_TAIP` register transmits the written word with the appended address bit set high. The data is moved into the `UART_TSR` register, where it is shifted out at the bit rate programmed by the `UART_CLK` register, with start, stop, address, and parity bits appended, as required.

If DMA is enabled, the DMA engine always writes the data into the UART_THR register, and the written word is transmitted with the appending address bit set low.

The polarity of transmit data is selectable, using the UART_CTL.TPOLC bit.

MDB Receive Operation

Receive operations use the same data format as the transmit configuration, except that the number of stop bits is always assumed to be 1. After detection of the start bit, the received word is shifted into the UART_RSR register at the programmed bit.

Normally, the receiver samples every incoming bit at exactly the 7th, 8th and 9th sample clock. If, however, the UART_CLK.EDBO bit is set, the receiver samples the bits roughly at 7/16th, 8/16th, and 9/16th of their period. This configuration achieves better bit rate granularity and accuracy needed at high operation speeds. Hardware design must ensure that the incoming signal is stable between 6/16th and 10/16th of the nominal bit period.

After the appropriate number of bits (including address, parity, and stop bits) is received, the UART_RSR register is transferred to the receive FIFO and accessible through the UART_RBR register.

The polarity of receive data is selectable, using the UART_CTL.RPOLC bit.

DMA Mode

In DMA mode, separate receive and transmit DMA channels move data between the UART and memory. The software does not have to move data; it just has to set up the appropriate transfers either through the descriptor mechanism or through autobuffer mode.

DMA channels provide a 4-deep FIFO, resulting in total buffer capabilities of 6 words at the transmit side and 9 words at the receive side. In DMA mode, the bus activity and arbitration mechanism determine the latency. The processor loading and interrupt priorities do not determine the latency.

To enable UART DMA, first set up the system DMA control registers. Then, enable the UART_IMSK.ERBFI or UART_IMSK.ETBEI interrupts. This sequence is necessary because these interrupt request lines double as DMA request lines. With DMA enabled, once these requests are received, the DMA control unit generates a direct memory access. If DMA is not enabled, the UART interrupt is passed on to the system interrupt handling unit. The status interrupt for the UART goes directly to one or more processor cores and event controllers, bypassing the DMA unit completely.

For transmit DMA, programs must set the DMA_CFG.SYNC bit. With this bit set, interrupt generation is delayed until the entire DMA FIFO is drained to the UART module. The UART transmit DMA interrupt service routine can disable the DMA or to clear the UART_IMSK.ETBEI control bit only when the DMA_CFG.SYNC bit is set. Otherwise, up to four data bytes can be lost.

When the UART_IMSK.ETBEI bit is set, an initial transmit DMA request is issued immediately. The program then clears the UART_IMSK.ETBEI bit through the DMA service routine.

In DMA transmit mode, the UART_IMSK.ETBEI bit enables the peripheral request to the DMA FIFO. The DMA_CFG.EN bit enables the strobe on the memory side. If the DMA count is less than the DMA FIFO depth,

which is 4, then the DMA interrupt can be requested before the UART_IMSK.ETBEI bit is set. If this behavior is unwanted, set the DMA_CFG.SYNC bit.

Regardless of the DMA_CFG.SYNC setting, the DMA stream has not left the UART transmitter completely at the time the interrupt request is generated. Transmission can abort in the middle of the stream, causing data loss, when the UART clock was disabled without extra synchronization with the UART_STAT.TEMT bit.

The UART provides functionality to avoid resource-consuming polling of the UART_STAT.TEMT bit. The UART_IMSK_SET.EDTPTI bit enables the UART_STAT.TEMT bit to trigger a DMA interrupt. To delay the DMA completion interrupt until the last data word of a STOP DMA has left the UART, keep the DMA_CFG.INT bit cleared and set the UART_IMSK_SET.EDTPTI bit instead. Then, the normal DMA completion interrupt is suppressed. Later, the UART_STAT.TEMT event triggers a DMA interrupt after the last word of the DMA has left the UART transmit buffers. If DMA_CFG.INT and UART_IMSK.EDTPTI are set, when finishing STOP mode, the DMA requests two interrupts.

The DMA of the UART module supports 8-bit and 16-bit operation, but not 32-bit operation. It does not support sign-extension.

Mixing DMA and Core Modes

Switching from DMA mode to core operation dynamically requires some consideration, especially for transmit operations. By default, the interrupt timing of the DMA is synchronized with the memory side of the DMA FIFOs. Normally, the transmit DMA completion interrupt is generated after the last byte is copied from the memory into the DMA FIFO. The transmit DMA interrupt service routine is not yet permitted to disable the DMA_CFG.EN bit. The interrupt is requested when the DMA_STAT.IRQDONE bit is set. The DMA_STAT.RUN bit, however, remains set until the data has completely left the transmit DMA FIFO.

When planning to switch from a DMA to core mode, set the DMA_CFG.SYNC bit in the word of the last descriptor or work unit before handing over control. Then, after the interrupt request occurs, software can write new data into the UART_THR register as soon as the UART_STAT.THRE bit permits. If the DMA_CFG.SYNC bit cannot be set, software can poll the DMA_STAT.RUN bit instead. Alternatively, using the UART_IMSK.EDTPTI bit can avoid expensive status bit polling.

When switching from core to DMA operation, ensure that the first DMA request is issued properly. If the DMA is enabled while the UART is still transmitting, no precaution is required. If, however, the DMA is enabled after the UART_STAT.TEMT bit is high, pulse the UART_IMSK.ETBEI bit to initiate DMA transmission.

Setting Up Hardware Flow Control

The following steps show how to set up UART hardware flow control:

1. Configure automatic or manual hardware flow control for the receiver through the UART_CTL.ARTS bit, or the transmitter through the UART_CTL.ACTS bit.
2. Configure $\overline{\text{UART\_CTS}}$ and $\overline{\text{UART\_RTS}}$ polarity through the UART_CTL.FCPOL bit.

On reset, when the UART is not yet enabled and the port multiplexing has not been programmed, the $\overline{\text{UART\_RTS}}$ pin is not driven. Some applications require a resistor pull the $\overline{\text{UART\_RTS}}$ signal to either state during reset.

UART Event Control

Status flags in the UART_STAT register are available to signal data reception, parity, and error conditions, if necessary.

DMA and Interrupt Multiplexing

See the *Direct Memory Access (DMA)* chapter on for information on DMA multiplexing. Several interrupts and DMA channels in thr UART can be multiplexed.

NOTE: To operate in interrupt mode without using DMA channels, set the UART_IMSK.ELSI bit. This configuration redirects receive and transmit requests to the status interrupt output. The status interrupt goes directly to the system without going through the DMA controller.

Interrupt Masks

Each UART features a set of interrupt mask registers: UART_IMSK, UART_IMSK_SET, and UART_IMSK_CLR. The UART_IMSK register supports read/write operations. Writing ones to the UART_IMSK_SET register enables interrupts, writing ones to the UART_IMSK_CLR register disables them. Reads from either register return the enabled bits. This way, different interrupt service routines can control transmit, receive, and status interrupt requests independently and easily.

The UART module uses the UART_IMSK registers to enable requests for system handling of empty or full states of data registers. Unless polling is used as a means of action, the UART_IMSK.ERBFI and UART_IMSK.ETBEI bits in this register are normally set.

Each UART module has three interrupt outputs. It uses one for transmission, one for reception, and one for reporting status events. The UART module routes transmit and receive requests through the DMA controller. The status request goes directly to the system for event processing.

If the associated DMA channel is enabled, the request functions as a DMA request. If the DMA channel is disabled, it simply forwards the request to the SEC. A DMA channel must be associated with the UART module to enable transmit and receive interrupts. Otherwise, transmit and receive requests cannot be forwarded.

NOTE: To operate in interrupt mode without using DMA channels, set the UART_IMSK.ELSI bit. This configuration redirects receive and transmit requests to the status interrupt request output. The status interrupt goes directly to the system without going through the DMA controller.

Interrupt Servicing

Interrupt service routines (ISRs) perform UART writes and reads. Separate interrupt lines are provided for transmit, receive, and status. The UART_IMSK register group enables the independent interrupts individually. To enable UART transmit interrupts, set the UART_CTL.EN bit.

The ISRs evaluate the status bits in the UART_STAT register to determine the signaling interrupt source. The system event controller for the processor assigns and unmasks interrupts. The ISRs must clear the interrupt latches explicitly. To reduce interrupt frequency on the receive side in core mode, use the UART_IMSK.ERFCI status

interrupt as an alternative to the regular UART_IMSK.ERBFI receive interrupt. Hardware must ensure that at least two (if UART_CTL.RFIT=0) or four (if UART_CTL.RFIT=1) words are available in the receive buffer by the time the interrupt is requested.

Transmit Interrupts

The UART module uses the UART_IMSK_SET.ETBEI bit to enable transmit interrupt requests.

The UART_THR and UART_TAIP registers are the same physical register, and both affect the signaling of the UART_STAT.TEMT, UART_STAT.TFI, and UART_STAT.THRE bits similarly.

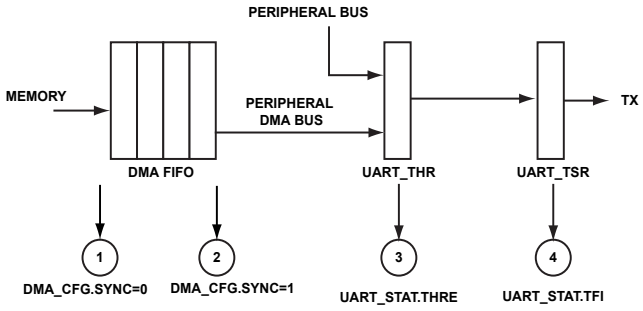


Figure 25-9: Transmit Interrupts

The UART module asserts the transmit request along with the UART_STAT.THRE bit, indicating that the transmit buffer is ready for new data. The UART_STAT.THRE bit resets to 1. When the UART_IMSK_SET.ETBEI bit is set, the UART module immediately issues an interrupt or DMA request. This way, no special handling of the first character is required when transmission of a string is initiated. Set the UART_IMSK_SET.ETBEI bit and let the interrupt service routine load the first character from memory and write it to the UART_THR register in the normal manner. ISRs can clear the UART_IMSK.ETBEI bit through the UART_IMSK_CLR register when the string transmission has completed.

Hardware clears the UART_STAT.THRE bit when new data is written to the UART_THR register. These write operations also clear the transmit interrupt request. However, they also initiate further transmission. If continued transmission is undesirable, the UART module can alternatively clear the transmit request through the UART_IMSK_CLR.ETBEI bit register. Transfers of data from the UART_THR register to the UART_TSR register reset this status flag in the UART_STAT register.

ISRs can interrogate the UART_STAT.TEMT bit to discover any ongoing transmission. The sticky counterpart of the UART_STAT.TEMT bit, UART_STAT.TFI, indicates when the transmit buffer has drained and can trigger a status interrupt. When data is pending in either one of these registers, the UART_STAT.TEMT flag is low. As soon as all data has left the UART_TSR register, the UART_STAT.TEMT bit goes high again and indicates that all pending transmit operations (including stop bits) have finished. Then, it is safe to disable the UART_CTL.EN bit or to three-state off-chip line drivers. Then, the UART module can generate an interrupt either through the status interrupt channel when the UART_IMSK.ETFI bit is set, or through the DMA controller when enabled by the UART_IMSK.EDTPTI bit.

When enabled by the UART_IMSK.ETBEI bit, the UART_STAT.THRE flag requests data along the peripheral command lines to the DMA controller (referred to as TXREQ). This signal is routed through the DMA controller. If the associated DMA channel is enabled, the TXREQ signal functions as a DMA request, otherwise the DMA controller simply forwards it to the system. Alternatively the UART_IMSK.ETXS bit can redirect the transmit interrupts to the UART status interrupt.

With interrupts disabled, the UART module can poll the status flags to determine when data is ready to move. Because polling is processor intensive, it is not typically used in real-time signal processing environments. Since read operations from UART_STAT registers have no side effects, different software threads can interrogate these registers without mutual impacts.

Receive Interrupts

The UART module uses the UART_IMSK_SET.ERBFI bit to enable receive interrupt requests. If set, the UART_STAT.DR flag requests an interrupt on the dedicated RXREQ output, indicating that new data is available in the UART_RBR register. This signal is routed through the DMA controller. If the associated DMA channel is enabled, the RXREQ signal functions as a DMA request; otherwise the DMA controller simply forwards it to the system. Alternatively, if no DMA channel is assigned to the UART, the UART_IMSK.ERXS bit can redirect the receive interrupts to the UART status interrupt. When software reads the UART_RBR register, hardware clears the UART_STAT.DR bit again, which, in turn, clears the receive interrupt request.

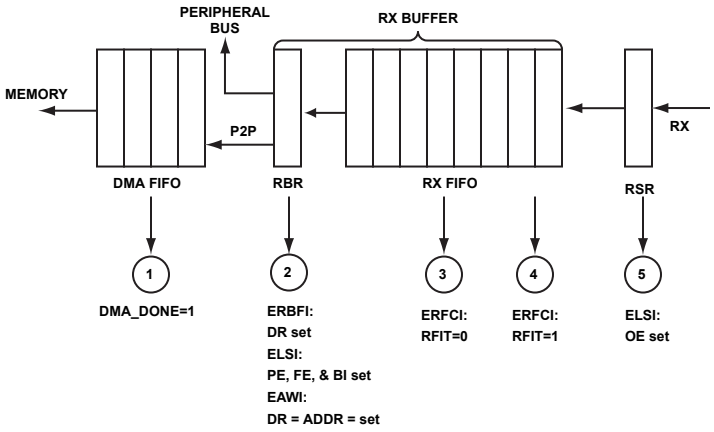


Figure 25-10: Receive Interrupts

Hardware updates the following:

- UART_STAT.DR bits
- UART_STAT.ADDR bits
- UART_STAT.ASTKY bits
- UART_STAT.PE bits
- UART_STAT.FE bits
- UART_STAT.BI bits

- UART_RBR register

The UART_STAT.OE bit is updated as soon as an overflow condition occurs (for example when a stop bit for a frame is received and the receive FIFO is full). When software does not read the UART_RBR register in time, the received data is protected from being overwritten by new data until software clears the UART_STAT.OE bit. Only the content of the UART_RSR register can be overwritten in the overrun case.

The UART module uses the UART_STAT.RFCS bit to monitor the state of the 8-deep receive FIFO. It uses the UART_CTL.RFIT bit to control the behavior of the buffer. If UART_CTL.RFIT is zero, the UART_STAT.RFCS bit is set when the receive buffer holds four or more words. If UART_CTL.RFIT is set, the UART_STAT.RFCS bit is set when the receive buffer holds seven or more words. Hardware clears the UART_STAT.RFCS bit when a core or DMA reads the UART_RBR register and when the buffer is flushed below the level of four (UART_CTL.RFIT=0) or seven (UART_CTL.RFIT=1). If the associated interrupt bit UART_IMSK.ERFCI is enabled, a status interrupt request is reported when the UART_STAT.RFCS bit is set.

If errors are detected during reception, an interrupt can be requested from the status interrupt output. This status interrupt request goes directly to the system. The bit enables status interrupt requests.

The controller detects the following error conditions, shown with their associated bits in the UART_STAT register.

- Overrun error (UART_STAT.OE bit)
- Parity error (UART_STAT.PE bit)
- Framing error or invalid stop bit (UART_STAT.FE bit)
- Break indicator (UART_STAT.BI bit)

Status Interrupts

The UART module uses status interrupt channels for the following purposes:

- Line status interrupt requests
- Flow control interrupt requests
- Receive FIFO threshold interrupt requests
- Transmission finished interrupt request

The UART module uses the UART_IMSK.ELSI bit to enable the line status interrupts. If set, the status interrupt request is asserted with one of the UART_STAT.BI, UART_STAT.FE, UART_STAT.PE, or UART_STAT.OE receive errors bits. A W1C operation to the UART_STAT register clears the error bits. Once all error conditions are cleared, the interrupt request deasserts.

The UART module uses the UART_IMSK_SET.ERFCI bit to enable the receive FIFO count interrupt. If set, a status interrupt request is generated when the UART_STAT.RFCS is active. The UART_STAT.RFCS bit indicates a receive buffer threshold level. If the UART_CTL.RFIT bit is cleared, software can safely read two words out of the UART_RBR register by the time the UART_STAT.RFCS interrupt occurs.

If the UART_CTL.RFIT bit is set, software can safely read four words. The interrupt request and the UART_STAT.RFCS bit are cleared when the UART_RBR is read enough of times, so that the receive buffer drains below the threshold of two (UART_CTL.RFIT=0) or four (UART_CTL.RFIT=1). Because in DMA mode a status service routine may not be permitted to read UART_RBR, this interrupt is only recommended in core mode. In DMA mode, use this functionality for error recovery only.

The UART module uses the UART_IMSK_SET.EDSSI bit to enable the flow control interrupts. If active, a status interrupt is generated when the sticky UART_STAT.SCTS bit register is set, indicating that the UART_CTS input of the transmitter been reasserted. A W1C operation to the UART_STAT.SCTS bit clears the interrupt request.

The UART module uses the UART_IMSK_SET.ETFI bit to enable the transmission finished interrupt. If active, a status interrupt request is asserted when the UART_STAT.TFI bit is set. The UART_STAT.TFI is the sticky version of the UART_STAT.TEMT bit, indicating that a byte that started transmission has finished. A W1C operation to the UART_STAT.TFI bit clears the interrupt request.

Multi-Drop Bus Events

Several status bits and interrupt features in the UART_STAT and UART_IMSK registers facilitate efficient data handling in multi-drop bus mode. These features include the address (UART_STAT.ADDR) bit, the address sticky (UART_STAT.ASTKY) bit and the enable address word interrupt (UART_IMSK.EAWI). One of the key features of the multi-drop bus protocol is its address bit. The address bit signifies to the slaves that the master is transmitting an address word (all read it) or a data word (only the addressed slave reads its). The UART hardware provides an efficient method of handling the situation described with the use of UART_STAT.ASTKY bit.

NOTE: The UART module uses the UART_STAT.ASTKY bit in multi-drop bus mode to indicate when an address operation for a peripheral is occurring. The UART_STAT.ASTKY bit is a sticky version of the UART_STAT.ADDR bit. Hardware sets the bit whenever the UART_STAT.ADDR bit is set. Only software clears it with a W1C operation. With the UART_STAT.ASTKY bit set, words are received irrespective of the mode bit or address bit setting. With the UART_STAT.ASTKY bit cleared, only address words (mode bit =1) are received and words with mode bit =0 are ignored in MDB mode. This bit does not affect reception in non-MDB modes. (Words with mode bit =0 are not moved from the UART_RSR register to the receive FIFO.)

UART Programming Model

The following sections provide basic procedures for configuring various UART operations.

Detecting Autobaud

To detect Autobaud:

1. Ensure that the timer is disabled.
2. Configure the following bits: UART_CTL.MOD =00, UART_CTL.LOOP_EN =1, UART_CTL.WLS =11 (8-bit data), and UART_CTL.EN =1

3. Configure the following bits: TIMER_TMR[n]_CFG.TMODE =1101, TIMER_TMR[n]_CFG.OUTDIS =1, TIMER_TMR[n]_CFG.IRQMODE =10 and enable the timer.
4. Send test data through the host device and wait for the timer interrupt and disable the timer.

   The bit rate can be derived from the timer period register value according to the formula provided in the Auto-baud Detection section.

Using Common Initialization Steps

When using the core or the DMA to execute transfers, the following steps are common to all UART modes.

1. All UART signals are multiplexed and compete with other functions at pin level. First, program the port registers according to the guidelines in the PORT chapter.
2. Program the UART_CLK register. Refer to Bit Rate Generation topic.
3. Program the UART_CTL register and enable the UART clock.

Using Core Transfers

Write data into the UART_THR register, when the UART_STAT.THRE bit is set, to initiate a core transmit operation. If the UART_STAT.DR bit is set, received data can be read from the UART_RBR register.

Using DMA Transfers

1. Make sure that the UART_IMSK.ETBEI or the UART_IMSK.ERBFI bits are cleared before configuring the DMA.
2. Configure the dedicated DMA channel.
3. Set the UART_IMSK.ETBEI or UART_IMSK.ERBFI bits to start the transfer.

Using Interrupts

Each UART features three interrupt signal outputs.

1. Enable individual interrupts in the appropriate processor or SEC fualt handler.
2. Register IRQ handlers.
3. Use the interrupts mask registers to enable specific IRQ events.

Setting Up Hardware Flow Control

1. Configure automatic or manual hardware flow control for the receiver through the UART_CTL.ARTS bit, or the transmitter through the UART_CTL.ACTS bit.
2. Configure $\overline{\text{UART\_CTS}}$ and $\overline{\text{UART\_RTS}}$ polarity through the UART_CTL.FCPOL bit.

CM41X_M4 UART Register Descriptions

UART (UART) contains the following registers.

Table 25-9: CM41X_M4 UART Register List

Name	Description
UART_CLK	Clock Rate Register
UART_CTL	Control Register
UART_IMSK	Interrupt Mask Register
UART_IMSK_CLR	Interrupt Mask Clear Register
UART_IMSK_SET	Interrupt Mask Set Register
UART_RBR	Receive Buffer Register
UART_RSR	Receive Shift Register
UART_RXCNT	Receive Counter Register
UART_SCR	Scratch Register
UART_STAT	Status Register
UART_TAIP	Transmit Address/Insert Pulse Register
UART_THR	Transmit Hold Register
UART_TSR	Transmit Shift Register
UART_TXCNT	Transmit Counter Register

Clock Rate Register

The UART_CLK register divides the system clock (SCLK) down to the bit clock.

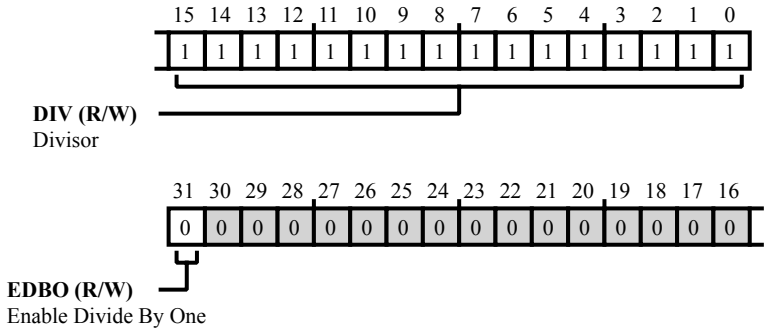


Figure 25-11: UART_CLK Register Diagram

Table 25-10: UART_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	EDBO	Enable Divide By One. The UART_CLK.EDBO bit enables the bypassing of the divide-by-16 prescaler in bit clock generation. This functionality improves bit rate granularity, especially at high bit rates. Do not set this bit in IrDA mode.	
		0	Bit clock prescaler = 16
		1	Bit clock prescaler = 1
15:0 (R/W)	DIV	Divisor. The UART_CLK.DIV provides the divisor for the UART's clock bit rate calculation. The bit rate is defined by: Bit Rate = SCLK / ($16^{(1-EDBo)}$ x UART_CLK.DIV)	

Control Register

The UART_CTL register provides enable and disable control for internal UART and for the IrDA mode of operation. This register also provides UART line control, permitting selection of the format of received and transmitted character frames. Modem feature control also is available from this register, including partial modem functionality to allow for hardware flow control and loopback mode.

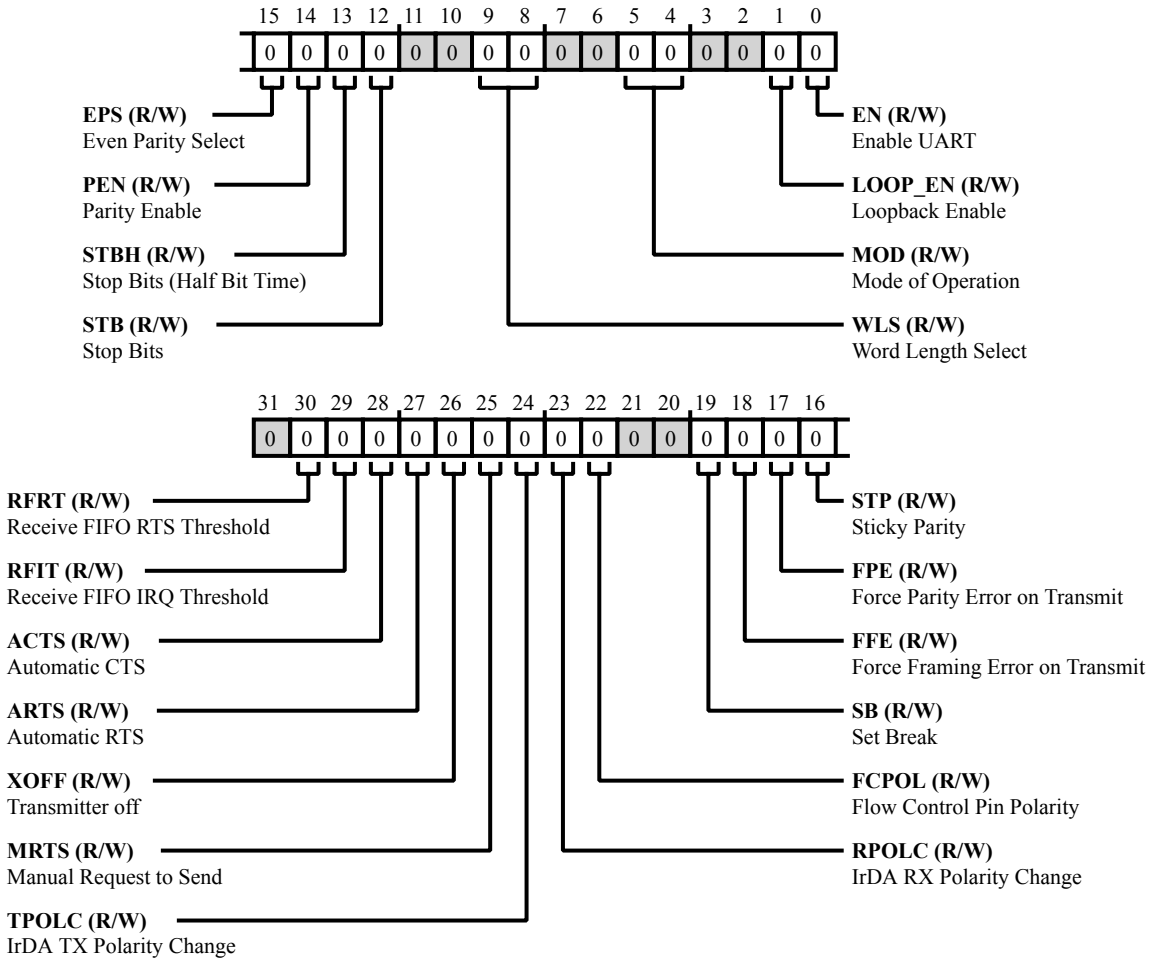


Figure 25-12: UART_CTL Register Diagram

Table 25-11: UART_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	RFRT	Receive FIFO RTS Threshold. The UART_CTL.RFRT bit controls UART_RTS pin assertion and deassertion timing. This bit is ignored if UART_CTL.ARTS is cleared. If set, the UART_RTS pin is deasserted when the receive buffer already holds seven words and an eighth start bit is detected. It is reasserted when the FIFO contains seven words or less. If cleared, the UART_RTS pin is deasserted when the RX buffer already holds four words and a fifth start bit is detected. The UART_RTS pin is reasserted when the RX buffer contains no more than 4 words.	
		0	Deassert RTS if RX FIFO word count > 4; assert if <= 4
		1	Deassert RTS if RX FIFO word count > 7; assert if <= 7
29 (R/W)	RFIT	Receive FIFO IRQ Threshold. The UART_CTL.RFIT bit controls the timing of the UART_STAT.RFCS bit. If UART_CTL.RFIT is cleared, the receive threshold is two. If UART_CTL.RFIT is set, the threshold is four words in the receive buffer.	
		0	Set RFCS=1 if RX FIFO count >= 4
		1	Set RFCS=1 if RX FIFO count >= 7
28 (R/W)	ACTS	Automatic CTS. The UART_CTL.ACTS bit must be set to enable the UART_CTS input pin for UART_TX handshaking. If enabled, the UART_STAT.CTS bit holds the value (if UART_CTL.FCPOL is set) or complement value (if UART_CTL.FCPOL is cleared) of the UART_CTS input pin. The UART_STAT.CTS bit can be used to determine whether the external device is ready to receive data (if UART_STAT.CTS is set) or whether it is busy (if UART_STAT.CTS is cleared). If UART_CTL.ACTS is cleared, the UART_TX handshaking protocol is disabled, and the UART_TX line transmits data whenever there is data to send, regardless of the value of UART_CTS. Software can pause ongoing transmission by setting the UART_CTL.XOFF bit.	
		0	Disable TX handshaking protocol
		1	Enable TX handshaking protocol
27 (R/W)	ARTS	Automatic RTS. The UART_CTL.ARTS bit must be set to enable the UART_RTS input pin for UART_TX handshaking. If set, the hardware guarantees a minimal UART_RTS pin deassertion pulse width of at least the number of data bits defined by the UART_CTL.WLS bit field. If cleared, the UART_RTS pin is not generated automatically by hardware. The UART_RTS pin can still be manually controlled by the UART_CTL.MRTS bit, and software is responsible for UART_RTS pulse width control (if needed).	
		0	Disable RX handshaking protocol.
		1	Enable RX handshaking protocol.

Table 25-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	XOFF	Transmitter off. The UART_CTL.XOFF bit (if set) turns off transmission (XOFF) by preventing the content of THR from being continued to TSR. When set, this bit turns on transmission (XON). The state of the UART_CTL.XOFF bit is ignored if the UART_CTL.ACTS bit is set.	
		0	Transmission ON, if ACTS=0
		1	Transmission OFF, if ACTS=0
25 (R/W)	MRTS	Manual Request to Send. The UART_CTL.MRTS bit controls the state of the $\overline{\text{UART\_RTS}}$ output pin when the UART_CTL.ARTS bit is cleared. When UART_CTL.MRTS is cleared, the UART deasserts the UART_RTS pin, signaling to the external device that the UART is not ready to receive. When UART_CTL.MRTS is set, the UART asserts the $\overline{\text{UART\_RTS}}$ pin, signaling to the external device that the UART is ready to receive.	
		0	Deassert RTS pin when ARTS=0
		1	Assert RTS pin when ARTS=0
24 (R/W)	TPOLC	IrDA TX Polarity Change. The UART_CTL.TPOLC bit selects the active low or high polarity for IrDA communications. This bit is effective only in IrDA mode. If set, in IrDA mode, the UART_TX pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the UART_TX pin idles low. In UART or MDB mode, it is NRZ.	
		0	Active-low TX polarity setting
		1	Active-high TX polarity setting
23 (R/W)	RPOLC	IrDA RX Polarity Change. The UART_CTL.RPOLC bit selects the active low or high polarity for IrDA communications. This bit is effective only in IrDA mode. If set, in IrDA mode, the UART_RX pin idles high. In UART or MDB mode, it is inverted-NRZ. If cleared, in IrDA mode, the UART_RX pin idles low. In UART or MDB mode, it is NRZ.	
		0	Active-low RX polarity setting
		1	Active-high RX polarity setting

Table 25-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	FCPOL	Flow Control Pin Polarity. The UART_CTL.FCPOL bit selects the polarities of the $\overline{\text{UART\_CTS}}$ and $\overline{\text{UART\_RTS}}$ pins. When the UART_CTL.FCPOL bit is cleared, the $\overline{\text{UART\_RTS}}$ and $\overline{\text{UART\_CTS}}$ pins are active low, and the UART is halted when the $\overline{\text{UART\_RTS}}$ and $\overline{\text{UART\_CTS}}$ pin state is high. When UART_CTL.FCPOL bit is set, the $\overline{\text{UART\_RTS}}$ and $\overline{\text{UART\_CTS}}$ pins are active high, and the UART is halted when the UART_RTS and UART_CTS pin state is low.	
		0	Active low CTS/RTS
		1	Active high CTS/RTS
19 (R/W)	SB	Set Break. If set, the UART_CTL.SB bit forces the UART_TX pin to low asynchronously, regardless of whether or not data is currently transmitted. This bit functions even when the UART clock is disabled. Because the UART_TX pin normally drives high, it can be used as a flag output pin, if the UART is not used. (For example, if UART_CTL.TPOLC is cleared, drive UART_TX pin low; or if UART_CTL.TPOLC is set, drive UART_TX pin high.)	
		0	No force
		1	Force TX pin to 0
18 (R/W)	FFE	Force Framing Error on Transmit. The UART_CTL.FFE bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.	
		0	Normal operation
		1	Force error
17 (R/W)	FPE	Force Parity Error on Transmit. The UART_CTL.FPE bit is intended for test purposes. This bit is useful for debugging software, especially in loopback mode.	
		0	Normal operation
		1	Force parity error

Table 25-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	STP	Sticky Parity. The UART_CTL.STP bit controls whether the parity is generated by hardware based on the data bits or whether it is set to a fixed value. If this bit is cleared, the hardware calculates the parity bit value based on the data bits. Then, the EPS bit determines whether odd or even parity mode is chosen. If this bit is set, odd parity is used. That means that the total count of logical-1 data bits including the parity bit must be an odd value. Even parity is chosen by UART_CTL.STP cleared and UART_CTL.EPS set. Then, the count of logical-1 bits must be a even value. If the UART_CTL.STP bit is set, hardware parity calculation is disabled. In this case, the sent and received parity equals the inverted UART_CTL.EPS bit.	
		0	No forced parity
		1	Force (Stick) parity to defined value (if PEN=1)
15 (R/W)	EPS	Even Parity Select.	
		0	Odd parity
		1	Even parity
14 (R/W)	PEN	Parity Enable. The UART_CTL.PEN bit enables parity transmission and parity check. The UART_CTL.PEN bit inserts one additional bit between the most significant data bit and the first stop bit. The polarity of this so-called parity bit depends on data and the UART_CTL.STP and UART_CTL.EPS control bits. Both transmitter and receiver calculate the parity value. The receiver compares the received parity bit with the expected value and issues a parity error if they do not match. If the UART_CTL.PEN bit is cleared, the UART_CTL.STP and the UART_CTL.EPS bits are ignored.	
		0	Disable
		1	Enable parity transmit and check
13 (R/W)	STBH	Stop Bits (Half Bit Time).	
		0	0 half-bit-time stop bit
		1	1 half-bit-time stop bit
12 (R/W)	STB	Stop Bits. The UART_CTL.STB bit controls how many stop bits are appended to transmitted data.	
		0	1 stop bit
		1	2 stop bits

Table 25-11: UART_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	WLS	Word Length Select. The UART_CTL.WLS field determines whether the transmitted and received UART word consists of 5, 6, 7, or 8 data bits.	
		0	5-bit word
		1	6-bit word
		2	7-bit word
		3	8-bit word
5:4 (R/W)	MOD	Mode of Operation. The UART_CTL.MOD selects the UART operation mode (UMOD).	
		0	UART mode
		1	MDB mode
		2	IrDA SIR mode
1 (R/W)	LOOP_EN	Loopback Enable. The UART_CTL.LOOP_EN bit enables UART loopback mode. When set, this bit disconnects the input of the receiver from the UART_RX pin, and internally redirects the transmit output to the receiver. The UART_TX pin remains active and continues to transmit data externally as well. Loopback mode also forces the UART_RTS pin to its deassertive state, disconnects the UART_CTS bit from the UART_CTS input pin, and directly connects the UART_CTL.MRTS bit to the UART_STAT.CTS bit. In loopback mode, setting the UART_CTL.MRTS bit sets the UART_STAT.CTS bit and enables the transmitter of the UART. Clearing the UART_CTL.MRTS bit clears the UART_STAT.CTS bit and disables the transmitter of the UART.	
		0	Disable
		1	Enable
0 (R/W)	EN	Enable UART. The UART_CTL.EN enables the UART clocks. This bit also resets the state machine and control registers when cleared. Using this bit to disable the UART -- when not used -- reduces power consumption.	
		0	Disable
		1	Enable

Interrupt Mask Register

The UART_IMSK register indicates the interrupt mask status (unmasked, if set, or masked, if cleared) of the UART status interrupt requests. This register is not a data register. Instead, it is controlled by the UART_IMSK_SET and UART_IMSK_CLR register pair. Writing ones to the UART_IMSK_SET register enables (unmasks) interrupt requests, and writing ones to the UART_IMSK_CLR register disables (masks) them. Reads from either register return the enabled bits.

The UART_IMSK register is used to enable requests for system handling of empty or full states of UART data registers. Unless polling is used as a means of action, the UART_IMSK.ERBFI and UART_IMSK.ETBEI bits are normally set. Setting this register without enabling system DMA causes the UART to notify the processor of the data inventory state using interrupts. For proper operation in this mode, system interrupts must be enabled, and appropriate interrupt handling routines must be present.

Each UART features three separate interrupt channels to handle the data transmit, data receive, and line status events independently, regardless of whether DMA is enabled or not. If no DMA channels are assigned to the UART, set the UART_IMSK.ELSI bit to reroute the transmit and receive interrupts to the status interrupt request output.

With system DMA enabled, the UART uses DMA to transfer data to or from the processor. Dedicated DMA channels are available for receive and transmit operations. Line error handling can be configured independently from the receive or transmit setup.

The DMA of the UART is enabled by first setting up the system DMA control registers and then enabling the UART_IMSK.ERBFI and UART_IMSK.ETBEI interrupts. This configuration is because the interrupt request lines double as DMA request lines. Depending on whether DMA is enabled or not, upon receiving these requests, the DMA control unit either generates a direct memory access or passes the UART interrupt on to the system interrupt handling unit(s). However, the error interrupt for the UART goes directly to the system interrupt handling unit(s), bypassing the DMA unit completely.

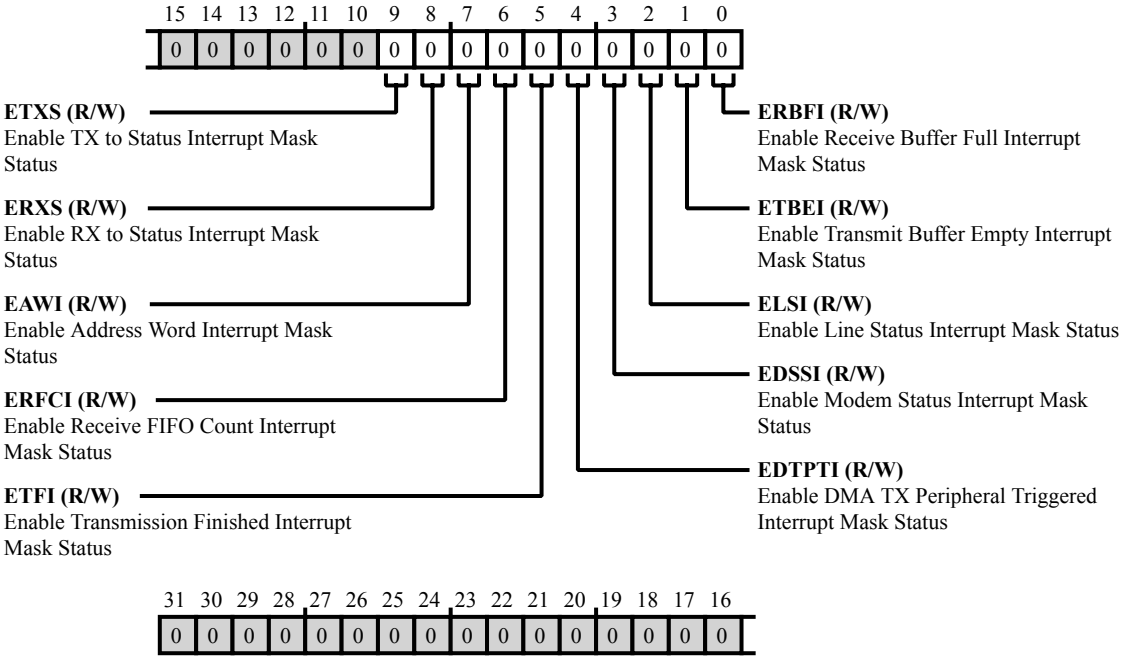


Figure 25-13: UART_IMSK Register Diagram

Table 25-12: UART_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	ETXS	Enable TX to Status Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ETXS` bit indicates re-direction of the TX interrupt requests to status interrupt output. If cleared, TX interrupt requests are routed to normal interrupt outputs.	
		0	Interrupt is masked
		1	Interrupt is unmasked
8 (R/W)	ERXS	Enable RX to Status Interrupt Mask Status. If set (interrupt unmasked), the `UART_IMSK.ERXS` bit indicates re-direction of RX interrupt requests to status interrupt output. If cleared, RX interrupt requests are routed to normal interrupt outputs.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 25-12: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	EAWI	Enable Address Word Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EAWI bit indicates generation of a status interrupt request when an Address word in MDB-mode is present in the UART_RBR. A received word is an address word if the UART_STAT.ADDR bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked
6 (R/W)	ERFCI	Enable Receive FIFO Count Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ERFCI bit indicates enabling of the receive buffer threshold interrupt request if signaled by the UART_STAT.RFCS bit. Read the UART_RBR register sufficient times to clear the interrupt request.	
		0	Interrupt is masked
		1	Interrupt is unmasked
5 (R/W)	ETFI	Enable Transmission Finished Interrupt Mask Status. If set (interrupt unmasked) the UART_IMSK.ETFI bit indicates enabling of interrupt generation on the status interrupt channel when the transmit buffer register, the transmit address register, and the transmit shift register are all empty as indicated by the UART_STAT.TFI. The UART_IMSK.ETFI interrupt can be used to avoid expensive polling of the UART_STAT.TEMT bit, when the UART clock or line drivers should be disabled after transmission has completed. W1C the UART_STAT.TFI bit to clear the interrupt request. In DMA operation, the UART_IMSK.ETFI bits functionality might be preferred.	
		0	Interrupt is masked
		1	Interrupt is unmasked
4 (R/W)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EDTPTI bit indicates enabling of the DMA completion interrupt request to be delayed until the data has left the UART completely. This bit is required for DMA transmit operation only. If set, the UART can generate a DMA interrupt request by the time the UART_STAT.TEMT bit goes high after the last DMA data word is transmitted. When UART_IMSK.EDTPTI is set, usually the DMA_CFG.INT field is cleared to 00 in a STOP mode DMA. This set up suppresses the normal completion interrupt request, and the UART_STAT.TEMT event is signaled through the DMA controller and triggers the DMA interrupt. If both (DMA_CFG.INT not 00 and UART_IMSK.EDTPTI set), two interrupts are requested at the end of a STOP mode DMA.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 25-12: UART_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	EDSSI	Enable Modem Status Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.EDSSI bit indicates enabling of a modem status interrupt request on the same status interrupt channel when the UART_STAT.SCTS bit is set. This indicates UART_CTS pin re-assertion. Write-1-to-clear (W1C) the UART_STAT.SCTS bit to clear the interrupt request.	
		0	Interrupt is masked
		1	Interrupt is unmasked
2 (R/W)	ELSI	Enable Line Status Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ELSI bit indicates that redirection of TX and RX interrupt requests to the status interrupt output of the UART by OR'ing them with the UART_STAT.OE, UART_STAT.PE, UART_STAT.FE, and UART_STAT.BI interrupt requests. Set this bit when no DMA channel is associated with the UART. Enabling UART_IMSK.ELSI disables the RX/TX interrupt channels and negates the UART_IMSK.EDTPTI bit.	
		0	Interrupt is masked
		1	Interrupt is unmasked
1 (R/W)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ETBEI bit indicates generation of a TX interrupt request if the UART_STAT.THRE bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked
0 (R/W)	ERBFI	Enable Receive Buffer Full Interrupt Mask Status. If set (interrupt unmasked), the UART_IMSK.ERBFI indicates generation of an RX interrupt request if the UART_STAT.DR bit is set.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Interrupt Mask Clear Register

The UART_IMSK indicates interrupt mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the UART_IMSK_SET and UART_IMSK_CLR register pair. Writing ones to UART_IMSK_SET enables (unmasks) interrupt requests, and writing ones to UART_IMSK_CLR disables (masks) them. Reads from either register return the enabled bits. For more information, see the UART_IMSK register description.

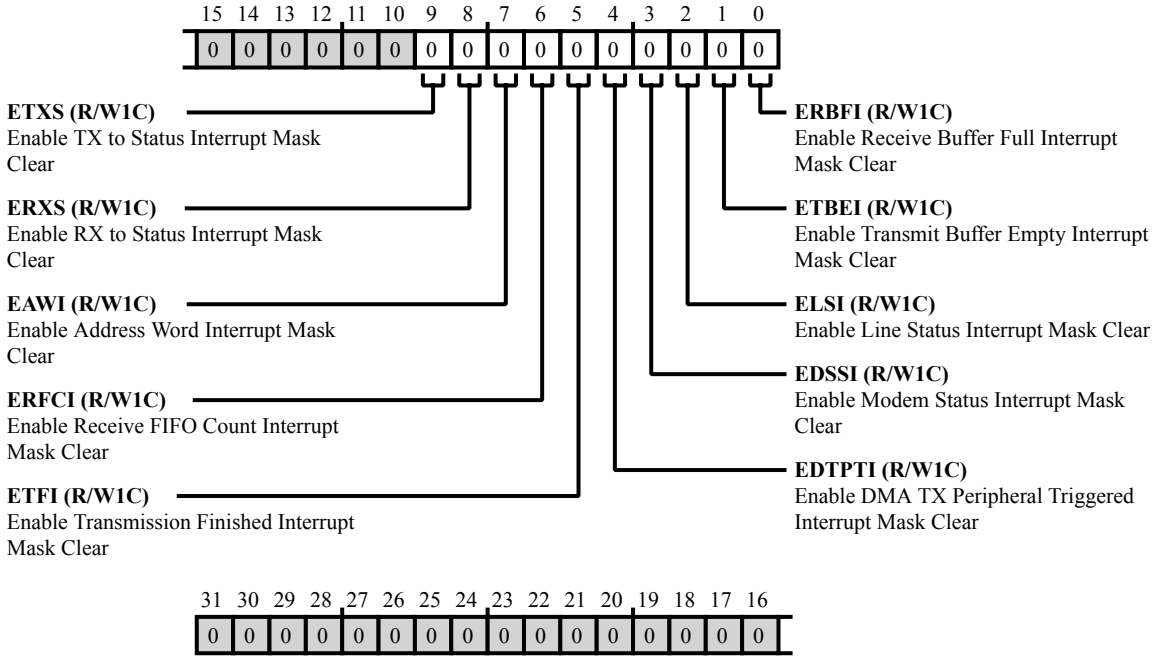


Figure 25-14: UART_IMSK_CLR Register Diagram

Table 25-13: UART_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	ETXS	Enable TX to Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
8 (R/W1C)	ERXS	Enable RX to Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
7 (R/W1C)	EAWI	Enable Address Word Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt

Table 25-13: UART_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	ERFCI	Enable Receive FIFO Count Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
5 (R/W1C)	ETFI	Enable Transmission Finished Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
4 (R/W1C)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
3 (R/W1C)	EDSSI	Enable Modem Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
2 (R/W1C)	ELSI	Enable Line Status Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
1 (R/W1C)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt
0 (R/W1C)	ERBFI	Enable Receive Buffer Full Interrupt Mask Clear.	
		0	No action
		1	Mask interrupt

Interrupt Mask Set Register

The UART_IMSK indicates interrupt request mask status (unmasked if set, masked if cleared) of UART status interrupts. This register is not a data register. Instead it is controlled by the UART_IMSK_SET and UART_IMSK_CLR register pair. Writing ones to UART_IMSK_SET enables (unmasks) interrupt requests, and writing ones to UART_IMSK_CLR disables (masks) them. Reads from either register return the enabled bits. For more information, see the UART_IMSK register description.

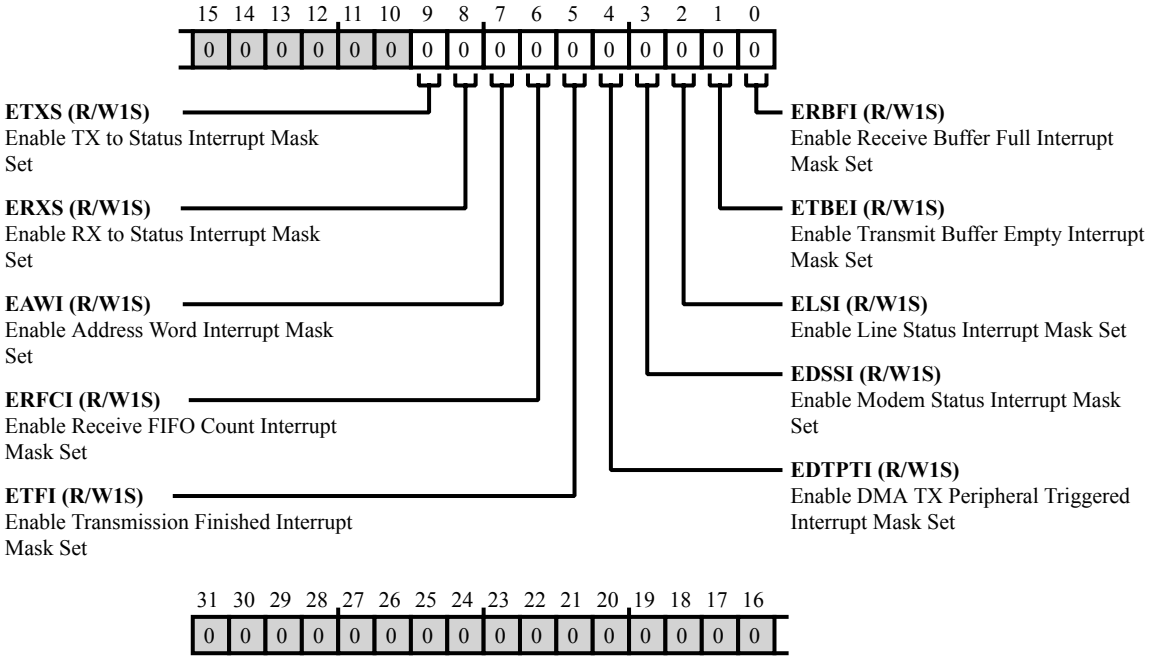


Figure 25-15: UART_IMSK_SET Register Diagram

Table 25-14: UART_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1S)	ETXS	Enable TX to Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
8 (R/W1S)	ERXS	Enable RX to Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
7 (R/W1S)	EAWI	Enable Address Word Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt

Table 25-14: UART_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1S)	ERFCI	Enable Receive FIFO Count Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
5 (R/W1S)	ETFI	Enable Transmission Finished Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
4 (R/W1S)	EDTPTI	Enable DMA TX Peripheral Triggered Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
3 (R/W1S)	EDSSI	Enable Modem Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
2 (R/W1S)	ELSI	Enable Line Status Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
1 (R/W1S)	ETBEI	Enable Transmit Buffer Empty Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt
0 (R/W1S)	ERBFI	Enable Receive Buffer Full Interrupt Mask Set.	
		0	No action
		1	Unmask interrupt

Receive Buffer Register

The read-only UART_RBR register is the UART's receive buffer. It is updated when there is pending data in the receive FIFO. Newly available data is signaled by the UART_STAT.DR bit.

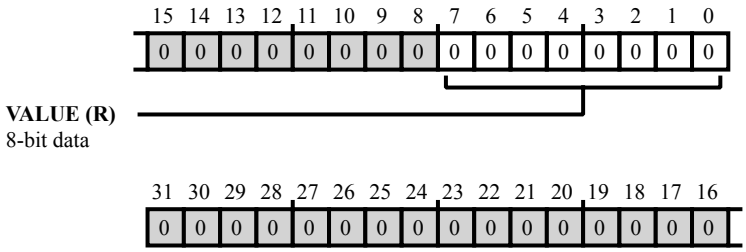


Figure 25-16: UART_RBR Register Diagram

Table 25-15: UART_RBR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	VALUE	8-bit data.

Receive Shift Register

The read only UART_RSR register which returns the content of the UART's receive shift register.

The frame data is moved into this shift register after polarity inversion, if any (including the native polarity inversion in the IrDA case).

In the case of the longest frame (MDB, with parity mode, and 8 bit data word-length), the start bit may be shifted out and not available for reading at the end of the frame reception. This register is NOT reset at the start of frame. If read, in the middle of a frame reception, data corresponding the previous frame may not have entirely shifted out (for example, the read data that have been read may NOT correspond entirely to the frame being received).

Because the UART is receiving only 1 stop bit, the UART_RSR contains only 1 stop bit even if more than one stop bit is present in the actual transfer. This register may be considered as storing the 10 most recently received bits (taking into consideration the stop bit receive limitation above).

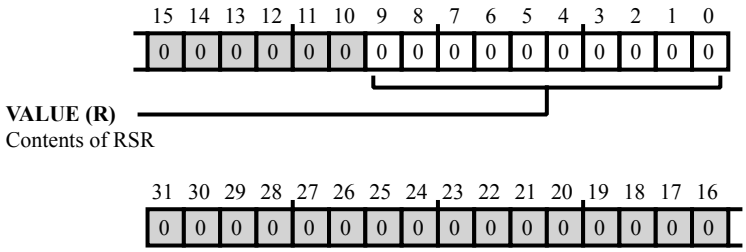


Figure 25-17: UART_RSR Register Diagram

Table 25-16: UART_RSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	VALUE	Contents of RSR.

Receive Counter Register

The UART_RXCNT register returns the content of 16-bit counter in the UART receiver. This count is used for baud rate clock generation (the lower [15:0] is the count data).

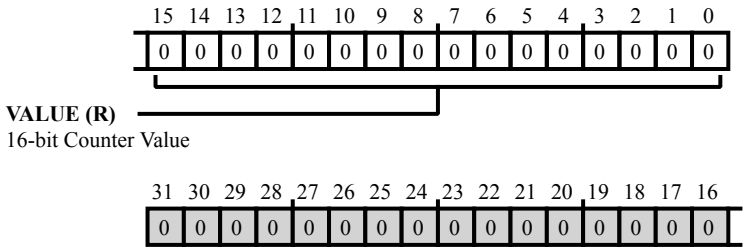


Figure 25-18: UART_RXCNT Register Diagram

Table 25-17: UART_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

Scratch Register

The UART_SCR registers contain 8-bit scratch pad data. These registers are used for general purpose data storage and do not control the UART hardware in any way.

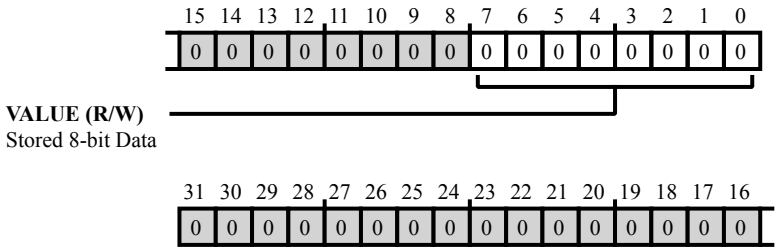


Figure 25-19: UART_SCR Register Diagram

Table 25-18: UART_SCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Stored 8-bit Data.

Status Register

The UART_STAT register contains the UART line status and UART modem status, as indicated by the current states of the UART's UART_CTS pin and internal receive buffers. Writes to this register can perform write-one-to-clear (W1C) operations on most status bits. Reading this register has no side effects.

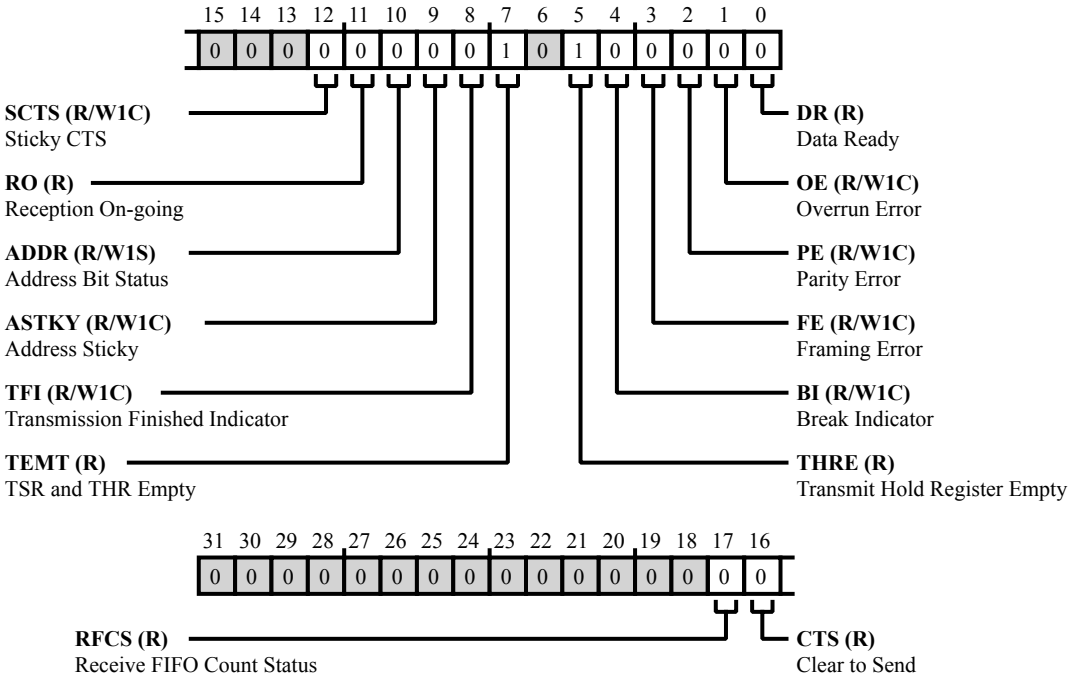


Figure 25-20: UART_STAT Register Diagram

Table 25-19: UART_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/NW)	RFCS	Receive FIFO Count Status. The UART_STAT.RFCS bit is set when the receive buffer holds more or equal entries than a certain threshold. The threshold is controlled by the UART_CTL.RFIT bit. If UART_CTL.RFIT is cleared, the threshold is four entries. If UART_CTL.RFIT is set, the threshold is seven entries. The UART_STAT.RFCS bit is cleared when the UART_RBR register is read sufficient times until the buffer is drained below the threshold. The UART_STAT.RFCS bit can trigger a status interrupt request if enabled by the UART_IMSK_SET.ERFCI bit.	
		0	RX FIFO has less than 4 (7) entries when RFIT=0 (1)
		1	RX FIFO has at least 4 (7) entries when RFIT=0 (1)

Table 25-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	CTS	Clear to Send. The UART_STAT.CTS bit holds the value (if UART_CTL.FCPOL set) or the complement value (if UART_CTL.FCPOL cleared) of the UART_CTS input pin. The UART_CTL.ACTS bit must be set to enable this feature. The core can read the value of the UART_STAT.CTS bit to determine whether the external device is ready to receive (UART_STAT.CTS set) or if it is busy (UART_STAT.CTS cleared). If UART_CTL.ACTS is cleared, the UART_TX handshaking protocol is disabled, and the UART transmits data as long as there is data to transmit, regardless of the value of UART_STAT.CTS. When UART_CTL.ACTS is cleared, the software can pause transmission temporarily by setting the XOFF bit. Note that in loopback mode (UART_CTL.LOOP_EN set), the UART_STAT.CTS bit is disconnected from the UART_CTS input pin. Instead, the bit is directly connected to the UART_CTL.MRTS bit.	
		0	Not clear to send (External device not ready to receive)
		1	Clear to send (External device ready to receive)
12 (R/W1C)	SCTS	Sticky CTS. The UART_STAT.SCTS bit is a sticky bit that is set when UART_STAT.CTS transitions from 0 to 1. The UART_STAT.SCTS bit is cleared by software with a W1C operation. This bit can trigger a line status interrupt request if enabled by the UART_IMSK_SET.EDSSI bit.	
		0	CTS has not transitioned from low to high
		1	CTS has transitioned from low to high
11 (R/NW)	RO	Reception On-going.	
		0	No data reception in progress
		1	Data reception in progress
10 (R/W1S)	ADDR	Address Bit Status. The UART_STAT.ADDR bit is used to mirror the address bit of the word in UART_RBR in multi-drop bus protocol, and is enabled only in MDB mode. The UART_STAT.ADDR bit is updated by hardware upon detecting a received word with the address bit in UART_RBR set or cleared. Additionally, software can set the ADDR bit with a write-1-to-set (W1S) operation.	
		0	Address bit is low
		1	Address bit is high

Table 25-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W1C)	ASTKY	Address Sticky. The UART_STAT.ASTKY bit is used in multi-drop bus mode to indicate whether a peripheral is currently being addressed. This bit is a sticky version of the UART_STAT.ADDR bit and is set by hardware when setting the UART_STAT.ADDR bit. The UART_STAT.ASTKY bit can only be cleared by software with a write-one-to-clear (W1C) operation. With the UART_STAT.ASTKY bit set, words will be received irrespective of the UART_CTL.MOD bit or UART_STAT.ADDR bit selection. With the UART_STAT.ASTKY bit cleared, only address words (UART_CTL.MOD bit set) will be received and words with UART_CTL.MOD bit cleared are ignored (not moved from the RSR to the RX FIFO) in MDB mode. The UART_STAT.ASTKY bit does not affect reception in non-MDB modes.	
		0	ADDR bit has not been set
		1	ADDR bit has been set
8 (R/W1C)	TFI	Transmission Finished Indicator. The UART_STAT.TFI bit is a sticky version of the UART_STAT.TEMT bit. While UART_STAT.TEMT is automatically cleared by hardware when new data is written to the UART_THR register, the sticky UART_STAT.TFI bit remains set, until it is cleared by software (W1C). The UART_STAT.TFI bit enables more flexible transmit interrupt request timing.	
		0	TEMT did not transition from 0 to 1
		1	TEMT transition from 0 to 1
7 (R/NW)	TEMT	TSR and THR Empty. The UART_STAT.TEMT bit indicates that the UART_THR and UART_TAIP registers and the UART_TSR register are empty. In this case, the program is permitted to write to the UART_THR and UART_TAIP registers twice without losing data. The UART_STAT.TEMT bit can also be used as indicator that pending UART transmission is completed. At that time, it is safe to disable the UART_CTL.EN bit or to three-state the off-chip line driver.	
		0	Not empty TSR/THR
		1	TSR/THR Empty

Table 25-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	THRE	Transmit Hold Register Empty. The UART_STAT.THRE bit indicates that the UART transmit channel is ready for new data and software can write to the UART_THR and UART_TAIP registers. Writes to the UART_THR and UART_TAIP registers clear the UART_STAT.THRE. The bit is set again when the UART_THR and UART_TAIP registers are empty and ready to accept data.	
		0	Not empty THR/TAIP
		1	Empty THR/TAIP
4 (R/W1C)	BI	Break Indicator. The UART_STAT.BI bit indicates that the first stop bit is sampled low and the entire data word, including parity bit, consists of low bits only. (This condition indicates that UART_RX was held low for more than the maximum word length.) The UART_STAT.BI bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The bit is sticky and can be cleared by W1C operations.	
		0	No break interrupt
		1	Break interrupt this indicates UARTxRX was held low(RPOLC=0) / high (RPOLC=1) for more than the maximum word length
3 (R/W1C)	FE	Framing Error. The UART_STAT.FE bit indicates that the first stop bit is sampled. This bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The UART_STAT.FE bit is sticky and can be cleared by W1C operations. Note that invalid stop bits can be simulated by setting the UART_CTL.FFE bit.	
		0	No error
		1	Invalid stop bit error
2 (R/W1C)	PE	Parity Error. The UART_STAT.PE bit indicates that the received parity bit does not match the expected value. This bit is updated simultaneously with the UART_STAT.DR bit, that is, by the time the first stop bit is received or when data is loaded from the receive FIFO to the UART_RBR register. The UART_STAT.PE bit is sticky and can be cleared by W1C operations. Note that invalid parity bits can be simulated by setting the UART_CTL.FPE bit.	
		0	No parity error
		1	Parity error

Table 25-19: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	OE	Overrun Error. The UART_STAT.OE bit indicates that further data is received while the internal receive buffer was full. This bit is set when sampling the stop bit of the sixth data word. To avoid overruns, read the UART_RBR register in time. In DMA receive mode, overruns are very unlikely to happen ever. After an overrun occurs, the UART_RBR and receive FIFO are protected from being overwritten by new data until the UART_STAT.OE bit is cleared by software. The content of the UART_RSR register is lost as soon as the overrun occurs. The UART_STAT.OE bit is sticky and can be cleared by W1C operations.	
		0	No overrun
		1	Overrun error
0 (R/NW)	DR	Data Ready. The UART_STAT.DR bit indicates that data is available in the receiver and can be read from the UART_RBR register. The bit is set by hardware when the receiver detects the first valid stop bit. The bit is cleared by hardware when the UART_RBR register is read.	
		0	No new data
		1	New data in RBR

Transmit Address/Insert Pulse Register

The UART_TAIP register and the UART_THR register share the same physical register, but UART_TAIP has different effect than the UART_THR register when UART_TAIP is written to in MDB and UART modes.

In MDB mode, data written to the UART_TAIP register is transmitted as an address frame (as with the UART_CTL.MOD bit set).

In UART mode, a write to UART_TAIP causes a pulse of value UART_TAIP [7] for a duration of UART_TAIP [6:0] x bit time. (There is additional inversion if the UART_CTL.TPOLC bit is set).

Bit time is defined by the UART_CLK register. The transmission of the pulse is followed by stop bit transmission as specified by the UART_CTL.STB and UART_CTL.STBH bits. This could be used for supporting line break command and inter-frame gap.

In IrDA mode, writes to UART_TAIP is treated the same as writes to UART_THR.

Accesses to the UART_TAIP register have the same affects as the UART_THR register with respect to the UART_STAT.THRE, UART_STAT.TEMT, and UART_STAT.TFI flags.

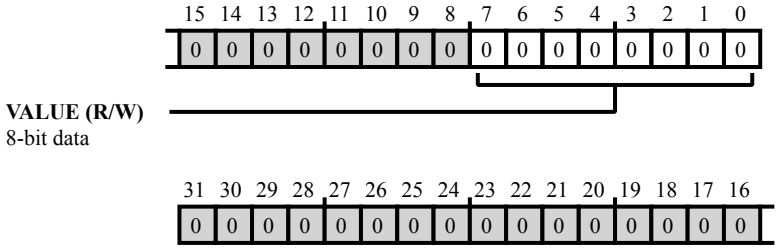


Figure 25-21: UART_TAIP Register Diagram

Table 25-20: UART_TAIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8-bit data.

Transmit Hold Register

The write-only UART_THR register is the UART's transmit buffer. The UART_STAT.THRE bit indicates whether data can be written to UART_THR. Writes to this register automatically propagate to the internal UART_TSR register as soon as UART_TSR is ready. Then, transmit operation is initiated immediately.

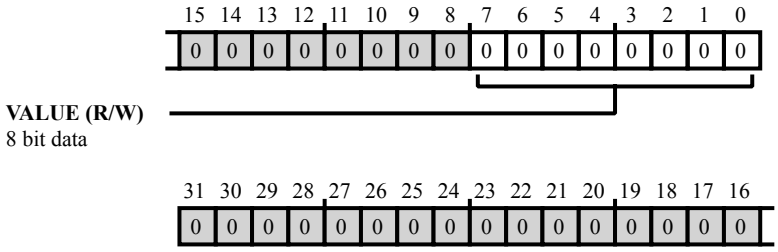


Figure 25-22: UART_THR Register Diagram

Table 25-21: UART_THR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	8 bit data.

Transmit Shift Register

The read only UART_TSR register which returns the content of the UART's transmit shift register.

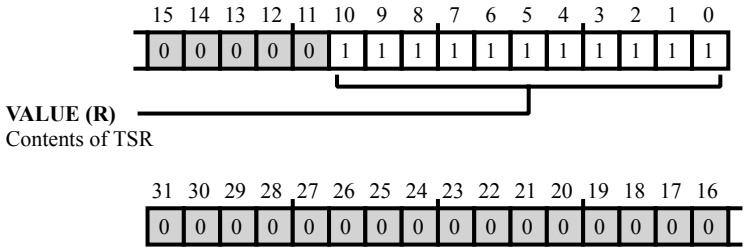


Figure 25-23: UART_TSR Register Diagram

Table 25-22: UART_TSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Contents of TSR.

Transmit Counter Register

The UART_TXCNT read only register returns the content of 16-bit counter in the UART transmitter. This count is used for baud rate clock generation (the lower [15:0] is the count data).

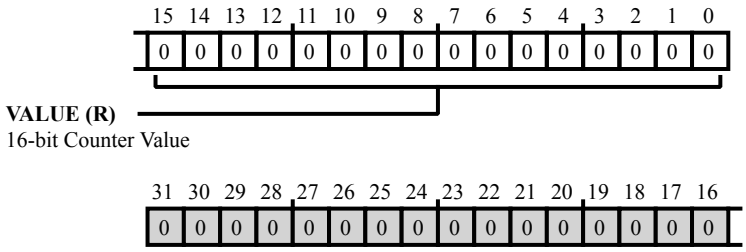


Figure 25-24: UART_TXCNT Register Diagram

Table 25-23: UART_TXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	16-bit Counter Value.

26 Two-Wire Interface (TWI)

The processor has a two-wire interface (TWI), that provides a simple exchange method of control data between multiple devices. The TWI module is compatible with the widely used I^2C bus standard. Additionally, the TWI module is fully compatible with serial camera control bus (SCCB) functionality for easier control of various CMOS camera sensor devices.

The TWI module offers the capabilities of simultaneous master and slave operation and support for both 7-bit addressing and multimedia data arbitration. The TWI interface uses two pins for transferring clock (`TWI_SCL`) and data (`TWI_SDA`) and supports the protocol at speeds up to 400K bits/sec. The TWI interface pins are compatible with 5-V logic levels.

To preserve processor bandwidth, the TWI module can be set up with transfer-initiated interrupts to only service FIFO buffer data reads and writes. Protocol-related interrupts are optional. The TWI externally moves 8-bit data while maintaining compliance with the I^2C bus protocol.

TWI Features

The TWI is fully compatible with the widely used I^2C bus standard.

The TWI controller includes the following features.

- Simultaneous master and slave operation on multiple device systems
- Support for multi-master bus arbitration
- 7-bit addressing
- 100K bits/second and 400K bits/second data rates
- General call address support
- Master clock synchronization and support for clock low extension
- Separate multiple-byte receive and transmit FIFOs
- Low interrupt rate
- Individual override control of data and clock lines in the event of bus lock-up
- Input filter for spike suppression

- Serial camera control bus support as specified in the *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*

TWI Functional Description

The TWI interface is a shift register that serially transmits and receives data bits. It moves data 1 bit at a time at the SCL rate, to and from other TWI devices. The SCL signal synchronizes the shifting and sampling of the data on the serial data pin.

CM41X_M4 TWI Register List

The Two-Wire Interface controller TWI allows a device to interface to an inter-IC bus as specified by the Philips I^2C Bus Specification version 2.1, dated January 2000. A set of registers governs TWI operations. For more information on TWI functionality, see the TWI register descriptions.

Table 26-1: CM41X_M4 TWI Register List

Name	Description
TWI_CLKDIV	SCL Clock Divider Register
TWI_CTL	Control Register
TWI_FIFOCTL	FIFO Control Register
TWI_FIFOSTAT	FIFO Status Register
TWI_IMSK	Interrupt Mask Register
TWI_ISTAT	Interrupt Status Register
TWI_MSTRADDR	Master Mode Address Register
TWI_MSTRCTL	Master Mode Control Registers
TWI_MSTRSTAT	Master Mode Status Register
TWI_RXDATA16	Rx Data Double-Byte Register
TWI_RXDATA8	Rx Data Single-Byte Register
TWI_SLVADDR	Slave Mode Address Register
TWI_SLVCTL	Slave Mode Control Register
TWI_SLVSTAT	Slave Mode Status Register
TWI_TXDATA16	Tx Data Double-Byte Register
TWI_TXDATA8	Tx Data Single-Byte Register

CM41X_M4 TWI Interrupt List

Table 26-2: CM41X_M4 TWI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
154	TWI0_DATA	TWI0 Data Interrupt	Level	

CM41X_M0 TWI Interrupt List

Table 26-3: CM41X_M0 TWI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
29	TWI0_DATA	TWI0 Data Interrupt	Level	

TWI Block Diagram

The ***TWI Block Diagram*** figure shows the basic blocks of the TWI interface.

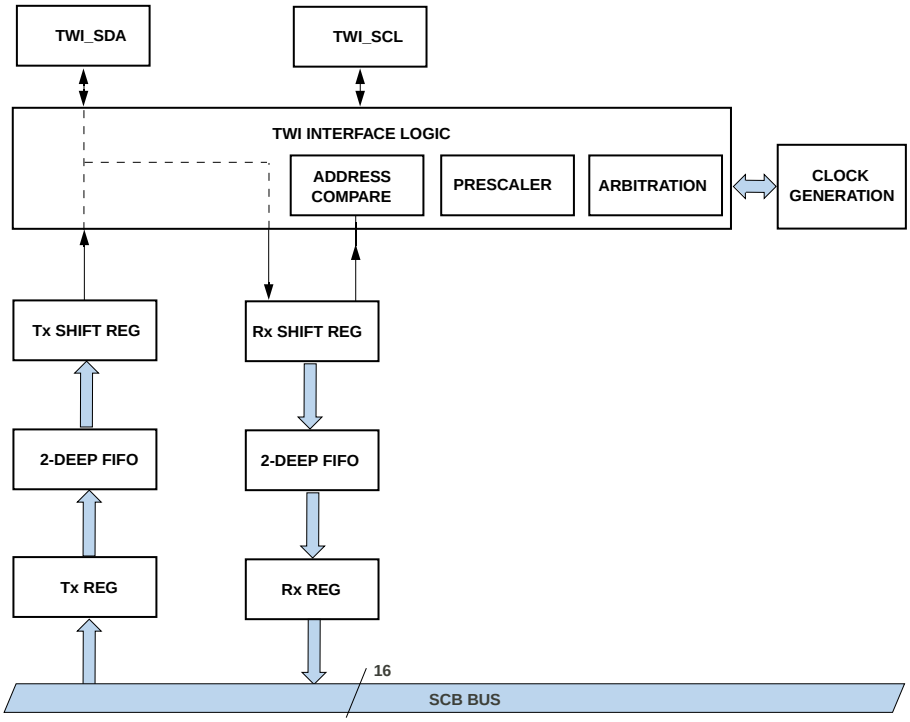


Figure 26-1: TWI Block Diagram

External Interface

The `TWI_SDA` (serial data) and `TWI_SCL` (serial clock) signals are open drain and require pull-up resistors. These bidirectional signals externally interface the TWI controller to the I^2C bus and no other external connections or logic are necessary.

Serial Clock Signal (SCL)

The serial clock signal (TWI_SCL) is an input in slave mode. In master mode, the TWI controller must set this signal to the desired frequency.

The TWI controller supports the standard mode of operation (up to 100 kHz) or fast mode (up to 400 kHz). The TWI control register (TWI_CTL) sets the TWI_CTL.PRESCALE value which sets the relationship between the system clock (SCLK) and the internally timed events of the TWI controller. The internal time reference is derived from SCLK using a prescaled value. The prescale value is the number of SCLK periods used in the generation of one internal time reference. Set the value of prescale to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value as follows.

PRESCALE = f_{SCLK}/10MHz

NOTE: It is not always possible to achieve 10-MHz accuracy. In such cases, it is safe to round up the PRESCALE value to the next highest integer. For example, if SCLK is 100 MHz, the PRESCALE value is calculated as 100 MHz/10 MHz = 10. A prescale value of 14 in this case ensures that all timing requirements are met.

During master mode operation, the TWI module uses the TWI_CLKDIV register values to create the minimum TWI_CLKDIV.CLKHI and TWI_CLKDIV.CLKLO durations of the TWI_SCL signal. The TWI_CLKDIV.CLKHI field specifies the minimum number of 10-MHz time reference periods the TWI_SCL waits before a new clock low period begins, assuming a single master. (The 10-MHz time reference periods are represented as an 8-bit binary value). The TWI uses the TWI_CLKDIV.CLKLO field to specify the minimum number of internal time reference periods (represented as an 8-bit binary value). The TWI_SCL signal is held low.

Serial clock frequencies can vary from 400 kHz to less than 20 kHz. The resolution of the clock generated is 1/10 MHz or 100 ns. The following equation describes the frequency.

TWI_CLKDIV = TWI_SCL period/10 MHz time reference.

For example, for an TWI_SCL of 400 kHz (period = 1/400 kHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns), the following equation applies:

TWI_CLKDIV = 2500 ns/100 ns = 25

Therefore, a TWI_SCL signal with a 30% duty cycle has TWI_CLKDIV.CLKLO=17 and TWI_CLKDIV.CLKHI=8. Adding TWI_CLKDIV.CLKLO and TWI_CLKDIV.CLKHI equals TWI_CLKDIV.

NOTE: The TWI_CLKDIV.CLKHI and TWI_CLKDIV.CLKLO fields are not intended to guarantee a certain frequency. Rather, they guarantee a certain minimum high and low duration for the TWI_SCL signal. Slew rate controls falling edges. The *RC* time constant governs the rising edges. The pull-up resistor and the TWI_SCL capacitance form the time constant. See the “Register Descriptions” section for more details.

Serial Data Signal (SDA)

The TWI transmits and receives serial data, depending on the direction of the transfer, on the bidirectional serial data signal (SDA).

Internal Interface

The peripheral bus interface supports the transfer of 16-bit wide data. The processor uses the interface in the support of register and FIFO buffer reads and writes. The TWI internal interface is comprised of the blocks described as follows.

Register block. Contains all control and status bits and reflects what can be written or read as outlined by the programming model. Each function block updates their corresponding status bits.

FIFO buffer. Configured as a 1-byte-wide, 2-deep transmit FIFO buffer and a 1-byte-wide, 2-deep receive FIFO buffer.

Transmit shift register. Serially shifts its data out externally off chip. The output can be controlled for generation of acknowledgments or it can be manually overwritten.

Receive shift register. Receives its data serially from off chip. The receive shift register is 1 byte wide and data received can either be transferred to the FIFO buffer or used in an address comparison.

Address compare block. Supports address comparison in the event the TWI controller module is accessed as a slave.

Prescaler block. Must be programmed to generate a 10-MHz time reference relative to the system clock. The block uses this time base for filtering of data and timing events specified by the electrical data sheet (See the Philips specification). The block uses the time base to generate the `TWI_SCL` clock as well.

Clock generation module. Generates an external `TWI_SCL` clock when in master mode. It includes the logic necessary for synchronization in a multi-master clock configuration and clock stretching when configured in slave mode.

NOTE: The TWI does not support DMA based operation.

TWI Architectural Concepts

The TWI controller follows the transfer protocol of the Philips I^2C Bus specification version 2.1 dated January 2000.

NOTE: The TWI unit does not support DMA-based operation.

TWI Protocol

The ***Data Transfer*** figure shows a simple complete transfer.

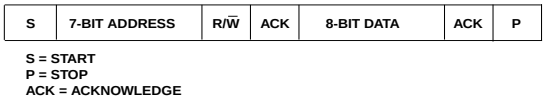


Figure 26-2: Data Transfer

The TWI controller register contents maps to a basic transfer. The ***Data Transfer with Bit Illustration*** figure details the same transfer from the ***Data Transfer*** figure noting the corresponding TWI controller bit names. In this illustration, the TWI controller successfully transmits 1 byte of data. The slave has acknowledged both address and data.

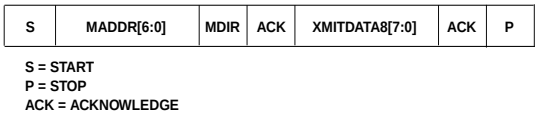


Figure 26-3: Data Transfer with Bit Illustration

Clock Generation and Synchronization

The TWI controller implementation only issues a clock during master mode operation and only at the time a transfer initiates. If arbitration for the bus is lost, the serial clock output immediately three-states. If multiple clocks attempt to drive the serial clock line, the TWI controller synchronizes its clock with the other remaining clocks. The ***Clock Synchronization*** figure shows this functionality.

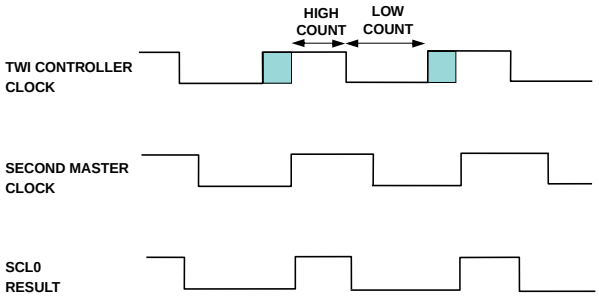


Figure 26-4: Clock Synchronization

The TWI controller serial clock (`TWI_SCL`) output follows these rules:

- Once the clock high (`TWI_CLKDIV.CLKHI`) count is complete, the serial clock output is driven low and the clock low (`TWI_CLKDIV.CLKLO`) count begins.
- Once the clock low count is complete, the serial clock line is three-stated. This state allows the external pull-up resistor to pull the `TWI_SCL` signal high. The clock synchronization logic enters into a delay mode (shaded area) until the `TWI_SCL` signal is detected at logic 1 level. Now, the clock high count begins.

Bus Arbitration

The TWI controller initiates a master mode transmission only when the bus is idle. If the bus is idle and two masters initiate a transfer, arbitration for the bus begins. The ***Bus Arbitration*** figure shows the arbitration.

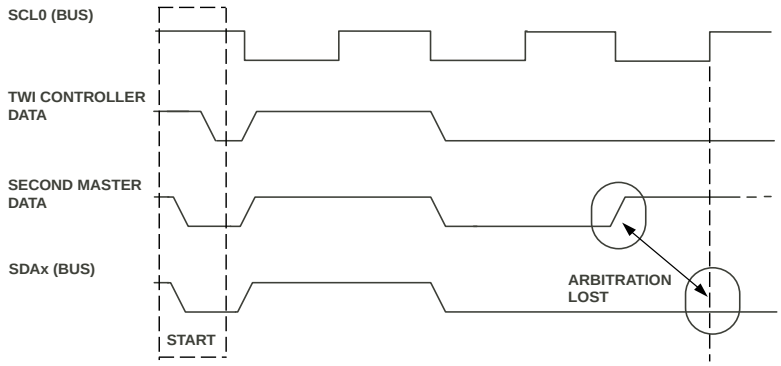


Figure 26-5: Bus Arbitration

The TWI controller monitors the serial data bus (SDA) while the TWI_SCL signal is high. If the TWI_SDA signal is determined to be an active logic 0 level while the data of the TWI controller is a logic 1 level, the TWI controller has lost arbitration. It stops generating the clock and data signals. Arbitration is not only performed at the serial clock edges, but also during the entire time the TWI_SCL signal is high.

Start and Stop Conditions

Start and stop conditions involve serial data transitions while the serial clock is a logic 1 level. The TWI controller generates and recognizes these transitions. Typically, start and stop conditions occur at the beginning and at the conclusion of a transmission, except repeated start combined transfers. The ***Start and Stop Conditions*** figure shows the transitions.

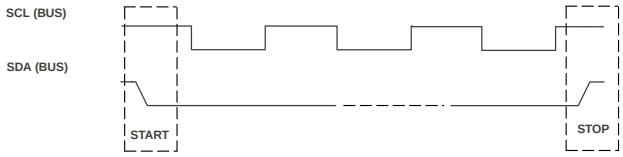


Figure 26-6: Start and Stop Conditions

The TWI special case start and stop conditions of the TWI controller include the following.

- Controller addressed as a slave-receiver. If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (TWI_ISTAT.SCOMP).
- Controller addressed as a slave-transmitter. If the master asserts a stop condition during the data phase of a transfer, the TWI controller concludes the transfer (TWI_ISTAT.SCOMP) and indicates a slave transfer error (TWI_ISTAT.SERR).
- Controller as a master-transmitter or master-receiver. If the stop bit (TWI_MSTRCTL.STOP) is set during an active master transfer, the TWI controller issues a stop condition as soon as possible avoiding any error conditions. The TWI controller operates as if data transfer count had been reached.

General Call Support

The TWI controller always decodes and acknowledges a general call address if:

- The TWI controller is enabled as a slave

- General call is enabled

The TWI_SLVCTL.GEN bit configures general call addressing (0x00) only when the TWI controller is a slave-receiver.

If the data associated with the transfer is (NAK) not acknowledged, the TWI_SLVCTL.NAK bit can be set. If the TWI controller issues a general call as a master-transmitter, set the appropriate address (TWI_MSTRADDR register) and transfer direction (TWI_MSTRCTL.DIR bit) and load the transmit FIFO data.

NOTE: The byte following the general call address usually defines the slaves response to the call. The interpretation of the command in the second byte is based on the value of its LSB. For a TWI slave device, the bytes received after the general call address are considered data.

Fast Mode

Fast mode essentially uses the same mechanics as the standard mode of operation. Fast mode affects electrical specifications and timing. When fast mode is enabled, (FAST) timing is modified to meet the following electrical requirements.

- Serial data rise times before arbitration evaluation (t_r)
- Stop condition set-up time from serial clock to serial data (t_{SUSTO})
- Bus free time between a stop and start condition (t_{BUF})

TWI Operating Modes

The TWI has two modes of operation: repeated start and clock stretching. The following sections describe the operating modes.

Repeated Start

A repeated start condition is the absence of a stop condition between two transfers. The two transfers can be of any direction type. Examples include a transmit followed by a receive, or a receive followed by a transmit. The following sections guide the programmer in developing a service routine.

Transmit Receive Repeated Start

The ***Repeated Start Followed by Data Receive*** figure shows a repeated start followed by a data receive sequence. The shading in the figure indicates that the slave has control of the bus.

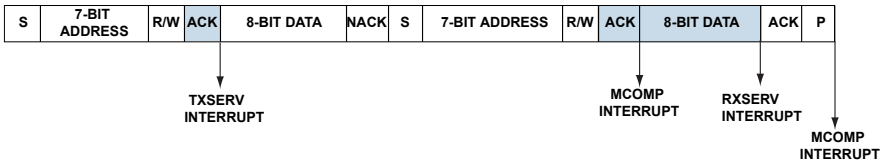


Figure 26-7: Repeated Start Followed by Data Receive

The tasks performed at each interrupt are:

- Transmit FIFO service (TWI_ISTAT.TXSERV) interrupt request. This interrupt is generated due to a FIFO access. Since this byte is the last of this transfer, the TWI uses the TWI_FIFOSTAT register to indicate that the transmit FIFO is empty. When read, TWI_MSTRCTL.DCNT bit field=0. Set the TWI_MSTRCTL.RSTART bit to indicate a repeated start and set the TWI_MSTRCTL.DIR bit if the following transfer is a data receive.
- Master transfer complete (TWI_ISTAT.MCOMP) interrupt. This interrupt request is generated when all data transfers (TWI_MSTRCTL.DCNT bit field=0). If no errors occur, a start condition initiates. Clear the TWI_MSTRCTL.RSTART bit and program the TWI_MSTRCTL.DCNT bits with the desired number of bytes to receive.
- Receive FIFO service (TWI_ISTAT.RXSERV) interrupt. This interrupt request is generated due to the arrival of a byte in the receive FIFO. Simple data handling is the only requirement.
- Master transfer complete (TWI_ISTAT.MCOMP) interrupt. This interrupt request is generated when the transfer completes.

Receive Transmit Repeated Start

The ***Repeated Start Data Receive Followed by Data Transmit*** figure illustrates a repeated start data receive followed by a data transmit sequence. The shading in the figure indicates that the slave has control of the bus.

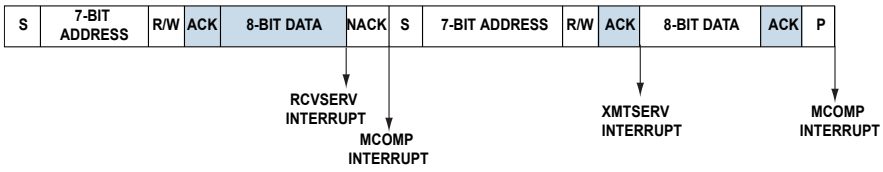


Figure 26-8: Repeated Start Data Receive Followed by Data Transmit

The tasks performed at each interrupt are:

- Receive FIFO service (TWI_ISTAT.RXSERV) interrupt. This interrupt request is generated due to the arrival of a data byte in the receive FIFO. Set the TWI_MSTRCTL.RSTART bit to indicate a repeated start and clear the TWI_MSTRCTL.DIR bit if the following transfer is a data transmit.
- Master transfer complete (TWI_ISTAT.MCOMP) interrupt. This interrupt request has occurred due to the completion of the data receive transfer. If no errors occur, a start condition initiates. Clear the TWI_MSTRCTL.RSTART bit and program the TWI_MSTRCTL.DCNT bits with the desired number of bytes to transmit.
- Transmit FIFO service (TWI_ISTAT.TXSERV) interrupt. This interrupt request is generated due to a FIFO access. Simple data handling is the only requirement.
- Master transfer complete (TWI_ISTAT.MCOMP) interrupt. This interrupt request is generated when the transfer completes.

NOTE: There is no timing constraint to meet the conditions—program the bits as required. Refer to Clock Stretching During Repeated Start section for more on how the controller stretches the clock during repeated start transfers.

Clock Stretching

Clock stretching is an added function of the TWI controller in master mode operation. This behavior uses self-induced stretching of the I ^{2}C clock while waiting to service interrupts. Hardware initiates stretching automatically. No programming is necessary. The TWI controller as a master supports three modes of clock stretching:

- Clock Stretching During FIFO Underflow
- Clock Stretching During FIFO Overflow
- Clock Stretching During Repeated Start

Clock Stretching During FIFO Underflow

During a master mode transmit, an interrupt request occurs the instant the transmit FIFO becomes empty. The most recent byte begins transmission. If the `TWI_ISTAT.TXSERV` interrupt request is not serviced, the concluding acknowledge phase of the transfer stretches.

Stretching of the clock continues until new data bytes are written to the transmit FIFO (`TWI_TXDATA8` or `TWI_TXDATA16` registers). No other action is required to release the clock and continue the transmission. This behavior continues until the transmission completes (`TWI_MSTRCTL.DCNT`=0). The transmission concludes (`TWI_ISTAT.MCOMP`). The ***Clock Stretching during FIFO Underflow*** figure and table show the stretching.

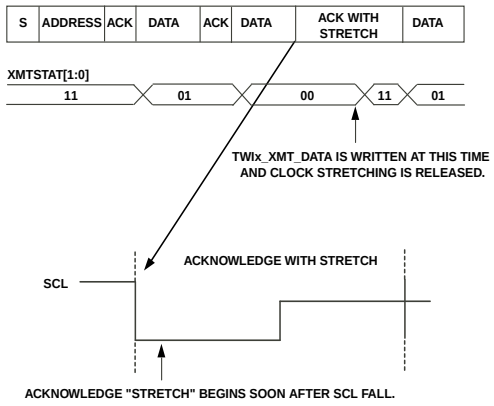


Figure 26-9: Clock Stretching during FIFO Underflow

TWI Controller	Processor
Interrupt: XMTSERV – Transmit FIFO buffer is empty.	Acknowledge: Clear the interrupt request source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transmit complete (DCNT= 0x00).	Acknowledge: Clear the interrupt request source bits.

Clock Stretching During FIFO Overflow

During a master mode receive operation, an interrupt occurs at the instant the receive FIFO becomes full. It is during the acknowledge phase of this received byte that clock stretching begins. The TWI module makes no attempt to initiate the reception of another byte. Stretching of the clock continues until the data bytes previously received are read from the receive FIFO buffer (`TWI_RXDATA8` or `TWI_RXDATA16` registers). No other action is required to release the clock and continue the reception of data. This behavior continues until the reception is complete (`TWI_MSTRCTL.DCNT=0`). Reception concludes (`TWI_ISTAT.MCOMP`). The ***Clock Stretching During FIFO Overflow*** figure and table show the clock stretching.

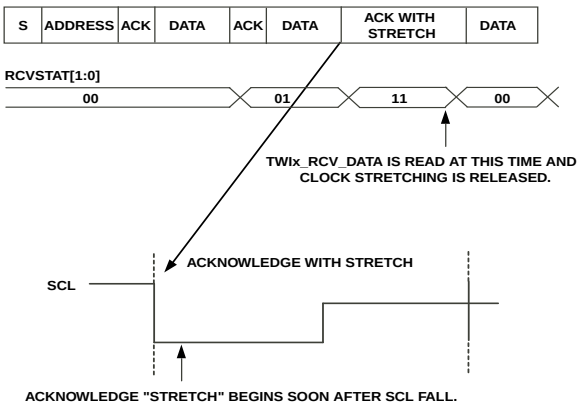


Figure 26-10: Clock Stretching During FIFO Overflow

TWI Controller	Processor
Interrupt: RCVSERV – Receive FIFO buffer is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Acknowledge: Clear the interrupt source bits.	Interrupt: MCOMP – Master receive complete.

Clock Stretching During Repeated Start

The repeated start feature in I^2C protocol requires a transition between two subsequent transfers. With the use of clock stretching, the task of managing transitions becomes simpler and common to all transfer types.

Once an initial TWI master transfer completes (transmit or receive), the clock initiates a stretch during the repeated start phase between transfers. Concurrent with this event, the initial transfer generates a `TWI_ISTAT.MCOMP` interrupt to signify the initial transfer has completed (`TWI_MSTRCTL.DCNT=0`). This initial transfer is handled without any special bit setting sequences or timing.

The clock stretching logic described applies here. With no system-related timing constraints, the subsequent transfer (receive or transmit) is set up and activated. This sequence can repeat as many times as required to string a series of repeated start transfers together. The ***Clock Stretching during Repeated Start Condition*** figure and table show the clock stretching.

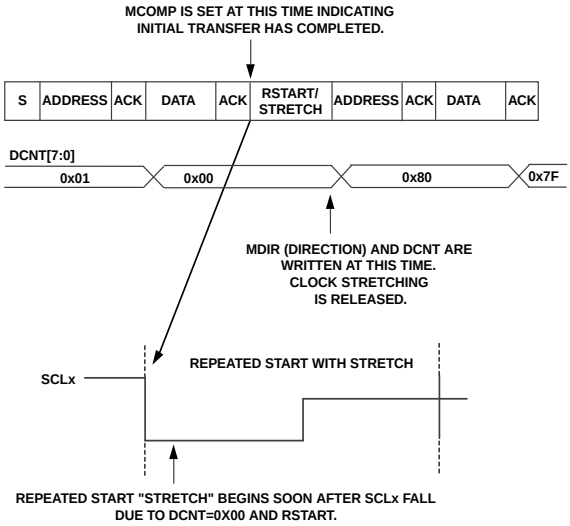


Figure 26-11: Clock Stretching during Repeated Start Condition

TWI Controller	Processor
Interrupt: MCOMP – Initial transmit has completed and DCNT = 0x00. Note: transfer in progress, RSTART previously set.	Acknowledge: Clear the interrupt request source bits. Write to TWIx_MASTER_CTL, setting MDIR (receive), clearing RSTART, and setting new DCNT value (nonzero).
Interrupt: RCVSERV – Receive FIFO is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master receive complete	Acknowledge: Clear the interrupt request source bits.

TWI Programming Model

The topics in this section provide information on the basic programming steps required to set up and run the two wire interface.

General Setup

General setup refers to register writes that are required for both slave mode and master mode operations.

Perform general setup before setting either the master or slave enable bits.

1. Program the `TWI_CTL.EN` bit to enable the TWI controller and set the prescale value (`TWI_CTL.PRESCALE` bit).
2. Program the prescale value to the binary representation of f_{SCLK0}/10 MHz. Round up all values to the next whole number.
3. Set the `TWI_CTL.EN` bit to enable the controller.

Once the TWI controller is enabled, a bus busy condition can be detected. This condition clears after t_{BUF} has expired, assuming no additional bus activity has been detected.

Slave Mode

When enabled, slave mode operation supports both receive and transmit data transfers.

It is not possible to enable only one data transfer direction and not acknowledge (NAK) the other. The following setup reflects this functionality.

1. Program the `TWI_SLVADDR` register. The TWI uses the appropriate 7 bits in determining a match during the address phase of the transfer.
2. Program the `TWI_TXDATA8.VALUE` or `TWI_TXDATA16` registers. These values are the initial data values for transmission when the slave is addressed and transmission is needed. This step is optional. If no data is written when the slave is addressed and transmission is needed, the serial clock (`TWI_SCL`) stretches. An interrupt is generated until data is written to the transmit FIFO.
3. Program the `TWI_IMSK` register. There are enable-bits associated with the desired interrupt sources. For example, programming the value 0x000F results in an interrupt request output to the processor, when the TWI module detects a valid address match. An interrupt request also occurs when a valid slave transfer completes or has an error, or a subsequent transfer has begun and the previous transfer has not been serviced.
4. Program the `TWI_SLVCTL` register. This step prepares and enables slave mode operation. For example, programming the value 0x0005 enables slave mode operation and requires 7-bit addressing. It indicates that data in the transmit FIFO buffer is for slave mode transmission.

The ***Slave Mode Interaction*** table and ***TWI Slave Mode Program Flow*** diagram represent the interaction between the TWI controller and the processor using this example.

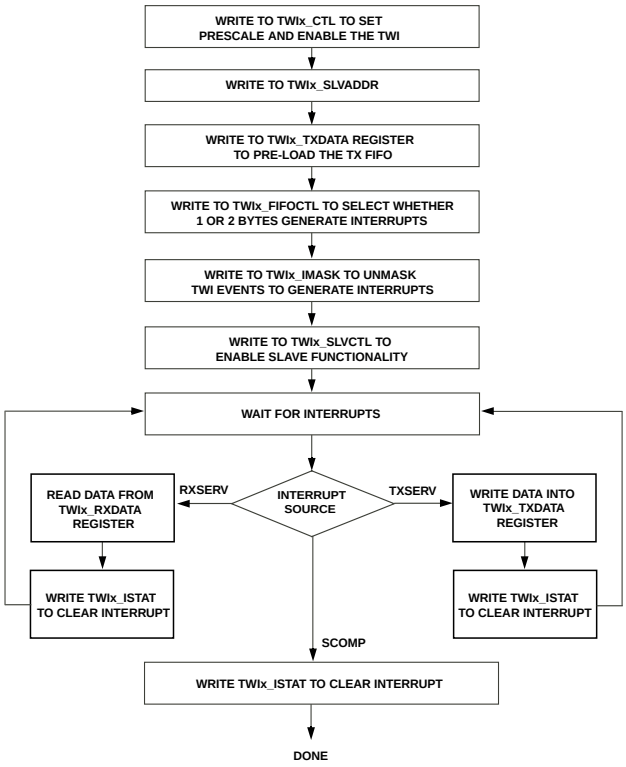


Figure 26-12: TWI Slave Mode Program Flow

Table 26-4: Slave Mode Interaction

TWI Controller	Processor
Interrupt: SINIT – Slave transfer in progress.	Acknowledge: Clear the interrupt source bits.
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt source bits. Read TWIx_FIFO_STAT. Read the receive FIFO buffer.
...	...
Interrupt: SCOMP – Slave transfer complete.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.

Master Mode Program Flow

The ***Master Mode Program Flow*** figure shows the program for the TWI in master mode.

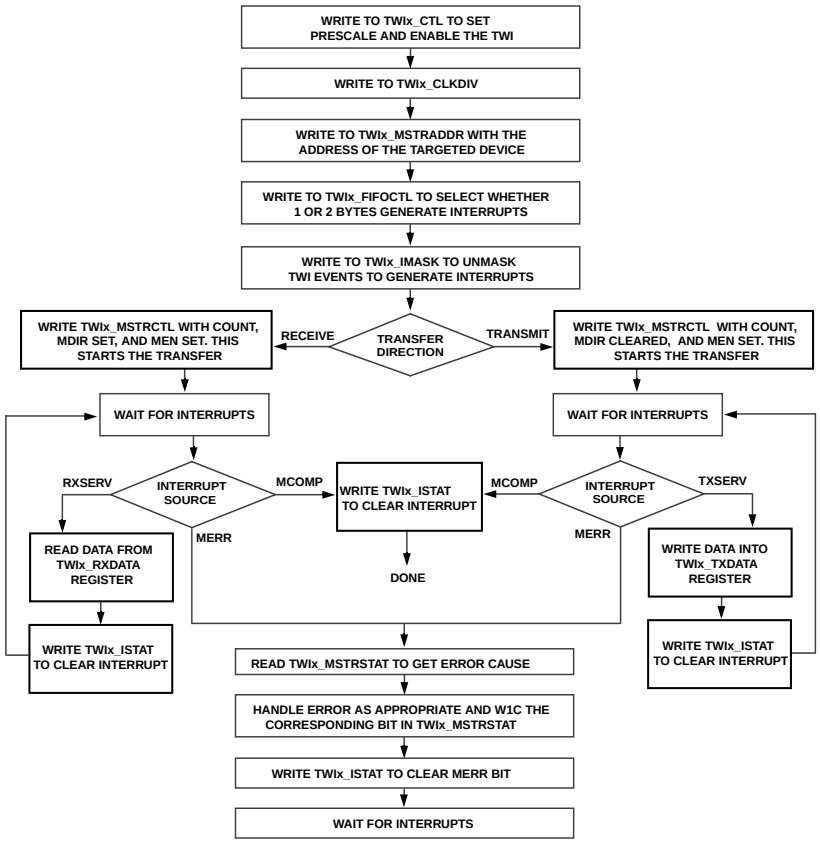


Figure 26-13: Master Mode Program Flow

Master Mode Clock Setup

Master mode operation is set up and executed on a per-transfer basis.

An example of programming steps for a receive and for a transmit is given separately in following sections. The programming step for clock setup listed here is common to both transfer types.

1. Program the TWI_CLKDIV register to define the minimum high and minimum low duration for the clock.

The TWI_CLKDIV.CLKHI and TWI_CLKDIV.CLKLO fields do not guarantee a certain frequency. Rather, they guarantee a certain minimum high and low duration for TWI_SCL. The slew rate controls falling edges. The RC time constant formed by the pull-up resistor and the SCL capacitance govern rising edges. See the “Register Descriptions” section for more details.

Master Mode Transmit

Follow these programming steps for a single master mode transmission:

1. Program the TWI_MSTRADDR register. This step defines the address transmitted during the address phase of the transfer.

2. Program the TWI_TXDATA8 or TWI_TXDATA16 register. This step configures the initial data transmitted. It is an error to complete the address phase of the transfer and not have data available in the transmit FIFO buffer.
3. Program the TWI_FIFOCTL register. The programming indicates if the transmit FIFO buffer interrupt requests occur with each byte transmitted (8-bits) or with every 2 bytes transmitted (16-bits).
4. Program the TWI_IMSK register. This step enables the bits associated with the desired interrupt request sources. For example, programming the value 0x0030 results in an interrupt output to the processor when the master transfer completes, and the master transfer has an error.
5. Program the TWI_MSTRCTL register. This step prepares and enables master mode operation. For example, programming the value 0x0201: enables master mode operation, generates a 7-bit address, sets the direction to master-transmit, uses standard mode timing, and transmits 8 data bytes before generating a stop condition.

The ***Master Mode Transmit Setup Interaction*** table represents the interaction between the TWI controller and the processor using this example.

Table 26-5: Master Mode Transmit Setup Interaction

TWI Controller	Processor
Interrupt: XMTSERV – Transmit buffer is empty.	Acknowledge: Clear the interrupt request source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt request source bits.

Master Mode Receive

Follow these programming steps for a single master mode receive.

1. Program the TWI_MSTRADDR register. This step defines the address transmitted during the address phase of the transfer.
2. Program the TWI_FIFOCTL register. This step indicates if the receive FIFO buffer interrupt requests occur with each byte received (8-bits) or with every 2 bytes received (16-bits).
3. Program the TWI_IMSK register. This step configures the enable bits associated with the desired interrupt sources. For example, programming the value 0x0030 results in an interrupt request output to the processor when the master transfer completes, and the master transfer has an error.
4. Program the TWI_MSTRCTL register. This step prepares and enables master mode operation. For example, programming the value 0x0205: enables master mode operation, generates a 7-bit address, sets the direction to master-receive, uses standard mode timing, and receives 8 data bytes before generating a stop condition.

The ***Master Mode Receive Setup Interaction*** table shows the interaction between the TWI controller and the processor using this example.

Table 26-6: Master Mode Receive Setup Interaction

TWI Controller	Processor
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt request source bits. Read the receive FIFO buffer.

NOTE: After the `TWI_MSTRCTL.DCNT` bit decrements to zero, the TWI master device sends a NAK to indicate to the slave transmitter to release the bus. This operation allows the master to send the stop signal to terminate the transfer.

CM41X_M4 TWI Register Descriptions

Two-Wire Interface (TWI) contains the following registers.

Table 26-7: CM41X_M4 TWI Register List

Name	Description
TWI_CLKDIV	SCL Clock Divider Register
TWI_CTL	Control Register
TWI_FIFOCTL	FIFO Control Register
TWI_FIFOSTAT	FIFO Status Register
TWI_IMSK	Interrupt Mask Register
TWI_ISTAT	Interrupt Status Register
TWI_MSTRADDR	Master Mode Address Register
TWI_MSTRCTL	Master Mode Control Registers
TWI_MSTRSTAT	Master Mode Status Register
TWI_RXDATA16	Rx Data Double-Byte Register
TWI_RXDATA8	Rx Data Single-Byte Register
TWI_SLVADDR	Slave Mode Address Register
TWI_SLVCTL	Slave Mode Control Register
TWI_SLVSTAT	Slave Mode Status Register
TWI_TXDATA16	Tx Data Double-Byte Register
TWI_TXDATA8	Tx Data Single-Byte Register

SCL Clock Divider Register

During master mode operation, the TWI_CLKDIV holds values, which the TWI uses to create the high and low durations of the serial clock (SCL). The clock signal SCL is an output in master mode and an input in slave mode. The values in the TWI_CLKDIV.CLKLO and TWI_CLKDIV.CLKHI fields add up to the CLKDIV value the following equation.

CLKDIV = TWI SCL period / 10 MHz time reference

Serial clock frequencies can vary from 400 KHz to less than 20 KHz. The resolution of the clock generated is 1/10 MHz or 100 ns. For example, for an SCL of 400 KHz (period = 1/400 KHz = 2500 ns) and an internal time reference of 10 MHz (period = 100 ns):

CLKDIV = 2500 ns / 100 ns = 25

For an SCL with a 30% duty cycle, use TWI_CLKDIV.CLKLO = 17 and TWI_CLKDIV.CLKHI = 8.

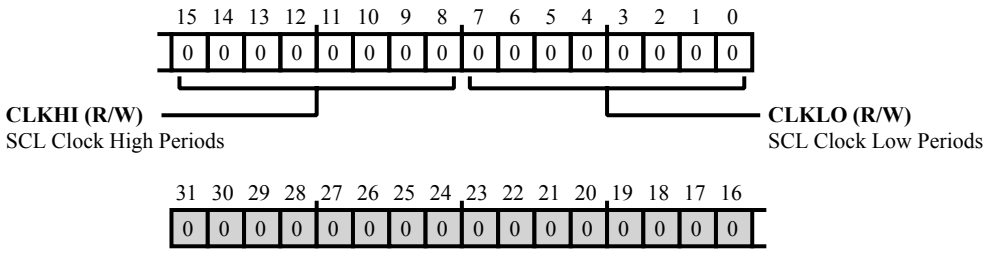


Figure 26-14: TWI_CLKDIV Register Diagram

Table 26-8: TWI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	CLKHI	SCL Clock High Periods. The TWI_CLKDIV.CLKHI specifies the number of 10 MHz time reference periods the serial clock (SCL) waits before a new clock low period begins, assuming a single master.
7:0 (R/W)	CLKLO	SCL Clock Low Periods. The TWI_CLKDIV.CLKLO specifies the number of internal time reference periods the serial clock (SCL) is held low.

Control Register

The TWI_CTL enables the TWI, establishes a relationship between the system clock (SCLK) and the TWI controller's internally timed events, and enables SCCB compatibility.

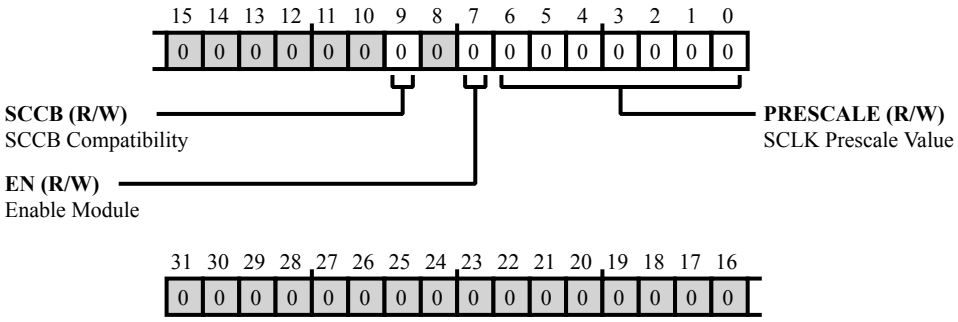


Figure 26-15: TWI_CTL Register Diagram

Table 26-9: TWI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	SCCB	SCCB Compatibility. The TWI_CTL.SCCB enables SCCB compatible operation for the TWI. SCCB compatibility is an optional feature and should not be used in an I^2C bus system. When this feature is enabled, all slave asserted acknowledgement bits are ignored by this master. This feature is valid only during transfers where the TWI is mastering an SCCB bus. Slave mode transfers should be avoided when this feature is enabled because the TWI controller always generates an acknowledge in slave mode.	
		0	Disable SCCB compatibility. When disabled, master transfers are not SCCB compatible.
		1	Enable SCCB compatibility. When enabled, master transfers are SCCB compatible. All slave-asserted acknowledgment bits are ignored by this master.
7 (R/W)	EN	Enable Module. The TWI_CTL.EN enables TWI controller operation for either master and/or slave mode of operation. It is recommended that this bit be set at the time TWI_CTL.PRESCALE is initialized and remain set. This method guarantees accurate operation of bus busy detection logic.	
		0	Disable
		1	Enable

Table 26-9: TWI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	PRESCALE	SCLK Prescale Value. The `TWI_CTL.PRESCALE` holds the pre-scaled value for the TWI internal time reference. This reference is derived from SCLK according to the formula: `TWI_CTL.PRESCALE` = f_{SCLK}/10MHz The `TWI_CTL.PRESCALE` specifies the number of system clock (SCLK) periods used in the generation of one internal time reference. The value of `TWI_CTL.PRESCALE` must be set to create an internal time reference with a period of 10 MHz. It is represented as a 7-bit binary value.

FIFO Control Register

The TWI_FIFOCTL control bits affect only the FIFO and are not tied in any way with master or slave mode operation.

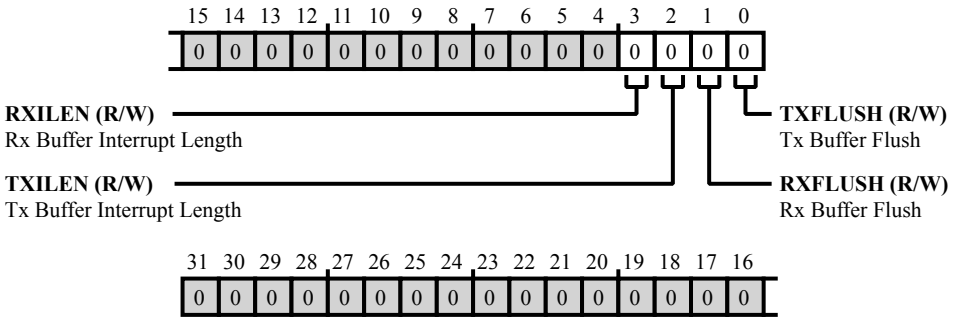


Figure 26-16: TWI_FIFOCTL Register Diagram

Table 26-10: TWI_FIFOCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	RXILEN	Rx Buffer Interrupt Length. The TWI_FIFOCTL.RXILEN determines the rate at which receive buffer interrupts are to be generated. Interrupts may be generated with each byte received or after two bytes are received. Interrupt status is available in TWI_FIFOSTAT.RXSTAT.	
		0	RXSERVI on 1 or 2 Bytes in FIFO
		1	RXSERVI on 2 Bytes in FIFO
2 (R/W)	TXILEN	Tx Buffer Interrupt Length. The TWI_FIFOCTL.TXILEN determines the rate at which transmit buffer interrupts are to be generated. Interrupts may be generated with each byte transmitted or after two bytes are transmitted. Interrupt status is available in TWI_FIFOSTAT.TXSTAT.	
		0	TXSERVI on 1 Byte of FIFO Empty
		1	TXSERVI on 2 Bytes of FIFO Empty
1 (R/W)	RXFLUSH	Rx Buffer Flush. The TWI_FIFOCTL.RXFLUSH directs the TWI to flush the contents of the receive buffer and update TWI_FIFOSTAT.RXSTAT to indicate the buffer is empty. This state is held until this bit is cleared. During an active receive, the receive buffer in this state responds to the receive logic as if it is full.	
		0	Normal Operation of Rx Buffer
		1	Flush Rx Buffer

Table 26-10: TWI_FIFOCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	TXFLUSH	Tx Buffer Flush. The `TWI_FIFOCTL.TXFLUSH` directs the TWI to flush the contents of the transmit buffer and update `TWI_FIFOSTAT.TXSTAT` to indicate the buffer is empty. This state is held until this bit is cleared. During an active transmit, the transmit buffer in this state responds to the transmit logic as if it is empty.	
		0	Normal Operation of Tx Buffer
		1	Flush Tx Buffer

FIFO Status Register

The TWI_FIFOSTAT fields indicate the state of the FIFO buffers' receive and transmit contents. The FIFO buffers do not discriminate between master data and slave data. By using the status and control bits provided, the FIFO can be managed to allow simultaneous master and slave operation.

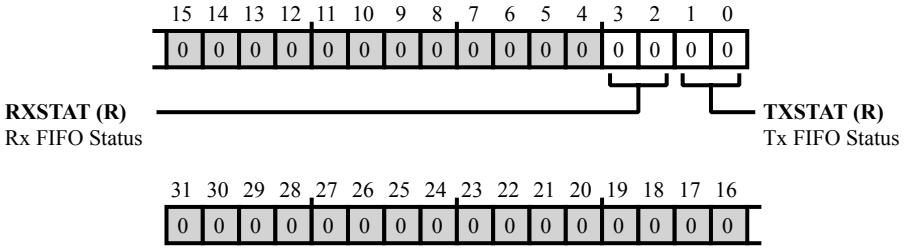


Figure 26-17: TWI_FIFOSTAT Register Diagram

Table 26-11: TWI_FIFOSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/NW)	RXSTAT	Rx FIFO Status. The read-only TWI_FIFOSTAT.RXSTAT indicates the number of valid data bytes in the receive FIFO buffer. The status is updated with each FIFO buffer read using the peripheral data bus or write access by the receive shift register. Simultaneous accesses are allowed.	
		0	Empty. The FIFO is empty.
		1	Contains 1 Byte. The FIFO contains one byte of data. A single byte peripheral read of the FIFO is allowed.
		2	Reserved
		3	Full. The FIFO is full and contains two bytes of data. Either a single or double byte peripheral read of the FIFO is allowed.
1:0 (R/NW)	TXSTAT	Tx FIFO Status. The read-only TWI_FIFOSTAT.TXSTAT field indicates the number of valid data bytes in the FIFO buffer. The status is updated with each FIFO buffer write using the peripheral data bus or read access by the transmit shift register. Simultaneous accesses are allowed.	
		0	Empty. The FIFO is empty. Either a single or double byte peripheral write of the FIFO is allowed.
		1	Contains 1 Byte. The FIFO contains one byte of data. A single byte peripheral write of the FIFO is allowed.
		2	Reserved
		3	Full. The FIFO is full and contains two bytes of data.

Interrupt Mask Register

The TWI_IMSK enables interrupt sources to assert the interrupt output. Each mask bit corresponds with one interrupt request source bit in TWI_ISTAT. Reading and writing TWI_IMSK does not affect the contents of the TWI_ISTAT.

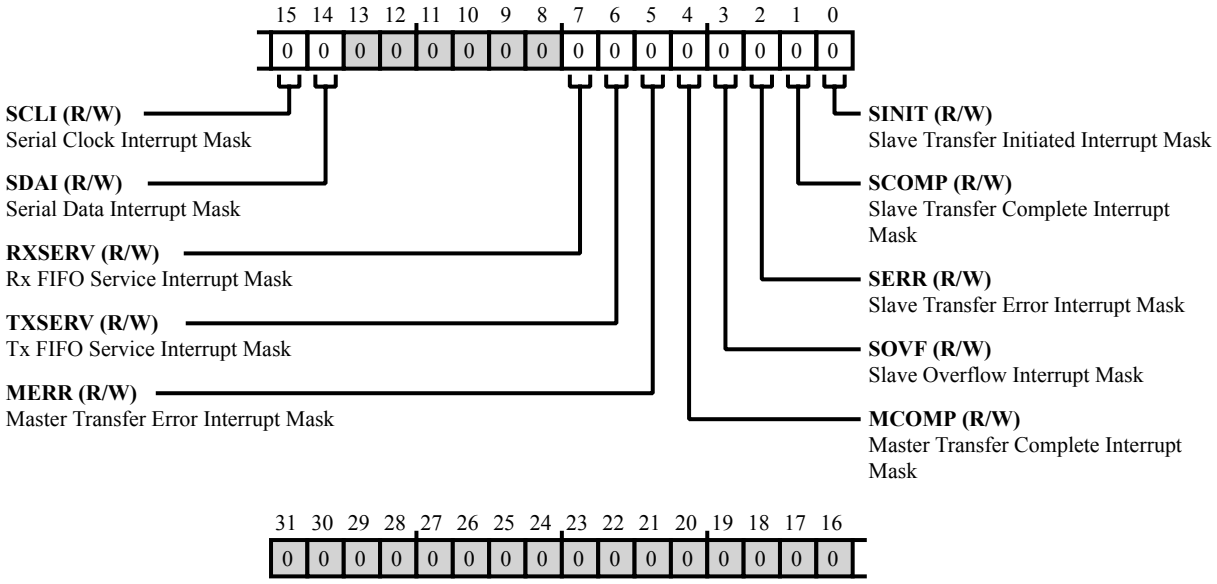


Figure 26-18: TWI_IMSK Register Diagram

Table 26-12: TWI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SCLI	Serial Clock Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
14 (R/W)	SDAI	Serial Data Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
7 (R/W)	RXSERV	Rx FIFO Service Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
6 (R/W)	TXSERV	Tx FIFO Service Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt

Table 26-12: TWI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	MERR	Master Transfer Error Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
4 (R/W)	MCOMP	Master Transfer Complete Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
3 (R/W)	SOVF	Slave Overflow Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
2 (R/W)	SERR	Slave Transfer Error Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
1 (R/W)	SCOMP	Slave Transfer Complete Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt
0 (R/W)	SINIT	Slave Transfer Initiated Interrupt Mask.	
		0	Mask (Disable) Interrupt
		1	Unmask (Enable) Interrupt

Interrupt Status Register

The TWI_ISTAT contains information about functional areas requiring servicing. Many of the bits serve as an indicator to further read and service various status registers. After servicing the interrupt source associated with a bit, the user must clear that interrupt source bit by writing a 1 to it.

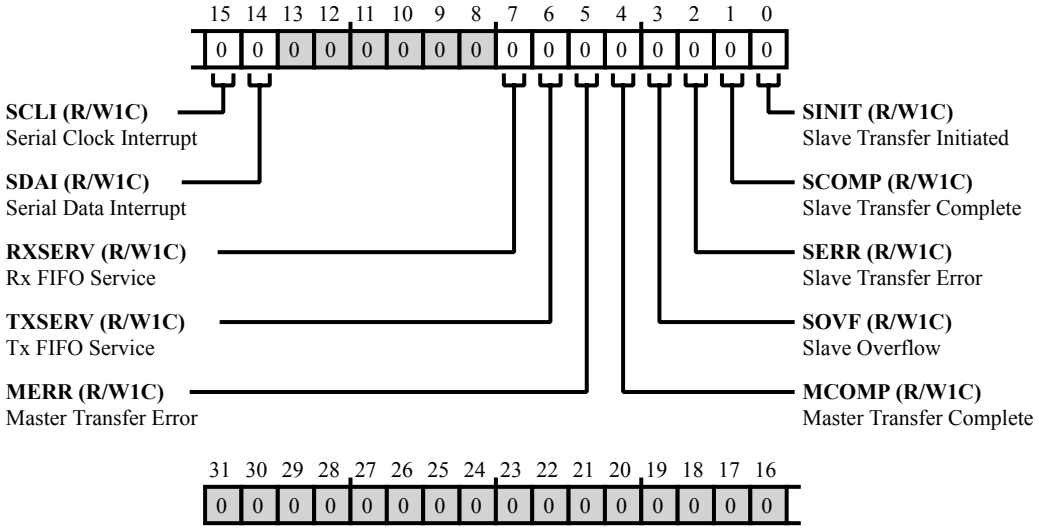


Figure 26-19: TWI_ISTAT Register Diagram

Table 26-13: TWI_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	SCLI	Serial Clock Interrupt. If the TWI is enabled (TWI_CTL.EN), SCLI is set on a high-to-low transition of the serial clock pin (SCLx). Normally, this bit is not required for I^2C bus transfers. It will be initially set on an I^2C transfer and does not require clearing.	
		0	No Interrupt. No transition was detected on the SCLx pin.
		1	Interrupt Detected. A high-to-low transition was detected on the SCLx pin. This bit is W1C.
14 (R/W1C)	SDAI	Serial Data Interrupt. If the TWI is enabled (TWI_CTL.EN), SDAI is set on a high-to-low transition of the serial data pin (SDAx). Normally, this bit is not required for I^2C bus transfers. It will be initially set on an I^2C transfer and does not require clearing.	
		0	No Interrupt. No transition was detected on the SDAx pin.
		1	Interrupt Detected. A high-to-low transition was detected on the SDAx pin. This bit is W1C.

Table 26-13: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	RXSERV	Rx FIFO Service. If TWI_FIFOCTL.RXILEN =0, the TWI_ISTAT.RXSERV is set each time the TWI_FIFOSTAT.RXSTAT field is updated to either 01 or 11. If TWI_FIFOCTL.RXILEN =1, the TWI_ISTAT.RXSERV is set each time TWI_FIFOSTAT.RXSTAT is updated to 11.	
		0	No Interrupt. The FIFO does not require servicing, or the TWI_FIFOSTAT.RXSTAT field has not changed since this bit was last cleared.
		1	Interrupt Detected. The receive FIFO buffer has one or two 8-bit words of data available to be read.
6 (R/W1C)	TXSERV	Tx FIFO Service. If TWI_FIFOCTL.TXILEN =0, the TWI_ISTAT.TXSERV is set each time the TWI_FIFOSTAT.TXSTAT field is updated to either 01 or 00. If TWI_FIFOCTL.TXILEN =1, the TWI_ISTAT.TXSERV is set each time TWI_FIFOSTAT.TXSTAT is updated to 00.	
		0	No Interrupt. FIFO does not require servicing, or the TWI_FIFOSTAT.TXSTAT field has not changed since this bit was last cleared.
		1	Interrupt Detected. The transmit FIFO buffer has one or two 8-bit locations available to be written.
5 (R/W1C)	MERR	Master Transfer Error. The TWI_ISTAT.MERR indicates that a master error has occurred. The conditions surrounding the error are indicated by the master status register (TWI_MSTRSTAT).	
		0	No Interrupt
		1	Interrupt Detected
4 (R/W1C)	MCOMP	Master Transfer Complete. The TWI_ISTAT.MCOMP indicates that the initiated master transfer has completed. In the absence of a repeat start, the bus has been released.	
		0	No Interrupt
		1	Interrupt Detected
3 (R/W1C)	SOVF	Slave Overflow. The TWI_ISTAT.SOVF indicates that the TWI_ISTAT.SCOMP bit was set at the time a subsequent transfer has acknowledged an address phase. The transfer continues, however, it may be difficult to delineate data of one transfer from another.	
		0	No Interrupt
		1	Interrupt Detected

Table 26-13: TWI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	SERR	Slave Transfer Error. The `TWI_ISTAT.SERR` indicates that a slave error has occurred. A restart or stop condition has occurred during the data receive phase of a transfer.	
		0	No Interrupt
		1	Interrupt Detected
1 (R/W1C)	SCOMP	Slave Transfer Complete. The `TWI_ISTAT.SCOMP` indicates that the transfer is complete and either a stop, or a restart was detected.	
		0	No Interrupt
		1	Interrupt Detected
0 (R/W1C)	SINIT	Slave Transfer Initiated. The `TWI_ISTAT.SINIT` indicates whether or not a slave transfer is in progress.	
		0	No Interrupt. A transfer is not in progress, or an address match has not occurred since the last time this bit was cleared.
		1	Interrupt Detected. The slave has detected an address match, and a transfer has been initiated.

Master Mode Address Register

During the addressing phase of a transfer, the TWI controller, with its master enabled, transmits the contents of TWI_MSTRADDR. When programming this register, omit the read/write bit. That is, only the upper 7 bits that make up the slave address should be written to this register. For example, if the slave address is b#1010000X, where X is the read/write bit, the TWI_MSTRADDR is programmed with b#1010000, which corresponds to 0x50. When sending out the address on the bus, the TWI controller appends the read/write bit as appropriate based on the state of the TWI_MSTRCTL.DIR bit.

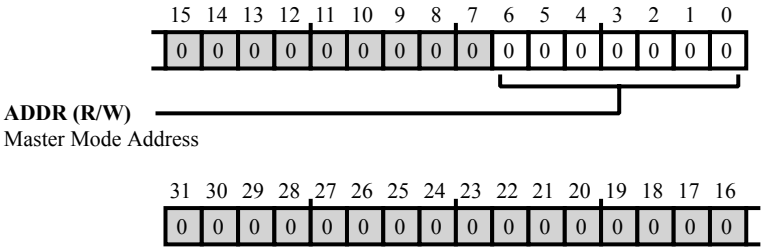


Figure 26-20: TWI_MSTRADDR Register Diagram

Table 26-14: TWI_MSTRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Master Mode Address.

Master Mode Control Registers

The TWI_MSTRCTL controls the logic associated with master mode operation. Bits in this register do not affect slave mode operation and should not be modified to control slave mode functionality.

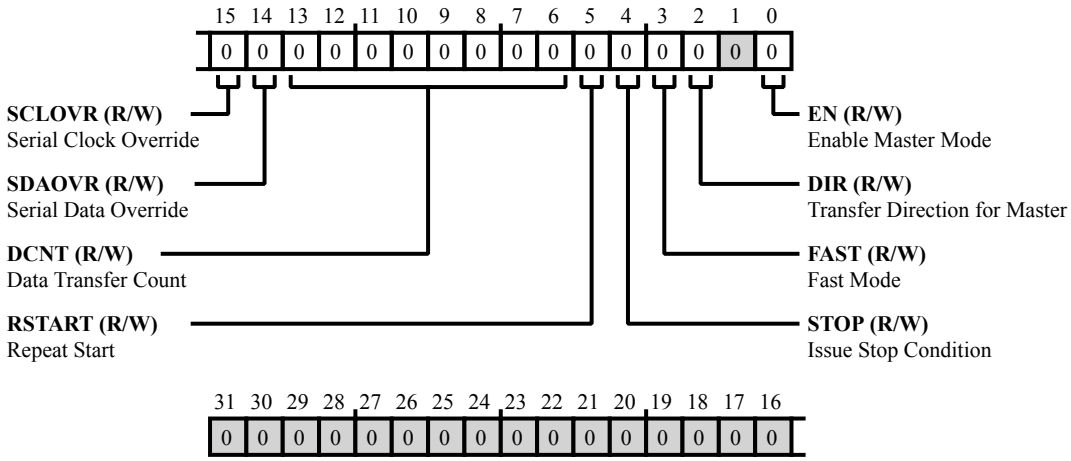


Figure 26-21: TWI_MSTRCTL Register Diagram

Table 26-15: TWI_MSTRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SCLOVR	Serial Clock Override. The TWI_MSTRCTL.SCLOVR provides direct control of the serial clock line when required. Normal master and slave mode operation should not require override operation. When TWI_MSTRCTL.SCLOVR is set, the TWI overrides normal serial clock output, driving it to an active 0 level and overriding all other logic. This state is held until this bit is cleared. When TWI_MSTRCTL.SCLOVR is cleared, the TWI permits normal serial clock operation under the control of master mode clock generation and slave mode clock stretching logic.	
		0	Permit Normal SCL Operation
		1	Override Normal SCL Operation
14 (R/W)	SDAOVR	Serial Data Override. The TWI_MSTRCTL.SDAOVR provides direct control of the serial data line when required. Normal master and slave mode operation should not require override operation. When TWI_MSTRCTL.SDAOVR is set, the TWI overrides normal serial data operation under the control of the transmit shift register and acknowledge logic, driving serial data output to an active 0 level and overriding all other logic. This state is held until this bit is cleared. When TWI_MSTRCTL.SDAOVR is cleared, the TWI permits normal serial data operation.	
		0	Permit Normal SDA Operation
		1	Override Normal SDA Operation

Table 26-15: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:6 (R/W)	DCNT	Data Transfer Count. The TWI_MSTRCTL.DCNT indicates the number of data bytes to transfer. As each data word is transferred, the TWI decrements this counter. When TWI_MSTRCTL.DCNT decrements to 0, a stop condition is generated. Setting TWI_MSTRCTL.DCNT to 0xFF disables the counter. In this transfer mode, data continues to be transferred until it is concluded by setting the TWI_MSTRCTL.STOP bit. In the event a master transmit is aborted due to a slave data NAK, the value of TWI_MSTRCTL.DCNT equals the number of bytes not sent. The byte which was NAK'ed by the slave is counted as a sent byte.	
5 (R/W)	RSTART	Repeat Start. The TWI_MSTRCTL.RSTART enables the TWI to issue a repeat start condition at the conclusion of the current transfer (TWI_MSTRCTL.DCNT =0) and begin the next transfer. The current transfer concludes with updates to the appropriate status and interrupt bits. If errors occurred during the previous transfer, a repeat start does not occur. In the absence of any errors, master enable (TWI_MSTRCTL.EN) does not self clear on a repeat start.	
		0	Disable Repeat Start
		1	Enable Repeat Start
4 (R/W)	STOP	Issue Stop Condition. The TWI_MSTRCTL.STOP directs the TWI to issue a stop condition. The transfer concludes as soon as possible avoiding any error conditions (as if data transfer count had been reached). At that time, the TWI_IMSK is updated along with any associated status bits.	
		0	Permit Normal Operation
		1	Issue Stop
3 (R/W)	FAST	Fast Mode. The TWI_MSTRCTL.FAST selects whether the TWI operates in fast mode or standard mode. In fast mode, the TWI uses timing specifications for transfers at up to 400K bits/s. In standard mode, the TWI uses timing specifications for transfers at up to 100K bits/s.	
		0	Select Standard Mode
		1	Select Fast Mode
2 (R/W)	DIR	Transfer Direction for Master. The TWI_MSTRCTL.DIR selects the transfer direction for the TWI as master initiated receive or transmit.	
		0	Master Transmit
		1	Master Receive

Table 26-15: TWI_MSTRCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable Master Mode. The `TWI_MSTRCTL.EN` enables master mode functionality. A start condition is generated if the bus is idle. This bit self clears at the completion of a transfer (after `TWI_MSTRCTL.DCNT` decrements to zero), including transfers terminated due to errors. If disabled (=0) during operation, the transfer is aborted, and all logic associated with master mode transfers are reset. Serial data and serial clock (SDA, SCL) are no longer driven. Write-1-to-clear status bits are not affected.	
		0	Disable
		1	Enable

Master Mode Status Register

The TWI_MSTRSTAT holds information during master mode transfers and at their conclusion. Generally, master mode status bits are not directly associated with the generation of interrupt requests, but these bits offer information on the current transfer. Slave mode operation does not affect master mode status bits.

Note that while TWI_MSTRSTAT.SCLSEN is set (this condition could be due to having no pull-up resistor on TWI_SCL or another agent is driving TWI_SCL low), the acknowledge bits (TWI_MSTRSTAT.ANAK and TWI_MSTRSTAT.DNAK) do not update. This result occurs because the acknowledge conditions are sampled during the high phase of TWI_SCL.

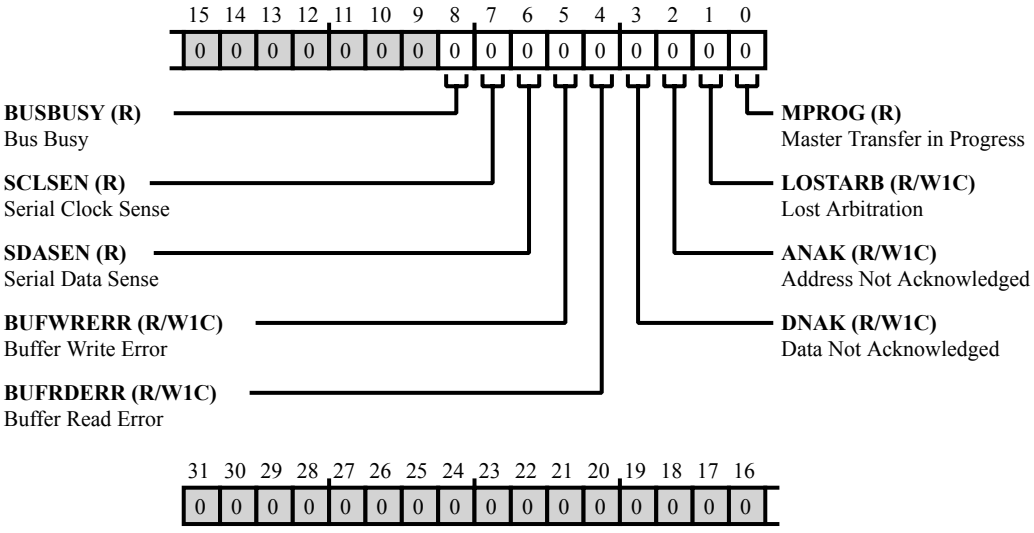


Figure 26-22: TWI_MSTRSTAT Register Diagram

Table 26-16: TWI_MSTRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	BUSBUSY	Bus Busy. The TWI_MSTRSTAT.BUSBUSY indicates whether the bus is currently busy or free. This indication is not limited to only this device but is for all devices. On a start condition, the setting of the register value is delayed due to the input filtering. On a stop condition the clearing of the register value occurs after t_{BUF}.	
		0	Bus Free. The bus is free. The clock and data bus signals have been inactive for the appropriate bus free time.
		1	Bus Busy. The bus is busy. Clock or data activity has been detected.

Table 26-16: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	SCLSEN	Serial Clock Sense. The `TWI_MSTRSTAT.SCLSEN` indicates the active or inactive state of the serial clock. Use this status bit when direct sensing of the serial clock line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature.	
		0	SCL Inactive "One". An inactive "one" is being sensed on the serial clock.
		1	SCL Active "Zero". An active "zero" is being sensed on the serial clock. The source of the active driver is not known and can be internal or external.
6 (R/NW)	SDASEN	Serial Data Sense. The `TWI_MSTRSTAT.SDASEN` indicates the active or inactive status of the serial data. Use this status bit when direct sensing of the serial data line is required. The register value is delayed due to the input filter (nominally 50 ns). Normal master and slave mode operation should not require this feature.	
		0	SDA Inactive "One". An inactive "one" is currently being sensed on the serial data line.
		1	SDA Active "Zero". An active "zero" is currently being sensed on the serial data line. The source of the active driver is not known and can be internal or external.
5 (R/W1C)	BUFWRERR	Buffer Write Error. The `TWI_MSTRSTAT.BUFWRERR` indicates whether the current master transfer was aborted due to a receive buffer write error. The receive buffer and receive shift register were both full at the same time. This bit is W1C.	
		0	No Status
		1	Buffer Write Error
4 (R/W1C)	BUFRDERR	Buffer Read Error. The `TWI_MSTRSTAT.BUFRDERR` indicates whether the current master transfer was aborted due to the detection of a NAK during data transmission. This bit is W1C.	
		0	No Status
		1	Buffer Read Error
3 (R/W1C)	DNAK	Data Not Acknowledged. The `TWI_MSTRSTAT.DNAK` indicates whether the current master transfer was aborted due to the detection of a NAK during data transmission. This bit is W1C.	
		0	No Status
		1	Data NAK

Table 26-16: TWI_MSTRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	ANAK	Address Not Acknowledged. The `TWI_MSTRSTAT.ANAK` indicates whether the current master transfer was aborted due to the detection of a NAK during the address phase of the transfer. This bit is W1C.	
		0	No Status
		1	Address NAK
1 (R/W1C)	LOSTARB	Lost Arbitration. The `TWI_MSTRSTAT.LOSTARB` indicates whether the current transfer was aborted due to the loss of arbitration with another master. This bit is W1C.	
		0	No Status
		1	Lost Arbitration
0 (R/NW)	MPROG	Master Transfer in Progress. The `TWI_MSTRSTAT.MPROG` indicates whether or not a master transfer is in progress. If clear (`TWI_MSTRSTAT.MPROG` =0), currently no transfer is taking place. This can occur after a transfer is complete or while an enabled master is waiting for an idle bus.	
		0	No Status
		1	Master Transfer in Progress

Rx Data Double-Byte Register

The TWI_RXDATA16 holds a 16-bit data value read from the FIFO buffer. To reduce interrupt output rates and peripheral bus access times, a double byte receive data access can be performed. Two data bytes can be read, effectively emptying the receive FIFO buffer with a single access.

The data is read in little endian byte order, where byte 0 is the first byte received and byte 1 is the second byte received. With each access, the receive status (TWI_FIFOSTAT.RXSTAT) field is updated to indicate it is empty. If an access is performed while the FIFO buffer is not full, the read data is unknown and the existing FIFO buffer data and its status remains unchanged.

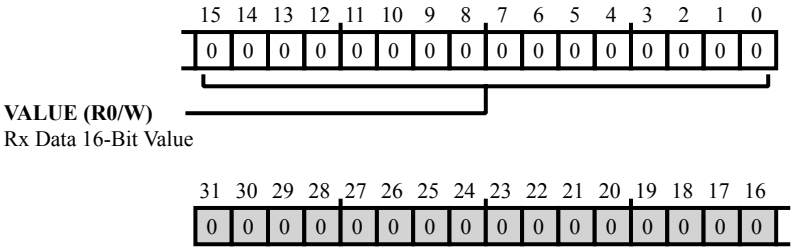


Figure 26-23: TWI_RXDATA16 Register Diagram

Table 26-17: TWI_RXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Rx Data 16-Bit Value.

Rx Data Single-Byte Register

The TWI_RXDATA8 holds an 8-bit data value read from the FIFO buffer. Receive data is read from the corresponding receive buffer in a first-in first-out order. Although peripheral bus reads are 16 bits, a read access to TWI_RXDATA8 accesses only one transmit data byte from the FIFO buffer. With each access, the receive status (TWI_FIFOSTAT.RXSTAT) field is updated. If an access is performed while the FIFO buffer is empty, the data is unknown and the FIFO buffer status remains indicating it is empty.

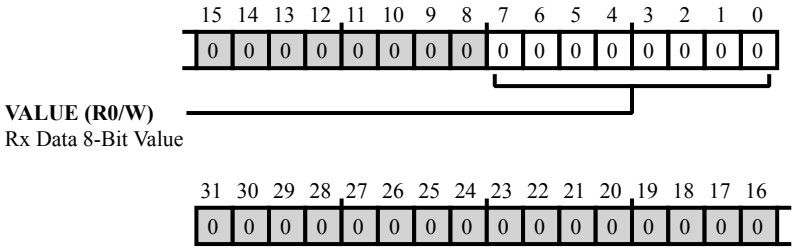


Figure 26-24: TWI_RXDATA8 Register Diagram

Table 26-18: TWI_RXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Rx Data 8-Bit Value.

Slave Mode Address Register

The TWI_SLVADDR holds the slave mode address, which is the valid address to which the slave-enabled TWI controller responds. The TWI controller compares this value with the received address during the addressing phase of a transfer.

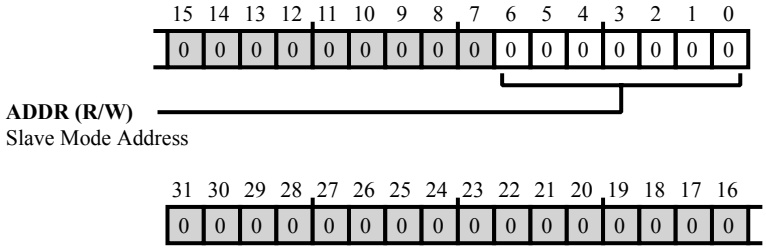


Figure 26-25: TWI_SLVADDR Register Diagram

Table 26-19: TWI_SLVADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	ADDR	Slave Mode Address.

Slave Mode Control Register

The TWI_SLVCTL controls the logic associated with slave mode operation. Settings in this register do not affect master mode operation and should not be modified to control master mode functionality.

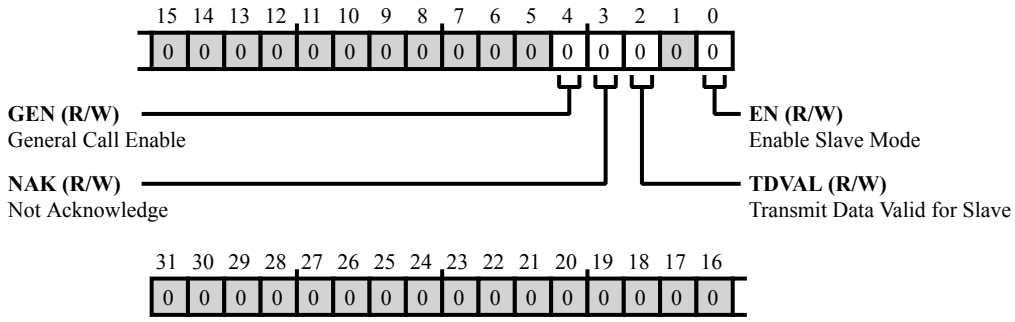


Figure 26-26: TWI_SLVCTL Register Diagram

Table 26-20: TWI_SLVCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	GEN	General Call Enable. The TWI_SLVCTL.GEN enables general call address matching. When enabled, a general call slave receive transfer is accepted. All status and interrupt source bits associated with transfers are updated. Note that general call address detection is available only when slave mode is enabled.	
		0	Disable General Call Matching
		1	Enable General Call Matching
3 (R/W)	NAK	Not Acknowledge. The TWI_SLVCTL.NAK directs the TWI to generate a NAK (if set) or an ACK (if cleared) at the conclusion of data transfer for slave receive. For NAK, the slave is still considered to be addressed at the conclusion of transfer.	
		0	Generate ACK
		1	Generate NAK
2 (R/W)	TDVAL	Transmit Data Valid for Slave. The TWI_SLVCTL.TDVAL selects whether the data in the transmit FIFO is available (valid) for slave transmission (TWI_SLVCTL.TDVAL set). If the FIFO data is not available (invalid) for slave transmission (TWI_SLVCTL.TDVAL cleared), the data in the transmit FIFO is for master mode transmits, and the data is not allowed to be used during a slave transmit; the transmit FIFO is treated as if it is empty.	
		0	Data Invalid for Slave Tx
		1	Data Valid for Slave Tx

Table 26-20: TWI_SLVCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable Slave Mode. The `TWI_SLVCTL.EN` enables slave operation. Enabling slave and master modes of operation concurrently is allowed. If disabled, no attempt is made to identify a valid address. If `TWI_SLVCTL.EN` is cleared during a valid transfer, clock stretching ceases, the serial data line is released, and the current byte is not acknowledged.	
		0	Disable
		1	Enable

Slave Mode Status Register

During and at the conclusion of register slave mode transfers, the TWI_SLVSTAT holds information on the current transfer. Generally slave mode status bits are not associated with the generation of interrupt requests. Master mode operation does not affect slave mode status bits.

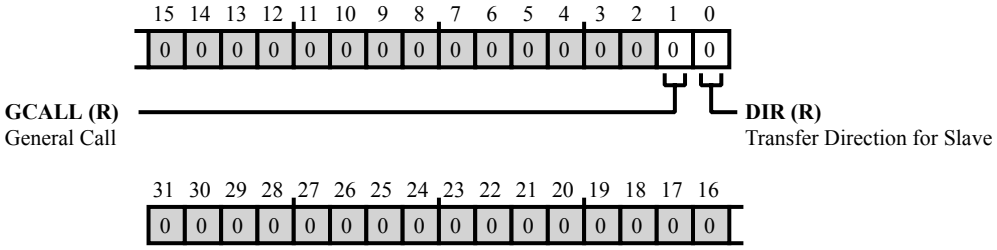


Figure 26-27: TWI_SLVSTAT Register Diagram

Table 26-21: TWI_SLVSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	GCALL	General Call. The TWI_SLVSTAT.GCALL indicates whether or not--at the time of addressing--the address was determined to be a general call. This bit self clears if slave mode is disabled (TWI_SLVCTL.EN =0).	
		0	Not a General Call Address
		1	General Call Address
0 (R/NW)	DIR	Transfer Direction for Slave. The TWI_SLVSTAT.DIR indicates whether--at the time of addressing--the transfer direction was determined to be slave transmit or receive. This bit self clears if slave mode is disabled (TWI_SLVCTL.EN =0).	
		0	Slave Receive
		1	Slave Transmit

Tx Data Double-Byte Register

The TWI_TXDATA16 register holds a 16-bit data value written into the FIFO buffer. To reduce interrupt latency output rates and peripheral bus access times, a double byte transfer data access can be done. Two data bytes can be written, effectively filling the transmit FIFO buffer with a single access.

The data is written in little endian byte order, where byte 0 is the first byte to be transferred and byte 1 is the second byte to be transferred. With each access, the transmit status (TWI_FIFOSTAT.TXSTAT) field is updated. If an access is performed while the FIFO buffer is not empty, the write is ignored and the existing FIFO buffer data and its status remains unchanged. This register when read back returns zero.

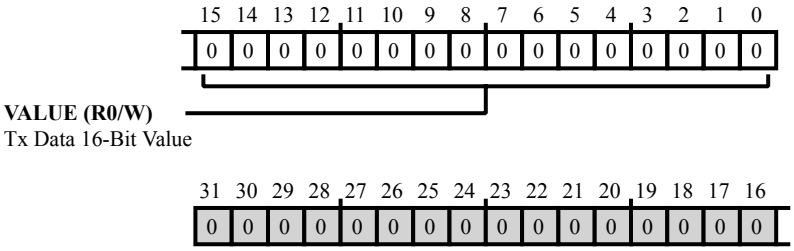


Figure 26-28: TWI_TXDATA16 Register Diagram

Table 26-22: TWI_TXDATA16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R0/W)	VALUE	Tx Data 16-Bit Value.

Tx Data Single-Byte Register

The TWI_TXDATA8 register holds an 8-bit data value written into the FIFO buffer. Transmit data is entered into the corresponding transmit buffer in a first-in first-out order. For 16-bit peripheral bus writes, a write access to this register adds only one transmit data byte to the FIFO buffer. With each access, the transmit status (TWI_FIFOSTAT.TXSTAT) field is updated. If an access is performed while the FIFO buffer is full, the write is ignored and the existing FIFO buffer data and its status remains unchanged. This register returns zero when read back.

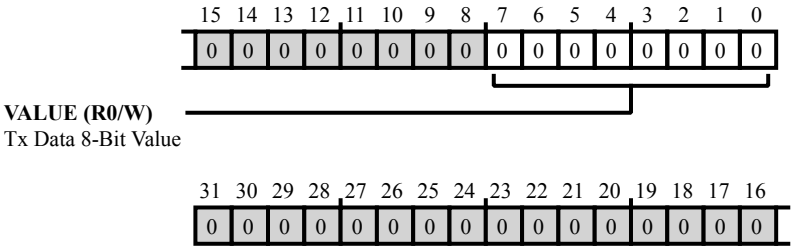


Figure 26-29: TWI_TXDATA8 Register Diagram

Table 26-23: TWI_TXDATA8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	VALUE	Tx Data 8-Bit Value.

27 Controller Area Network (CAN)

The processor contains a Controller Area Network (CAN) module based on the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. The CAN protocol is compatible with the control applications. It can communicate reliably over a network and incorporates CRC checking, message error tracking, and fault node confinement.

NOTE: This document assumes familiarity with the CAN standard. For more information, refer to Version 2.0 of the CAN specification from Robert Bosch GmbH.

CAN Features

Key features of the CAN module include:

- Conformity to the CAN 2.0B (active) standard
- Dedicated acceptance mask for each mailbox
- Support for data rates of up to 1M bit/s
- Support for standard (11-bit) and extended (29-bit) identifiers
- 32 mailboxes (8 transmit, 8 receive, 16 configurable)
- Data filtering (first 2 bytes) for acceptance filtering (DeviceNet™ mode)
- Error status and warning registers
- Universal counter-module
- Readable receive and transmit pin values
- Support for remote frames
- Active or passive network support
- Interrupts, including transmit or receive complete, error, and global
- Clock derived from SCLK through a programmable divider, eliminating the need for an extra crystal

CAN Functional Description

The following sections provide information on the functional operation of the CAN module. This section also provides listings of the CAN registers and interrupts.

CM41X_M4 CAN Register List

The controller area network (CAN) module implements the CAN 2.0B (active) protocol. This protocol is an asynchronous communications protocol used in both industrial and automotive control systems. A set of registers govern CAN operations. For more information on CAN functionality, see the CAN register descriptions.

Table 27-1: CM41X_M4 CAN Register List

Name	Description
CAN_AA1	Abort Acknowledge 1 Register
CAN_AA2	Abort Acknowledge 2 Register
CAN_AM[nn]H	Acceptance Mask (H) Register
CAN_AM[nn]L	Acceptance Mask (L) Register
CAN_CEC	Error Counter Register
CAN_CLK	Clock Register
CAN_CTL	CAN Master Control Register
CAN_DBG	Debug Register
CAN_ESR	Error Status Register
CAN_EWR	Error Counter Warning Level Register
CAN_GIF	Global CAN Interrupt Flag Register
CAN_GIM	Global CAN Interrupt Mask Register
CAN_GIS	Global CAN Interrupt Status Register
CAN_INT	Interrupt Pending Register
CAN_MBIM1	Mailbox Interrupt Mask 1 Register
CAN_MBIM2	Mailbox Interrupt Mask 2 Register
CAN_MBRIF1	Mailbox Receive Interrupt Flag 1 Register
CAN_MBRIF2	Mailbox Receive Interrupt Flag 2 Register
CAN_MBTD	Temporary Mailbox Disable Register
CAN_MBTIF1	Mailbox Transmit Interrupt Flag 1 Register
CAN_MBTIF2	Mailbox Transmit Interrupt Flag 2 Register
CAN_MB[nn]_DATA0	Mailbox Word 0 Register
CAN_MB[nn]_DATA1	Mailbox Word 1 Register

Table 27-1: CM41X_M4 CAN Register List (Continued)

Name	Description
CAN_MB[nn]_DATA2	Mailbox Word 2 Register
CAN_MB[nn]_DATA3	Mailbox Word 3 Register
CAN_MB[nn]_ID0	Mailbox ID 0 Register
CAN_MB[nn]_ID1	Mailbox ID 1 Register
CAN_MB[nn]_LENGTH	Mailbox Length Register
CAN_MB[nn]_TIMESTAMP	Mailbox Time Stamp Register
CAN_MC1	Mailbox Configuration 1 Register
CAN_MC2	Mailbox Configuration 2 Register
CAN_MD1	Mailbox Direction 1 Register
CAN_MD2	Mailbox Direction 2 Register
CAN_OPSS1	Overwrite Protection/Single Shot Transmission 1 Register
CAN_OPSS2	Overwrite Protection/Single Shot Transmission 2 Register
CAN_RFH1	Remote Frame Handling 1 Register
CAN_RFH2	Remote Frame Handling 2 Register
CAN_RML1	Receive Message Lost 1 Register
CAN_RML2	Receive Message Lost 2 Register
CAN_RMP1	Receive Message Pending 1 Register
CAN_RMP2	Receive Message Pending 2 Register
CAN_STAT	Status Register
CAN_TA1	Transmission Acknowledge 1 Register
CAN_TA2	Transmission Acknowledge 2 Register
CAN_TIMING	Timing Register
CAN_TRR1	Transmission Request Reset 1 Register
CAN_TRR2	Transmission Request Reset 2 Register
CAN_TRS1	Transmission Request Set 1 Register
CAN_TRS2	Transmission Request Set 2 Register
CAN_UCCNF	Universal Counter Configuration Mode Register
CAN_UCCNT	Universal Counter Register
CAN_UCRC	Universal Counter Reload/Capture Register

CM41X_M4 CAN Interrupt List

Table 27-2: CM41X_M4 CAN Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
146	CAN1_TX	CAN1 Transmit	Level	
147	CAN1_RX	CAN1 Receive	Level	
148	CAN1_STAT	CAN1 Status	Level	
149	CAN0_TX	CAN0 Transmit	Level	
150	CAN0_RX	CAN0 Receive	Level	
151	CAN0_STAT	CAN0 Status	Level	

CM41X_M0 CAN Interrupt List

Table 27-3: CM41X_M0 CAN Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
28	CAN0_RX	CAN0 Receive	Level	
28	CAN0_STAT	CAN0 Status	Level	
28	CAN0_TX	CAN0 Transmit	Level	

CM41X_M0 CAN Trigger List

Table 27-4: CM41X_M0 CAN Trigger List Masters

Trigger ID	Name	Description	Sensitivity
19	CAN0_STAT	CAN0 Status	Level

Table 27-5: CM41X_M0 CAN Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

CM41X_M4 CAN Trigger List

Table 27-6: CM41X_M4 CAN Trigger List Masters

Trigger ID	Name	Description	Sensitivity
53	CAN0_STAT	CAN0 Status	Level
54	CAN1_STAT	CAN1 Status	Level

Table 27-7: CM41X_M4 CAN Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

External Interface

The interface to the CAN bus is a simple two-wire line. The ***Representation of CAN Transceiver Interconnection*** shows a symbolic representation of the CAN transceiver interconnection. Typically, the CAN_TX output and CAN_RX input pins of the processor connect to an external CAN CAN_TX and CAN_RX pins (respectively) of the tranceiver. The CAN_TX and CAN_RX pins operate with TTL levels and are appropriate for operation with CAN bus transceivers according to ISO/DIS 11898.

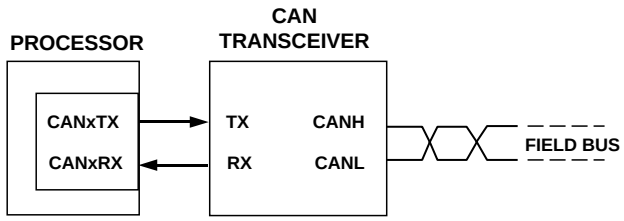


Figure 27-1: Representation of CAN Transceiver Interconnection

CAN data is either dominant (logic 0) or recessive (logic 1). The default state of the CAN_TX output is recessive.

Architectural Concepts

The full CAN controller features 32 message buffers called mailboxes. Eight mailboxes are dedicated for message transmission, eight are for reception, and 16 are programmable in direction.

The CAN module architecture is based around a 32-entry mailbox RAM. The CAN serial interface or the processor core accesses the mailbox sequentially. Each mailbox consists of eight 16-bit control and data registers and two optional 16-bit acceptance mask registers. Configure all of these registers before enabling the mailbox.

Since the mailbox area is implemented as RAM, the reset values of these registers are undefined. The ***CAN Mailbox Area*** figure shows the mailbox area. The data is divided into fields, which include a message identifier, a time stamp, a byte count, up to 8 bytes of data, and several control bits.

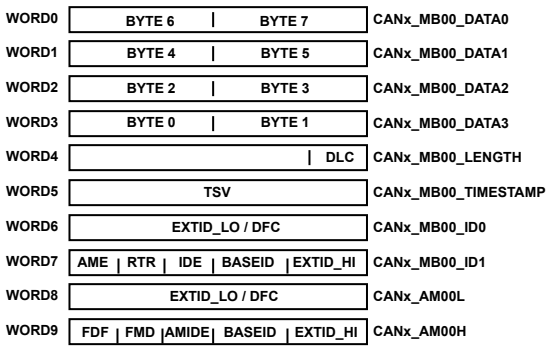


Figure 27-2: CAN Mailbox Area

The CAN mailbox identification register pair (CAN_MB[nn]_ID0/1) includes:

- The 29-bit identifier (base part CAN_AM[nn]H.BASEID plus the extended part CAN_AM[nn]L.EXTID/ CAN_AM[nn]H.EXTID)
- The acceptance mask enable bit (CAN_MB[nn]_ID1.AME)
- The remote transmission request bit (CAN_MB[nn]_ID1.RTR)
- The identifier extension bit (CAN_MB[nn]_ID1.IDE)

NOTE: Do not write to the identifier of a message object while the mailbox is enabled for the CAN module (the corresponding bit in CAN_MC1 is set).

The other mailbox area registers and bits are:

- The data length code bit (CAN_MB[nn]_LENGTH.DLC). The upper 12 bits of this register for each mailbox are marked as reserved. Always, set these 12 bits to zero.
- The mailbox word registers (CAN_MB[nn]_DATA0/1/2/3) supply up to 8 bytes for the data field. The data is sent MSB first based on the number of bytes defined in the CAN_MB[nn]_LENGTH.DLC bit. For example, if only one byte is transmitted or received (CAN_MB[nn]_LENGTH.DLC=1), then it is stored in the most significant byte of the CAN_MB[nn]_DATA3 register.
- The time stamp value bits (CAN_MB[nn]_TIMESTAMP.TSV)

The final registers in the mailbox area are the acceptance mask registers (CAN_AM[nn]H and CAN_AM[nn]L). The acceptance mask is enabled when the CAN_MB[nn]_ID1.AME bit is set.

Setting the CAN_CTL.DNM and CAN_AM[nn]H.FDF bits enables the *filtering on data field* option. When enabled, the CAN_MB[nn]_ID0.EXTID[15:0] bits are reused as acceptance code (DFC) for the data field filtering.

Block Diagram

The ***CAN Controller Block Diagram*** figure shows a block diagram of the CAN module.

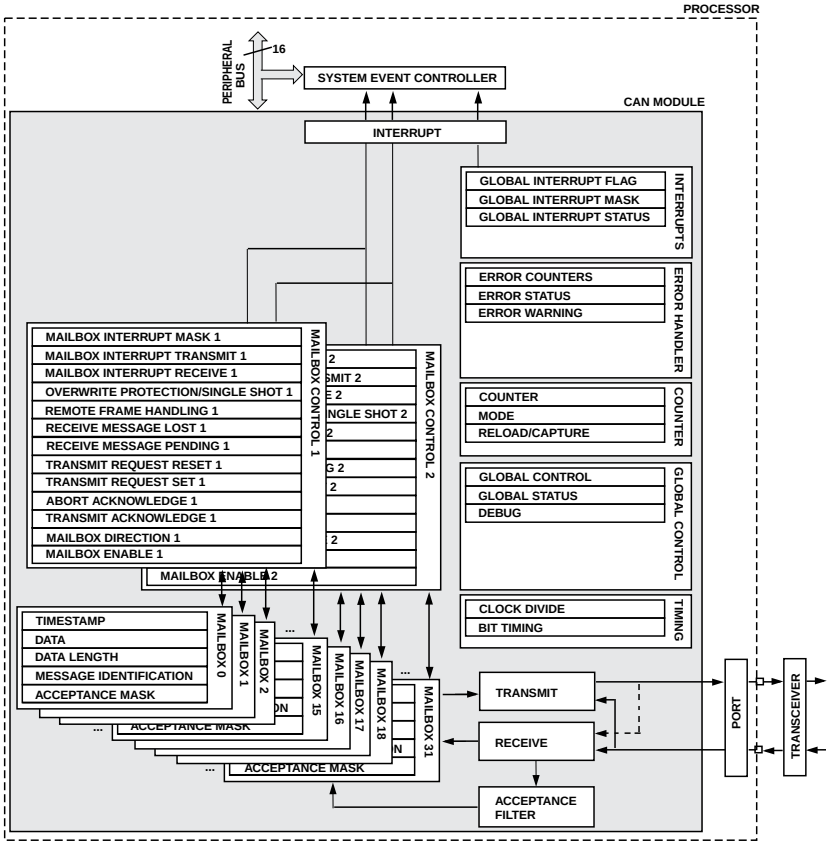


Figure 27-3: CAN Controller Block Diagram

Mailbox Control

Mailbox control memory-mapped registers (MMRs) function as control and status registers for the 32 mailboxes. Each bit in these registers represents one specific mailbox. Since CAN MMRs are all 16 bits wide, pairs of registers manage certain functionality for all 32 individual mailboxes. Mailboxes 0–15 are configured or monitored in registers with a suffix of 1. Similarly, mailboxes 16–31 use the same named register with a suffix of 2. For example, the CAN mailbox direction registers (`CAN_MD1` / `CAN_MD2`) control mailboxes. See the ***CAN Mailbox Register Pair*** figure. The ***CAN Register List*** table shows the mailbox control registers.

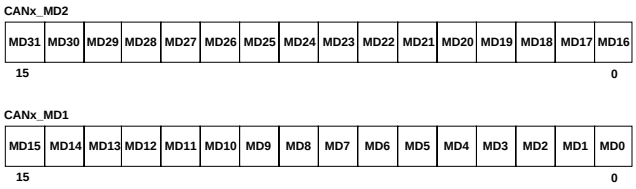


Figure 27-4: CAN Mailbox Register Pair

Mailboxes 24–31 support transmit operation only and mailboxes 0–7 are receive-only mailboxes. Therefore, the lower 8 bits in the 1 registers and the upper 8 bits in the 2 registers are sometimes reserved or are restricted in their use.

Protocol Fundamentals

Although the CAN_RX and CAN_TX pins are TTL-compliant signals, the CAN signals beyond the transceiver have asymmetric drivers. A low state on the CAN_TX pin activates strong drivers while a high state activates weak drivers. So, the active low state is the *dominant* state and the active high state is the *recessive* state. If the CAN module is passive, the CAN_TX pin is always high. If two CAN nodes transmit at the same time, dominant bits overwrite recessive bits.

The CAN protocol specifies that all nodes trying to send a message on the CAN bus attempt to send a frame once the bus is available. The ***Standard CAN Frame*** figure shows the frame. The Start of Frame (SOF) indicator signals the beginning of a new frame. Each CAN node then begins transmitting its message starting with the message ID.

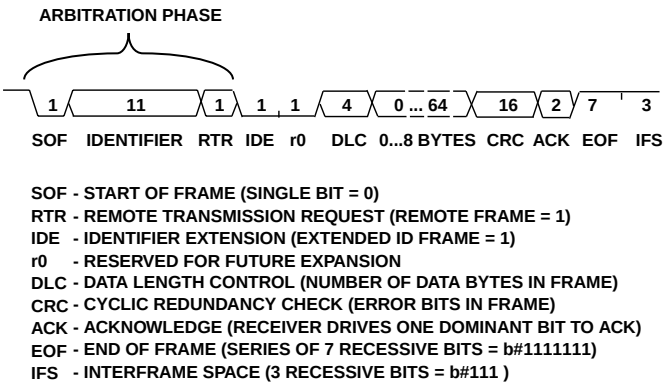


Figure 27-5: Standard CAN Frame

While transmitting, the CAN controller samples the CAN_RX pin to verify that the driven logic level is the value it placed on the CAN_TX pin. The names for the logic levels apply here. When a transmitting node places a recessive 1 on the CAN_TX pin and detects a dominant 0 on the CAN_RX pin, another node has placed a dominant bit on the bus. In this case, the dominant bit from from the other node has a higher priority.

Therefore, if the value sensed on the CAN_RX pin is the value driven on the CAN_TX pin, transmission continues. Otherwise, the CAN controller senses that it has lost arbitration. Module configuration determines the next course of action.

The ***Standard CAN Frame*** figure shows a basic 11-bit identifier frame. The CAN_MB[nn]_ID1.RTR bit follows the SOF and identifier. The CAN_MB[nn]_ID1.RTR bit indicates whether the frame contains data (data frame) or is a request for data associated with the message identifier in the frame sent (remote frame).

NOTE: In the CAN protocol, a dominant bit in the CAN_MB[nn]_ID1.RTR field wins arbitration against a remote frame request (CAN_MB[nn]_ID1.RTR=1) for the same message ID. This functionality allows a remote request to be a lower priority than a data frame.

The next field of interest in the frame is the CAN_MB[nn]_ID1.IDE bit. When set, it indicates that the message is an extended frame with a 29-bit identifier instead of an 11-bit identifier. In an extended frame, the first part of the message resembles the ***Extended CAN Frame*** figure.

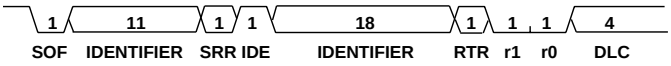


Figure 27-6: Extended CAN Frame

For the CAN_MB[nn]_ID1.RTR field, a dominant bit in the CAN_MB[nn]_ID1.IDE field wins arbitration against an extended frame with the same lower 11 bits. Standard frames have a higher priority than extended frames.

The internal logic automatically generates the Substitute Remote Request (SRR), the reserved bits r0 and r1, and the checksum (CRC). (The SRR is always sent as recessive; reserved bits r0 and r1 are always sent as dominant).

CAN Operating Modes

The CAN controller is in configuration mode when coming out of processor reset. Hardware behavior can be altered only when CAN is in configuration mode. Before initializing the mailboxes, configure the CAN bit timing to work on the CAN bus. The controller connect to the CAN bus.

Data Transfer Modes

The following sections provide information on the data transfer modes supported by the CAN controller.

Transmit Operations

The *CAN Transmit Operation Flowchart* shows the CAN transmit operation. Mailboxes 24–31 are dedicated transmitters. Configure mailboxes 8–23 as transmitters by writing 0 to the corresponding bit in the CAN_MD1 or CAN_MD2 registers. Enable mailbox n (CAN_MC1.MB=1). After writing the data and the identifier into the mailbox area, the message is sent. Then, the corresponding transmit request bit is set (CAN_TRS1.MB=1).

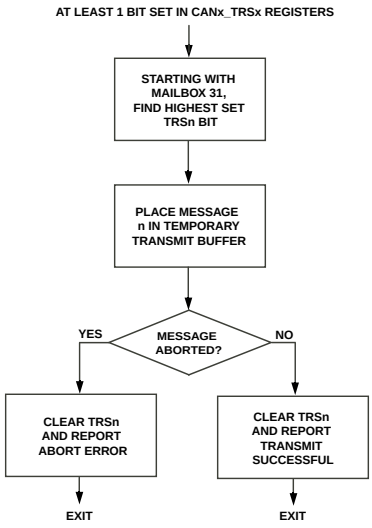


Figure 27-7: CAN Transmit Operation Flowchart

When a transmission completes, the corresponding bits in the CAN_TRS1 or CAN_TRS2 and CAN_TRR1 or CAN_TRR2 registers are cleared. If the transmission is successful, the corresponding bit in the CAN_TA1/CAN_TA2 register is set. If the transmission aborts due to lost arbitration or a CAN error, the corresponding bit in

the CAN_AA1/CAN_AA2 register is set. A requested transmission can also be manually aborted by setting the corresponding bit in the CAN_TRR1/CAN_TRR2 register.

Software sets multiple CAN_TRS1.MB bits simultaneously. These bits are reset after either a successful or an aborted transmission.

The CAN hardware sets these bits in the following cases:

- When using the auto-transmit mode of the universal counter
- When a message loses arbitration and the single-shot CAN_OPSS1.MB bit is not set
- When a remote frame request occurs (only possible for receive or transmit mailboxes if the feature for automatic remote frame handling is enabled (CAN_RFH1.MB=1)).

NOTE: Manage the mailbox area when a CAN_TRS1 or CAN_TRS2 bit is set. Write access to the mailbox is permissible with a bit set. But, changing data in such a mailbox can lead to unexpected data during transmission.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS1 or CAN_TRS2 bit associated with a disabled mailbox can result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS1 or CAN_TRS2 bit is reset by the internal logic can cause unpredictable results.

Retransmission

Normally, the current message object is resent after the loss of arbitration or error frame detection on the CAN bus line. If there is more than one transmit message object pending, the message object with the highest mailbox transmits first. See the ***Transmit Flow*** figure. The currently aborted transmission restarts after any messages with higher priority are sent.

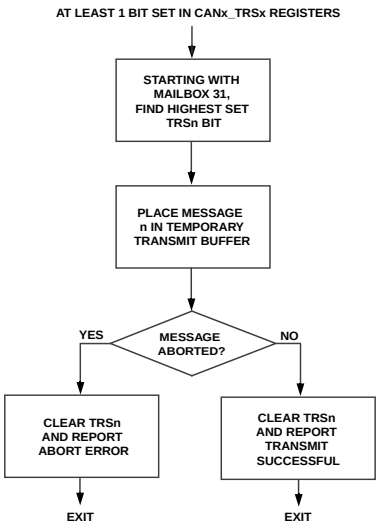


Figure 27-8: Transmit Flow

A message written into the mailbox does not replace a message under preparation. The message under preparation is copied into the temporary transmit buffer when the internal transmit request for the CAN core module is set. The message in the buffer is not replaced until:

- The message is sent successfully
- The arbitration on the CAN bus line is lost
- There is an error frame on the CAN bus line

Single-Shot Transmission

When using the single-shot transmission feature (CAN_OPSS1.MB=1), the corresponding CAN_TRS1 bit is cleared after the message is successfully sent. The bit is cleared even if the transmission aborts due to a lost arbitration or an error frame on the CAN bus line. Therefore, there is no further attempt to transmit the message again when the initial try failed, and the abort error is reported (CAN_AA1.MB=1).

Auto-Transmission

In auto-transmit mode, the message in mailbox 11 (MB11) can be sent periodically using the universal counter. This mode often broadcasts heartbeats to all CAN nodes. So, messages sent this way usually have a high priority.

The period value is written to the CAN_UCRC register. Auto-transmission mode is enabled by setting the CAN_UCCNF.UCCNF field to 0x03. When enabled, the counter CAN_UCCNT is loaded with the value in the CAN_UCRC register. The counter decrements to 0 at the CAN bit clock rate and is then reloaded from CAN_UCRC. Each time the counter reaches a value of 0, internal logic automatically sends the CAN_TRS1.MB bit. The corresponding message from mailbox 11 transfers.

For proper auto-transmit operation, configure mailbox 11 as a transmit mailbox. The mailbox must contain valid data (identifier, control bits, and data) before the counter expires and after this mode is enabled.

Receive Operation

The CAN hardware autonomously receives messages and discards invalid messages. Once a valid message is successfully received, the receive logic interrogates all enabled receive mailboxes. The logic interrogates sequentially, from mailbox 23 down to mailbox 0, whether the message is of interest to the local node or not.

Each incoming data frame is compared to all identifiers stored in the active receive and transmit mailboxes with the feature for remote frame handling enabled (=1). The active receive mailboxes indices of CAN_MD1 and CAN_MC1 registers are set to 1. The message identifier of the received message, along with the identifier extension (CAN_MB[nn]_ID1.IDE) and remote transmission request (CAN_MB[nn]_ID1.RTR) bits, are compared with the register settings of each mailbox. In standard mode, the message is compared with the content of the CAN_MB[nn]_ID1 register. In extended mode, the content of the CAN_MB[nn]_ID0 register must also match.

If the acceptance mask enable CAN_MB[nn]_ID1.AME bit is not set, a match is signaled only if CAN_MB[nn]_ID1.IDE, CAN_MB[nn]_ID1.RTR, and all (11 or 29) identifier bits are exact. If, however, the CAN_MB[nn]_ID1.AME bit is set, the acceptance mask registers (CAN_AM[nn]H/L) determine which of the CAN_MB[nn]_ID1.IDE and CAN_MB[nn]_ID1.RTR bits must match.

The following logic applies:

[(Received Message ID) XNOR (CAN_MB[nn]_ID0/1)] OR [(CAN_MB[nn]_ID1.AME) AND (CAN_AM[nn]H /L)].

This logic appears graphically in the ***CAN Message Receive Logic*** figure.

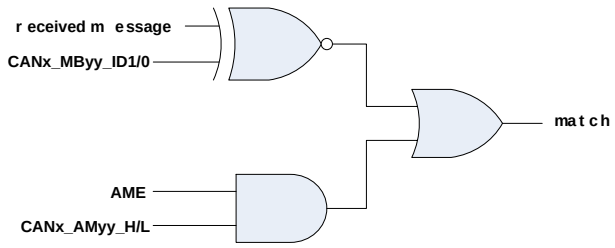


Figure 27-9: CAN Message Receive Logic

A one (1) at the respective bit position in the CAN_AM[nn]H/CAN_AM[nn]L mask registers means that the bit does not need to match when CAN_MB[nn]_ID1.AME=1. This way, a mailbox can accept a group of messages.

Table 27-8: Mailbox Used for Acceptance Filtering

MCn	MDn	RFHn	Mailbox n	Comment
0	X	X	Ignored	Mailbox n disabled
1	0	0	Ignored	Mailbox n enabled; Mailbox n configured for transmit; Remote frame handling disabled
1	0	1	Used	Mailbox n enabled; Mailbox n configured for transmit; Remote frame handling enabled
1	1	X	Used	Mailbox n enabled; Mailbox n configured for receive

If the acceptance filter finds a matching identifier, the content of the received data frame is stored in that mailbox. A received message is stored only once, even if multiple receive mailboxes match its identifier. If the current identifier does not match any mailbox, the message is not stored.

The ***CAN Receive Operation Flowchart*** illustrates the decision tree of the receive logic when processing the individual mailboxes.

If a message is received for a mailbox and that mailbox still contains unread data (CAN_RMP1.MB), then the program decides whether to overwrite the old message. If the CAN_OPSS1.MB bit is cleared, the corresponding CAN_RML1.MB bit is set, and the stored message is overwritten. The receive message lost interrupt request occurs (CAN_GIS.RMLIS is set). If, however, the CAN_OPSS1.MB bit is set, the next mailboxes are checked for another matching identifier. If no match is found, the message is discarded, and the next message is checked.

NOTE: If a receive mailbox is disabled, an ongoing receive message for that mailbox is lost even if a second mailbox is configured to receive the same identifier.

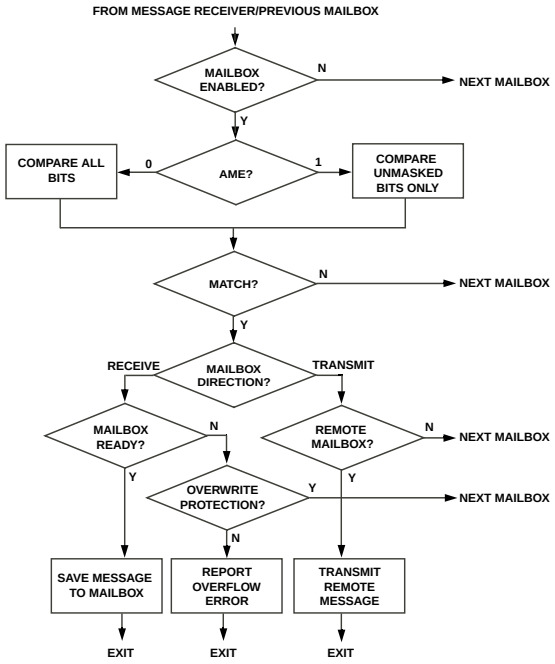


Figure 27-10: CAN Receive Operation Flowchart

Data Acceptance Filtering

If Device Net mode is enabled (`CAN_CTL.DNM` = 1) and the mailbox is set up for filtering on data field, the filtering occurs on the standard ID of the message and data fields. The data field filtering can be programmed for either the first byte only or the first 2 bytes, as shown the ***Data Field Filtering*** table.

If the `CAN_AM[nn]H.FDF` bit is set, the corresponding `CAN_AM[nn]L` register holds the data field mask (DFM bits 15:0]). If the `CAN_AM[nn]H.FDF` bit is cleared, the corresponding `CAN_AM[nn]L` register holds the extended identifier mask (`CAN_AM[nn]H.EXTID` bits 15:0).

Table 27-9: Data Field Filtering

FDF (Filter on Data Field)	FMD (Full Mask Data Field)	Description
0	0	Do not allow filtering on the data field
0	1	Not allowed. FMF must be 0 when FDF is 0
1	0	Filter on first data byte only
1	1	Filter on first two data bytes

Watchdog Mode

Watchdog mode ensures that messages are received periodically. It also observes whether a certain node on the network is alive and functioning properly. Watchdog mode detects and manages the failure cases, as needed.

Enable this mode by programming the universal counter to watchdog mode by setting the `CAN_UCCNF.UCCNF` to 0x2. Once enabled, the `CAN_UCCNT` register is loaded with the predefined value contained in `CAN_UCRC`. This counter decrements at the CAN bit rate.

If the CAN_UCCNF.UCCT and CAN_UCCNF.UCRC bits are set and a message is received in mailbox 4 before the counter counts down to 0, the counter is reloaded with the CAN_UCRC contents. If the counter has counted down to 0 without receiving a message in mailbox 4, then the CAN_GIS.UCEIS bit is set. The counter reloads automatically with the contents of the CAN_UCRC register. If an interrupt request is desired for this event, set the CAN_GIM.UCEIM bit. With the mask bit set, when a watchdog interrupt request occurs, the CAN_GIF.UCEIF bit is also set.

Write to the CAN_UCCNF register to reload the counter with the contents of CAN_UCRC or to disable the register.

The CAN_UCRC register controls the time period it takes for the watchdog interrupt request to occur.

Time Stamps

To get an indication of the time of the receive or transmit time for each message, program the CAN universal counter to time stamp mode. Enable this mode by setting the CAN_UCCNF.UCCNF field to 0x01.

If enabled, the value of the 16-bit free-running counter (CAN_UCCNT) is written into the CAN_MB[nn]_TIMESTAMP register of the corresponding mailbox. The operation occurs when a received message is stored or a message is transmitted.

The time stamp value is captured at the sample point of the Start of Frame (SOF) bit of each incoming or outgoing message. Afterwards, this time stamp value is copied to the CAN_MB[nn]_TIMESTAMP register of the corresponding mailbox.

If the mailbox is configured for automatic remote frame handling (CAN_RFH1.MB = 1), the time stamp value is written for transmission of a data frame or the reception of the requested data frame. The mailbox is configured for transmit or receive.

Clear the counter by setting the CAN_UCCNF.UCRC bit to 1. Or, disable the counter by clearing the CAN_UCCNF.UCE bit. Write to the CAN_UCCNT register to load the counter with a value.

It is also possible to clear the counter (CAN_UCCNT) by the reception of a message in mailbox number 4 (synchronization of all time stamp counters in the system). This operation is accomplished by setting the CAN_UCCNF.UCCT bit.

The CAN_GIS.UCEIS bit is set when the counter overflows. A global CAN interrupt request can optionally occur by unmasking the CAN_GIM.UCEIM bit. If the interrupt source is unmasked, the CAN_GIF.UCEIF bit is also set.

Remote Frame Handling

Automatic handling of remote frames for a transmit mailbox is enabled by setting the corresponding CAN_RFH1.MB bit.

Remote frames are data frames that have no data field and the CAN_MB[nn]_ID1.RTR bit is set. The data length code (DLC) of the requesting remote frame overrules the DLC of the responding data frame. A DLC can be programmed with values in the range of 0–15, but DLC values greater than 8 are considered as 8.

A remote frame contains:

- The identifier bits
- The control field CAN_MB[nn]_LENGTH.DLC (data length count)
- The remote transmission request (CAN_MB[nn]_ID1.RTR) bit

Only configurable mailboxes MB8–MB23 can process remote frames, but all mailboxes can receive and transmit remote frame requests. The CAN_OPSS1 register has no effect when configured for automatic remote frame handling. All content of a mailbox is always overwritten by an incoming message.

NOTE: If a remote frame is received, the DLC of the corresponding mailbox is overwritten with the received value.

Erroneous behavior can result when the CAN_RFH1.MB bit is changed while the corresponding mailbox is processing. To avoid the risk of inconsistent messages, programs must temporarily disable the mailbox while its data registers are updating.

Temporarily Disabling CAN Mailbox

If a mailbox is enabled and configured to transmit, monitor the write accesses to the data field to avoid transmitting inconsistent messages. Be careful if the mailbox is transmitting (or attempting to transmit) repeatedly. Also, if this mailbox is used for automatic remote frame handling, the data field must be updated without losing an incoming remote request frame and without sending inconsistent data. Therefore, the CAN controller allows for temporarily disabling the mailbox using the mailbox temporary disable register (CAN_MBTD).

The pointer to the requested mailbox to the CAN_MBTD.TDPTR field is written, and the CAN_MBTD.TDR bit is set. Internal logic then sets the corresponding CAN_MBTD.TDA flag.

If a mailbox is configured as transmit (CAN_MD1 = 0) and the CAN_MBTD.TDA bit is set, the content of the data field of that mailbox can be updated. If there is an incoming remote request frame while the mailbox is temporarily disabled,

- Internal logic sets the corresponding transmit request bit (CAN_TRS1.MB), and
- The data length code (DLC) of the incoming message is written to the corresponding mailbox.

However, the requested message is not sent until the CAN_MBTD.TDR bit is cleared. Similarly, all transmit requests for temporarily disabled mailboxes are ignored until the CAN_MBTD.TDR bit is cleared. Additionally, transmission of a message immediately aborts when the mailbox is temporarily disabled and the corresponding transmission request reset (CAN_TRR1.MB) bit for this mailbox is set.

If a mailbox is configured to receive (CAN_MD1 = 1), then after issuing a temporary disable request, the CAN_MBTD.TDA flag is set. The mailbox is not processed. If there is an incoming message for a temporarily disabled mailbox, the internal logic waits until reception is complete or there is an error on the CAN bus before setting CAN_MBTD.TDA. Once this flag is set, the mailbox can then be disabled (CAN_MC1 = 0) without the risk of losing an incoming frame. The CAN_MBTD.TDR bit must then be reset as soon as possible.

When the CAN_MBTD.TDA flag is set for a given mailbox, only the data field of that mailbox can be updated. Accesses to the control bits and the identifier are denied.

Bit Timing

The CAN controller does not have a dedicated clock. Instead, the CAN clock is derived from the system clock based on a configurable number of time quanta. The time quantum (TQ) is derived from the formula:

TQ = (BRP + 1)/SCLK

where BRP is the 10-bit bit rate prescaler field in the CAN_CLK register.

Although the CAN_CLK.BRP field can be set to any value, it is recommended that the value be greater than or equal to 4. Restrictions apply to the bit timing configuration when BRP is less than 4.

The CAN_CLK register defines the TQ value, and multiple time quanta make up the duration of a CAN bit on the bus. The CAN_TIMING register controls the nominal bit time and the sample point of the individual bits in the CAN protocol. The ***Three Phases of a CAN Bit*** figure shows the three phases of a CAN bit: the synchronization segment, the segment before the sample point, and the segment after the sample point.

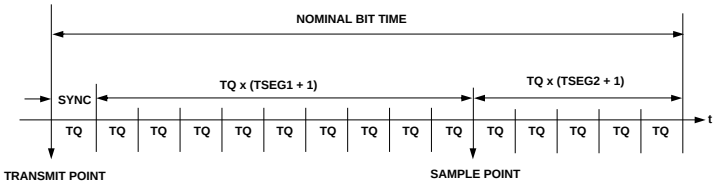


Figure 27-11: Three Phases of a CAN Bit

The synchronization segment is fixed to one TQ. Synchronize the nodes on the bus. All signal edges are expected to occur within this segment.

The CAN_TIMING.TSEG1 and CAN_TIMING.TSEG2 fields control how many TQs the CAN bits consist of, resulting in the CAN bit rate. The following formula gives the nominal bit time.

t_{BIT} = TQ × [1 + (1 + TSEG1) + (1 + TSEG2)]

For safe receive operations on given physical networks, the sample point is programmable by the CAN_TIMING.TSEG1 field. The CAN_TIMING.TSEG2 field holds the number of TQs to complete the bit time. Often, best sample reliability is achieved with sample points in the high 80% range of the bit time. Never use sample points lower than 50%. Therefore, CAN_TIMING.TSEG1 must always be greater than or equal to CAN_TIMING.TSEG2.

The CAN module does not distinguish between the propagation segment and the phase segment-1 as defined by the standard. The CAN_TIMING.TSEG1 value is intended to cover both of them. The CAN_TIMING.TSEG2 value represents the phase segment-2.

If the CAN module detects a recessive-to-dominant edge outside the synchronization segment, it can automatically move the sampling point such that the CAN bit is still handled properly. The synchronization jump width (CAN_TIMING.SJW) field specifies the maximum number of TQs, ranging from 1 to 4 (SJW + 1), allowed for such a resynchronization attempt. The SJW value must not exceed CAN_TIMING.TSEG2 or CAN_TIMING.TSEG1. Therefore, the fundamental rule for writing CAN_TIMING is:

SJW ≤ TSEG2 ≤ TSEG1

In addition to this fundamental rule, CAN_TIMING.TSEG2 must also be greater than or equal to the information processing time (IPT). IPT is the time required by the logic to sample the CAN_RX input, which is 3 system clock cycles.

Therefore, restrictions apply to the minimal value of CAN_TIMING.TSEG2 if CAN_CLK.BRP is less than 2. If CAN_CLK.BRP is set to 0, the CAN_TIMING.TSEG2 field must be greater than or equal to 2. If CAN_CLK.BRP is set to 1, the minimum CAN_TIMING.TSEG2 value is 1.

NOTE: Use the same nominal bit rate for all nodes on a CAN bus.

With all the timing parameters set, the final consideration is sampling performance. The default behavior of the CAN controller is to sample the CAN bit once. The controller samples at the point described by the CAN_TIMING register and controlled by the CAN_TIMING.SAM bit. If this bit is set, however, the input signal is oversampled three times at the system clock rate. The resulting value is generated by a majority decision of the three sample values. Always keep the CAN_TIMING.SAM bit cleared if the BRP value is less than 4.

Do not modify the CAN_CLK and CAN_TIMING registers during normal operation. Always enter configuration mode first. Writes to these registers have no effect when CAN is not in configuration or debug mode. If not coming out of processor reset, enter configuration mode by setting the CAN_CTL.CCR bit and poll the CAN_STAT register until CAN_STAT.CCA is set.

NOTE: If the CAN_TIMING.TSEG1 field is programmed to 0, the module does not leave the configuration mode.

During configuration mode, the module is not active on the CAN bus line. The CAN_TX output pin remains recessive and the module does not receive or transmit messages or error frames. After leaving the configuration mode, all CAN internal core registers and the CAN error counters are set to their initial values.

A soft reset does not change the values of CAN_CLK and CAN_TIMING. Therefore, an ongoing transfer through the CAN bus cannot be corrupted by changing the bit timing parameter or initiating the soft reset (by setting the CAN_CTL.SRS bit).

CAN Low Power Features

The CAN module includes built-in sleep and suspend modes to save power.

The following sections describe the behavior of the CAN module in these modes.

Built-In Suspend Mode

The most modest of power savings mode is the suspend mode. This mode is entered by setting the CAN_CTL.CSR bit. The module enters the suspend mode after the current operation of the CAN bus finishes. Then, the internal logic sets the CAN_STAT.CSA bit. Once CAN enters this mode, the module is no longer active on the CAN bus line, slightly reducing power consumption.

In suspend mode, the CAN_TX output pin remains in a recessive state, and the module does not receive or transmit messages or error frames. The content of the CAN_CEC register remains unchanged. Clear CAN_CTL.CSR to exit suspend mode.

The only difference between suspend mode and configuration mode is that the CAN_CTL and CAN_STAT registers are not reset when exiting suspend mode.

Built-In Sleep Mode

The next level of power savings can be realized by using the built-in sleep mode for the module. This mode is entered by setting the CAN_CTL.SMR bit. The module enters the sleep mode after the current operation of the CAN bus finishes. Once this mode is entered, many of the internal CAN module clocks are shut off, reducing power consumption, and the CAN_INT.SMACK bit is set.

When the CAN module is in sleep mode, all register reads return the contents of CAN_INT instead of the usual contents. All register writes, except to CAN_INT, are ignored in sleep mode. A small part of the module is clocked continuously to allow for waking up out of sleep mode.

A write to the CAN_INT register ends sleep mode. If the CAN_CTL.WBA bit is set before entering sleep mode, a dominant bit on the CAN_RX pin also ends sleep mode. When software sets the CAN_CTL.SMR bit, hardware sets the CAN_CTL.CSR bit as well, making sleep mode a super set of suspend mode. When the controller wakes up from sleep mode, hardware automatically clears CAN_CTL.SMR and CAN_CTL.CSR. If, however, the controller never enters sleep mode because the wake-up condition was met before CAN_INT.SMACK bit turns to 1, the CAN_CTL.SMR and CAN_CTL.CSR bits do not always automatically clear. Therefore, clear the two bits using software, when returning from sleep mode.

Soft Reset

The CAN controller features a build-in reset mechanism called soft reset. Soft reset is entered immediately after software has set the CAN_CTL.SRS bit. Soft reset brings all control registers to a defined state. Mailbox and error registers remain unaffected. Soft reset does not alter the CAN_TIMING and CAN_CLK registers and does not disturb the on-going transmission of a currently pending message, acknowledge bit or error frame. However, when recovering from soft reset, software can lose track of transmission or reception reports and interrupt requests.

CAN Event Control

The following section describe how the CAN module generates and controls events.

CAN Interrupt Signals

The CAN module provides three independent interrupt requests: two mailbox interrupt requests (mailbox receive interrupt request (MBRIRQ) and mailbox transmit interrupt request (MBTIRQ)) and a global CAN status interrupt request (GIRQ). The values of these three interrupt requests can also be read through the CAN_GIS register.

Mailbox Interrupts

Each of the 32 mailboxes in the CAN module can generate a receive or transmit interrupt request, depending on the mailbox configuration. To enable a mailbox to generate an interrupt request, set the corresponding CAN_MBIM1 bit.

If a mailbox is configured as a receive mailbox, the corresponding CAN_MBRIF1 bit and CAN_RMP1 bit are set after a received message is stored in mailbox n. When using the feature for automatic remote frame handling (CAN_RFH1=1), the receive interrupt flag is set after the requested data frame is stored in the mailbox.

If any CAN_MBRIF1 bits are set, the mailbox generates a CAN_INT.MBRIRQ interrupt request. To clear the CAN_INT.MBRIRQ interrupt request, software must clear all of the set CAN_MBRIF1 bits by writing a 1 to those bit locations in CAN_MBRIF1. Prior to this operation, software must clear the corresponding CAN_RMP1 bit.

If a mailbox is configured as a transmit mailbox, the corresponding CAN_MBTIF1 bit in the transmit interrupt flag is set after the message in mailbox n is sent correctly. The corresponding CAN_TA1 bit is also set. The CAN_TA1 bits maintain their state even after the corresponding mailbox n is disabled (CAN_MC1=0). When using the feature for automatic remote frame handling, the transmit interrupt flag is set after the requested data frame is sent from the mailbox.

If any CAN_MBTIF1.MB bits are set, the MBTIRQ interrupt output is raised in the CAN_INT register. To clear the MBTIRQ interrupt request, software must clear all of the bits that are set in the CAN_MBTIF1 register by writing a 1 to those bit locations. Additionally, software must clear the associated CAN_TA1 bit or set the associated CAN_TRS1 bit to clear the interrupt source that asserts the CAN_MBTIF1 bit.

Global Interrupt

The global CAN interrupt logic implements with three registers:

- The CAN_GIM register, where each interrupt source can be enabled or disabled separately
- The CAN_GIS register
- The CAN_GIF register

The interrupt mask bits only affect the content of the CAN_GIF register. If the mask bit is not set in the CAN_GIM register, the corresponding flag bit is not set when the event occurs. The interrupt status bits in the CAN_GIS register, however, are always set when the corresponding interrupt event occurs, independent of the mask bits. Thus, the interrupt status bits can be used to poll interrupt events.

The CAN_INT.GIRQ bit is only asserted if a bit in the CAN_GIF register is set. The read-only CAN_INT.GIRQ bit remains set as long as at least 1 bit in CAN_GIF is set. All bits in the interrupt status and interrupt flag registers remain set until cleared by software or a soft reset has occurred.

NOTE: The CAN_GIF register is read-only (RO). In the global CAN interrupt ISR, clear the interrupt latch by writing a 1 to the corresponding bit of the CAN_GIS register. The operations clear the related bits of the CAN_GIS and CAN_GIF registers, as well as the CAN_INT.GIRQ bit.

There are several interrupt events that can activate this GIRQ interrupt request:

- Access denied event interrupt (CAN_GIM.ADIM, CAN_GIS.ADIS, CAN_GIF.ADIF): At least one access to the mailbox RAM occurred during a data update by internal logic.

- Universal counter exceeded event interrupt (CAN_GIM.UCEIM, CAN_GIS.UCEIS, CAN_GIF.UCEIF): There is an overflow of the universal counter (in time stamp mode or event counter mode) or the counter has reached the value 0x0000 (in watchdog mode).
- Receive message lost event interrupt (CAN_GIM.RMLIM, CAN_GIS.RMLIS, CAN_GIF.RMLIF): A message is received for a mailbox that currently contains unread data. At least 1 bit in the CAN_RMLn register is set. If the bit in CAN_GIS and CAN_GIF registers is cleared and there is at least 1 bit in CAN_RML1 still set, then the bit in the CAN_GIS and CAN_GIF registers is not set again. The internal interrupt source signal is only active if a new bit in CAN_RML1 is set.
- Abort acknowledge event interrupt (CAN_GIM.AAIM, CAN_GIS.AAIS, CAN_GIF.AAIF): At least 1 CAN_AA1.MB bit in the CAN_AA1 registers is set. If the bit in the CAN_GIS and CAN_GIF registers is cleared and there is at least 1 bit in CAN_AA1 still set, then the bit in the CAN_GIS and CAN_GIF registers is not set again. The internal interrupt source signal is only active if a new bit in CAN_AA1 is set. The CAN_AA1.MB bits maintain state even after the corresponding mailbox n is disabled (CAN_MC1 = 0).
- Access to unimplemented address event interrupt (CAN_GIM.UIAIM, CAN_GIS.UIAIS, CAN_GIF.UIAIF): There was a CPU access to an address which is not implemented in the controller module.
- Wake-up event interrupt (CAN_GIM.WUIM, CAN_GIS.WUIS, CAN_GIF.WUIF): The CAN module has left the sleep mode because of detected activity on the CAN bus line.
- Bus-off event interrupt (CAN_GIM.BOIM, CAN_GIS.BOIS, CAN_GIF.BOIF): The CAN module has entered the bus-off state. This interrupt source is active if the status of the CAN core changes from normal operation mode to the bus-off mode. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the bus-off mode is still active, then this bit is not set again. If the module leaves the bus-off mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.
- Error-passive event interrupt (CAN_GIM.EPIM, CAN_GIS.EPIS, CAN_GIF.EPIF): The CAN module has entered the error-passive state. This interrupt source is active if the status of the CAN module changes from the error-active mode to the error-passive mode. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the error-passive mode is still active, then this bit is not set again. If the module leaves the error-passive mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.
- Error warning receive event interrupt (CAN_GIM.EWRIM, CAN_GIS.EWRIS, CAN_GIF.EWRIF): The CAN receive error counter (CAN_CEC.RXECNT) has reached the warning limit. If the bit in the CAN_GIS and CAN_GIF registers) is cleared and the error warning mode is still active, this bit is not set again. If the module leaves the error warning mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.
- Error warning transmit interrupt (CAN_GIM.EWTIM, CAN_GIS.EWTIS, CAN_GIF.EWTIF): The CAN transmit error counter (CAN_CEC.TXECNT) has reached the warning limit. If the bit in the CAN_GIS and CAN_GIF registers is cleared and the error warning mode is still active, this bit is not set again. If the module leaves the error warning mode, the bit in the CAN_GIS and CAN_GIF registers remains set, if not explicitly cleared.

Event Counter

For diagnostic functions, it is possible to use the universal counter as an event counter. The counter can be programmed in the CAN_UCCNF[3:0] field to increment on one of these conditions:

- 0x6 – CAN error frame. Counter increments if there is an error frame on the CAN bus line.
- 0x7 – CAN overload frame. Counter increments if there is an overload frame on the CAN bus line.
- 0x8 – Lost arbitration. Counter increments every time arbitration on the CAN line is lost during transmission.
- 0x9 – Transmission aborted. Counter increments every time arbitration is lost or a transmit request is canceled (CAN_AA1 is set).
- 0xA – Transmission succeeded. Counter increments every time a message sends without detected errors (CAN_TA1 is set).
- 0xB – Receive message rejected. Counter increments every time a message is received without detected errors but not stored in a mailbox because there is no matching identifier found.
- 0xC – Receive message lost. Counter increments every time a message is received without detected errors but not stored in a mailbox because the mailbox contains unread data (CAN_RML1 is set).
- 0xD – Message received. Counter increments every time a message is received without detected errors, whether the received message is rejected or stored in a mailbox.
- 0xE – Message stored. Counter increments every time a message is received without detected errors, has an identifier that matches an enabled receive mailbox, and is stored in the receive mailbox (CAN_RMP1 is set).
- 0xF – Valid message. Counter increments every time a valid transmit or receive message is detected on the CAN bus line.

CAN Warnings and Errors

The processor controls CAN warnings and errors using the error counter (CAN_CEC) register, the error status (CAN_ESR) register, and the error counter warning level (CAN_EWR) register. The following sections describe error handling.

Programmable Warning Limits

Programs can set the warning level for CAN_GIS.EWTIS and CAN_GIS.EWRIS separately by writing to the CAN_EWR.EWLREC and CAN_EWR.EWLTEC fields. After power-on reset, the CAN_EWR register is set to the default warning level of 96 for both error counters. After a soft reset, the contents of this register remain unchanged.

Error Handling

Error management is a part of the CAN standard. Several different kinds of bus errors can occur during transmissions:

- Bit error – Only the transmitting node detects this error. Whenever a node transmits, it continuously monitors its receive pin (CAN_RX) and compares the received bit with the transmitted bit. During the arbitration phase,

the node postpones the transmission if the received and transmitted bits do not match. However, after the arbitration phase, a bit error is signaled any time the value on CAN_RX does not equal what is transmitted on the CAN_TX pin. (The arbitration phase completes when the CAN_MB[nn]_ID1.RTR bit is sent successfully.)

- Form error. Occurs when a fixed-form bit position in the CAN frame contains one or more illegal bits. Occurs when a dominant bit is detected at a delimiter or end of frame bit position.
- Acknowledge error. Occurs whenever a message is sent and no receivers drive an acknowledge bit.
- CRC error. Occurs whenever a receiver calculates the CRC on the data it received and finds it different than the CRC that transmitted on the bus itself.
- Stuff error. The CAN specification requires the transmitter to insert an extra stuff bit of opposite value after 5 bits have transmitted with the same value. The receiver disregards the value of the stuff bits. However, it takes advantage of the signal edge to resynchronize itself. A stuff error occurs on receiving nodes whenever the sixth consecutive bit value is the same as the previous 5 bits.

Once the CAN module detects any of the errors, it updates the CAN_ESR and CAN_CEC registers. In addition to the standard errors, the CAN_ESR.SAO flag signals when the CAN_RX pin sticks at dominant level, indicating a possibility of shorted wires.

Error Frames

It is important that all nodes on the CAN bus ignore data frames that any single node failed to receive. Every node sends an error frame as soon as it has detected an error as shown in the ***CAN Error Example*** figure.

A device that has detected an error still completes the ongoing bit. It initiates an error frame by sending six dominant and eight recessive bits to the bus. Since this activity is a violation of the bit stuffing rule, all nodes are signaled to discard the ongoing frame. (All receivers that did not detect the transmission error in the first instance now detect a stuff bit error.)

The transmitter can detect a normal bit error sooner. It aborts the transmission of the ongoing frame and tries resending it later.

When all nodes on the bus have detected the error, they also send six dominant and eight recessive bits to the bus. The resulting error frame consists of two different fields. The first field is the superposition of error flags contributed from the different stations, which are a sequence of 6–12 dominant bits. The second field is the error delimiter and consists of eight recessive bits indicating the end of frame.

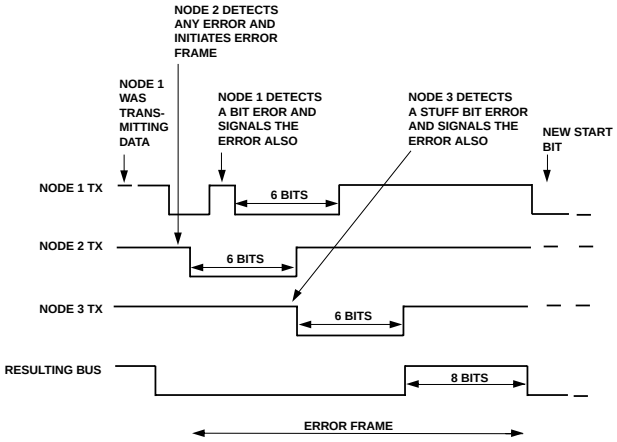


Figure 27-12: CAN Error Example

For CRC errors, the error frame initiates at the end of the frame, rather than immediately after the failing bit.

After having received eight recessive bits, every node knows that the error condition is resolved and, if messages are pending, starts transmission. The transmitter that had to abort its operation must win the new arbitration again; otherwise its message is delayed as determined by priority.

Because the transmission of an error frame destroys the frame under transmission, a faulty node erroneously detecting an error can block the bus. So, there are two node states which determine a nodes right to signal an error—error-active and error-passive.

- *Error-active* nodes have an error detection rate below a certain limit. These nodes drive an active error flag of six dominant bits.
- *Error-passive* nodes have a higher error detection rate and can have a local problem and therefore have a limited right to signal errors. These nodes drive a passive error flag consisting of six recessive bits. Therefore, an error-passive transmitting node is still able to inform the other nodes about the aborting of a self-transmitted frame. But, it is no longer able to destroy correctly received frames of other nodes.

Error Levels

The CAN specification requires each node in the system to operate at one of three levels. The ***CAN Error Level Description*** table describes the levels. This functionality prevents nodes with high error rates from blocking the entire network, as local hardware can cause the errors. The CAN module provides an error counter for transmit (TEC) and an error counter for receive (REC). The `CAN_CEC` register contains each of these 8-bit counters.

After initialization, both the TEC and the REC counters are 0. Each time a bus error occurs, one of the counters increments by either 1 or 8, depending on the error situation. Refer to version 2.0 of the CAN specification. Successful transmit or receive operations decrement the respective counter by 1.

If either of the error counters exceeds 127, the CAN module goes into an error-passive state and the `CAN_STAT.EP` bit is set. Once this state occurs, the module is not allowed to send any more active error frames. However, the module can still transmit messages and signal passive error frames in case the transmission fails due to bit errors.

If one of the counters exceeds 255 (that is, when an 8-bit counter overflows), the CAN module disconnects from the bus and it goes into bus-off mode. In this mode, the CAN_STAT.EBO bit is set. Software intervention is needed for recovery from this state, unless the CAN_CTL.ABO bit is enabled. The bit puts the module into active mode after the bus-off recovery sequence.

Table 27-10: CAN Error Level Description

Level	Condition	Description
Error active	Transmit and receive error counters <128	This level is the initial condition level. As long as errors stay below 128, the node drives active error flags during error frames.
Error passive	Transmit or receive error counter-value from 128 through 255, inclusive	Errors have accumulated to a level that requires the node to drive passive error flags during error frames
Bus off	Transmit or receive error counters greater than 255	CAN module goes into bus-off mode

In addition to the three levels in the table, the CAN module also generates separate transmit and receive warnings (CAN specification enhancement). By default, when one of the error counters exceeds 96, it signals and reports a warning in the CAN_STAT register. The CAN receive warning flag (CAN_STAT.WR) bit is set when CAN_CEC.RXECNT exceeds 96. The CAN transmit warning flag (CAN_STAT.WT) bit is set when CAN_CEC.TXECNT exceeds 96. The error warning level can be programmed using the error warning register (CAN_EWR).

Additionally, interrupt requests can occur for all of these levels by unmasking them in the global CAN interrupt mask register (CAN_GIM). These sources include: the bus-off interrupt (CAN_GIM.BOIM), the error-passive interrupt (CAN_GIM.EPIM), the error warning receive interrupt (CAN_GIM.EWRIM), and the error warning transmit interrupt (CAN_GIM.EWTIM).

During the bus-off recovery sequence, internal logic sets the configuration mode request CAN_CTL.CCR bit. The CAN core module does not automatically come out of the bus-off mode. The CAN_CTL.CCR bit cannot be reset until the bus-off recovery sequence completes.

NOTE: Set the CAN_CTL.ABO bit to override this behavior. After exiting the bus-off or configuration modes, the CAN error counters are reset.

CAN Debug and Test Modes

The CAN module contains test mode features that aid in the debugging of the CAN software and system.

NOTE: When using these features, the CAN module does not always comply to the CAN specification. Enable or disable all test modes only when the module is in configuration mode (CAN_STAT.CCA=1) or suspend mode (CAN_STAT.CSA=1).

The CAN_DBG.CDE bit provides access to all of the debug features. Set this bit to enable the test mode. Write to the bit first before writing to the CAN_DBG register. When the CAN_DBG.CDE bit is cleared, all debug features are disabled.

When the CAN_DBG.CDE bit is set, it enables writes to the other bits of the CAN_DBG register. It also enables these features, which are not compliant to the CAN standard:

- Bit timing registers can be changed anytime, not only during configuration mode. The group includes the CAN_CLK and CAN_TIMING registers.
- Write access is allowed to the normally read-only CAN_CEC register.

The following list describes other bits in the debug register.

- The CAN module uses the CAN_DBG.MRB bit to enable the read back mode. In this mode, a message transmitted on the CAN bus (or through an internal loopback mode) is received back directly to the internal receive buffer. After a correct transmission, the internal logic treats this transfer as a normal receive message. This feature allows testing of most of the CAN features without an external device.
- The CAN_DBG.MAA bit allows the CAN module to generate its own acknowledge during the ACK slot of the CAN frame. No external devices or connections are necessary to read back a transmit message. In this mode, the sent message is automatically stored in the internal receive buffer. In auto-acknowledge mode, the module itself transmits the acknowledge. This acknowledge can be programmed to appear on the CAN_TX pin, if CAN_DBG.DIL=1 and CAN_DBG.DTO= 0. If the acknowledge is only used internally, then set these test mode bits to CAN_DBG.DIL= 0 and CAN_DBG.DTO=1.
- The CAN module uses the CAN_DBG.DIL bit to internally enable the transmit output to be routed back to the receive input.
- The CAN module uses the CAN_DBG.DTO bit to disable the CAN_TX output pin. When this bit is set, the CAN_TX pin continuously drives recessive bits.
- The CAN module uses the CAN_DBG.DRI bit to disable the CAN_RX input. When set, the internal logic receives recessive bits or receives the internally-generated transmit value in the case of the internal loop enabled (CAN_DBG.DIL= 0). In either case, the value on the CAN_RX input pin is ignored.
- The CAN module uses the CAN_DBG.DEC bit to disable the transmit and receive error counters in the CAN_CEC register. When this bit is set, the CAN_CEC holds its current contents and is not allowed to increment or decrement the error counters. This mode does not conform to the CAN specification.

NOTE: Write to the error counter registers in debug mode only. Write-access during reception can lead to undefined values. The maximum value which can be written into the error counters is 255. Therefore, the error counter value of 256, which forces the module into the bus off state, cannot be written into the error counter registers.

Table 27-11: Common CAN Test Mode Bit Combinations

MRB	MAA	DIL	DTO	DRI	CDE	Functional Description
X	X	X	X	X	0	Normal mode, not debug mode
0	X	X	X	X	X	No readback of transmit message

Table 27-11: Common CAN Test Mode Bit Combinations (Continued)

MRB	MAA	DIL	DTO	DRI	CDE	Functional Description
1	0	1	0	0	1	Normal transmission on CAN bus line. Read back. External acknowledge from external device required.
1	1	1	0	0	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. CAN_RX input is enabled.
1	1	0	0	0	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge transmit on CAN bus line. CAN_RX input and internal loop are enabled (internal OR of TX and RX)
1	1	0	0	1	1	Normal transmission on CAN bus line. Read back. No external acknowledge required. Transmit message and acknowledge are transmitted on CAN bus line. CAN_RX input is ignored. Internal loop is enabled.
1	1	0	1	1	1	No transmission on CAN bus line. Read back. No external acknowledge required. Nether transmit message nor acknowledge are transmitted on CAN_TX. CAN_RX input is ignored. Internal loop is enabled.

CM41X_M4 CAN Register Descriptions

Controller Area Network (CAN) contains the following registers.

Table 27-12: CM41X_M4 CAN Register List

Name	Description
CAN_AA1	Abort Acknowledge 1 Register

Table 27-12: CM41X_M4 CAN Register List (Continued)

Name	Description
CAN_AA2	Abort Acknowledge 2 Register
CAN_AM[nn]H	Acceptance Mask (H) Register
CAN_AM[nn]L	Acceptance Mask (L) Register
CAN_CEC	Error Counter Register
CAN_CLK	Clock Register
CAN_CTL	CAN Master Control Register
CAN_DBG	Debug Register
CAN_ESR	Error Status Register
CAN_EWR	Error Counter Warning Level Register
CAN_GIF	Global CAN Interrupt Flag Register
CAN_GIM	Global CAN Interrupt Mask Register
CAN_GIS	Global CAN Interrupt Status Register
CAN_INT	Interrupt Pending Register
CAN_MBIM1	Mailbox Interrupt Mask 1 Register
CAN_MBIM2	Mailbox Interrupt Mask 2 Register
CAN_MBRIF1	Mailbox Receive Interrupt Flag 1 Register
CAN_MBRIF2	Mailbox Receive Interrupt Flag 2 Register
CAN_MBTD	Temporary Mailbox Disable Register
CAN_MBTIF1	Mailbox Transmit Interrupt Flag 1 Register
CAN_MBTIF2	Mailbox Transmit Interrupt Flag 2 Register
CAN_MB[nn]_DATA0	Mailbox Word 0 Register
CAN_MB[nn]_DATA1	Mailbox Word 1 Register
CAN_MB[nn]_DATA2	Mailbox Word 2 Register
CAN_MB[nn]_DATA3	Mailbox Word 3 Register
CAN_MB[nn]_ID0	Mailbox ID 0 Register
CAN_MB[nn]_ID1	Mailbox ID 1 Register
CAN_MB[nn]_LENGTH	Mailbox Length Register
CAN_MB[nn]_TIMESTAMP	Mailbox Time Stamp Register
CAN_MC1	Mailbox Configuration 1 Register
CAN_MC2	Mailbox Configuration 2 Register
CAN_MD1	Mailbox Direction 1 Register

Table 27-12: CM41X_M4 CAN Register List (Continued)

Name	Description
CAN_MD2	Mailbox Direction 2 Register
CAN_OPSS1	Overwrite Protection/Single Shot Transmission 1 Register
CAN_OPSS2	Overwrite Protection/Single Shot Transmission 2 Register
CAN_RFH1	Remote Frame Handling 1 Register
CAN_RFH2	Remote Frame Handling 2 Register
CAN_RML1	Receive Message Lost 1 Register
CAN_RML2	Receive Message Lost 2 Register
CAN_RMP1	Receive Message Pending 1 Register
CAN_RMP2	Receive Message Pending 2 Register
CAN_STAT	Status Register
CAN_TA1	Transmission Acknowledge 1 Register
CAN_TA2	Transmission Acknowledge 2 Register
CAN_TIMING	Timing Register
CAN_TRR1	Transmission Request Reset 1 Register
CAN_TRR2	Transmission Request Reset 2 Register
CAN_TRS1	Transmission Request Set 1 Register
CAN_TRS2	Transmission Request Set 2 Register
CAN_UCCNF	Universal Counter Configuration Mode Register
CAN_UCCNT	Universal Counter Register
CAN_UCRC	Universal Counter Reload/Capture Register

Abort Acknowledge 1 Register

The CAN_AA1 register indicates a transmission abort (due to lost arbitration or a CAN error) for mailboxes 8 through 15. Each bit in this register indicates a transmission abort for the corresponding mailbox when set (=1). Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

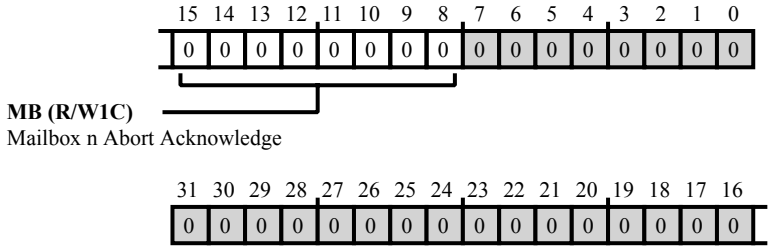


Figure 27-13: CAN_AA1 Register Diagram

Table 27-13: CAN_AA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Abort Acknowledge.

Abort Acknowledge 2 Register

The CAN_AA2 register indicates a transmission abort (due to lost arbitration or a CAN error) for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates a transmission abort for the corresponding mailbox when set (=1).

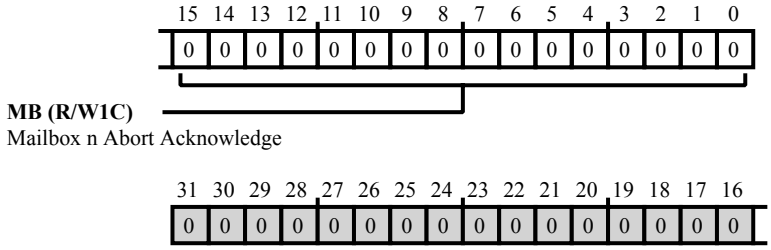


Figure 27-14: CAN_AA2 Register Diagram

Table 27-14: CAN_AA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Abort Acknowledge.

Acceptance Mask (H) Register

The CAN_AM[nn]H register and CAN_AM[nn]L register manage acceptance mask operation. For information about acceptance mask operation, see the Receive Operation section.

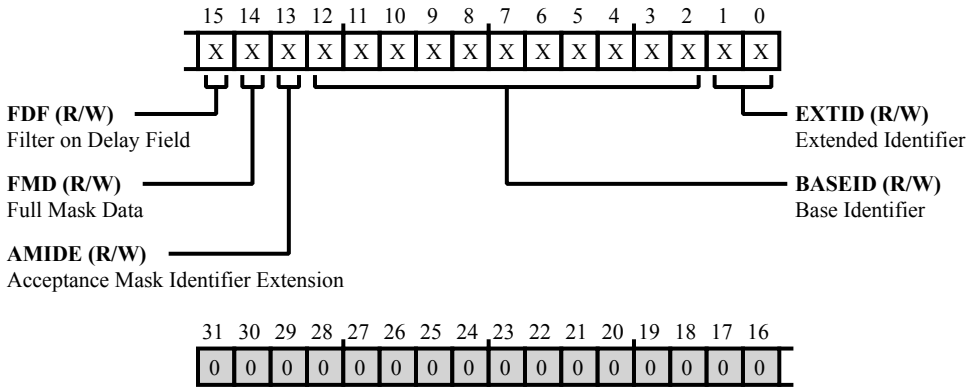


Figure 27-15: CAN_AM[nn]H Register Diagram

Table 27-15: CAN_AM[nn]H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	FDF	Filter on Delay Field. The CAN_AM[nn]H.FDF bit selects the operation of the CAN_AM[nn]H register and CAN_AM[nn]L register when the CAN_CTL.DNM bit is enabled. If the CAN_AM[nn]H.FDF bit is set, the corresponding CAN_AM[nn]L.EXTID bits hold the data field mask. If the CAN_AM[nn]H.FDF bit is cleared, the corresponding CAN_AM[nn]L.EXTID bits hold the high bits of the extended identifier mask.
14 (R/W)	FMD	Full Mask Data. The CAN_AM[nn]H.FMD bit works with the CAN_AM[nn]H.FDF bit to determine data field filtering. For information about data field filtering, see the Receive Operation section.
13 (R/W)	AMIDE	Acceptance Mask Identifier Extension. The CAN_AM[nn]H.AMIDE bit enables the comparison of the received message ID to the value in the CAN_AM[nn]H.EXTID and CAN_AM[nn]L.EXTID bits.
12:2 (R/W)	BASEID	Base Identifier. The CAN_AM[nn]H.BASEID bits hold the base ID for acceptance mask operations.
1:0 (R/W)	EXTID	Extended Identifier. The CAN_AM[nn]H.EXTID bits hold the extended ID (upper two bits) for acceptance mask operations.

Acceptance Mask (L) Register

The CAN_AM[nn]L register and CAN_AM[nn]H register manage acceptance mask operation. For information about acceptance mask operation, see the Receive Operation section.

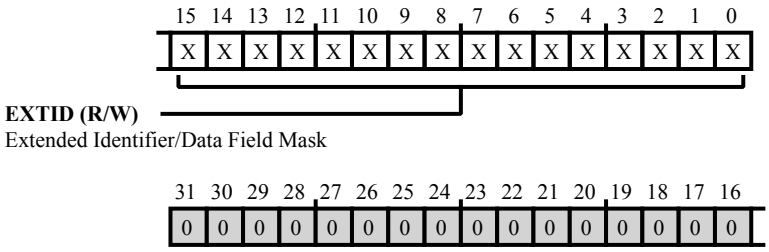


Figure 27-16: CAN_AM[nn]L Register Diagram

Table 27-16: CAN_AM[nn]L Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	EXTID	Extended Identifier/Data Field Mask. The CAN_AM[nn]L.EXTID bits hold the extended ID (lower 16 bits) for data field mask in acceptance mask operations.

Error Counter Register

The CAN_CEC register, CAN_ESR register, and CAN_EWR register control CAN warnings and errors. For detailed information about error and warning operations, see the Event Control section.

The CAN_CEC register holds an error counter for transmit (CAN_CEC.TXECNT) and an error counter for receive (CAN_CEC.RXECNT). After initialization, both counters are 0. Each time a bus error occurs, one of the counters is incremented by either 1 or 8, depending on the error situation (documented in Version 2.0 of CAN Specification). Successful transmit and receive operations decrement the respective counter by 1.

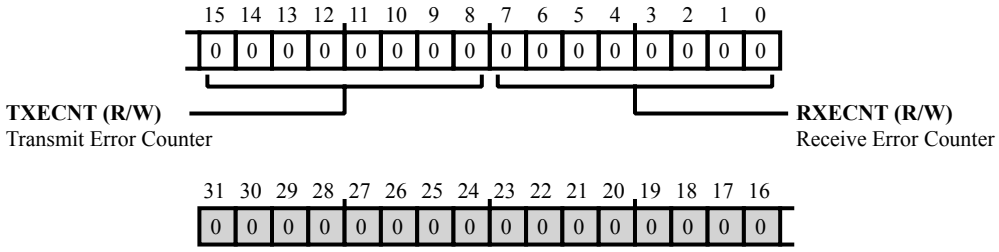


Figure 27-17: CAN_CEC Register Diagram

Table 27-17: CAN_CEC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	TXECNT	Transmit Error Counter. The CAN_CEC.TXECNT bits hold the transmit error counter, which is incremented for errors (by either 1 or 8) and is decremented (by 1) for successful transmit operations.
7:0 (R/W)	RXECNT	Receive Error Counter. The CAN_CEC.RXECNT bits hold the receive error counter, which is incremented for errors (by either 1 or 8) and is decremented (by 1) for successful receive operations.

Clock Register

The CAN_CLK register selects the bit rate prescaler for calculating the time quantum (TQ), which is used to derive the CAN clock from the system clock (SCLK). For more information about bit timing and clock operation, see the CAN Operating Modes section.

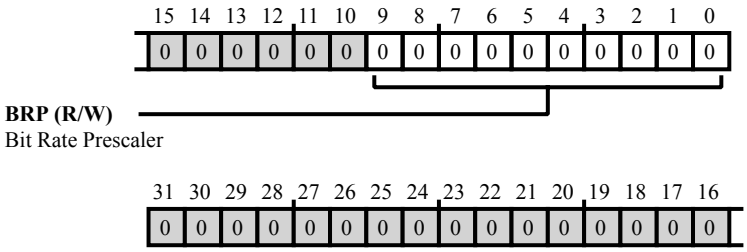


Figure 27-18: CAN_CLK Register Diagram

Table 27-18: CAN_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/W)	BRP	Bit Rate Prescaler. The CAN_CLK.BRP bits select the bit rate prescaler value, which is used to calculate the time quantum for CAN bit timing. The formula using CAN_CLK.BRP to calculate the time quantum is: TQ = (BRP+1) / SCLK Note that it is recommended that the CAN_CLK.BRP value be greater than or equal to 4. For more information about bit timing, see the Operating Modes section.

CAN Master Control Register

The CAN_CTL register controls CAN mode requests, including soft reset.

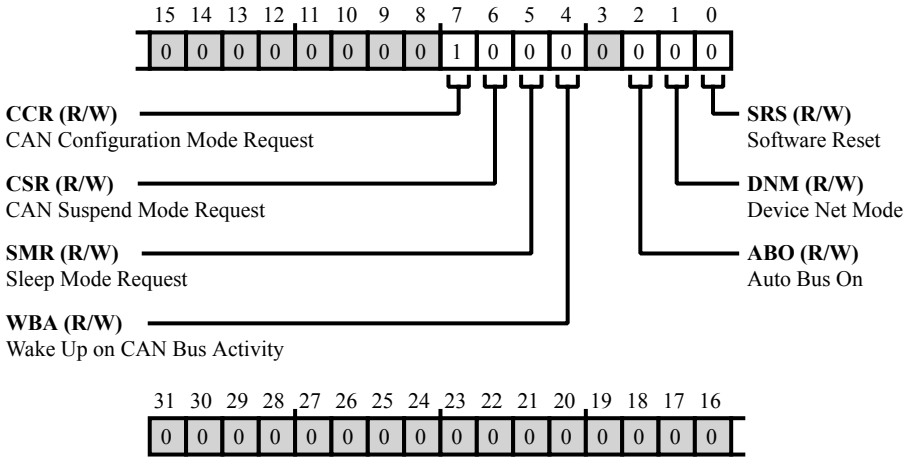


Figure 27-19: CAN_CTL Register Diagram

Table 27-19: CAN_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CCR	CAN Configuration Mode Request. The CAN_CTL.CCR bit requests that the CAN enter configuration mode. Note that the CAN should always be put in configuration mode before modifying the CAN_CLK or CAN_TIMING registers.	
		0	No Request (Exit Configuration Mode)
		1	Request Configuration Mode
6 (R/W)	CSR	CAN Suspend Mode Request. The CAN_CTL.CSR bit requests that the CAN enter suspend mode. The CAN enters suspend mode after the current operation of the CAN bus is finished.	
		0	No Request (Exit Suspend Mode)
		1	Request Suspend Mode
5 (R/W)	SMR	Sleep Mode Request. The CAN_CTL.SMR bit requests that the CAN enter sleep mode. The CAN enters sleep mode after the current operation of the CAN bus is finished.	
		0	No Request (Exit Sleep Mode)
		1	Request Sleep Mode

Table 27-19: CAN_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	WBA	Wake Up on CAN Bus Activity. The CAN_CTL.WBA bit enables wake on CAN bus activity. When enabled, a dominant bit on the CAN_RX pin ends sleep mode (also, the default wake up condition of a write to the CAN_INT register).	
		0	Disable Wake on Bus Activity
		1	Enable Wake on Bus Activity
2 (R/W)	ABO	Auto Bus On. The CAN_CTL.ABO bit selects whether (if enabled) the CAN enters active mode after the bus-off recovery sequence or (if disabled) the CAN enters configuration mode after the bus-off recovery sequence.	
		0	Disable Auto Bus On
		1	Enable Auto Bus On
1 (R/W)	DNM	Device Net Mode. The CAN_CTL.DNM bit enables mailbox filtering on a data field. The filtering is done on the standard ID of the message and data fields. For more information, see the CAN_AM[nn]H.FDF bit description.	
		0	Disable Device Net Mode
		1	Enable Device Net Mode
0 (R/W)	SRS	Software Reset. The CAN_CTL.SRS bit resets the CAN, bringing all control registers to a defined state. Soft reset is entered immediately after software has set the CAN_CTL.SRS bit.	
		0	No Action
		1	Reset CAN

Debug Register

The CAN_DBG register controls CAN debug modes, including CAN_TX and CAN_RX pin enable and disable.

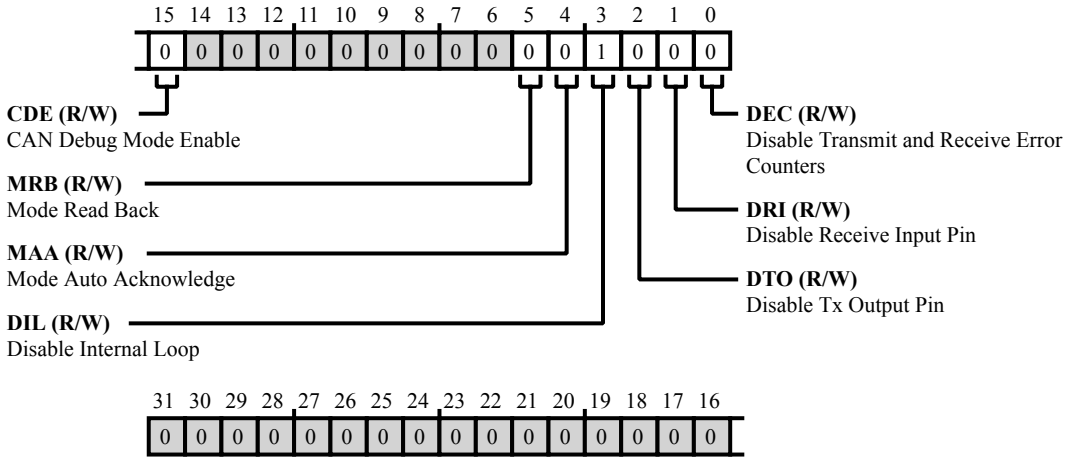


Figure 27-20: CAN_DBG Register Diagram

Table 27-20: CAN_DBG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	CDE	CAN Debug Mode Enable. The CAN_DBG.CDE bit enables debug mode. This bit must be written first before subsequent writes to the CAN_DBG register. When the CAN_DBG.CDE bit is cleared, all CAN debug features are disabled.	
		0	Disable Debug Mode
		1	Enable Debug Mode
5 (R/W)	MRB	Mode Read Back. The CAN_DBG.MRB bit enables read back mode. When enabled, a message transmitted on the CAN bus or through an internal loop back mode is received back directly to the internal receive buffer.	
		0	Disable Read Back Mode
		1	Enable Read Back Mode

Table 27-20: CAN_DBG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	MAA	Mode Auto Acknowledge. The CAN_DBG.MAA bit enables auto acknowledge mode, allowing the CAN to generate its own acknowledge during the ACK slot of the CAN frame. The CAN_DBG.MAA acknowledge appears on the CAN_TX pin if CAN_DBG.DIL =1 and CAN_DBG.DTO =0. If the acknowledge is only going to be used internally, these test mode bits should be set to CAN_DBG.DIL = 0 and CAN_DBG.DTO =1.	
		0	Disable Auto Acknowledge Mode
		1	Enable Auto Acknowledge Mode
3 (R/W)	DIL	Disable Internal Loop. The CAN_DBG.DIL bit disables internal loop mode, which routes the transmit output to the receive input.	
		0	Enable Internal Loop
		1	Disable Internal Loop
2 (R/W)	DTO	Disable Tx Output Pin. The CAN_DBG.DTO bit disables the CAN_TX pin.	
		0	Enable Tx Output Pin
		1	Disable Tx Output Pin, Drive Recessive
1 (R/W)	DRI	Disable Receive Input Pin. The CAN_DBG.DRI bit disables the CAN_RX pin.	
		0	Enable Rx Input Pin
		1	Disable Rx Input Pin, Drive Recessive Internally
0 (R/W)	DEC	Disable Transmit and Receive Error Counters. The CAN_DBG.DEC bit disables the transmit and receive error counters in the CAN_CEC register. When set, the CAN_CEC holds its current contents and is not allowed to increment or decrement the error counters. Note that this mode does not conform to the CAN specification.	
		0	Enable CEC Tx and Rx Error Counters
		1	Disable CEC Tx and Rx Error Counters

Error Status Register

The CAN_ESR register, CAN_CEC register, and CAN_EWR register control CAN warnings and errors. All bits in the CAN_ESR register are W1C. Note that the CAN updates the CAN_CEC register when error status is detected in the CAN_ESR register. For detailed information about error and warning operations, see the Operating Modes section.

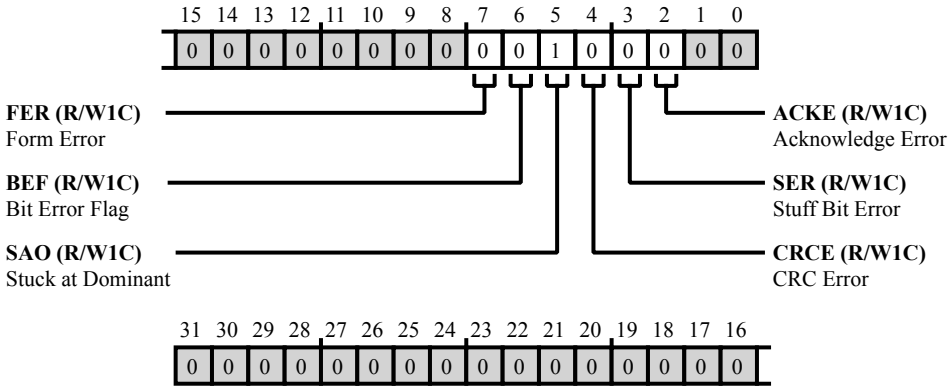


Figure 27-21: CAN_ESR Register Diagram

Table 27-21: CAN_ESR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	FER	Form Error. The CAN_ESR.FER bit indicates when a form error occurs, indicating that a fixed-form bit position in the CAN frame contains one or more illegal bits. This occurs when a dominant bit is detected at a delimiter or end-of-frame bit position.	
		0	No Status
		1	Form Error
6 (R/W1C)	BEF	Bit Error Flag. The CAN_ESR.BEF bit indicates (detected by the transmitting node only) when the value on the CAN_RX pin does not equal what is being transmitted on the CAN_TX pin. When a node is transmitting, it continuously monitors its receive pin (CAN_RX) and compares the received data with the transmitted data. The node postpones the transmission (during the arbitration phase) if the received and transmitted data do not match. After the arbitration phase (CAN_MB[nn]_ID1.RTR bit sent successfully), a bit error is signaled when the value on the CAN_RX pin does not equal what is being transmitted on the CAN_TX pin.	
		0	No Status
		1	Bit Error Flag

Table 27-21: CAN_ESR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	SAO	Stuck at Dominant. The CAN_ESR.SAO bit indicates when the CAN_RX pin sticks at dominant level, indicating that shorted wires are likely.	
		0	No Status
		1	Stuck At Dominant
4 (R/W1C)	CRCE	CRC Error. The CAN_ESR.CRCE bit indicates when a CRC error occurs. This error may occur when a receiver calculates the CRC on the data it received and finds the value different than the CRC that was transmitted on the bus.	
		0	No Status
		1	CRC Error
3 (R/W1C)	SER	Stuff Bit Error. The CAN_ESR.SER bit indicates when a stuff bit error (stuffed 6th consecutive bit value is the same as the previous five bits) occurs. The CAN specification requires that the transmitter insert an extra stuff bit of opposite value after 5 bits have been transmitted with the same value. The receiver disregards the value of these stuff bits. The receiver takes advantage of the signal edge to re-synchronize itself. A stuff bit error occurs on receiving nodes when the 6th consecutive bit value is the same as the previous five bits.	
		0	No Status
		1	Stuff Bit Error Receive
2 (R/W1C)	ACKE	Acknowledge Error. The CAN_ESR.ACKE bit indicates when an acknowledge error occurs, indicating that a message is sent and no receivers drive an acknowledge bit.	
		0	No Status
		1	Acknowledge Error

Error Counter Warning Level Register

The CAN_EWR register, CAN_CEC register, and CAN_ESR register control CAN warnings and errors. For detailed information about error and warning operations, see the Operating Modes section.

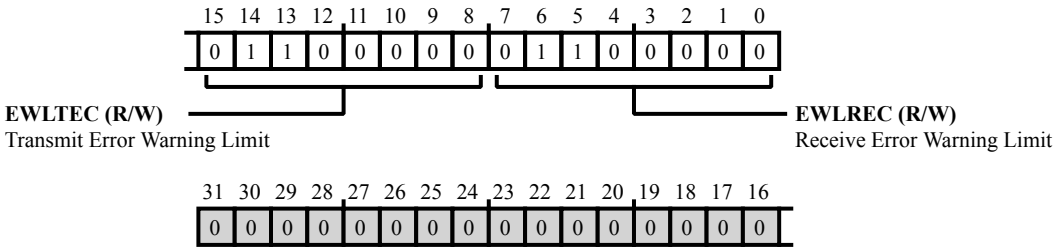


Figure 27-22: CAN_EWR Register Diagram

Table 27-22: CAN_EWR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	EWLTEC	Transmit Error Warning Limit. The CAN_EWR.EWLTEC bits select the transmit error warning limit, which is used as a condition for the CAN_GIS.EWTIS interrupt.
7:0 (R/W)	EWLREC	Receive Error Warning Limit. The CAN_EWR.EWLREC bits select the receive error warning limit, which is used as a condition for the CAN_GIS.EWRIS interrupt.

Global CAN Interrupt Flag Register

The CAN_GIF register, CAN_GIF register, and CAN_GIM register control CAN interrupt requests. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIF register holds the interrupt flag. The CAN_INT.GIRQ bit is only asserted if a bit in the CAN_GIF is set. The CAN_INT.GIRQ bit remains set as long as at least one bit in the CAN_GIF register is set. All bits in this register are W1C.

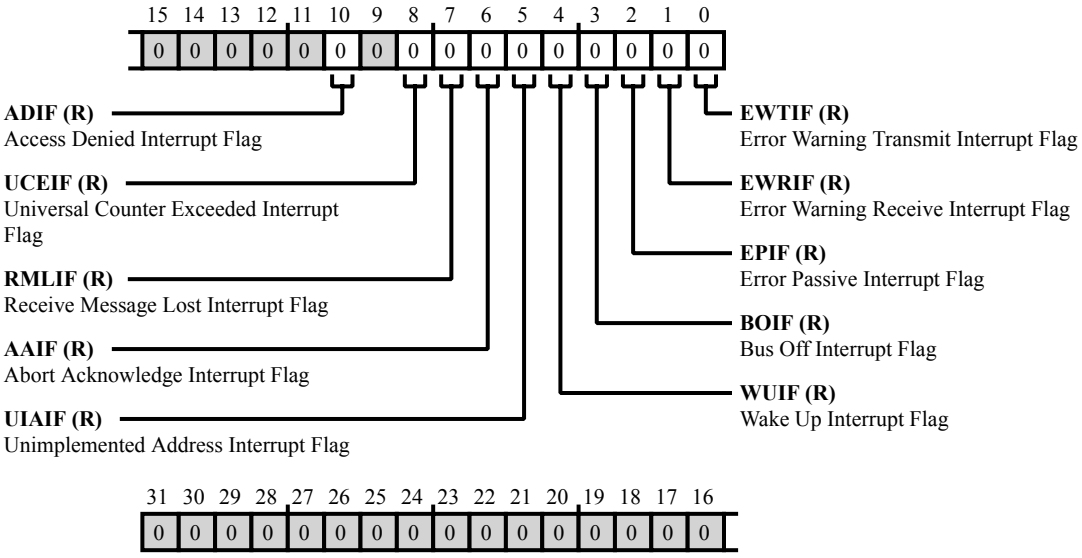


Figure 27-23: CAN_GIF Register Diagram

Table 27-23: CAN_GIF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	ADIF	Access Denied Interrupt Flag. The CAN_GIF.ADIF bit indicates that the access denied interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
8 (R/NW)	UCEIF	Universal Counter Exceeded Interrupt Flag. The CAN_GIF.UCEIF bit indicates that the universal counter exceeded interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Table 27-23: CAN_GIF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	RMLIF	Receive Message Lost Interrupt Flag. The CAN_GIF.RMLIF bit indicates that the receive message lost interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
6 (R/NW)	AAIF	Abort Acknowledge Interrupt Flag. The CAN_GIF.AAIF bit indicates that the abort acknowledge interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
5 (R/NW)	UIAIF	Unimplemented Address Interrupt Flag. The CAN_GIF.UIAIF bit indicates that the unimplemented address interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
4 (R/NW)	WUIF	Wake Up Interrupt Flag. The CAN_GIF.WUIF bit indicates that the wake up interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
3 (R/NW)	BOIF	Bus Off Interrupt Flag. The CAN_GIF.BOIF bit indicates that the bus off interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
2 (R/NW)	EPIF	Error Passive Interrupt Flag. The CAN_GIF.EPIF bit indicates that the error passive mode interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)
1 (R/NW)	EWRIF	Error Warning Receive Interrupt Flag. The CAN_GIF.EWRIF bit indicates that the error warning receive interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Table 27-23: CAN_GIF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	EWTIF	Error Warning Transmit Interrupt Flag. The `CAN_GIF.EWTIF` bit indicates that the error warning transmit interrupt flag is set (latched).	
		0	No Interrupt Flag
		1	Interrupt Flag Set (Latched)

Global CAN Interrupt Mask Register

The CAN_GIM register, CAN_GIF register, and CAN_GIF register control CAN interrupt requests. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIM register holds the interrupt mask. The interrupt mask bits only affect the content of the CAN_GIF register. If the mask bit is not set (enabled/unmasked), the corresponding flag bit is not set when the event occurs.

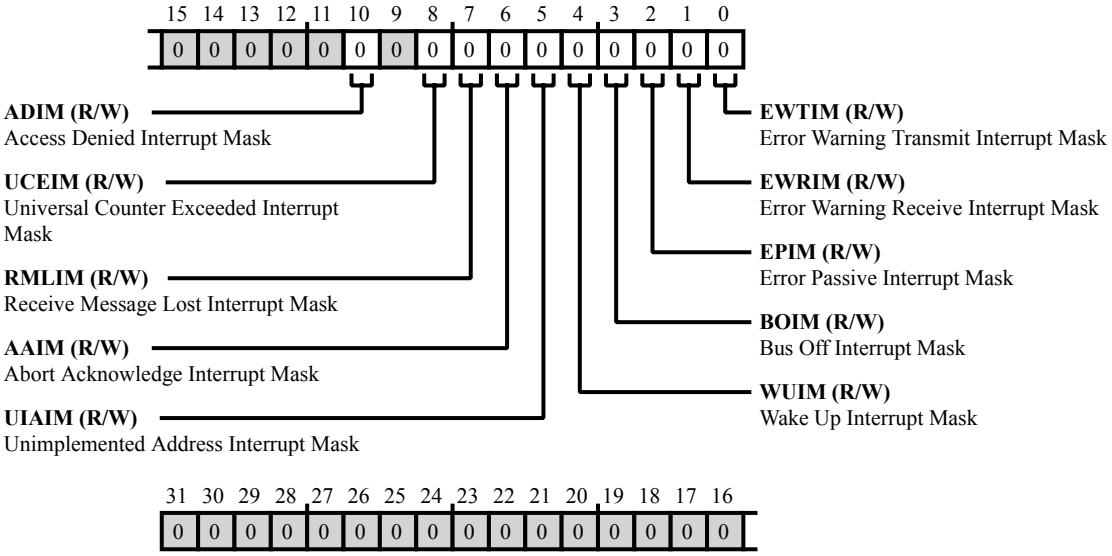


Figure 27-24: CAN_GIM Register Diagram

Table 27-24: CAN_GIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	ADIM	Access Denied Interrupt Mask. The CAN_GIM.ADIM bit enables (unmasks) the access denied interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
8 (R/W)	UCEIM	Universal Counter Exceeded Interrupt Mask. The CAN_GIM.UCEIM bit enables (unmasks) the universal counter exceeded interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Table 27-24: CAN_GIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	RMLIM	Receive Message Lost Interrupt Mask. The CAN_GIM.RMLIM bit enables (unmasks) the receive message lost interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
6 (R/W)	AAIM	Abort Acknowledge Interrupt Mask. The CAN_GIM.AAIM bit enables (unmasks) the abort acknowledge interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
5 (R/W)	UIAIM	Unimplemented Address Interrupt Mask. The CAN_GIM.UIAIM bit enables (unmasks) the unimplemented address interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
4 (R/W)	WUIM	Wake Up Interrupt Mask. The CAN_GIM.WUIM bit enables (unmasks) the wake up interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
3 (R/W)	BOIM	Bus Off Interrupt Mask. The CAN_GIM.BOIM bit enables (unmasks) the bus off interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
2 (R/W)	EPIM	Error Passive Interrupt Mask. The CAN_GIM.EPIM bit enables (unmasks) the error passive mode interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)
1 (R/W)	EWRIM	Error Warning Receive Interrupt Mask. The CAN_GIM.EWRIM bit enables (unmasks) the error warning receive interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Table 27-24: CAN_GIM Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EWTIM	Error Warning Transmit Interrupt Mask. The `CAN_GIM.EWTIM` bit enables (unmasks) the error warning transmit interrupt request.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Global CAN Interrupt Status Register

The CAN_GIS register, CAN_GIF register, and CAN_GIM register control CAN interrupt requests. For detailed information about interrupt operations, see the Event Control section.

The CAN_GIS register holds the interrupt status. All bits in this register are W1C.

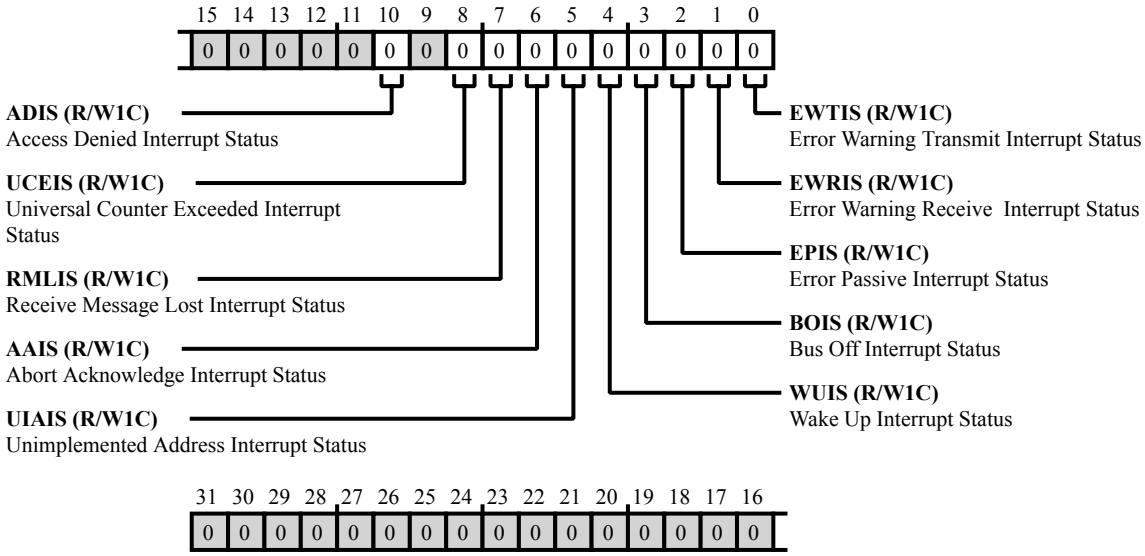


Figure 27-25: CAN_GIS Register Diagram

Table 27-25: CAN_GIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	ADIS	Access Denied Interrupt Status. The CAN_GIS.ADIS bit indicates when at least one access to the mailbox RAM occurred during a data update by internal logic.	
		0	No Interrupt Pending
		1	Interrupt Pending
8 (R/W1C)	UCEIS	Universal Counter Exceeded Interrupt Status. The CAN_GIS.UCEIS bit indicates when there has been an overflow of the universal counter (in time stamp mode or event counter mode) or the counter has reached the value 0x0000 (in watchdog mode).	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 27-25: CAN_GIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	RMLIS	Receive Message Lost Interrupt Status. The CAN_GIS.RMLIS bit indicates when a message is received for a mailbox that currently contains unread data. At least one bit in the receive message lost register (CAN_RML1 or CAN_RML2) is set. If the bit in CAN_GIS (and CAN_GIF) is reset and there is at least one bit in CAN_RML1 or CAN_RML2 still set, the bit in CAN_GIF (and CAN_GIF) is not set again. The internal interrupt source signal is only active if a new bit in CAN_RML1 or CAN_RML2 is set.	
		0	No Interrupt Pending
		1	Interrupt Pending
6 (R/W1C)	AAIS	Abort Acknowledge Interrupt Status. The CAN_GIS.AAIS bit indicates when At least one abort acknowledge bit is set in the CAN_AA1 or the CAN_AA2 registers. If the bit in CAN_GIS (and CAN_GIF) is reset and there is at least one bit in CAN_AA1 or CAN_AA2 still set, the bit in CAN_GIS (and CAN_GIF) is not set again. The internal interrupt source signal is only active if a new bit in CAN_AA1 or CAN_AA2 is set. The abort acknowledge bits maintain state even after the corresponding mailbox n is disabled.	
		0	No Interrupt Pending
		1	Interrupt Pending
5 (R/W1C)	UIAIS	Unimplemented Address Interrupt Status. The CAN_GIS.UIAIS bit indicates when there was a processor core access to an address that is not implemented in the CAN.	
		0	No Interrupt Pending
		1	Interrupt Pending
4 (R/W1C)	WUIS	Wake Up Interrupt Status. The CAN_GIS.WUIS bit indicates when the CAN has left the sleep mode because of detected activity on the CAN bus line.	
		0	No Interrupt Pending
		1	Interrupt Pending
3 (R/W1C)	BOIS	Bus Off Interrupt Status. The CAN_GIS.BOIS bit indicates when the CAN has entered the bus-off state. This interrupt source is active if the status of the CAN changes from normal operation mode to the bus-off mode. If the bit in CAN_GIS (and CAN_GIF) is reset and the bus-off mode is still active, this bit is not set again. If the module leaves the bus-off mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 27-25: CAN_GIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	EPIS	Error Passive Interrupt Status. The CAN_GIS.EPIS bit indicates when the CAN has entered the error passive state. This interrupt source is active if the status of the CAN changes from the error active mode to the error passive mode. If the bit in CAN_GIS (and CAN_GIF) is reset and the error passive mode is still active, this bit is not set again. If the CAN leaves the error passive mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
1 (R/W1C)	EWRIS	Error Warning Receive Interrupt Status. The CAN_GIS.EWRIS bit indicates when the CAN_CEC.RXECNT has reached the warning limit. If the bit in CAN_GIS (and CAN_GIF) is reset and the error warning mode is still active, this bit is not set again. If the CAN leaves the error warning mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending
0 (R/W1C)	EWTIS	Error Warning Transmit Interrupt Status. The CAN_GIS.EWTIS bit indicates when the CAN_CEC.TXECNT has reached the warning limit. If the bit in CAN_GIS (and CAN_GIF) is reset and the error warning mode is still active, this bit is not set again. If the CAN leaves the error warning mode, the bit in CAN_GIS (and CAN_GIF) remains set.	
		0	No Interrupt Pending
		1	Interrupt Pending

Interrupt Pending Register

The CAN_INT register indicates the status of pending CAN interrupts and indicates the state of the CAN_RX and CAN_TX pins. Though this register is read-only, a write is allowed to exit the built-in sleep mode of the module on processors supporting this feature.

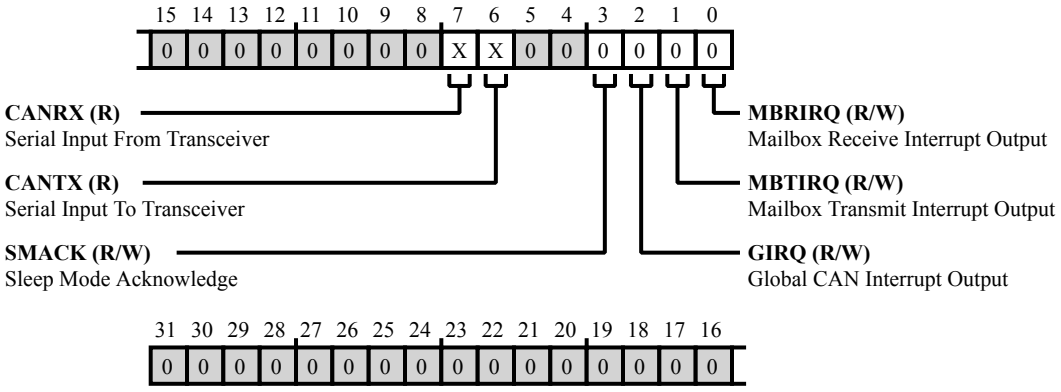


Figure 27-26: CAN_INT Register Diagram

Table 27-26: CAN_INT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	CANRX	Serial Input From Transceiver. The CAN_INT.CANRX bit indicates the logic value that the CAN detects on the CAN_RX pin. Note that the reset/default value for CAN_INT.CANRX is dependent on pin values.	
		0	Dominant Value (Low Active)
		1	Recessive Value (High Active)
6 (R/NW)	CANTX	Serial Input To Transceiver. The CAN_INT.CANTX bit indicates the logic value that the CAN detects on the CAN_TX pin. Note that the reset/default value for CAN_INT.CANTX is dependent on pin values.	
		0	Dominant Value (Low Active)
		1	Recessive Value (High Active)
3 (R/W)	SMACK	Sleep Mode Acknowledge. The CAN_INT.SMACK bit indicates when the CAN has entered sleep mode.	
		0	Not in Sleep Mode
		1	Sleep Mode

Table 27-26: CAN_INT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	GIRQ	Global CAN Interrupt Output. The CAN_INT.GIRQ bit indicates when at least one bit is set in the CAN_GIF register, indicating at least one unmasked CAN is flagged (latched). The CAN_INT.GIRQ bit remains set as long as at least one bit is set in the CAN_GIF register.	
		0	No CAN Global Interrupt Flag Set
		1	CAN Global Interrupt Flag (1 or More) Set
1 (R/W)	MBTIRQ	Mailbox Transmit Interrupt Output. The CAN_INT.MBTIRQ bit indicates when any bits are set in the CAN_MBTIF1 register or CAN_MBTIF2 register, indicating transmit.	
		0	No CAN Transmit Flags Set
		1	CAN Transmit Flags Set (1 or More)
0 (R/W)	MBRIRQ	Mailbox Receive Interrupt Output. The CAN_INT.MBRIRQ bit indicates when any bits are set in the CAN_MBRIF1 register or CAN_MBRIF2 register, indicating receive.	
		0	No CAN Receive Flags Set
		1	CAN Receive Flags Set (1 or More)

Mailbox Interrupt Mask 1 Register

The CAN_MBIM1 register enables transmit and receive interrupt requests for mailboxes 0 through 15. Each bit in this register requests enables the transmit or receive interrupt request for the corresponding mailbox when set (=1).

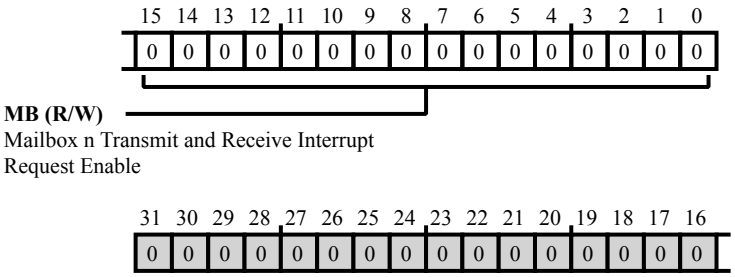


Figure 27-27: CAN_MBIM1 Register Diagram

Table 27-27: CAN_MBIM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Request Enable.

Mailbox Interrupt Mask 2 Register

The CAN_MBIM2 register enables transmit and receive interrupt requests for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests enables the transmit or receive interrupt request for the corresponding mailbox when set (=1).

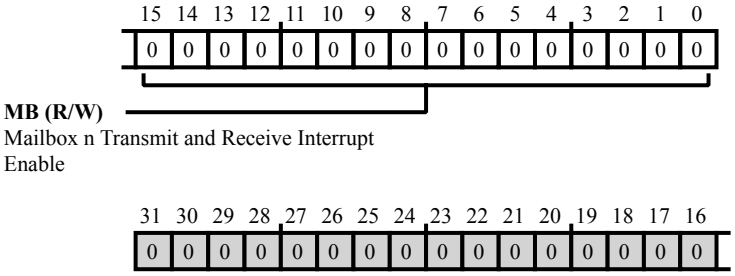


Figure 27-28: CAN_MBIM2 Register Diagram

Table 27-28: CAN_MBIM2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Enable.

Mailbox Receive Interrupt Flag 1 Register

The CAN_MBRIF1 register indicates when a receive interrupt request is pending---due to successful reception (corresponding CAN_RMP1 bit set) and the interrupt is enabled (corresponding CAN_MBIM1 bit set)---for mailboxes 0 through 15. Each bit in this register indicates the receive interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBRIF1 is set, the CAN receive interrupt request is raised (CAN_INT.MBRIRQ bit set). To clear the interrupt request, all of the set bits in CAN_RMP1 must be cleared by software, then the associated bits set in CAN_MBRIF1 must be cleared (W1C).

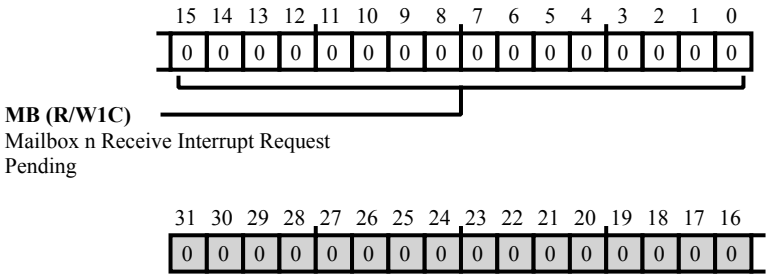


Figure 27-29: CAN_MBRIF1 Register Diagram

Table 27-29: CAN_MBRIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Receive Interrupt Request Pending.

Mailbox Receive Interrupt Flag 2 Register

The CAN_MBRIF2 register indicates when a receive interrupt request is pending---due to successful reception (corresponding CAN_RMP2 bit set) and the interrupt is enabled (corresponding CAN_MBIM2 bit set)---for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the receive interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBRIF2 is set, the CAN receive interrupt request is raised (CAN_INT.MBRIRQ bit set). To clear the interrupt request, all of the set bits in CAN_RMP2 must be cleared by software, then the associated bits set in CAN_MBRIF2 must be cleared (W1C). Bits 8 through 15 are reserved and read-only, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

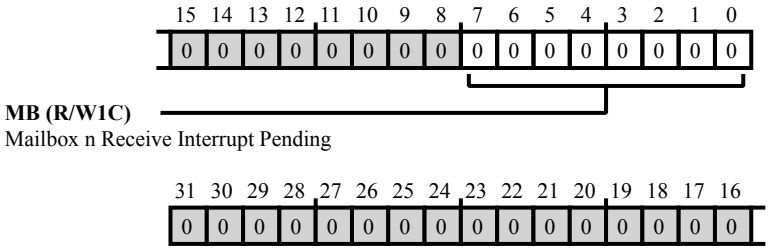


Figure 27-30: CAN_MBRIF2 Register Diagram

Table 27-30: CAN_MBRIF2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	MB	Mailbox n Receive Interrupt Pending.

Temporary Mailbox Disable Register

The CAN_MBTD register supports temporarily and selectively disabling CAN mailboxes. For more information about this feature, see the Operating Modes section.

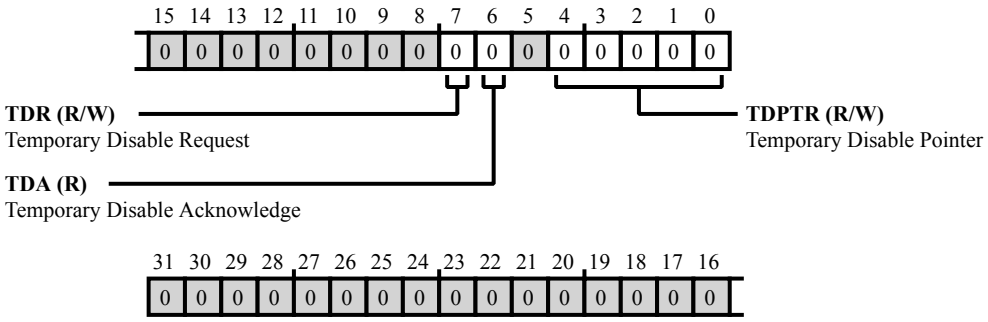


Figure 27-31: CAN_MBTD Register Diagram

Table 27-31: CAN_MBTD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	TDR	Temporary Disable Request. The CAN_MBTD.TDR bit holds the pointer to mailbox, which is disabled when the CAN_MBTD.TDR bit is set.	
		0	No Request
		1	Request Temporary Mailbox Disable
6 (R/NW)	TDA	Temporary Disable Acknowledge. The CAN_MBTD.TDA bit indicates when the mailbox (to which the CAN_MBTD.TDPTR bit points) is disabled. When this bit is set for a mailbox, only the data field of that mailbox may be updated. Accesses that mailboxs control bits and the identifier are denied.	
		0	No Acknowledge
		1	Acknowledge Temporary Mailbox Disable
4:0 (R/W)	TDPTR	Temporary Disable Pointer. The CAN_MBTD.TDPTR bits hold the pointer to mailbox, which is disabled when the CAN_MBTD.TDR bit is set.	

Mailbox Transmit Interrupt Flag 1 Register

The CAN_MBTIF1 register indicates when a transmit interrupt request is pending---due to successful transmission (corresponding CAN_TA1 bit is set) and the interrupt is enabled (corresponding CAN_MBIM1 bit is set)---for mailboxes 8 through 15. Each bit in this register indicates the transmit interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBTIF1 is set, the CAN transmit interrupt request is raised (CAN_INT.MBTIRQ bit set). To clear the interrupt request, all of the set bits in CAN_MBTIF1 must be cleared by software (W1C). Also, software must clear the associated bits set in CAN_TA1 or set the associated bits in CAN_TRS1 bit to clear the interrupt source asserting the bits in CAN_MBTIF1. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

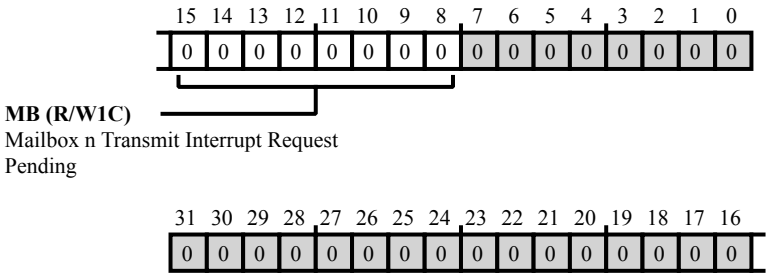


Figure 27-32: CAN_MBTIF1 Register Diagram

Table 27-32: CAN_MBTIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Interrupt Request Pending.

Mailbox Transmit Interrupt Flag 2 Register

The CAN_MBTIF2 register indicates when a transmit interrupt request is pending---due to successful transmission (corresponding CAN_TA2 bit is set) and the interrupt is enabled (corresponding CAN_MBIM2 bit is set)---for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates the transmit interrupt pending status for the corresponding mailbox when set (=1). When any bit in CAN_MBTIF2 is set, the CAN transmit interrupt request is raised (CAN_INT.MBTIRQ bit is set). To clear the interrupt request, all of the set bits in CAN_MBTIF2 must be cleared by software (W1C). Also, software must clear the associated bits set in CAN_TA2 or set the associated bits in CAN_TRS2 bit to clear the interrupt source asserting the bits in CAN_MBTIF2.

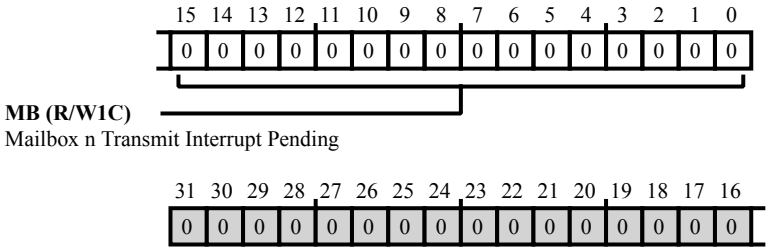


Figure 27-33: CAN_MBTIF2 Register Diagram

Table 27-33: CAN_MBTIF2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Transmit Interrupt Pending.

Mailbox Word 0 Register

The CAN_MB[nn]_DATA0 register holds mailbox data bytes.

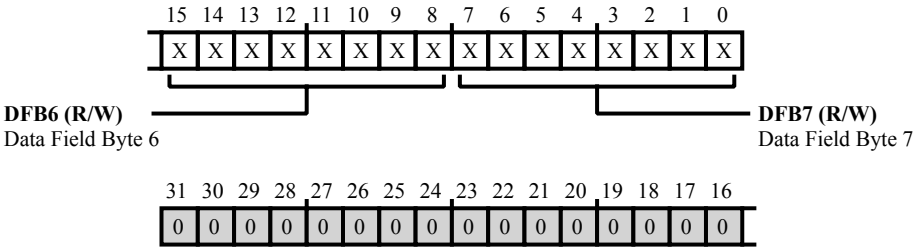


Figure 27-34: CAN_MB[nn]_DATA0 Register Diagram

Table 27-34: CAN_MB[nn]_DATA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB6	Data Field Byte 6. The CAN_MB[nn]_DATA0.DFB6 bits hold mailbox data.
7:0 (R/W)	DFB7	Data Field Byte 7. The CAN_MB[nn]_DATA0.DFB7 bits hold mailbox data.

Mailbox Word 1 Register

The `CAN_MB[nn]_DATA1` register holds mailbox data bytes.

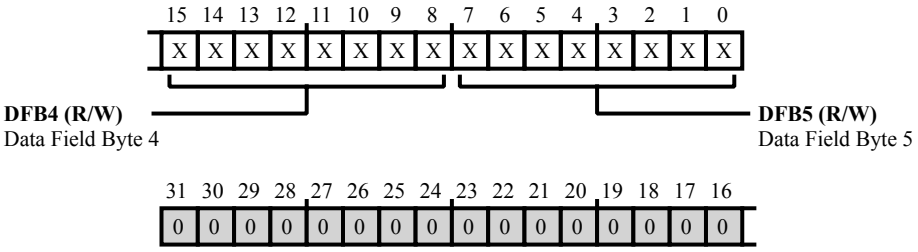


Figure 27-35: CAN_MB[nn]_DATA1 Register Diagram

Table 27-35: CAN_MB[nn]_DATA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB4	Data Field Byte 4. The `CAN_MB[nn]_DATA1.DFB4` bits hold mailbox data.
7:0 (R/W)	DFB5	Data Field Byte 5. The `CAN_MB[nn]_DATA1.DFB5` bits hold mailbox data.

Mailbox Word 2 Register

The CAN_MB[nn]_DATA2 register holds mailbox data bytes.

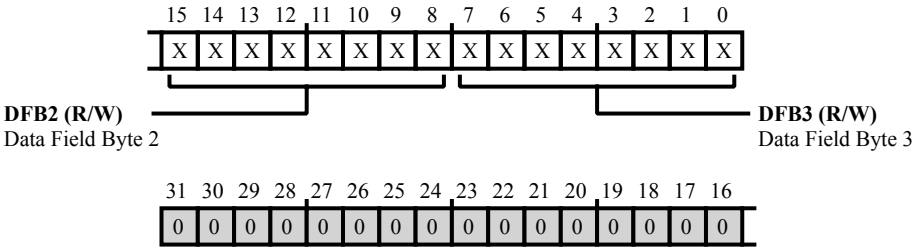


Figure 27-36: CAN_MB[nn]_DATA2 Register Diagram

Table 27-36: CAN_MB[nn]_DATA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB2	Data Field Byte 2. The CAN_MB[nn]_DATA2.DFB2 bits hold mailbox data.
7:0 (R/W)	DFB3	Data Field Byte 3. The CAN_MB[nn]_DATA2.DFB3 bits hold mailbox data.

Mailbox Word 3 Register

The CAN_MB[nn]_DATA3 register holds mailbox data bytes.

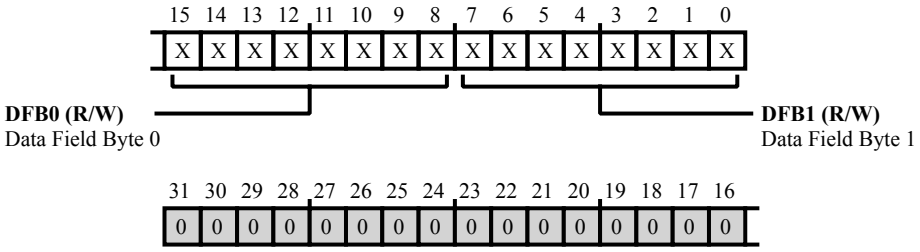


Figure 27-37: CAN_MB[nn]_DATA3 Register Diagram

Table 27-37: CAN_MB[nn]_DATA3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	DFB0	Data Field Byte 0. The CAN_MB[nn]_DATA3.DFB0 bits hold mailbox data.
7:0 (R/W)	DFB1	Data Field Byte 1. The CAN_MB[nn]_DATA3.DFB1 bits hold mailbox data.

Mailbox ID 0 Register

The `CAN_MB[nn]_ID0` register contains the lower 16 bits of the 18-bit extended identifier.

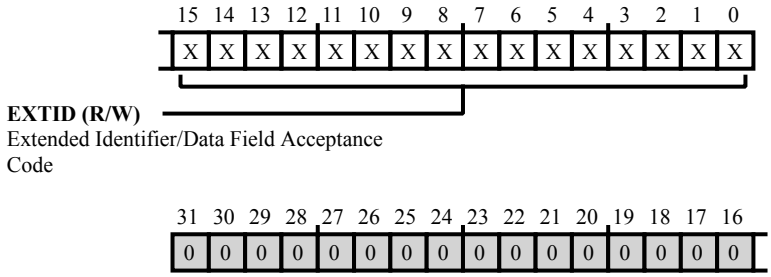


Figure 27-38: CAN_MB[nn]_ID0 Register Diagram

Table 27-38: CAN_MB[nn]_ID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	EXTID	Extended Identifier/Data Field Acceptance Code. The `CAN_MB[nn]_ID0.EXTID` bits hold the lower 16 bits of the 18-bit extended ID.

Mailbox ID 1 Register

The CAN_MB[nn]_ID1 register contains the identifier bits of mailbox. The 11-bit BASE_ID is mapped to The CAN_MB[nn]_ID1.BASEID field. It also enables the extended identification and contains upper two bits of 18-bit extended identifier.

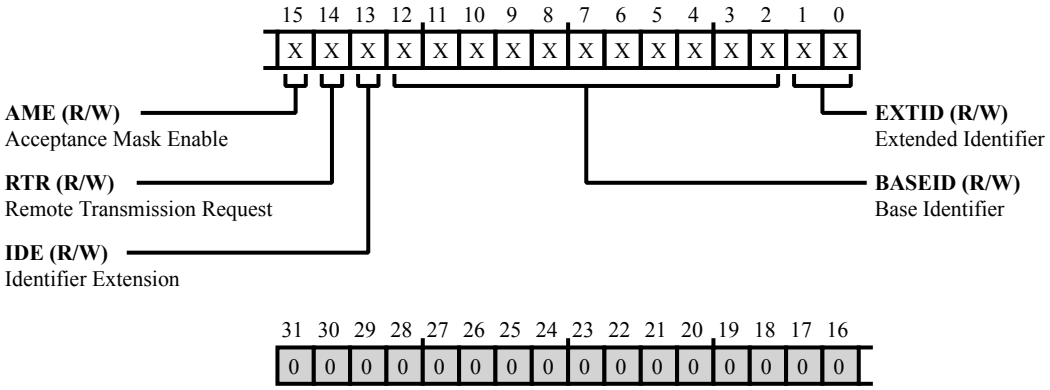


Figure 27-39: CAN_MB[nn]_ID1 Register Diagram

Table 27-39: CAN_MB[nn]_ID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/W)	AME	Acceptance Mask Enable. The CAN_MB[nn]_ID1.AME bit enables acceptance mask operations if the mailbox is configured as receiver. When enabled (=1), only those bits that have the corresponding mask bit cleared are compared to the received message ID. A bit position that is set in the mask register does not need to match. This bit should be set to 0 when the mailbox is configured in transmit mode.
14 (R/W)	RTR	Remote Transmission Request. The CAN_MB[nn]_ID1.RTR bit selects whether the frame contains data (data frame) or contains a request for data associated with the message identifier in the frame being sent (remote frame).
13 (R/W)	IDE	Identifier Extension. The CAN_MB[nn]_ID1.IDE bit enables the comparison of the received message ID to the value in the CAN_MB[nn]_ID1.EXTID and CAN_MB[nn]_ID0.EXTID bits. When configured as transmitter, it sends the extended identifier in addition to the base identifier.
12:2 (R/W)	BASEID	Base Identifier. The CAN_MB[nn]_ID1.BASEID bits hold the base identifier for acceptance mask operations.

Table 27-39: CAN_MB[nn]_ID1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	EXTID	Extended Identifier. The `CAN_MB[nn]_ID1.EXTID` bits hold the upper two bits of 18-bit extended identifier.

Mailbox Length Register

The CAN_MB[nn]_LENGTH register holds the data length code for the received remote frame. For more information about remote frames, see the Remote Frame Handling section.

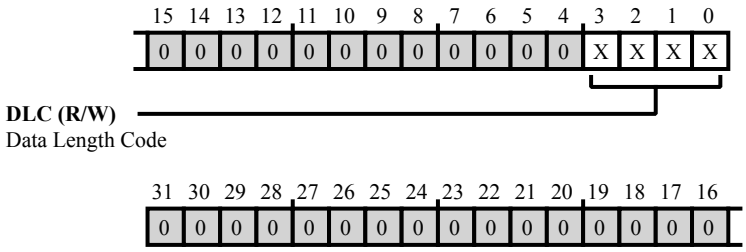


Figure 27-40: CAN_MB[nn]_LENGTH Register Diagram

Table 27-40: CAN_MB[nn]_LENGTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	DLC	Data Length Code. The CAN_MB[nn]_LENGTH.DLC bits hold the DLC value of the received remote frame. The received value overwrites any previous value.

Mailbox Time Stamp Register

The CAN_MB[nn]_TIMESTAMP register holds an indication of the time of reception or transmission for each message, when the universal counter is in time stamp mode (CAN_UCCNF.UCCNF =0x1). In this mode, the CAN writes the value of the counter (CAN_UCCNT) to the CAN_MB[nn]_TIMESTAMP register when a received message is stored or a message is transmitted. For more information about time stamps, see the Time Stamps section.

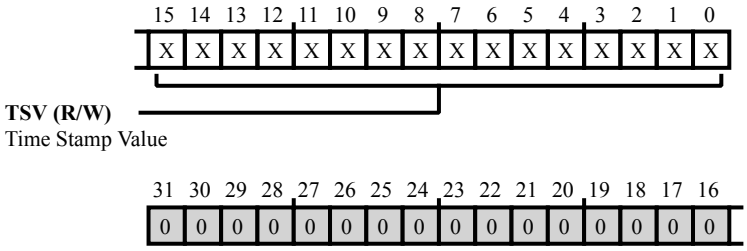


Figure 27-41: CAN_MB[nn]_TIMESTAMP Register Diagram

Table 27-41: CAN_MB[nn]_TIMESTAMP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TSV	Time Stamp Value. The CAN_MB[nn]_TIMESTAMP.TSV bits hold the message time stamp value.

Mailbox Configuration 1 Register

The CAN_MC1 register enables mailboxes 0 through 15. Each bit in this register enables or disables the corresponding mailbox. For all bits, set the bit (=1) to enable the mailbox, and clear the bit (=0) to disable the mailbox.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS1 bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS1 bit is reset by the internal logic can cause unpredictable results.

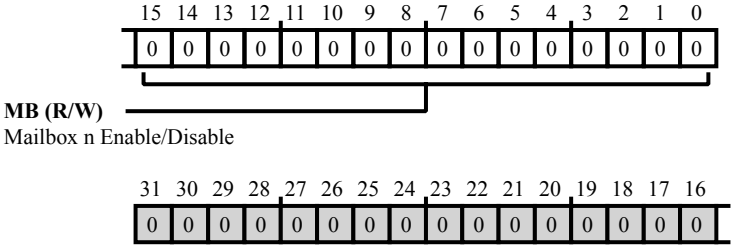


Figure 27-42: CAN_MC1 Register Diagram

Table 27-42: CAN_MC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Enable/Disable.

Mailbox Configuration 2 Register

The CAN_MC2 register enables mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables or disables the corresponding mailbox. For all bits, set the bit (=1) to enable the mailbox, and clear the bit (=0) to disable the mailbox.

Enabling and disabling mailboxes has an impact on transmit requests. Setting the CAN_TRS2 bit associated with a disabled mailbox may result in erroneous behavior. Similarly, disabling a mailbox before the associated CAN_TRS2 bit is reset by the internal logic can cause unpredictable results.

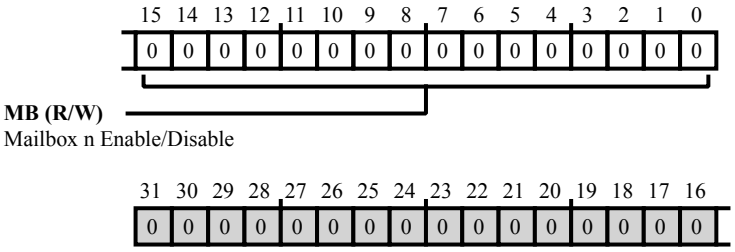


Figure 27-43: CAN_MC2 Register Diagram

Table 27-43: CAN_MC2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Enable/Disable.

Mailbox Direction 1 Register

The CAN_MD1 register selects the data transfer direction for mailboxes 0 through 15. Each bit in this register selects receive mode or transmit mode for the corresponding mailbox. For all bits, set the bit (=1) for receive mode from the mailbox, and clear the bit (=0) for transmit mode to the mailbox. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

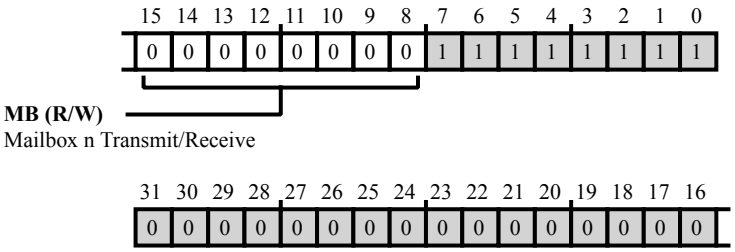


Figure 27-44: CAN_MD1 Register Diagram

Table 27-44: CAN_MD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit/Receive.

Mailbox Direction 2 Register

The CAN_MD2 register selects the data transfer direction for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register selects receive mode or transmit mode for the corresponding mailbox. For all bits, set the bit (=1) for receive mode from the mailbox, and clear the bit (=0) for transmit mode to the mailbox. Bits 8 through 15 are read-only, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

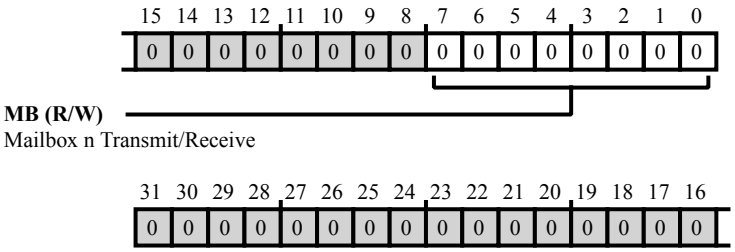


Figure 27-45: CAN_MD2 Register Diagram

Table 27-45: CAN_MD2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MB	Mailbox n Transmit/Receive.

Overwrite Protection/Single Shot Transmission 1 Register

The CAN_OPSS1 register enables overwrite protection for mailboxes 0 through 15. Each bit in this register enables overwrite protection for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote overwrite protection, see the detailed feature description in the CAN Functional Description section. For more information about how this feature affects transmit and receive operations, see the CAN Operating Modes sections, describing transmit and receive operations.

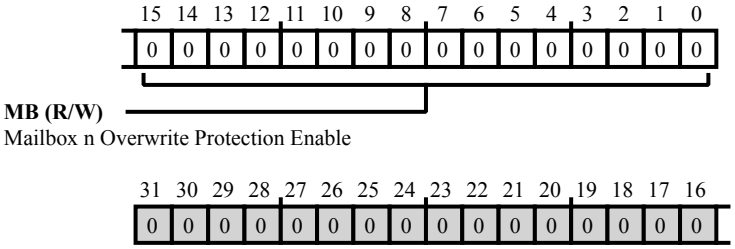


Figure 27-46: CAN_OPSS1 Register Diagram

Table 27-46: CAN_OPSS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Overwrite Protection Enable.

Overwrite Protection/Single Shot Transmission 2 Register

The CAN_OPSS2 register enables overwrite protection for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables overwrite protection for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote overwrite protection, see the detailed feature description in the CAN Functional Description section. For more information about how this feature affects transmit and receive operations, see the CAN Operating Modes sections, describing transmit and receive operations.

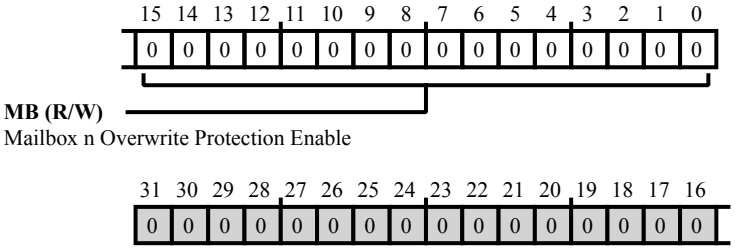


Figure 27-47: CAN_OPSS2 Register Diagram

Table 27-47: CAN_OPSS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Overwrite Protection Enable.

Remote Frame Handling 1 Register

The CAN_RFH1 register enables remote frame handling for mailboxes 8 through 15. Each bit in this register enables remote frame handling for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote frame handling, see the CAN Operating Modes sections, describing transmit and receive operations. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

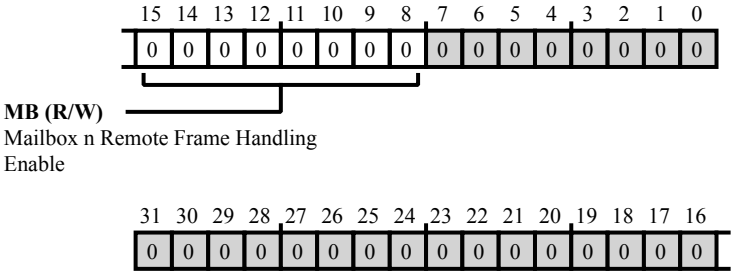


Figure 27-48: CAN_RFH1 Register Diagram

Table 27-48: CAN_RFH1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Remote Frame Handling Enable.

Remote Frame Handling 2 Register

The CAN_RFH2 register enables remote frame handling for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register enables remote frame handling for the corresponding mailbox when set (=1). Note that enabling this bit affects transmit and receive operations for mailboxes. For more information about remote frame handling, see the CAN Operating Modes sections, describing transmit and receive operations.

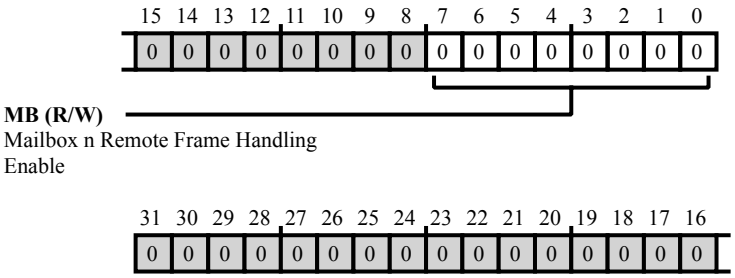


Figure 27-49: CAN_RFH2 Register Diagram

Table 27-49: CAN_RFH2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MB	Mailbox n Remote Frame Handling Enable.

Receive Message Lost 1 Register

The CAN_RML1 register indicates when a message is lost---due to a message coming while there is pending data (corresponding CAN_RMP1 bit set) and overwrite protection is disabled (CAN_OPSS1 bit cleared)---for mailboxes 0 through 15. Each bit in this register indicates the message lost status for the corresponding mailbox when set (=1).

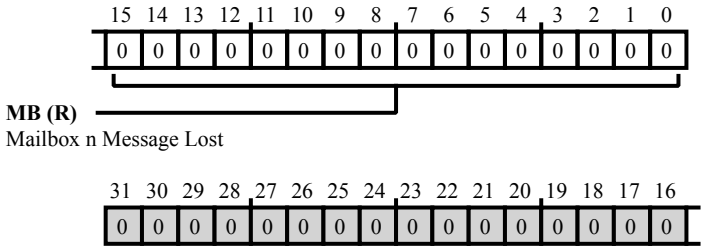


Figure 27-50: CAN_RML1 Register Diagram

Table 27-50: CAN_RML1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	MB	Mailbox n Message Lost.

Receive Message Lost 2 Register

The CAN_RML2 register indicates when a message is lost---due to a message coming while there is pending data (corresponding CAN_RMP2 bit set) and overwrite protection is disabled (CAN_OPSS2 bit cleared)---for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the message lost status for the corresponding mailbox when set (=1). Bits 8 through 15 are reserved, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

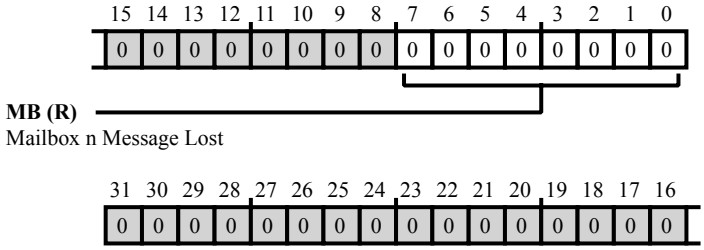


Figure 27-51: CAN_RML2 Register Diagram

Table 27-51: CAN_RML2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	MB	Mailbox n Message Lost.

Receive Message Pending 1 Register

The CAN_RMP1 register indicates when a message is pending (unread data) for mailboxes 0 through 15. Each bit in this register indicates the message pending status for the corresponding mailbox when set (=1).

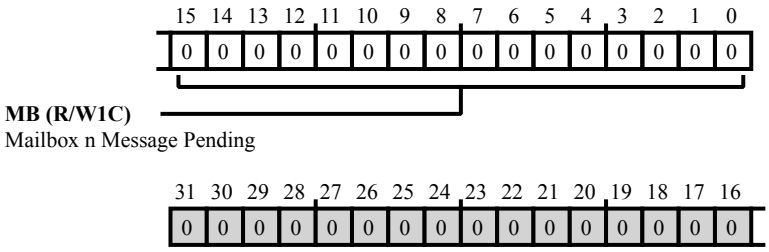


Figure 27-52: CAN_RMP1 Register Diagram

Table 27-52: CAN_RMP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Message Pending.

Receive Message Pending 2 Register

The CAN_RMP2 register indicates when a message is pending (unread data) for mailboxes 16 (bit 0) through 23 (bit 7). Each bit in this register indicates the message pending status for the corresponding mailbox when set (=1). Bits 8 through 15 are reserved, as the corresponding mailboxes (24 through 31) are transmit-only mailboxes.

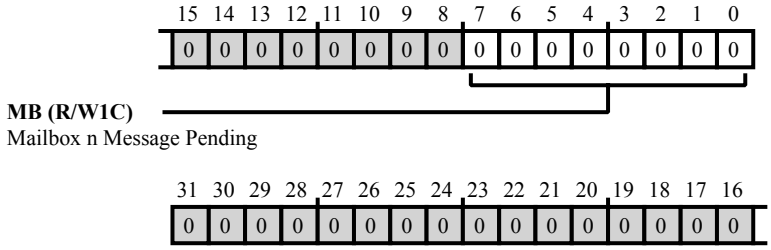


Figure 27-53: CAN_RMP2 Register Diagram

Table 27-53: CAN_RMP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	MB	Mailbox n Message Pending.

Status Register

The CAN_STAT register indicates status for CAN modes and error conditions.

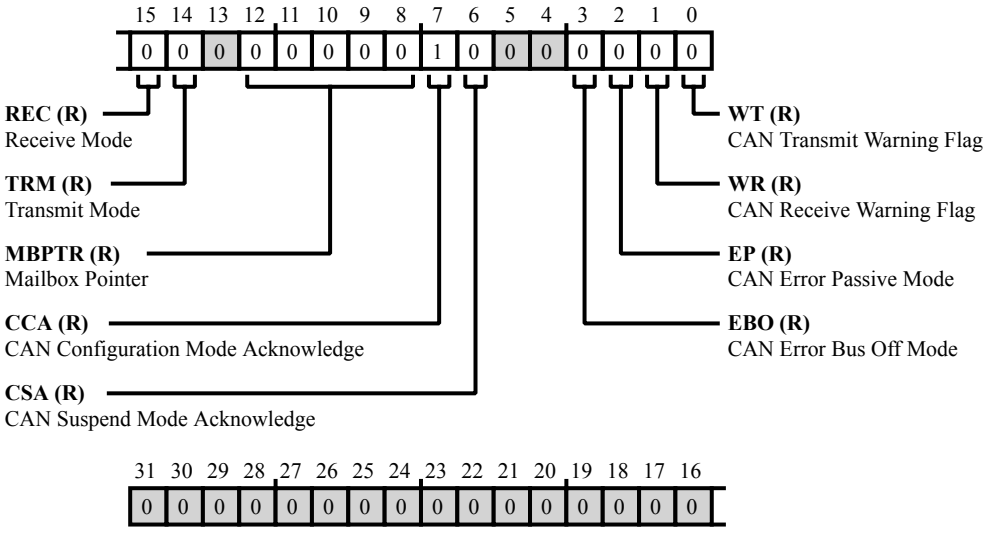


Figure 27-54: CAN_STAT Register Diagram

Table 27-54: CAN_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	REC	Receive Mode. The CAN_STAT.REC bit indicates whether the CAN is in receive mode.	
		0	Not in Receive Mode
		1	Receive Mode
14 (R/NW)	TRM	Transmit Mode. The CAN_STAT.TRM bit indicates whether the CAN is in transmit mode.	
		0	Not in Transmit Mode
		1	Transmit Mode
12:8 (R/NW)	MBPTR	Mailbox Pointer. The CAN_STAT.MBPTR bits represent the mailbox number of the current transmit message. After a successful transmission, these bits remain unchanged.	
		0-31	Processing Mailbox 0 to31 Message
7 (R/NW)	CCA	CAN Configuration Mode Acknowledge. The CAN_STAT.CCA bit indicates whether the CAN is in configuration mode.	
		0	Not in Configuration Mode
		1	Configuration mode

Table 27-54: CAN_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/NW)	CSA	CAN Suspend Mode Acknowledge. The `CAN_STAT.CSA` bit indicates whether the CAN is in suspend mode.	
		0	Not in Suspend Mode
		1	Suspend mode
3 (R/NW)	EBO	CAN Error Bus Off Mode. The `CAN_STAT.EBO` bit indicates whether the CAN is in error bus off mode.	
		0	TXECNT Below 256
		1	TXECNT Above Bus Off Limit
2 (R/NW)	EP	CAN Error Passive Mode. The `CAN_STAT.EP` bit indicates whether the CAN is in error passive mode.	
		0	TXECNT and RXECNT Below 128
		1	TXECNT or RXECNT Above EP Level
1 (R/NW)	WR	CAN Receive Warning Flag. The `CAN_STAT.WR` bit indicates whether the CAN has detected a receive warning flag condition.	
		0	RXECNT Below Limit
		1	RXECNT at Limit
0 (R/NW)	WT	CAN Transmit Warning Flag. The `CAN_STAT.WT` bit indicates whether the CAN detected a transmit warning flag condition.	
		0	TXECNT Below Limit
		1	TXECNT at Limit

Transmission Acknowledge 1 Register

The CAN_TA1 register indicates transmission success for mailboxes 8 through 15. Each bit in this register indicates transmission success for the corresponding mailbox when set (=1). Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

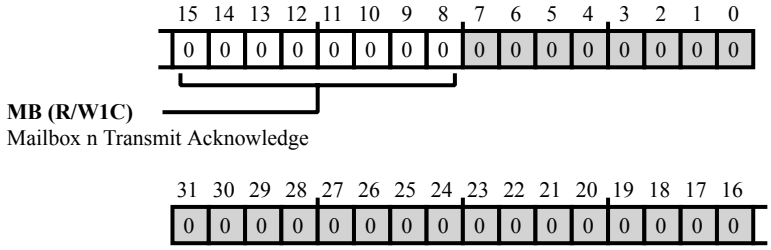


Figure 27-55: CAN_TA1 Register Diagram

Table 27-55: CAN_TA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Acknowledge.

Transmission Acknowledge 2 Register

The CAN_TA2 register indicates transmission success for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register indicates transmission success for the corresponding mailbox when set (=1).

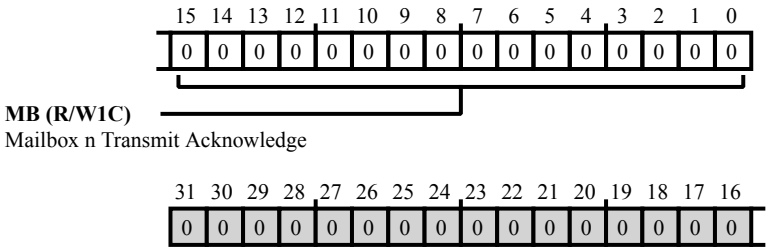


Figure 27-56: CAN_TA2 Register Diagram

Table 27-56: CAN_TA2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Transmit Acknowledge.

Timing Register

The CAN_TIMING register select the time segments, sampling, and synchronization for CAN bit timing. For more information about bit timing and clock operation, see the CAN Operating Modes section.

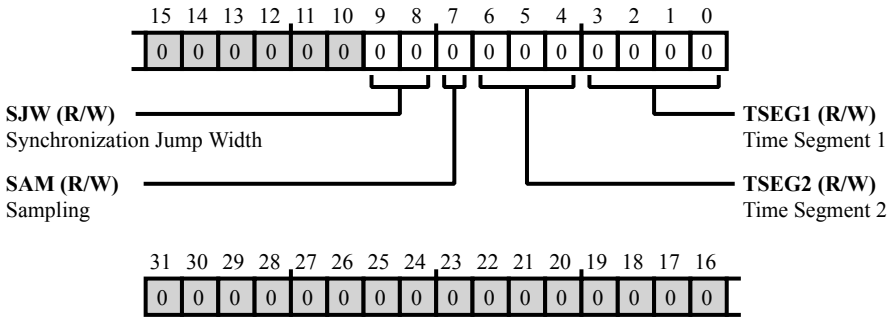


Figure 27-57: CAN_TIMING Register Diagram

Table 27-57: CAN_TIMING Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:8 (R/W)	SJW	Synchronization Jump Width. The CAN_TIMING.SJW bits select the maximum number of time quanta, ranging from 1 to 4(SJW + 1). This selection allows for a re-synchronization attempt when the CAN detects a recessive-to-dominant edge outside the synchronization segment. The re-synchronization automatically moves the sampling point such that the CAN bit is still handled properly. Note that the CAN_TIMING.SJW value should not exceed CAN_TIMING.TSEG2 or CAN_TIMING.TSEG1.
7 (R/W)	SAM	Sampling. The CAN_TIMING.SAM bit selects whether the CAN performs normal sampling (once at the sampling point described by the CAN_TIMING register) or performs over sampling. If CAN_TIMING.SAM is set, the CAN over samples the input signal at three times at the SCLK rate. The resulting value is generated by a majority decision of the three sample values. Note that the CAN_TIMING.SAM bit should always be cleared if the CAN_CLK.BRP value is less than 4.
6:4 (R/W)	TSEG2	Time Segment 2. The CAN_TIMING.TSEG2 bits and CAN_TIMING.TSEG1 bits control how many time quanta of which the CAN bits consist, resulting in the CAN bit rate. For more information about bit timing and clock operation, see the CAN Operating Modes section. Note that the CAN_TIMING.TSEG1 value should always be greater than or equal to the CAN_TIMING.TSEG2 value.

Table 27-57: CAN_TIMING Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TSEG1	Time Segment 1. The `CAN_TIMING.TSEG1` bits and `CAN_TIMING.TSEG2` bits control how many time quanta of which the CAN bits consist, resulting in the CAN bit rate. For more information about bit timing and clock operation, see the CAN Operating Modes section. Note that the `CAN_TIMING.TSEG1` value should always be greater than or equal to the `CAN_TIMING.TSEG2` value.

Transmission Request Reset 1 Register

The CAN_TRR1 register requests transmit abort for mailboxes 8 through 15. Bits in this register request transmit abort for the corresponding mailbox when set (=1). When a transmission completes, the corresponding bits in the transmit request set register (CAN_TRS1) and in the CAN_TRR1 are cleared. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

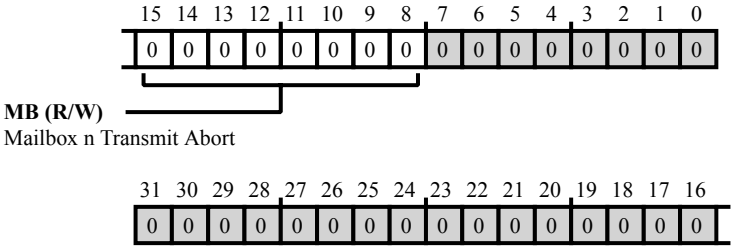


Figure 27-58: CAN_TRR1 Register Diagram

Table 27-58: CAN_TRR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit Abort.

Transmission Request Reset 2 Register

The CAN_TRR2 register requests transmit abort for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests transmit abort for the corresponding mailbox when set (=1). When a transmission completes, the corresponding bits in the transmit request set register (CAN_TRS2) and in the CAN_TRR2 are cleared.

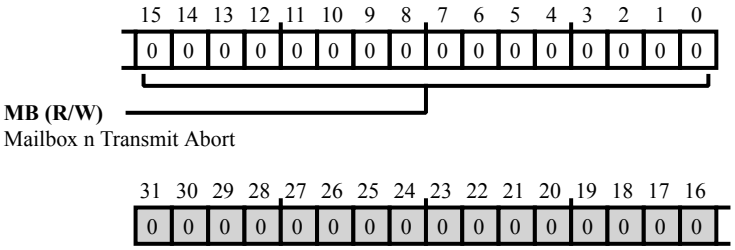


Figure 27-59: CAN_TRR2 Register Diagram

Table 27-59: CAN_TRR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit Abort.

Transmission Request Set 1 Register

The CAN_TRS1 register requests transmit for mailboxes 8 through 15. Bits in this register request transmit for the corresponding mailbox when set (=1). After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (with the corresponding bit in CAN_MC1 = 1), and (subsequently) the corresponding transmit request bit is set (in CAN_TRS1). When a transmission completes, the corresponding bits in CAN_TRS1 and in the transmit request reset register (CAN_TRR1) are cleared. Bits 0 through 7 are read-only, as the corresponding mailboxes are receive-only mailboxes.

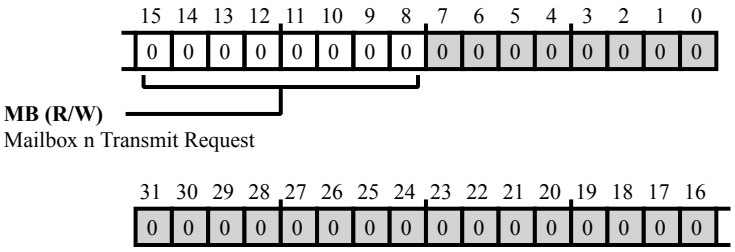


Figure 27-60: CAN_TRS1 Register Diagram

Table 27-60: CAN_TRS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	MB	Mailbox n Transmit Request.

Transmission Request Set 2 Register

The CAN_TRS2 register requests transmit for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests transmit for the corresponding mailbox when set (=1). After writing the data and the identifier into the mailbox area, the message is sent after mailbox n is enabled (with the corresponding bit in CAN_MC2 = 1), and (subsequently) the corresponding transmit request bit is set (in CAN_TRS2). When a transmission completes, the corresponding bits in CAN_TRS2 and in the transmit request reset register (CAN_TRR2) are cleared.

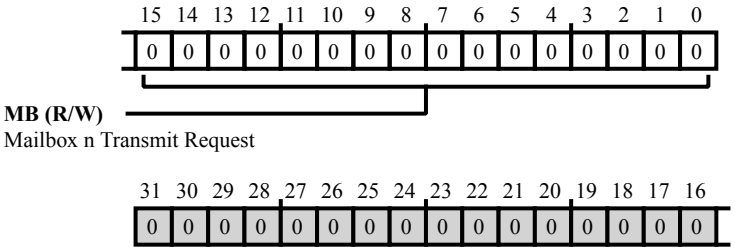


Figure 27-61: CAN_TRS2 Register Diagram

Table 27-61: CAN_TRS2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit Request.

Universal Counter Configuration Mode Register

The CAN_UCCNF register controls the operation of the universal counter, including counter enable and counter mode selection.

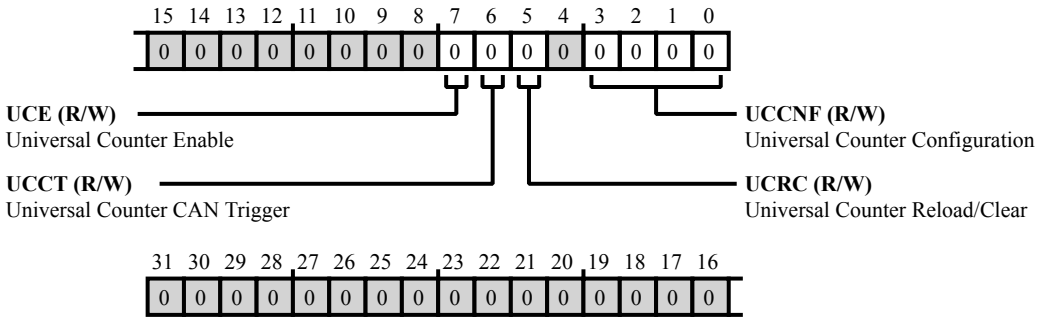


Figure 27-62: CAN_UCCNF Register Diagram

Table 27-62: CAN_UCCNF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	UCE	Universal Counter Enable. The CAN_UCCNF.UCE bit enables universal counter operation in the mode selected by the CAN_UCCNF.UCCNF bits.	
		0	Disable Counter
		1	Enable Counter
6 (R/W)	UCCT	Universal Counter CAN Trigger. The CAN_UCCNF.UCCT bit enables the universal counter trigger, directing the CAN to re-load the counter on mailbox 4 reception in watchdog mode and clear the counter on mailbox 4 reception in time stamp mode. This bit has no effect in all other modes.	
		0	Disable Trigger
		1	Enable Trigger
5 (R/W)	UCRC	Universal Counter Reload/Clear. The CAN_UCCNF.UCRC bit re-loads or clears the universal counter, depending on the counter mode. In watchdog mode, setting this bit directs the CAN to re-load the counter. In all other modes, setting this bit directs the CAN to clear the counter.	
		0	No Action
		1	Re-load or Clear the Counter
3:0 (R/W)	UCCNF	Universal Counter Configuration. The CAN_UCCNF.UCCNF bits select the universal counter operating mode. For more information about these modes, see the Operating Modes section.	
		0	Reserved

Table 27-62: CAN_UCCNF Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1	Time Stamp Mode
		2	Watchdog Mode
		3	Auto-transmit Mode
		4	Reserved
		5	Reserved
		6	Count Error Frames
		7	Count Overload Frames
		8	Count Arbitration Lost
		9	Count Aborted Transmissions
		10	Count Successful Transmissions
		11	Count Rejected Receive Messages
		12	Count Receive Message Lost
		13	Count Successful Receptions
		14	Count Stored Receptions
		15	Count Valid Messages

Universal Counter Register

The CAN_UCCNT register holds the current universal count. This register is reloaded from the CAN_UCRC register when counter the decrements to zero in auto-transmit mode.

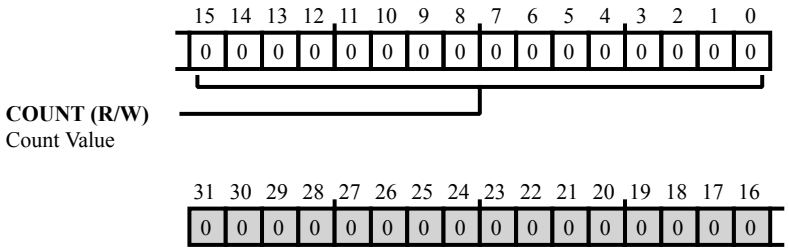


Figure 27-63: CAN_UCCNT Register Diagram

Table 27-63: CAN_UCCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count Value. The CAN_UCCNT.COUNT bits hold the current universal count value.

Universal Counter Reload/Capture Register

The CAN_UCRC register holds the period value (universal count), which is used in auto-transmit mode as the period for sending the message in mailbox 11 (broadcast heartbeat) to all CAN nodes. Accordingly, messages sent this way usually have high priority.

The period value is written to the CAN_UCRC register. When auto-transmit mode is enabled (CAN_UCCNF.UCCNF = 0x3), the CAN loads the counter with the value in CAN_UCRC. The counter decrements to 0 at the CAN bit clock rate, then is reloaded. Each time the counter decrements to 0, the CAN sets the CAN_TRS1.MB bit for mailbox 11 and sends the corresponding message from mailbox 11.

Note that for auto-transmit mode, mailbox 11 must be configured as a transmit mailbox and must contain valid data (identifier, control bits, and data). This setup must occur before the counter first expires after this mode is enabled.

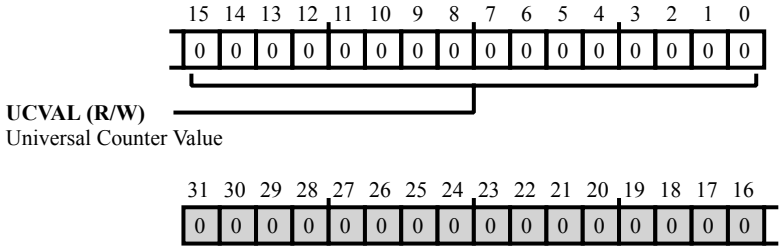


Figure 27-64: CAN_UCRC Register Diagram

Table 27-64: CAN_UCRC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	UCVAL	Universal Counter Value. The CAN_UCRC.UCVAL bits hold the value for the universal count period, which is used in auto-transmit mode.

28 Serial Peripheral Interface (SPI)

The serial peripheral interface is an industry-standard synchronous serial link that supports communication with multiple SPI-compatible devices. The baseline SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. The two data pins allow full-duplex operation to other SPI-compatible devices. Two extra (optional) data pins are provided on specific SPIs to support quad SPI operation. Enhanced modes of operation such as flow control, fast mode, and dual-I/O mode (DIOM) are also supported. In addition, a direct memory access (DMA) mode allows for transferring several words with minimal CPU interaction.

With a range of configurable options, the SPI ports provide a glueless hardware interface with other SPI-compatible devices in master mode, slave mode, and multimaster environments. The SPI peripheral includes programmable baud rates, clock phase, and clock polarity. The peripheral can operate in a multimaster environment by interfacing with several other devices, acting as either a master device or a slave device. In a multimaster environment, the SPI peripheral uses open-drain outputs to avoid data bus contention. The flow control features enable slow slave devices to interface with fast master devices by providing an SPI ready pin which flexibly controls the transfers.

SPI Features

The SPI module supports the following features:

- Full-duplex, synchronous serial interface
- 8, 16-bit and 32-bit word sizes
- Programmable baud rate, clock phase, and polarity
- Programmable inter-frame latency
- Flow control
- Support for Fast and DIOM modes
- Quad and memory-mapped modes are supported by SPI2 only
- Independent receive and transmit DMA channels
- Burst transfer mode for non-DMA write accesses

SPI Functional Description

This section provides information on the function of the SPI module.

Shift register functionality

The SPI is essentially a shift register that serially transmits and receives data bits to or from other SPI devices. During an SPI transfer, data is simultaneously transmitted (shifted out serially) and received (shifted in serially). A serial clock line synchronizes shifting and sampling of the information on the two serial data lines.

Master slave functionality

During a data transfer, one SPI system acts as the link master which controls the data flow. The other system acts as the slave, which has data shifted into and out of it by the master. Different devices can take turn being masters, and one master can simultaneously shift data into multiple slaves (broadcast mode). However, only one slave can drive its output to write data back to the master at any given time. This rule must be enforced in the broadcast mode. Several slaves can be selected to receive data from the master in this mode. But only one slave can be enabled to send data back to the master.

Enhanced operating modes

SPI supports enhanced modes of operation like fast mode, DIOM, and Quad-SPI, and optional flow control. In fast mode, received data is sampled on the transmit edge instead of the standard receive edge, thus enabling a full-cycle path for the received data. In DIOM, both MOSI and MISO are configured as input or output pins, and 2 bits are shifted in or out on each receive or transmit edge. In Quad-SPI mode, SPI_D3:0 are configured as input or output pins and 4 bits are shifted in or out on each receive or transmit edge. A slower slave can use flow control to stall a faster master device.

Single and multi-master use

The SPI can be used in a single master as well as multi-master environment. The `SPI_MOSI`, `SPI_MISO`, and the `SPI_CLK` signals are all tied together in both configurations. SPI transmission and reception can be enabled simultaneously or individually, depending on `SPI_RXCTL` and `SPI_TXCTL` settings. In broadcast mode, several slaves can be enabled to receive, but only one slave must be in transmit mode and driving the `SPI_MISO` line.

CM41X_M4 SPI Register List

The Serial Peripheral Interface (SPI) provides a full-duplex, synchronous serial interface, which supports both master/slave modes and multi-master environments. The SPI's baud rate and clock phase/polarities are programmable, and it has integrated DMA channels for both transmit and receive data streams. A set of registers governs SPI operations. For more information on SPI functionality, see the SPI register descriptions.

Table 28-1: CM41X_M4 SPI Register List

Name	Description
SPI_CLK	Clock Rate Register
SPI_CTL	Control Register
SPI_DLY	Delay Register
SPI_ILAT	Masked Interrupt Condition Register
SPI_ILAT_CLR	Masked Interrupt Clear Register
SPI_IMSK	Interrupt Mask Register
SPI_IMSK_CLR	Interrupt Mask Clear Register
SPI_IMSK_SET	Interrupt Mask Set Register
SPI_RFIFO	Receive FIFO Data Register
SPI_RWC	Received Word Count Register
SPI_RWCR	Received Word Count Reload Register
SPI_RXCTL	Receive Control Register
SPI_SLVSEL	Slave Select Register
SPI_STAT	Status Register
SPI_TFIFO	Transmit FIFO Data Register
SPI_TWC	Transmitted Word Count Register
SPI_TWCR	Transmitted Word Count Reload Register
SPI_TXCTL	Transmit Control Register

CM41X_M0 SPI Interrupt List

Table 28-2: CM41X_M0 SPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
25	SPI0_ERR	SPI0 Error	Level	
25	SPI0_STAT	SPI0 Status	Level	

CM41X_M4 SPI Interrupt List

Table 28-3: CM41X_M4 SPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
58	SPI1_ERR	SPI1 Error	Level	

Table 28-3: CM41X_M4 SPI Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
59	SPI0_ERR	SPI0 Error	Level	
135	SPI1_STAT	SPI1 Status	Level	
136	SPI0_STAT	SPI0 Status	Level	

28.2.4 SPI ITrigger List

CM41X_M4 SPI DMA Channel List

Table 28-4: CM41X_M4 SPI DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPI0_TXDMA	SPI0 TX DMA Channel
DMA1	SPI0_RXDMA	SPI0 RX DMA Channel
DMA6	SPI1_TXDMA	SPI1 TX DMA Channel
DMA7	SPI1_RXDMA	SPI1 RX DMA Channel

SPI Block Diagram

The ***SPI Controller Block Diagram*** illustrates the blocks of the SPI module. The module is comprised of three primary parts:

- SPI core contains the receive and transmit FIFOs and their associated shift registers
- Control blocks contain the synchronizer and logic to control the data flow through the data pipelines
- Register block

Transfer Protocol

The SPI module implements two channels that are independent of each other. The SPI module uses the `SPI_RXCTL` and `SPI_TXCTL` dedicated control registers to control these channels. Except in dual and quad modes, SPI can enable and use both channels simultaneously.

The SPI protocol supports four different combinations of serial clock phase and polarity. These combinations are selected through the `SPI_CTL.CPOL` and `SPI_CTL.CPHA` bits.

The ***SPI Transfer Protocol*** figures demonstrate the two basic transfer formats as defined by the CPHA bit. Two waveforms are shown for `SPI_CLK`; one for `SPI_CTL.CPOL`=0 and the other for `SPI_CTL.CPOL`=1. The diagrams can be interpreted as master or slave timing diagrams since the `SPI_CLK`, `SPI_MISO`, and `SPI_MOSI` pins are directly connected between the master and the slave. The `SPI_MISO` signal is the output from the slave (slave transmission), and the `SPI_MOSI` signal is the output from the master (master transmission). The master

generates the SPI_CLK signal. The SPI_SS signal is the slave device select input to the slave from the master. The diagrams represent an 8-bit transfer (SPI_CTL.SIZE=0) with the MSB first (SPI_CTL.LSBF=0). Any combination of the SPI_CTL.SIZE and SPI_CTL.LSBF bits is permissible. For example, a 16-bit transfer with the LSB first is another possible configuration.

The clock polarity and the clock phase could be identical for the master device and the slave device involved in the communication link. The transfer format from the master can be changed between transfers to adjust for various requirements of a slave device.

The SPI module uses the SPI_CTL.ASSEL bit to determine when the SPI hardware or software control the SPI_SEL[n] line. When SPI_CTL.ASSEL=1, the slave select line must be set to the polarity set in the SPI_CTL.SELST field between each serial transfer. The actual behavior of SPI_SEL[n] also depends on the parameters programmed into the SPI_DLY register. The SPI hardware logic automatically controls this functionality. When SPI_CTL.ASSEL=0, SPI_SEL[n] can either remain active between successive transfers or be inactive. The software must control this activity through manipulation of the SPI_SLVSEL register.

The ***SPI Transfer Protocol*** pair of figures illustrates the case when SPI_CTL.ASSEL = 1 and the SPI_SEL[n] line is inactive between frames. If ASSEL = 0, the SPI_SEL[n] line can remain active between frames; however, the first bit is only driven when an active transition of SPI_CLK occurs.

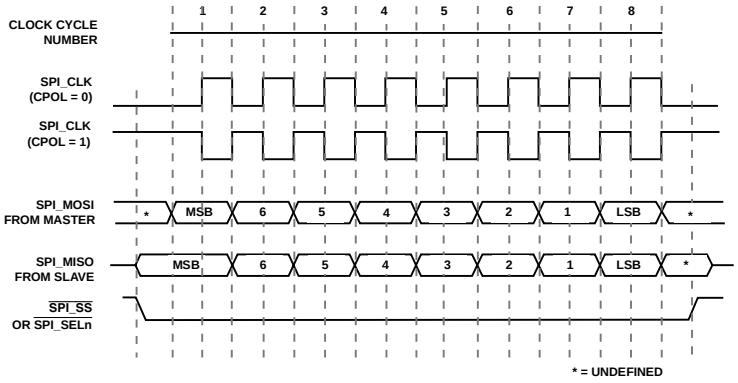


Figure 28-1: SPI Transfer Protocol for CPHA=0

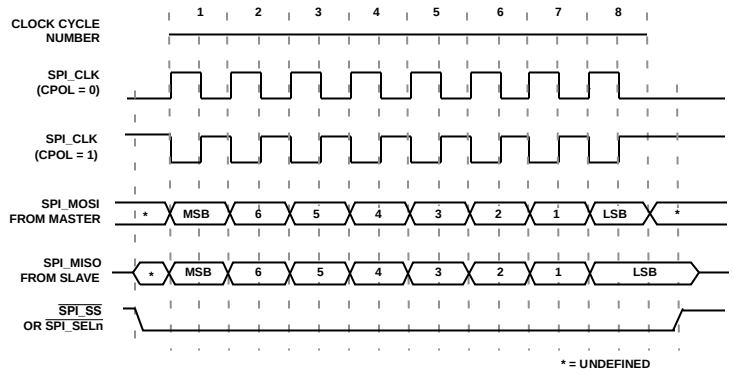


Figure 28-2: SPI Transfer Protocol for CPHA=1

Clock Considerations

The SPI_CLK signal is a gated clock that is only active during data transfers, for the time of the transferred word. In normal mode, the number of active edges is equal to the number of bits to be transmitted or received. In dual-I/O mode, it is half of the number of bits to be transmitted or received, and in quad-SPI mode it is one-fourth of the number. The clock rate can be as high as the SCLK/2, and both even and odd dividers from SCLK are supported. The clock rate can be derived using both even and odd dividers from

For master devices, the SPI uses the SPI_CLK register value to determine the clock rate, whereas this value is ignored for slave devices.

When the SPI controller is a master, SPI_CLK is an output signal. Conversely, when the SPI controller is a slave, SPI_CLK is an input signal. Slave devices ignore the SPI clock when the slave select input is driven inactive. The SPI uses the SPI_CLK signal to shift out and shift in the data driven onto the SPI_MISO and SPI_MOSI lines. The data is always shifted out on one edge of the clock (the active edge) and sampled on the opposite edge of the clock (the sampling edge). Clock polarity and clock phase relative to data are programmable through the SPI_CTL register and define the transfer format.

Controlling Delay Between Frames

The ***SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)*** figure illustrates SPI timing using the SPI_DLY.LEADX and SPI_DLY.LAGX programming. The SPI uses the SPI_DLY.LAGX bits to control the timing between the slave select (SPI_SS) signal assertion and the first SPI_CLK edge. The SPI uses the SPI_DLY.LEADX bits to control the timing between the last SPI_CLK edge and deassertion of the SPI_SS signal. The lead and lag timing can be extended by a 1 SPI_CLK duration to ease timing restrictions on the slave device.

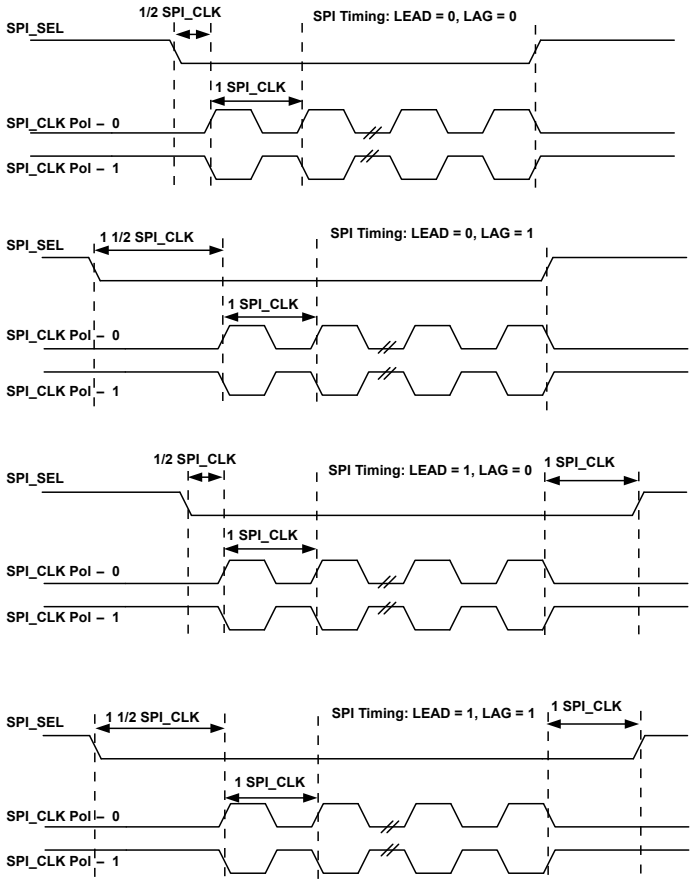


Figure 28-3: SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)

The ***SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)*** figure illustrates SPI timing with STOP programming. The SPI module uses this timing to insert multiples of `SPI_CLK` period delays between transfers. The `SPI_SS` line is deasserted for the duration specified in the `SPI_DLY.STOP` bit field, assuming the `SPI_CTL.SELST` bit is configured for deassertion between transfers.

If the `SPI_DLY.STOP` bit =0, the master operates in a *continuous mode*. This mode causes an immediate start of the second word after the last bit is transferred from the first word. During this mode of operation, the slave select line is continuously asserted.

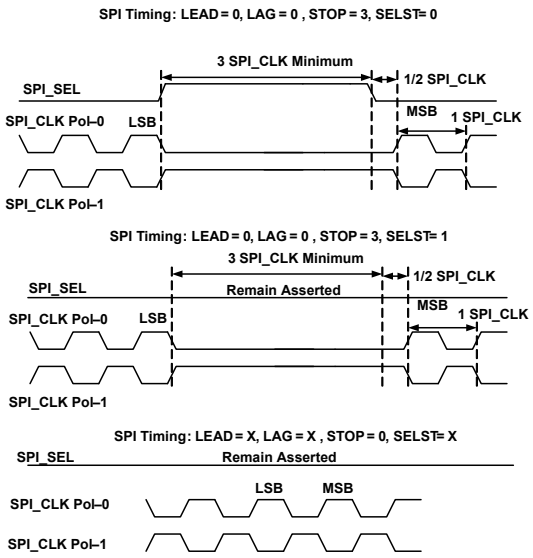


Figure 28-4: SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)

When the SPI_DLY.STOP bit =0 and initial conditions for a transfer are not met, the interface pauses before the next transfer. During this pause, the SPI uses the SPI_CTL.SELST bit to determine the state of the slave select pin. The SPI uses the SPI_DLY.LEADX and SPI_DLY.LAGX bits to determine the timing between SPI_CLK edges and the slave select line.

Flow Control

In master mode, the slave device must drive the SPI_RDY pin. The pin acts as an input signal. The slave can deassert the SPI_RDY pin to stop the master from initiating any new transfer. If SPI_RDY is deasserted in the middle of a transfer, the current transfer continues, and the next transfer will not start unless the slave asserts the SPI_RDY signal. Whenever the slave deasserts SPI_RDY and stalls the master, the SPI controller goes into a waiting state, and the SPI_STAT.FCS bit is set. When the slave asserts SPI_RDY, the SPI controller resumes operation, and the SPI_STAT.FCS bit is cleared.

In slave mode, the SPI_RDY pin acts as an output signal. Flow control can be configured on either the TX channel or the RX channel. The SPI uses the SPI_CTL.FCCH bit to control this configuration. If flow control is configured on the TX channel, as the SPI_TFIFO status nears the empty condition, the SPI_RDY pin is deasserted. If flow control is configured on the RX channel, as the SPI_RFIFO status nears the full condition, the SPI_RDY pin is deasserted. The SPI uses the SPI_CTL.FCWM bits to control the FIFO status at which SPI_RDY deassertion takes place. Flow control in slave mode is purely based on the FIFO status and does not depend on the word counters.

The ***SPI Flow Control Timing in Master Mode*** figure illustrates this timing.

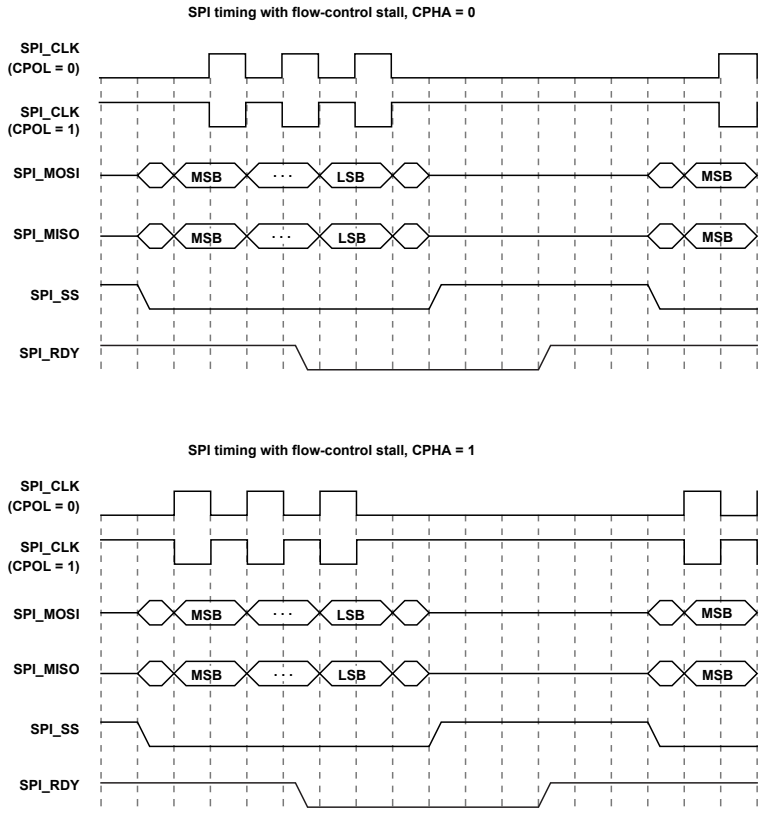


Figure 28-5: SPI Flow Control Timing in Master Mode.

Slave Select Operation

If the SPI is in slave mode, $\overline{\texttt{SPI\_SS}}$ acts as the slave select input. When SPI is enabled as a master, $\overline{\texttt{SPI\_SS}}$ can serve as an error detection input for the SPI in a multi-master environment. The SPI_CTL.PSSE bit enables this feature. When SPI_CTL.PSSE=1, the $\overline{\texttt{SPI\_SS}}$ input is the master mode error input. Otherwise, $\overline{\texttt{SPI\_SS}}$ is ignored.

The $\overline{\texttt{SPI\_SS}}$ signal is an active-low signal. The master asserts the signal during the transfer. The signal can be deasserted or remain asserted between transfers. When $\overline{\texttt{SPI\_SS}}$ is deasserted, SPI_CLK and inputs are ignored, and outputs are three-stated.

The slave select bits (SPI_SLVSEL.SSEL1 – SPI_SLVSEL.SSEL7) are used in a multiple-slave SPI environment. For example, if there are eight SPI devices in the system including a processor master, the master processor can support the SPI mode transactions across the other seven devices. This configuration requires only one master processor in this multi-slave environment.

For example, assume that the SPI of the processor is the master. The SPI_SLVSEL.SSEL1 – SPI_SLVSEL.SSEL7 bits on the processor can be connected to the slave select pin of each slave device. In this configuration, the slave select bits can be used in three ways. In cases 1 and 2, the processor is the master and the seven microcontrollers or peripherals with SPI interfaces are slaves. The processor can do one of the following:

1. Transmit to all seven SPI devices at the same time in a broadcast mode. Here, all slave select bits are set.

2. Receive and transmit from one SPI device by enabling only one slave SPI device at a time.
3. If all the slaves are also processors, then the requester can receive data from only one processor at a time. (The functionality is enabled by clearing the SPI_CTL.EMISO bit in the six other slave processors.) The requestor can transmit broadcast data to all seven at the same time. This MISO enabling feature is available in some other microcontrollers. Therefore, it is possible to use the MISO enabling feature with any other SPI device that includes this functionality.

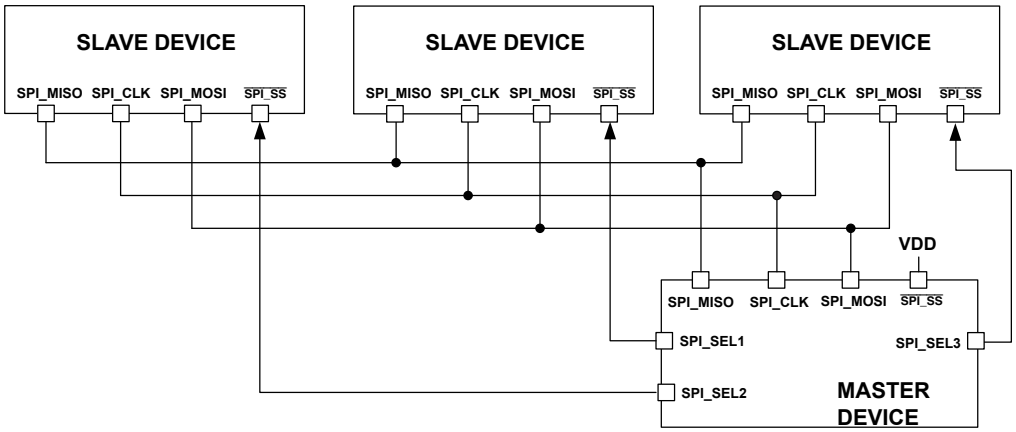


Figure 28-6: Single-Master, Multiple-Slave Configuration

Beginning and Ending a Non-DMA SPI Transfer

The start and finish of a non-DMA SPI transfer depend on the following settings.

1. Whether the device is configured as a master or a slave.
2. The state of the SPI_CTL.ASSEL bit, which selects between hardware and software control over SPI_SLVSEL.

When SPI_CTL.CPHA=0, the enabled slave select outputs are driven active. However, the SPI_CLK signal remains inactive for the first half of the first cycle of SPI_CLK. For a slave with SPI_CTL.CPHA=0, the transfer starts as soon as the SPI_SS input goes low.

When SPI_CTL.CPHA=1, a transfer starts with the first active edge of SPI_CLK for both slave and master devices. For a master device, a transfer is complete after it sends the last data and simultaneously receives the last data bit. A transfer for a slave device ends after the last sampling edge of SPI_CLK. If SPI_CTL.ASSEL=0, the hardware maintains responsibility for toggling SPI_SS between frames. If SPI_CTL.ASSEL=1, software controls the SPI_SS line and can keep it active between frames.

The SPI_STAT.RFE bit defines when the receive buffer can be read, indicating that SPI_RFIFO is not empty. The SPI_STAT.TFF bit defines when the transmit buffer can be written, indicating that the SPI_TFIFO is not full. The end of a single word transfer occurs when the SPI_STAT.RFE bit is cleared. The status indicates that a new word has been received and written into the receive FIFO. The SPI_STAT.RFE bit remains cleared as long as the receive FIFO has valid data.

To maintain software compatibility with other SPI devices, the SPI_STAT.SPIF bit is also available for polling. This bit can have a slightly different behavior from other commercially available devices.

In master mode with the SPI_CTL.ASSEL bit cleared, software manually asserts the required slave select signal before starting the transaction. After all data transfers, software typically releases the slave select line.

When the receive or transmit word counters are enabled in the SPI_TXCTL or SPI_RXCTL registers, the SPI generates a finish interrupt at the end of the transfer. It signals the end of all transfers related to that transaction.

Transmit Operation in Non-DMA Mode

The transmit operation in non-DMA mode is enabled through the SPI_TXCTL.TEN bit. It can be enabled independently from the receive operation, and the transmit channel can become the initiating channel based on the SPI_TXCTL.TTI bit setting.

Transmit underrun is not possible in this mode, as no new transfer initiates unless the transmit FIFO is empty (in the case that SPI_TXCTL.TTI =1). A receive overflow is detected when data from a new frame transfer replaces older data in a full receive FIFO. This event can occur if SPI_TXCTL.TTI =1 and the receive channel is enabled in a non-initiating capacity.

An SPI transmit interrupt request is signaled once the transmit channel has been enabled and the transmit FIFO is not full. The SPI uses the SPI_TXCTL.TDR bit setting to control the frequency of the interrupt request.

Receive Operation in Non-DMA Mode

The receive operation in non-DMA mode is enabled through the SPI_RXCTL.REN bit. It can be enabled independently from the transmit operation, and the receive channel can become the initiating channel based on the SPI_RXCTL.RTI bit setting.

Receive overflow is not possible in this mode, as no new transfer initiates when the receive FIFO is full (in the case of SPI_RXCTL.RTI =1). A transmit underrun can occur (SPI_TXCTL.TDU bit) when no valid data is in the SPI_TFIFO register when a transfer is initiated. This event can occur if SPI_RXCTL.RTI =1 and the transmit channel is enabled in a non-initiating capacity.

An SPI receive interrupt request is signaled once the receive channel has been enabled and there is data waiting to be read. The SPI uses the SPI_RXCTL.RDR bit setting to control the frequency of the interrupt request.

DMA and Interrupt Multiplexing

Please refer to the *Direct Memory Access (DMA)* chapter for information about DMA multiplexing. Several interrupts and DMA channels in the SPI can be multiplexed.

The SPI1 DMA units are multiplexed with other periperals and may require configuration to support SPI DMA.

Dual I/O Mode

In Dual I/O mode, the SPI_MISO and SPI_MOSI pins are configured to operate in the same direction which doubles bandwidth. The SPI uses the SPI_CTL.SOSI bit to determine the order of bits on the pins. When set,

the processor sends the first bit on the SPI_MOSI pin and the second bit on the SPI_MISO pin. If the SPI_CTL.SOSI bit is cleared, the order is reversed. Since dual I/O mode uses both pins to transmit or receive data, only one channel can be enabled, either transmit or receive. Flow control through the SPI_RDY pin is supported. Interrupt request generation is unaffected by dual I/O mode. However, the interrupt service interval is reduced, since the individual transfer latency is halved.

Changing to quad SPI mode must be done when the SPI is in a quiescent state.

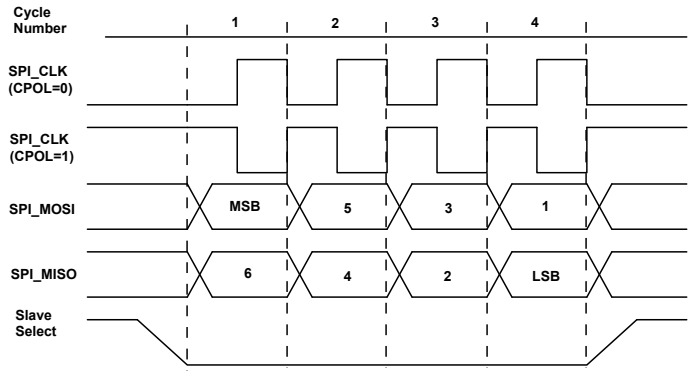


Figure 28-7: Dual I/O Mode Transfer Protocol for CPHA=0, SOSI=1, 8-Bit Transfer, LSBF=0.

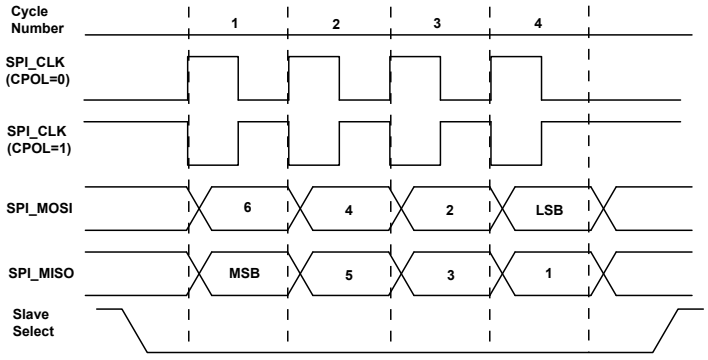


Figure 28-8: Dual I/O Mode Transfer Protocol for CPHA=1, SOSI=0, 8-Bit Transfer, LSBF=0.

Quad I/O Mode (SPI2 only)

In quad SPI mode, the SPI_MISO and SPI_MOSI pins, in tandem with the SPI_D2 and SPI_D3 pins, are configured to operate in the same direction. The SPI uses the SPI_CTL.SOSI bit to determine the order of bits on the pins. When set, the processor sends:

- The first bit on the SPI_MOSI pin
- The second bit on the SPI_MISO pin
- The third bit on the SPI_D2 pin
- The fourth bit on the SPI_D3 pin

If the SPI_CTL.SOSI bit is cleared, the order is reversed. Since quad SPI mode uses all four pins to transmit or receive data, only one channel can be enabled as either transmit or receive. Flow control through the SPI_RDY pin is supported.However, the interrupt service interval is reduced, since the individual transfer latency is halved.

Changing to quad SPI mode must be done when the SPI is in a quiescent state.

While using dual or quad I/O mode for communicating with flash devices, program the SPI_CTL.CPHA and the SPI_CTL.CPOL bits =1. This programming avoids bus contention during read operations, because the flash device starts driving out the bits immediately after dummy cycles in read header.

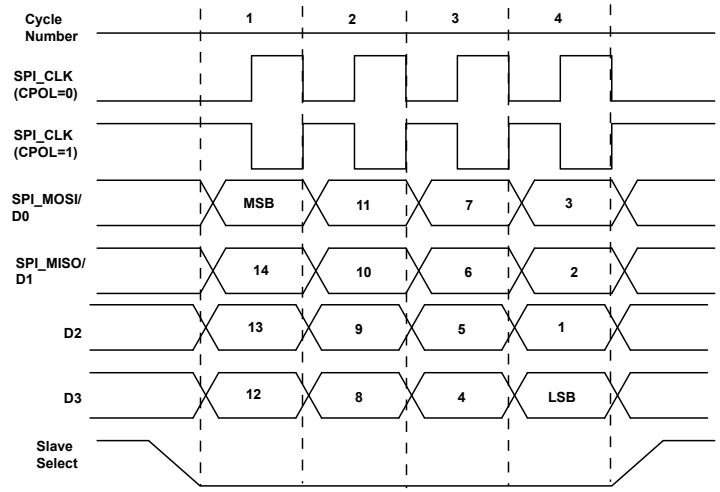


Figure 28-9: Quad Mode Timing for CPHA=0, SOSI=1, 16-Bit Transfer, LSBF=0.

NOTE: The SPI does support quad SPI 8-bit transfer in slave continuous mode of operation with an SCLK:SPI_CLK ratio of less than 1:2. A minimum of 2 SCLK cycles is required between transfers in 8-bit quad SPI slave mode with an SCLK:SPI_CLK ratio of less than 1:2.

Fast Mode

Fast mode is similar to the normal mode of operation when transmitting. When receiving, data is sampled at the next transmit edge allowing a full cycle of timing in the receive direction. This mode is valid in master mode operation only. When the SPI operates in fast mode, the slave drives the data for one full cycle.

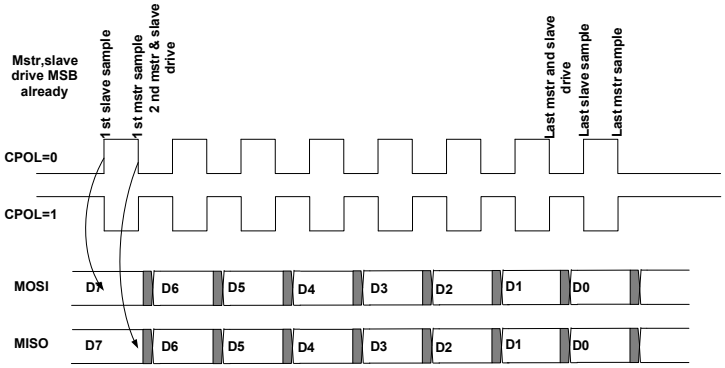


Figure 28-10: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 0

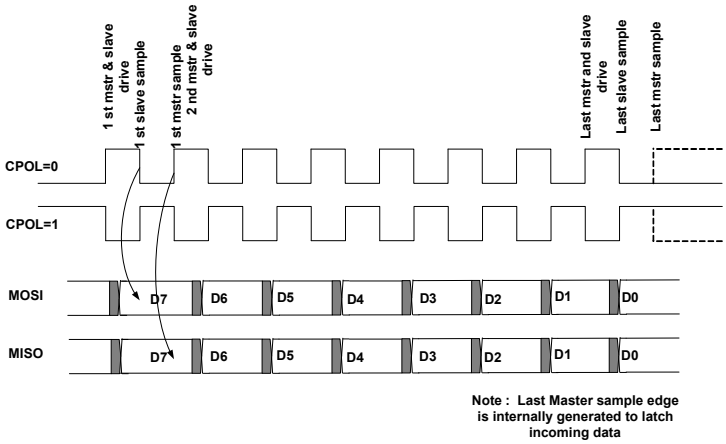


Figure 28-11: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 1

SPI Interrupt Signals

The SPI controller supports three types of interrupt request signals that correspond to data, status, and error conditions.

Data Interrupts

The SPI peripheral supports two data interrupt channels – receive and transmit. These interrupt signals are multiplexed into the DMA request lines. Since the peripheral interfaces with separate read and write interfaces with DMA, the read and write data interrupts are independent. When the DMA channels are not used, the interrupts are routed directly to the system event controller. The interrupts occupy the same vector locations as the corresponding DMA channels.

When the DMA channels are not used, but still properly configured using the system multiplexing scheme, the interrupt requests are routed through DMA unit to the system for event processing.

Each of the data interrupt requests can be individually controlled. Program the `SPI_RXCTL.RDR` and `SPI_TXCTL.TDR` bit fields for receive and transmit, respectively. When receive is enabled, the RX interrupt request is issued whenever there is data available in the receive datapath for reading. (The event occurs according to the `SPI_RXCTL.RDR` bit setting.) When transmit is enabled, the TX interrupt request is issued whenever the transmit datapath can be written. (The event occurs according to the `SPI_TXCTL.TDR` setting.) DMA data interrupts are compatible with second-generation DMA to incorporate urgent data requests and transfer finish interrupt requests apart from the usual data request interrupts. Transmit interrupt requests operate independently from the word counter-value in the `SPI_TWC` register.

Status Interrupts

The SPI controller supports several status interrupt requests to indicate different conditions of the receiver and transmitter. All status interrupt requests can be masked. Status interrupt requests are signaled directly through a single SPI status IRQ line. The line cannot be combined with the SPI error IRQ line for some processors. The ***SPI Status Interrupts*** table describes the status interrupt requests that are available for the SPI controller.

Table 28-5: SPI Status Interrupts

SPI_STAT Bit	Description
SPI_STAT.RUWM	Receive FIFO urgent watermark interrupt request. Issued when the level of the RFIFO breaches the watermark set in the SPI_RXCTL.RUWM field. It is cleared when the level of the RFIFO reaches the watermark set in the SPI_RXCTL.RRWM field. If the RX channel is configured in DMA mode, SPI_RXCTL.RUWM is multiplexed with the data request.
SPI_STAT.TUWM	Transmit FIFO urgent watermark interrupt request. Issued when the level of the TFIFO breaches the watermark set using the SPI_TXCTL.TUWM bit. It is cleared when the level of the TFIFO reaches the watermark set in the SPI_TXCTL.TRWM field. If the TX channel is configured in DMA mode, SPI_STAT.TUWM is multiplexed with the data request.
SPI_STAT.TS	Transmit start interrupt request. Issued when the start of a transmit burst is detected by loading of the SPI_TWC register with the contents of the SPI_TWCR register.
SPI_STAT.RS	Receive start interrupt request. Issued when the start of a receive burst is detected by the loading of SPI_RWC with the contents of SPI_RWCR.
SPI_STAT.TF	Transmit finish interrupt request. Issued when a transmit burst completes (SPI_TWC decrements to zero).
SPI_STAT.RF	Receive finish interrupt request. Issued when a receive burst completes (SPI_RWC decrements to zero).

Error Conditions

The SPI controller supports interrupt requests upon several different error conditions. All interrupt requests are maskable. The individual error indications combine into a single SPI error IRQ signal, which can be multiplexed on some processors with the aggregated SPI status IRQ signal. The ***SPI Error Interrupts*** table details the possible error indications.

Error conditions arise depending on which of the channels (transmit or receive) are enabled. If a channel is disabled, all errors related to it are ignored. When both channels are enabled, errors from both channels are enabled.

Table 28-6: SPI Error Interrupts

Bit	Description
SPI_STAT.MF	Mode fault. Signaled when another device also tries to be a master in a multi-master system and drives the SPI_SS input low. This error is signaled in master mode operation.
SPI_STAT.TUR	Transmission error. Signaled when an underflow condition occurs on the transmit channel. This event occurs when a new transfer starts but SPI_TFIFO is empty. This error does not occur in master transmit initiating mode since SPI_TFIFO *Not Empty* is one of the conditions for transfer initiation.
SPI_STAT.ROR	Reception error. Signaled when an overflow condition occurs on the receive channel. This event occurs when a new data word is received, but the SPI_RFIFO is full. This error condition does not occur in master receive initiating mode since SPI_RFIFO *Not Full* is one of the conditions for transfer initiation.
SPI_STAT.TC	Transmit collision error. Signaled when loading data to the transmit shift register happens near the first transmitting edge of SPI_CLK. In slave mode of operation, the SPI controller is unaware of when the next transfer starts. Loading of data to the transmit shift register can happen just after the transmitting edge. This event results in the setup time not being met for the first bit transmitted. The transmitted data is corrupt. In SPI_CTL.CPHA 1 mode, the first SPI_CLK edge is taken as the first transmitting edge. If

Table 28-6: SPI Error Interrupts (Continued)

Bit	Description
	SPI_CTL.CPHA =0, then the last SPI_CLK edge of the last transmission (SPI_CTL.SELST =1) or slave select deassertion (SPI_CTL.SELST =0) is taken as the first transmitting edge. This error is signaled only in the slave mode of operation. In master mode of operation, loading of data happens before the first transmitting edge of SPI_CLK.

SPI Programming Concepts

The following sections provide general programming guidelines and procedures.

Programming Guidelines

It is acceptable to program SPI_RXCTL and SPI_TXCTL registers after programming the SPI_CTL register. However, program the initiating mode register and its counter-register, if enabled, after the non-initiating mode register. For example, if transmit is the initiating mode and receive is the non-initiating mode, then program the SPI_RXCTL and SPI_RWC registers before the SPI_TXCTL and SPI_TWC registers. If enabling both transmit and receive in initiating mode, enable the SPI_CTL register after programming both the SPI_RXCTL and SPI_TXCTL registers.

These programming guidelines prevent SPI from starting a transfer when SPI registers are not fully programmed. Other ways of programming are also allowed as long as the initiating conditions prevent the start of communication until after programming of SPI registers is complete.

Avoid data corruption when changing the SPI module configuration. Do not change the configuration during a data transfer. Additionally, change the clock polarity only when no slave is selected. However, an exception to this rule exists. When an SPI communication link consists of a single master and slave, SPI_CTL.ASSEL = 0. The slave select input of the slave is permanently tied low. In this case, the slave is always selected. Avoid data corruption by enabling the slave only after both the master and slave devices are configured.

The module supports 8, 16-bit and 32-bit word sizes. To ensure correct operation, configure both the master and slave with the same word size.

Master Operation in Non-DMA Modes

This section describes the operation of the SPI as a master in non-DMA mode.

1. Write to the SPI_SLVSEL register, setting one or more of the SPI select enable bits. This operation ensures that the desired slaves are properly deselected while the master is configured.
2. The SPI_RXCTL.RTI and SPI_TXCTL.TTI bits determine the SPI initiating mode. The initiating mode defines the primary transfer channel, and also the initiating condition for the transfer.
3. Write to the SPI_CLK, SPI_CTL, SPI_RXCTL, and SPI_TXCTL registers. This operation enables the device as a master and configures the SPI system. It specifies the transfer modes and channels, appropriate word length, transfer format, baud rate, and other control information.

ADDITIONAL INFORMATION: If SPI_RXCTL.RTI is enabled and SPI_TXCTL.TTI is not, write to the SPI_RXCTL register after writing into SPI_CTL, SPI_TXCTL, and SPI_TFIFO registers to prevent a transmit underrun for the first transfer.

4. If SPI_CTL.ASSEL=0, activate the desired slaves by clearing one or more of the SPI_SLVSEL flag bits. Otherwise, the SPI hardware performs slave activation.
5. The SPI controller then generates the programmed clock pulses on SPI_CLK and simultaneously shifts data out of SPI_MOSI while shifting data in from SPI_MISO. Before a shift, the shift register is loaded with the contents of the SPI_TFIFO register. At the end of the transfer, the contents of the shift register are loaded into SPI_RFIFO.
6. Whenever the initiating conditions are satisfied, the SPI continues to send and receive words. If the transmit buffer remains empty or the receive buffer remains full, the device operates according to the states of the SPI_TXCTL.TDU and SPI_RXCTL.RDO bits.
7. It is possible to program a secondary channel in addition to the initiating channel. This feature allows usage of available channel resources for receives or transmits simultaneously with the initiating channel.

Slave Operation in Non-DMA Modes

When a device is enabled as a slave in a non-DMA mode, a transition of the $\overline{\text{SPI\_SS}}$ select signal to the active state (low) triggers the the start of a transfer. Or, the first active edge of SPI_CLK triggers the start, depending on the state of SPI_CTL.CPHA bit. The interface operates in the following manner.

1. The core writes to the SPI_CTL, SPI_RXCTL, and SPI_TXCTL registers. The operation defines the mode of the serial link to be the same as the mode setup in the SPI master.
2. To prepare for the data transfer, the core writes data to be transmitted into SPI_TFIFO.
3. Once the $\overline{\text{SPI\_SS}}$ falling edge is detected, the slave starts sending data on active SPI_CLK edges and sampling data on inactive SPI_CLK edges.
4. Reception or transmission continues until $\overline{\text{SPI\_SS}}$ is released or until the slave has received the proper number of clock cycles.
5. The slave device continues to receive or transmit with each new falling edge transition on $\overline{\text{SPI\_SS}}$ or active SPI_CLK edge. If the transmit buffer remains empty or the receive buffer remains full, the device operates according to the states of the SPI_TXCTL.TDU and SPI_RXCTL.RDO bits.

Configuring DMA Master Mode

The SPI interface supports a write DMA channel and a read DMA channel. It can use these functions individually or in a lock-step manner in duplex mode (SPI_TXCTL.TTI= SPI_RXCTL.RTI=1).

1. Write to the appropriate DMA registers to enable the SPI DMA channel and to configure the necessary work units, access direction, word count, and so on.
2. Write to the SPI_SLVSEL register, setting one or more of the SPI flag select bits.

3. Write to the SPI_CLK and SPI_CTL registers, enabling the device as a master and configuring the SPI system by specifying the appropriate word length, transfer format, baud rate, and so forth.
4. Write to SPI_RXCTL to configure SPI master receive mode, or write to SPI_TXCTL to configure SPI master transmit mode.
5. Finally, write to the SPI_RXCTL.REN bit to enable the receive channel, or write to SPI_TXCTL.TEN to enable the transmit channel.
6. If the SPI_RXCTL.RTI bit is enabled, a receive transfer is initiated upon enabling SPI_CTL.EN bit. If the receive word counter is enabled (SPI_RXCTL.RWCEN), then the SPI_RWC register must be non-zero for a transfer to initiate.

   ADDITIONAL INFORMATION: If enabling both receive and transmit DMA channels, but not enabling SPI_TXCTL.TTI, write to the SPI_RXCTL register after writing the SPI_CTL and SPI_TXCTL registers. In this way, a transmit underrun can be prevented for the first transfer. Subsequent transfers are initiated as the SPI reads data from the receive shift register and writes to the SPI receive FIFO. The SPI then requests a write from DMA to memory. Upon a DMA grant, the DMA engine reads a word from the SPI receive FIFO and writes to memory. New requests continue to be initiated as long as the receive FIFO does not fill up, when SPI_RWC does not become zero while SPI_RXCTL.RWCEN=1.
7. If SPI_TXCTL.TTI is enabled, the SPI controller requests DMA reads from memory as long as there is space for more data in the transmit pipe. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO. As long as transmit data is available in the FIFO, and the SPI_TWC register is non-zero when SPI_TXCTL.TWCEN=1, the SPI continues to initiate transfers until disabled.
8. If both the SPI_TXCTL.TTI and SPI_RXCTL.RTI bits are enabled, the SPI controller requests a DMA read from memory. However, there must be space for more data in the transmit pipe and the number of words written into the SPI must be less than SPI_TWC if SPI_TXCTL.TWCEN=1. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO.

   ADDITIONAL INFORMATION: As the SPI writes data from the transmit FIFO into the transmit shift register, it initiates a transfer on the SPI link. Data received from the transfer is moved from the SPI receive shift register to the receive FIFO. The SPI controller requests a write from DMA to memory. Upon a DMA grant, the DMA engine reads a word from the receive FIFO and writes to memory. Transfer continues to be initiated as long as both receives and transmits can accommodate new data.
9. If the receive pipe fills up due to unavailability of DMA grants, the transmit pipe stalls until the pipe is drained. If the transmit pipe fills up, the SPI stops requesting for DMA writes. If the value in SPI_RWC expires, further write-requests to DMA stop. However, data already written into the transmit FIFO is sent, and read requests to DMA continue until the receive data is read from the receive FIFO.
10. The SPI then generates the programmed clock pulses on SPI_CLK and simultaneously shifts data out of SPI_MOSI while shifting data in from SPI_MISO. For receive transfers, the value in the shift register is loaded into the SPI_RFIFO register at the end of the transfer. For transmit transfers, the value in the SPI_TFIFO register is loaded into the shift register at the start of the transfer.

Configuring DMA Slave Mode Operation

This mode occurs when the SPI is enabled as a slave and the DMA engine is configured to transmit or receive data. A transition of the SPI_SS signal to the active-low state triggers the start of a transfer. Or, the first active edge of SPI_CLK triggers the start of a transfer, depending on the state of the SPI_CTL.CPHA bit. The following steps illustrate the SPI receive or transmit DMA sequence in an SPI slave (in response to a master command). The SPI supports a receive DMA channel and a transmit DMA channel.

1. Write to the appropriate DMA registers to enable the SPI DMA channel and configure the necessary work units, access direction, word count, and so on.
2. Write to the SPI_CTL, SPI_RXCTL, and SPI_TXCTL registers to define the mode of the serial link to be the same as the mode configured in the SPI master.
3. If the receive channel is enabled (SPI_RXCTL.REN is asserted), the following actions occur:
    a. Once the slave select input is active, the slave starts receiving and transmitting data on active SPI_CLK edges.
    b. The value in the shift register is loaded into the SPI_RFIFO register at the end of the transfer.
    c. Once SPI_RFIFO has valid data, it requests a write from DMA to memory.
    d. Upon a DMA grant, the DMA engine reads a word from the receive FIFO and writes to memory.
    e. As long as there is data in the receive FIFO, the SPI slave continues to request a DMA write to memory. The DMA engine continues to read a word from the FIFO and writes to memory. The SPI slave continues receiving words on active SPI_CLK edges as long as the SPI_SS input is active.
    f. If the data collected in the receive pipe breaches the set level, and the DMA engine cannot keep up with the receive rate, the slave can deassert the SPI_RDY signal. This signaling throttles the master. The receive pipe level is set according to the SPI_CTL.FCWM field. The signal is deasserted as the DMA drains the receive FIFO. Alternatively, the SPI can use the SPI_RXCTL.RDO bit to decide when the incoming data is discarded or overwritten into the receive FIFO (when SPI_CTL.FCEN is inactive).
4. If the transmit channel is enabled (SPI_TXCTL.TEN is asserted), the following actions occur:
    a. The SPI requests a DMA read from memory.
    b. Upon a DMA grant, the DMA engine reads a word from memory and writes to the transmit FIFO.
    c. The SPI then reads DMA data from the transmit FIFO and writes to the transmit shift register, awaiting the start of the next transfer.
    d. Once the slave select input is active, the slave starts receiving and transmitting data on active SPI_CLK edges.
    e. As long as there is room in the transmit FIFO, the SPI slave continues to request a DMA read from memory. The DMA engine continues to read a word from memory and write to the transmit FIFO. The SPI slave continues transmitting words on active SPI_CLK edges as long as the SPI_SS input is active.

f. If the number of outstanding data entries in the transmit pipe breaches the level set and the DMA cannot keep up with the transmit rate, the slave deasserts the SPI_RDY signal. This signaling throttles the master. The transmit pipe level is set according to the SPI_CTL.FCWM field. The signal is deasserted as the DMA fills the transmit FIFO. Alternately, the SPI_TXCTL.TDU bit decides the state of the transmit data (when SPI_CTL.FCEN is deasserted).

5. If both receive and transmit channels are enabled, the following actions occur after the actions for each channel. Transfers continue as long as both receive and transmit channels can accommodate new data.

   a. If the receive pipe fills up due to the unavailability of DMA grant, the SPI interface stalls the master by asserting the SPI_RDY pin. This signal is deasserted as the DMA drains the receive FIFO. Alternately, the SPI uses the SPI_RXCTL.RDO bit to decide when the incoming data is discarded or overwritten in the receive FIFO (when SPI_CTL.FCEN is deasserted).

   b. If the transmit pipe fills up, the SPI stops requesting DMA writes until the pipe clears.

   c. If there is an underflow problem in the transmit pipe, the slave stalls the master by deasserting SPI_RDY while the DMA fills the transmit FIFO. Alternately, the SPI uses the SPI_TXCTL.TDU bit to decide the state of the transmit data (when SPI_CTL.FCEN is deasserted).

CM41X_M4 SPI Register Descriptions

Serial Peripheral Interface (SPI) contains the following registers.

Table 28-7: CM41X_M4 SPI Register List

Name	Description
SPI_CLK	Clock Rate Register
SPI_CTL	Control Register
SPI_DLY	Delay Register
SPI_ILAT	Masked Interrupt Condition Register
SPI_ILAT_CLR	Masked Interrupt Clear Register
SPI_IMSK	Interrupt Mask Register
SPI_IMSK_CLR	Interrupt Mask Clear Register
SPI_IMSK_SET	Interrupt Mask Set Register
SPI_RFIFO	Receive FIFO Data Register
SPI_RWC	Received Word Count Register
SPI_RWCR	Received Word Count Reload Register
SPI_RXCTL	Receive Control Register
SPI_SLVSEL	Slave Select Register
SPI_STAT	Status Register

Table 28-7: CM41X_M4 SPI Register List (Continued)

Name	Description
SPI_TFIFO	Transmit FIFO Data Register
SPI_TWC	Transmitted Word Count Register
SPI_TWCR	Transmitted Word Count Reload Register
SPI_TXCTL	Transmit Control Register

Clock Rate Register

The SPI_CLK register selects the baud rate for SPI data transfers, relating this rate to the SPI serial clock (SPI clock) and the system clock (SCLK).

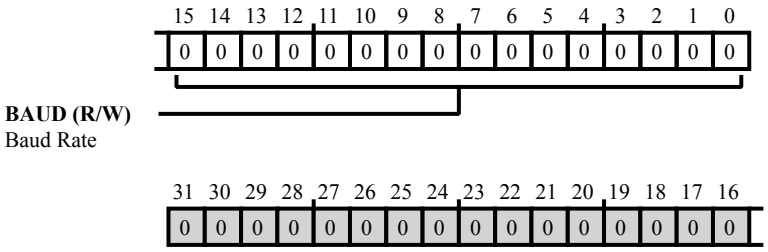


Figure 28-12: SPI_CLK Register Diagram

Table 28-8: SPI_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	BAUD	Baud Rate. The SPI_CLK.BAUD bits set the SPI baud rate according to the formula: BAUD = (SCLK / SPI Clock) - 1

Control Register

The SPI_CTL register enables the SPI and configures settings for operating modes, communication protocols, and buffer operations.

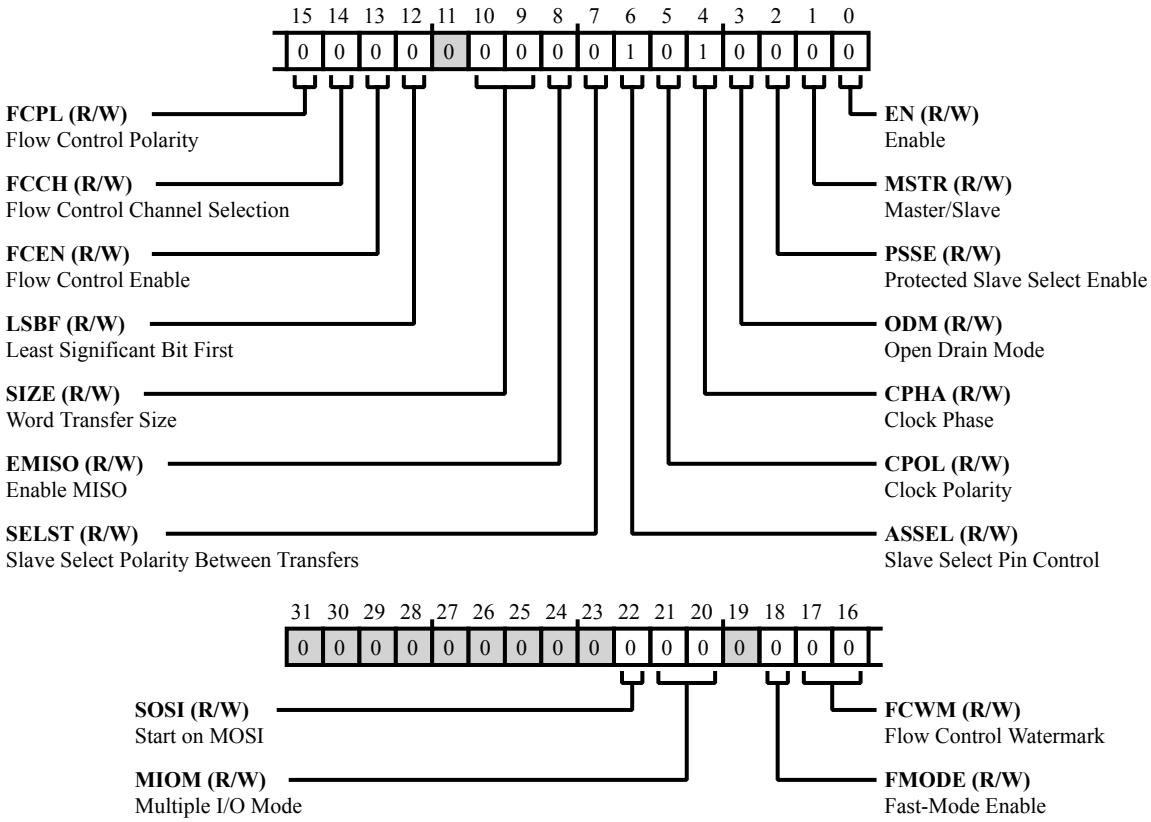


Figure 28-13: SPI_CTL Register Diagram

Table 28-9: SPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	SOSI	Start on MOSI. The SPI_CTL.SOSI bit is valid only when SPI_CTL.MIOM is enabled for either DIOM or QIOM, and this bit selects the starting pin and the bit placement on pins for these modes. In DIOM, by default, (SPI_CTL.SOSI =0) SPI sends the first bit on the SPI_MISO pin and the second bit on the SPI_MOSI pin. In QIOM, by default, the SPI sends the first bit on the SPI_D3 pin, the second bit on the SPI_D2 pin, the third bit on the SPI_MISO pin and the fourth bit on the SPI_MOSI pin. This order can be reversed by setting the SPI_CTL.SOSI bit. When this bit is set, the SPI sends the first bit on the SPI_MOSI pin. The first bit referred to here depends on the SPI_CTL.LSBF bit setting (MSB bit or LSB bit).	
		0	Start on MISO (DIOM) or start on SPI_D3
		1	Start on MOSI
21:20 (R/W)	MIOM	Multiple I/O Mode. The SPI_CTL.MIOM bits enable SPI operation in dual I/O mode (DIOM) or quad I/O mode (QIOM). These bits can only be changed when the SPI is disabled (SPI_CTL.EN =0).	
		0	No MIOM (disabled)
		1	DIOM operation
		2	QIOM operation
		3	Reserved
18 (R/W)	FMODE	Fast-Mode Enable. The SPI_CTL.FMODE bit enables fast mode operation for SPI receive transfers. SPI transmit operations in fast mode are the same as normal mode.	
		0	Disable
		1	Enable

Table 28-9: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:16 (R/W)	FCWM	Flow Control Watermark. The SPI_CTL.FCWM bits select the watermark level of the transmit channel (SPI_TFIFO buffer) or receive channel (SPI_RFIFO buffer) that triggers flow control operation. These bits are applicable only when the SPI is a slave (SPI_CTL.MSTR = 0) and flow control is enabled (SPI_CTL.FCEN =1). When the watermark condition is met, the SPI slave deasserts the SPI_RDY pin.	
		0	TFIFO empty or RFIFO full
		1	TFIFO 75% or more empty, or RFIFO 75% or more full
		2	TFIFO 50% or more empty, or RFIFO 50% or more full
		3	Reserved
15 (R/W)	FCPL	Flow Control Polarity. The SPI_CTL.FCPL bit selects flow control polarity for the SPI_RDY pin when flow control is enabled. When the SPI_RDY pin is active, the SPI is indicating it is ready for data transfer.	
		0	Active-low RDY
		1	Active-high RDY
14 (R/W)	FCCH	Flow Control Channel Selection. The SPI_CTL.FCCH bit selects whether the SPI applies flow control to the transmit channel (SPI_TFIFO buffer) or receive channel (SPI_RFIFO buffer). This bit is applicable only when the SPI is a slave and flow control is enabled.	
		0	Flow control on RX buffer
		1	Flow control on TX buffer
13 (R/W)	FCEN	Flow Control Enable. The SPI_CTL.FCEN bit enables SPI flow control operation, which permits slow slave devices to interface with fast master devices. This bit controls the operation of the SPI_RDY pin. Note that options for flow control operation are available using the SPI_CTL.FCCH, SPI_CTL.FCPL, and SPI_CTL.FCWM bits.	
		0	Disable
		1	Enable

Table 28-9: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	LSBF	Least Significant Bit First. The SPI_CTL.LSBF bit selects whether the SPI transmits/receives data as LSB first (little endian) or MSB first (big endian). This bit can only be changed when the SPI is disabled.	
		0	MSB sent/received first (big endian)
		1	LSB sent/received first (little endian)
10:9 (R/W)	SIZE	Word Transfer Size. The SPI_CTL.SIZE bits select the SPI transfer word size as 8, 16 or 32 bits. To ensure correct operation, both the master and slave must be configured with the same word size. This bit can only be changed when the SPI is disabled (SPI_CTL.EN =0).	
		0	8-bit word
		1	16-bit word
		2	32-bit word
		3	Reserved
8 (R/W)	EMISO	Enable MISO. The SPI_CTL.EMISO bit enables master-in-slave-out (MISO) mode. This SPI mode is applicable only when the SPI is a slave.	
		0	Disable
		1	Enable
7 (R/W)	SELST	Slave Select Polarity Between Transfers. The SPI_CTL.SELST bit selects the state (polarity) for the $\overline{\text{SPI\_SEL[n]}}$ pin between SPI transfers when the SPI is a master and hardware slave select assertion is enabled (SPI_CTL.ASSEL =1). In slave mode, this bit affects the detection of both transmit collision (SPI_STAT.TC and underrun (SPI_STAT.TUR) errors.	
		0	Deassert slave select (high)
		1	Assert slave select (low)

Table 28-9: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	ASSEL	Slave Select Pin Control. The SPI_CTL.ASSEL bit selects whether the SPI hardware sets the $\overline{\texttt{SPI\_SEL[n]}}$ pin output value (ignoring the slave select SPI_SLVSEL.SSEL1 - $\overline{\texttt{SPI\_SLVSEL.SSEL7}}$ bits) or whether software control of the slave select bits set the $\overline{\texttt{SPI\_SEL[n]}}$ pin output value. This feature is applicable only when the SPI is a master. When hardware control is enabled, the $\overline{\texttt{SPI\_SEL[n]}}$ pin output is asserted during the transfers, and the pin polarity between transfers is selected by the SPI_CTL.SELST bit. When software control is enabled, the $\overline{\texttt{SPI\_SEL[n]}}$ pin output value is set through software control of the slave select bits, and as such, the pin may either remain asserted (low) or be deasserted between transfers.	
		0	Software slave select control
		1	Hardware slave select control
5 (R/W)	CPOL	Clock Polarity. The SPI_CTL.CPOL bit selects whether the SPI uses an active-low or active-high signal for the SPI clock (SPI_CLK). This bit works with the SPI_CTL.CPHA bit to select combinations of clock phase and polarity for the SPI_CLK pin. This bit can only be changed when the SPI is disabled.	
		0	Active-high SPI CLK
		1	Active-low SPI CLK
4 (R/W)	CPHA	Clock Phase. The SPI_CTL.CPHA bit selects whether the SPI starts toggling the signal for the SPI clock (SPI_CLK) from the start of the first data bit or from the middle of the first data bit. The SPI_CTL.CPHA bit works with the SPI_CTL.CPOL bit to select combinations of clock phase and polarity for the SPI_CLK pin. This bit can only be changed when the SPI is disabled.	
		0	SPI CLK toggles from middle
		1	SPI CLK toggles from start

Table 28-9: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	ODM	Open Drain Mode. The SPI_CTL.ODM bit configures the data output pins (SPI_MOSI and SPI_MISO) to behave as open drain outputs, which prevents contention and possible damage to pin drivers in multi-master or multi-slave SPI systems. When SPI_CTL.ODM is enabled and the SPI is a master, the SPI three-states the SPI_MOSI pin when the data driven out on MOSI is a logic-high. The SPI does not three-state the SPI_MOSI pin when the driven data is a logic-low. When SPI_CTL.ODM is enabled and the SPI is a slave, the SPI three-states the SPI_MISO pin when the data driven out on SPI_MISO is a logic-high. Note that an external pull-up resistor is required on both the SPI_MOSI and SPI_MISO pins when SPI_CTL.ODM is enabled.	
		0	Disable
		1	Enable
2 (R/W)	PSSE	Protected Slave Select Enable. The SPI_CTL.PSSE bit enables the $\overline{\text{SPI\_SS}}$ pin to provide error detection input in a multi-master environment when the SPI is in master mode. If some other device in the system asserts the $\overline{\text{SPI\_SS}}$ pin while SPI is enabled as master (and SPI_CTL.PSSE is enabled), this condition causes a mode fault error.	
		0	Disable
		1	Enable
1 (R/W)	MSTR	Master/Slave. The SPI_CTL.MSTR bit toggles the SPI between master mode and slave mode. This bit can only be changed when the SPI is disabled.	
		0	Slave
		1	Master
0 (R/W)	EN	Enable. The SPI_CTL.EN bit enables SPI operation.	
		0	Disable SPI module
		1	Enable

Delay Register

The SPI_DLY register selects a transfer delay and the lead/lag timing between slave select signals and SPI clock edge assertion/deassertion.

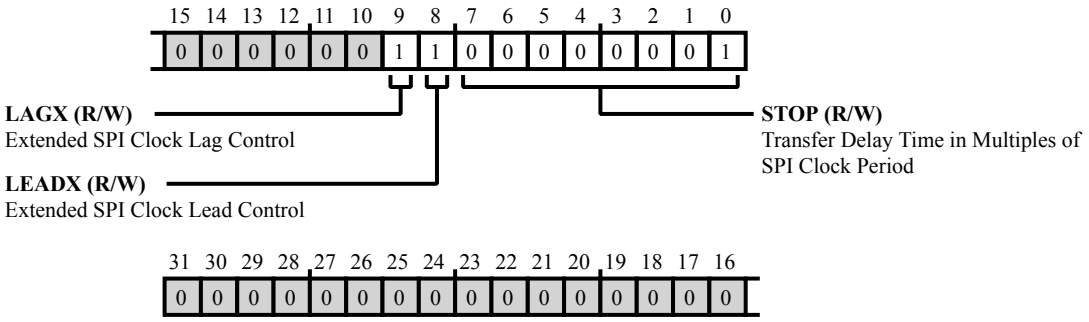


Figure 28-14: SPI_DLY Register Diagram

Table 28-10: SPI_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	LAGX	Extended SPI Clock Lag Control. The SPI_DLY.LAGX bit enables insertion of a 1-SPI_CLK cycle lag (extend lag) in the timing between the slave select (SPI_SEL[n]) assertion and first SPI clock edge.	
		0	Disable
		1	Enable
8 (R/W)	LEADX	Extended SPI Clock Lead Control. The SPI_DLY.LEADX bit enables insertion of a 1-SPI_CLK cycle lead (extend lead) in the timing between the slave select (SPI_SEL[n]) deassertion and last SPI clock edge.	
		0	Disable
		1	Enable
7:0 (R/W)	STOP	Transfer Delay Time in Multiples of SPI Clock Period. The SPI_DLY.STOP bits select a delay (number of stop bits in multiples of SPI clock duration) at the end of each SPI transfer. The default delay is the minimum value required to comply with the SPI protocol (1-bit duration). The SPI_DLY.STOP bits can be programmed with smaller delay values, resulting in continuous operation (for example, stop bits =0).	

Masked Interrupt Condition Register

The SPI_ILAT register latches interrupts, queuing the interrupt requests for service. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt request is unmasked in SPI_IMSK, the SPI latches the interrupt request bit in SPI_ILAT.

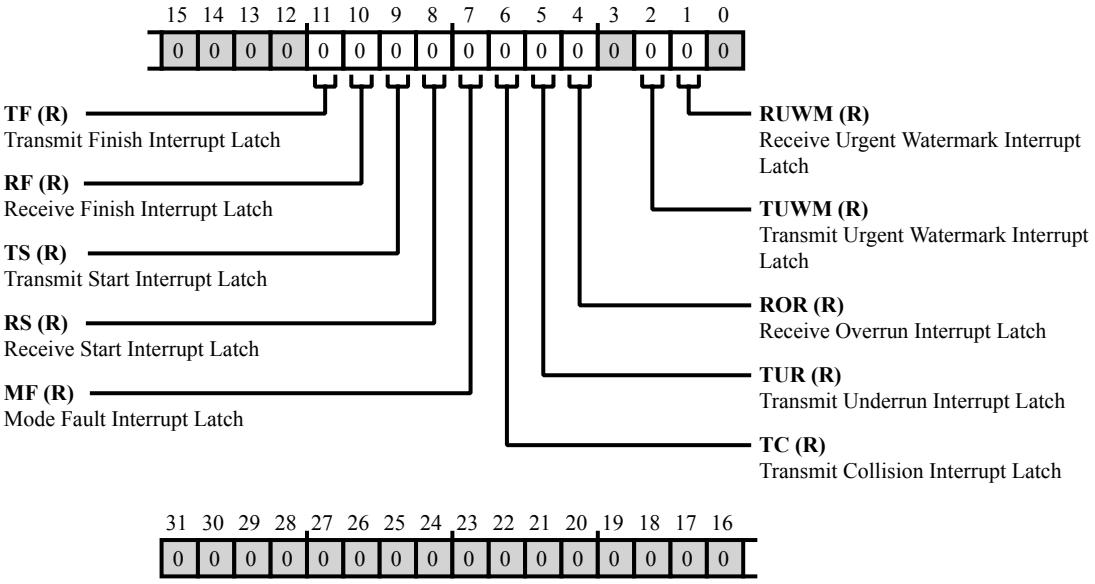


Figure 28-15: SPI_ILAT Register Diagram

Table 28-11: SPI_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
10 (R/NW)	RF	Receive Finish Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
9 (R/NW)	TS	Transmit Start Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
8 (R/NW)	RS	Receive Start Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request

Table 28-11: SPI_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	MF	Mode Fault Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
6 (R/NW)	TC	Transmit Collision Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
5 (R/NW)	TUR	Transmit Underrun Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
4 (R/NW)	ROR	Receive Overrun Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
2 (R/NW)	TUWM	Transmit Urgent Watermark Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request
1 (R/NW)	RUWM	Receive Urgent Watermark Interrupt Latch.	
		0	No interrupt request
		1	Latched interrupt request

Masked Interrupt Clear Register

The SPI_ILAT_CLR register permits clearing individual mask bits in the SPI_ILAT register without affecting other bits in the register. Use write-1-to-clear on a bit in the SPI_ILAT_CLR register to clear the corresponding bit in the SPI_ILAT register.

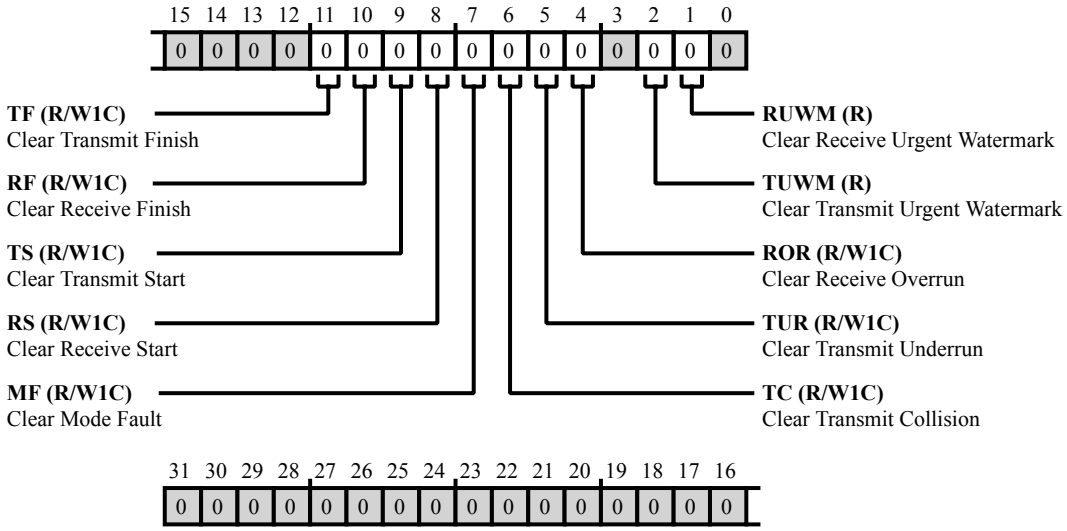


Figure 28-16: SPI_ILAT_CLR Register Diagram

Table 28-12: SPI_ILAT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	TF	Clear Transmit Finish. The SPI_ILAT_CLR.TF bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
10 (R/W1C)	RF	Clear Receive Finish. The SPI_ILAT_CLR.RF bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
9 (R/W1C)	TS	Clear Transmit Start. The SPI_ILAT_CLR.TS bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit

Table 28-12: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	RS	Clear Receive Start. The SPI_ILAT_CLR.RS bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
7 (R/W1C)	MF	Clear Mode Fault. The SPI_ILAT_CLR.MF bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
6 (R/W1C)	TC	Clear Transmit Collision. The SPI_ILAT_CLR.TC bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
5 (R/W1C)	TUR	Clear Transmit Underrun. The SPI_ILAT_CLR.TUR bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
4 (R/W1C)	ROR	Clear Receive Overrun. The SPI_ILAT_CLR.ROR bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit
2 (R/NW)	TUWM	Clear Transmit Urgent Watermark. The SPI_ILAT_CLR.TUWM bit clears the corresponding mask bit in the SPI_ILAT register.	
		0	No effect
		1	Clear mask bit

Table 28-12: SPI_ILAT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	RUWM	Clear Receive Urgent Watermark. The `SPI_ILAT_CLR.RUWM` bit clears the corresponding mask bit in the `SPI_ILAT` register.	
		0	No effect
		1	Clear mask bit

Interrupt Mask Register

The SPI_IMSK register unmasks (enables) or masks (disables) SPI interrupt requests. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt request is unmasked in SPI_IMSK, the SPI latches the interrupt request bit in the SPI_ILAT register, queuing the interrupt request for service.

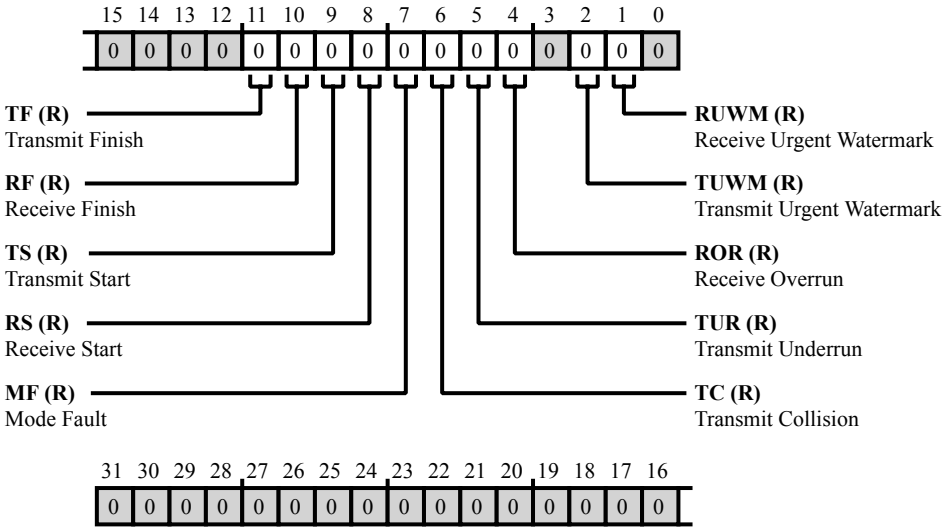


Figure 28-17: SPI_IMSK Register Diagram

Table 28-13: SPI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish. The SPI_IMSK.TF bit unmasks (enables) or masks (disables) the TF interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
10 (R/NW)	RF	Receive Finish. The SPI_IMSK.RF bit unmasks (enables) or masks (disables) the RF interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
9 (R/NW)	TS	Transmit Start. The SPI_IMSK.TS bit unmasks (enables) or masks (disables) the TS interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request

Table 28-13: SPI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	RS	Receive Start. The `SPI_IMSK.RS` bit unmasks (enables) or masks (disables) the RS interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
7 (R/NW)	MF	Mode Fault. The `SPI_IMSK.MF` bit unmasks (enables) or masks (disables) the MF interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
6 (R/NW)	TC	Transmit Collision. The `SPI_IMSK.TC` bit unmasks (enables) or masks (disables) the TC interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
5 (R/NW)	TUR	Transmit Underrun. The `SPI_IMSK.TUR` bit unmasks (enables) or masks (disables) the TUR interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
4 (R/NW)	ROR	Receive Overrun. The `SPI_IMSK.ROR` bit unmasks (enables) or masks (disables) the ROR interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
2 (R/NW)	TUWM	Transmit Urgent Watermark. The `SPI_IMSK.TUWM` bit unmasks (enables) or masks (disables) the TUWM interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request
1 (R/NW)	RUWM	Receive Urgent Watermark. The `SPI_IMSK.RUWM` bit unmasks (enables) or masks (disables) the RUWM interrupt.	
		0	Disable (mask) interrupt request
		1	Enable (unmask) interrupt request

Interrupt Mask Clear Register

The SPI_IMSK_CLR register permits clearing individual mask bits in the SPI_IMSK register without affecting other bits in the register. Use write-1-to-clear on a bit in the SPI_IMSK_CLR register to clear the corresponding bit in the SPI_IMSK register.

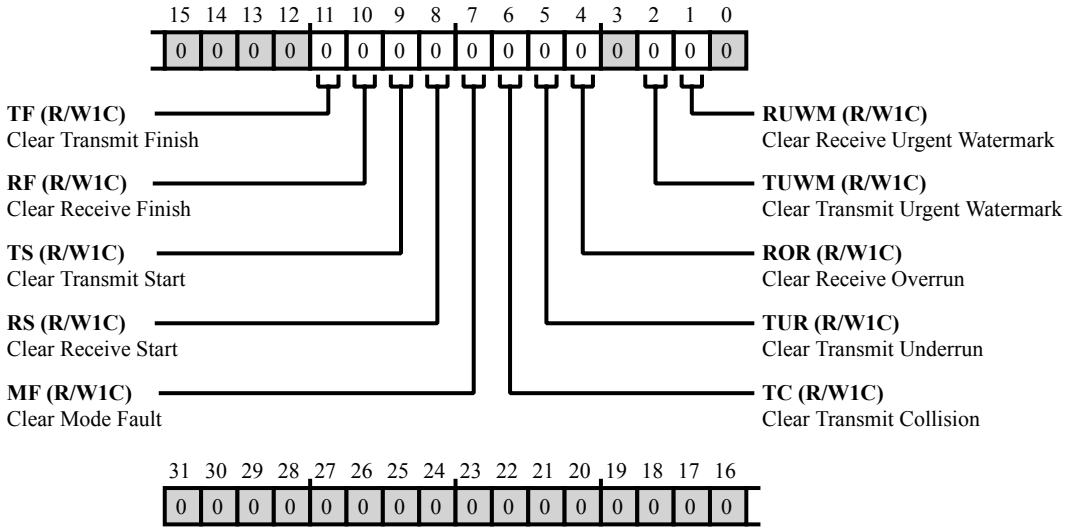


Figure 28-18: SPI_IMSK_CLR Register Diagram

Table 28-14: SPI_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1C)	TF	Clear Transmit Finish. The SPI_IMSK_CLR.TF bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
10 (R/W1C)	RF	Clear Receive Finish. The SPI_IMSK_CLR.RF bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
9 (R/W1C)	TS	Clear Transmit Start. The SPI_IMSK_CLR.TS bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit

Table 28-14: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	RS	Clear Receive Start. The SPI_IMSK_CLR.RS bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
7 (R/W1C)	MF	Clear Mode Fault. The SPI_IMSK_CLR.MF bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
6 (R/W1C)	TC	Clear Transmit Collision. The SPI_IMSK_CLR.TC bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
5 (R/W1C)	TUR	Clear Transmit Underrun. The SPI_IMSK_CLR.TUR bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
4 (R/W1C)	ROR	Clear Receive Overrun. The SPI_IMSK_CLR.ROR bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit
2 (R/W1C)	TUWM	Clear Transmit Urgent Watermark. The SPI_IMSK_CLR.TUWM bit clears the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Clear mask bit

Table 28-14: SPI_IMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	RUWM	Clear Receive Urgent Watermark. The `SPI_IMSK_CLR.RUWM` bit clears the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Clear mask bit

Interrupt Mask Set Register

The SPI_IMSK_SET register permits setting individual mask bits in the SPI_IMSK register without affecting other bits in the register. Use write-1-to-set on a bit in the SPI_IMSK_SET register to set the corresponding bit in the SPI_IMSK register.

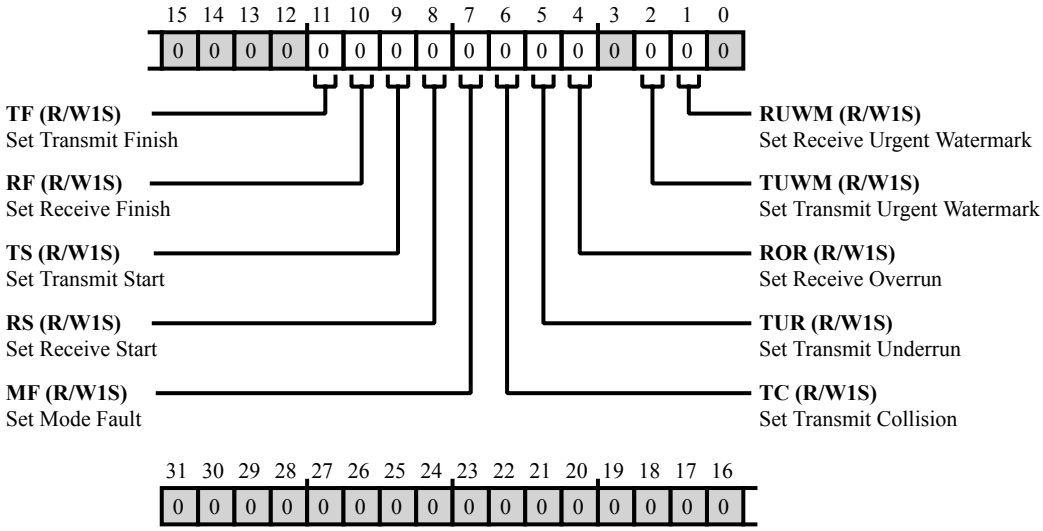


Figure 28-19: SPI_IMSK_SET Register Diagram

Table 28-15: SPI_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W1S)	TF	Set Transmit Finish. The SPI_IMSK_SET.TF bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
10 (R/W1S)	RF	Set Receive Finish. The SPI_IMSK_SET.RF bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
9 (R/W1S)	TS	Set Transmit Start. The SPI_IMSK_SET.TS bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit

Table 28-15: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1S)	RS	Set Receive Start. The SPI_IMSK_SET.RS bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
7 (R/W1S)	MF	Set Mode Fault. The SPI_IMSK_SET.MF bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
6 (R/W1S)	TC	Set Transmit Collision. The SPI_IMSK_SET.TC bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
5 (R/W1S)	TUR	Set Transmit Underrun. The SPI_IMSK_SET.TUR bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
4 (R/W1S)	ROR	Set Receive Overrun. The SPI_IMSK_SET.ROR bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit
2 (R/W1S)	TUWM	Set Transmit Urgent Watermark. The SPI_IMSK_SET.TUWM bit sets the corresponding mask bit in the SPI_IMSK register.	
		0	No effect
		1	Set mask bit

Table 28-15: SPI_IMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	RUWM	Set Receive Urgent Watermark. The `SPI_IMSK_SET.RUWM` bit sets the corresponding mask bit in the `SPI_IMSK` register.	
		0	No effect
		1	Set mask bit

Receive FIFO Data Register

The SPI_RFIFO register has an interface to the receive shift register in the SPI and has an interface to the processor's data buses. The top level of the buffer is visible to programs as the 32-bit SPI_RFIFO register, but the size (number of word locations) of the receive FIFO is actually flexible with transfer word size. The size of the receive FIFO is 8 if the word size is 8-bit, or the size is 4 if the word size is 16-bit, or the size is 2 if the word size is 32-bit.

Both masters and slaves may stop or stall receive transfers based on FIFO status. When the receive FIFO is full, the SPI master stops initiating new transfers on the SPI if SPI_RXCTL.RTI is enabled. A slave may stall the SPI interface when the content of the FIFO crosses the selected watermark. If data reception continues after SPI_RFIFO is full, the data in the receive FIFO is invalid. The SPI indicates this condition with receive overrun (SPI_STAT.ROR) error. This condition is possible when SPI_RXCTL.RTI =0 and SPI_RXCTL.REN =1 for a master, or for a slave that does not exercise flow control.

Note that the receive FIFO is reset (cleared) when the SPI is disabled after being enabled.

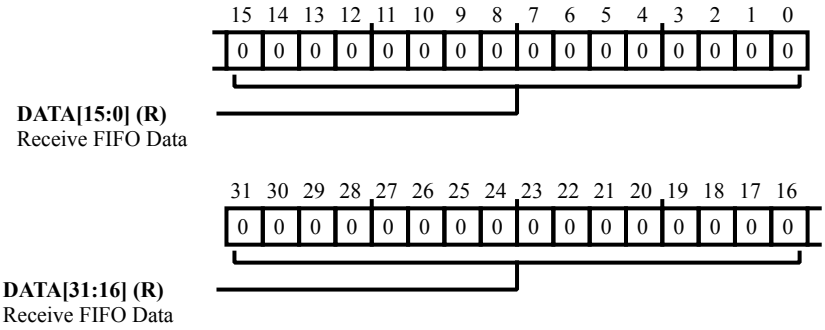


Figure 28-20: SPI_RFIFO Register Diagram

Table 28-16: SPI_RFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Receive FIFO Data. The SPI_RFIFO.DATA bit field contains the FIFO receive data.

Received Word Count Register

The SPI_RWC register holds a count of the number of words remaining to be received by the SPI. To start the decrement of the word count in SPI_RWC, enable the receive word counter (SPI_RXCTL.RWCEN =1). The SPI uses the word count to control the duration of transfers and to signal the completion of a burst of transfers with the receive finish interrupt (SPI_ILAT.RF). In DMA mode, the SPI uses the SPI_RWC register to ensure that the number of frames received during a DMA transfer is equal to the number of words programmed in the DMA channel controller. The values programmed into the SPI_RWC registers should match the word count in the DMA configuration. The SPI_RWC register maintains the number of frames to be received in a transfer. The SPI_RWC should only be changed when the counter is disabled.

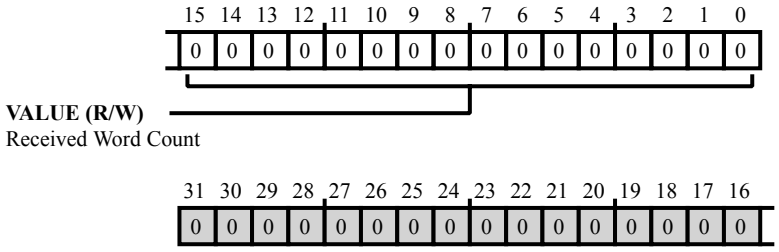


Figure 28-21: SPI_RWC Register Diagram

Table 28-17: SPI_RWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count. The SPI_RWC.VALUE bits hold the receive transfer word count.

Received Word Count Reload Register

The SPI_RWCR register holds the receive word count value that the SPI loads into the SPI_RWC register when the transfer count decrements to zero. To prevent the SPI from reloading the counter, use zero for the reload count value. The SPI_RWCR register should only be changed when the counter is disabled.

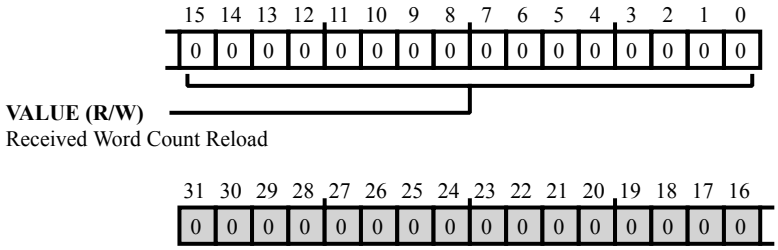


Figure 28-22: SPI_RWCR Register Diagram

Table 28-18: SPI_RWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Received Word Count Reload. The SPI_RWCR.VALUE bits hold the receive transfer word count reload value.

Receive Control Register

The SPI_RXCTL register enables the SPI receive channel, initiates receive transfers, and configures SPI_RFIFO buffer watermark settings.

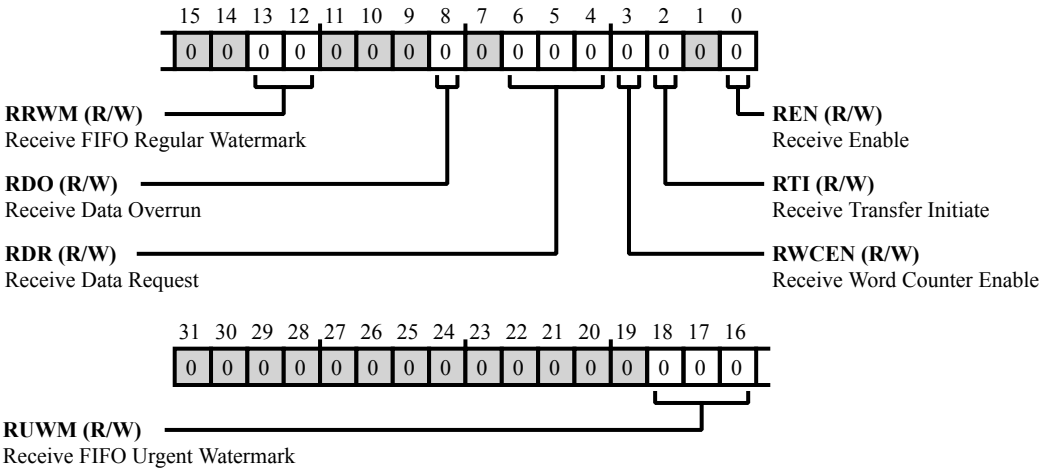


Figure 28-23: SPI_RXCTL Register Diagram

Table 28-19: SPI_RXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	RUWM	Receive FIFO Urgent Watermark. The SPI_RXCTL.RUWM bits select the receive FIFO (SPI_RFIFO) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the SPI_ILAT.RUWM interrupt. When an urgent SPI_RFIFO watermark is enabled with SPI_RXCTL.RUWM, the SPI_RXCTL.RRWM selection is used as the deassertion condition for any SPI_ILAT.RUWM interrupts that are latched.	
		0	Disabled
		1	25% full RFIFO
		2	50% full RFIFO
		3	75% full RFIFO
		4	Full RFIFO
		5	Reserved
		6	Reserved
		7	Reserved

Table 28-19: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:12 (R/W)	RRWM	Receive FIFO Regular Watermark. The SPI_RXCTL.RRWM bits select the receive FIFO (SPI_RFIFO) watermark level for regular data bus requests. When an urgent SPI_RFIFO watermark is enabled with SPI_RXCTL.RUWM, the SPI_RXCTL.RRWM selection is used as the deassertion condition for any SPI_ILAT.RUWM interrupts that are latched.	
		0	Empty RFIFO
		1	RFIFO less than 25% full
		2	RFIFO less than 50% full
		3	RFIFO less than 75% full
8 (R/W)	RDO	Receive Data Overrun. The SPI_RXCTL.RDO bit selects handling for receive data requests when the receive buffer (SPI_RFIFO) is full. If enabled and SPI_RFIFO is full, the SPI overwrites old data in the buffer with incoming data. If disabled and SPI_RFIFO is full, the SPI keeps old data in the buffer and discards incoming data.	
		0	Discard incoming data if SPI_RFIFO is full
		1	Overwrite old data if SPI_RFIFO is full
6:4 (R/W)	RDR	Receive Data Request. The SPI_RXCTL.RDR bits select receive FIFO (SPI_RFIFO) watermark conditions that direct the SPI to generate a receive data request.	
		0	Disabled
		1	Not empty RFIFO
		2	25% full RFIFO
		3	50% full RFIFO
		4	75% full RFIFO
		5	Full RFIFO
		6	Reserved
		7	Reserved
3 (R/W)	RWCEN	Receive Word Counter Enable. The SPI_RXCTL.RWCEN bit enables the decrement of the SPI_RWC register when the count is not zero and SPI_RXCTL.RTI is enabled. Enabling SPI_RXCTL.RWCEN prevents receive overrun errors from occurring. The SPI_RXCTL.RWCEN bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable

Table 28-19: SPI_RXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	RTI	Receive Transfer Initiate. The SPI_RXCTL.RTI bit enables initiation of receive transfers if the receive FIFO (SPI_RFIFO) is not full. The bit also enables this initiation if SPI_RWC is not zero when SPI_RXCTL.RWCEN is enabled. Enabling SPI_RXCTL.RTI prevents receive overrun errors from occurring. The SPI_RXCTL.RTI bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
0 (R/W)	REN	Receive Enable. The SPI_RXCTL.REN bit enables SPI receive channel operation.	
		0	Disable
		1	Enable

Slave Select Register

The SPI_SLVSEL register enables the SPI_SEL[n] pins for output and indicates the state (high or low) of these pins when enabled.

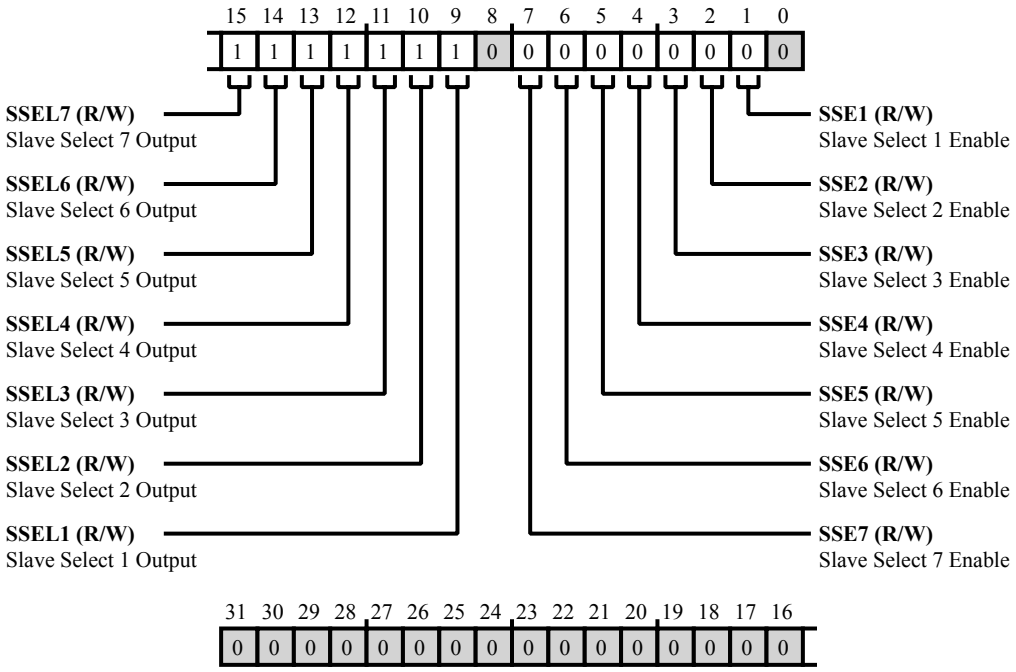


Figure 28-24: SPI_SLVSEL Register Diagram

Table 28-20: SPI_SLVSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	SSEL7	Slave Select 7 Output. The SPI_SLVSEL.SSEL7 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High
14 (R/W)	SSEL6	Slave Select 6 Output. The SPI_SLVSEL.SSEL6 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High

Table 28-20: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	SSEL5	Slave Select 5 Output. The SPI_SLVSEL.SSEL5 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High
12 (R/W)	SSEL4	Slave Select 4 Output. The SPI_SLVSEL.SSEL4 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High
11 (R/W)	SSEL3	Slave Select 3 Output. The SPI_SLVSEL.SSEL3 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High
10 (R/W)	SSEL2	Slave Select 2 Output. The SPI_SLVSEL.SSEL2 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High
9 (R/W)	SSEL1	Slave Select 1 Output. The SPI_SLVSEL.SSEL1 bit state indicates the value driven on the related SPI_SEL[n] pin.	
		0	Low
		1	High
7 (R/W)	SSE7	Slave Select 7 Enable. The SPI_SLVSEL.SSE7 bit enables the related SPI_SEL[n] pin for output. If disabled, the SPI three-states the related SPI_SEL[n] pin.	
		0	Disable
		1	Enable

Table 28-20: SPI_SLVSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SSE6	Slave Select 6 Enable. The SPI_SLVSEL.SSE6 bit enables the related $\overline{\text{SPI\_SEL[n]}}$ pin for output. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
5 (R/W)	SSE5	Slave Select 5 Enable. The SPI_SLVSEL.SSE5 bit enables the related $\overline{\text{SPI\_SEL[n]}}$ pin for output. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
4 (R/W)	SSE4	Slave Select 4 Enable. The SPI_SLVSEL.SSE4 bit enables the related $\overline{\text{SPI\_SEL[n]}}$ pin for output. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
3 (R/W)	SSE3	Slave Select 3 Enable. The SPI_SLVSEL.SSE3 bit enables the related $\overline{\text{SPI\_SEL[n]}}$ pin for output. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
2 (R/W)	SSE2	Slave Select 2 Enable. The SPI_SLVSEL.SSE2 bit enables the related $\overline{\text{SPI\_SEL[n]}}$ pin for output. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable
1 (R/W)	SSE1	Slave Select 1 Enable. The SPI_SLVSEL.SSE1 bit enables the related $\overline{\text{SPI\_SEL[n]}}$ pin for output. See the SPI_SLVSEL.SSE7 bit description for more information.	
		0	Disable
		1	Enable

Status Register

The SPI_STAT register indicates SPI status including FIFO status, error conditions, and interrupt conditions. When an interrupt condition from this register is unmasked (enabled) by the corresponding bit in the SPI_IMSK register, the interrupt request is latched into the corresponding bit in the SPI_ILAT register.

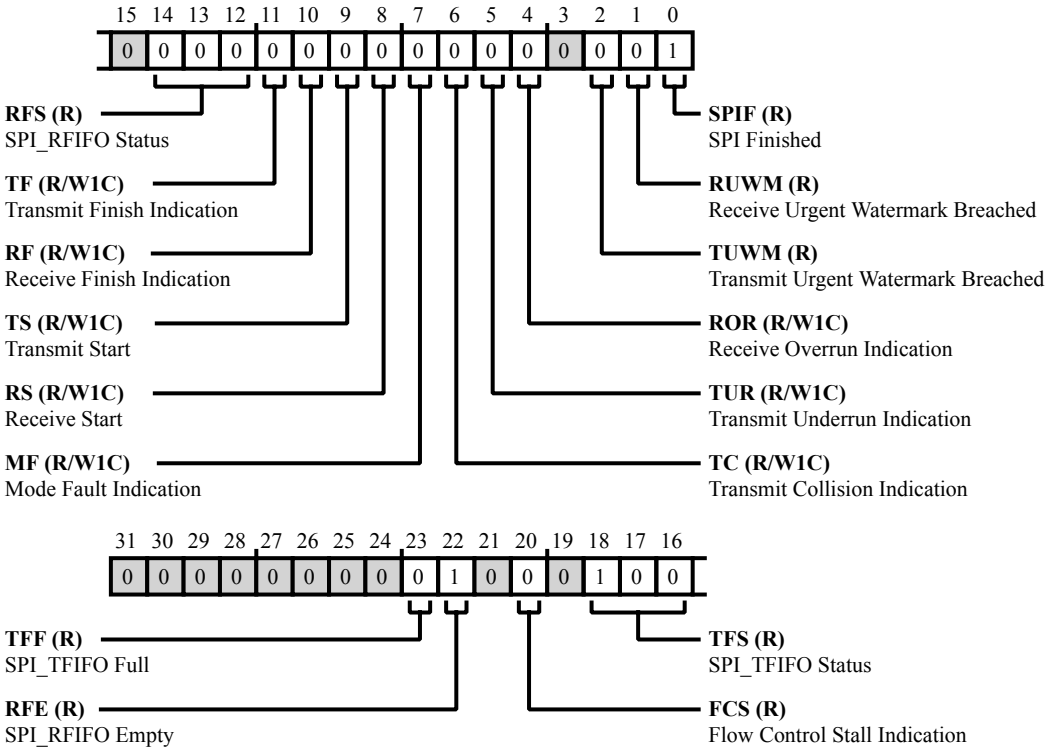


Figure 28-25: SPI_STAT Register Diagram

Table 28-21: SPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23 (R/NW)	TFF	SPI_TFIFO Full. The SPI_STAT.TFF bit indicates whether the SPI_TFIFO is full or not full.	
		0	Not full Tx FIFO
		1	Full Tx FIFO
22 (R/NW)	RFE	SPI_RFIFO Empty. The SPI_STAT.RFE bit indicates whether the SPI_RFIFO is empty or not empty.	
		0	Rx FIFO not empty
		1	Rx FIFO empty

Table 28-21: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	FCS	Flow Control Stall Indication. The SPI_STAT.FCS bit indicates whether a slave has deasserted the SPI_RDY pin to stall the SPI master while the slave is unable to service the SPI masters request. This bit is valid only when the SPI is a master (SPI_CTL.MSTR =1) and flow control is enabled (SPI_CTL.FCEN =1).	
		0	No Stall (RDY pin asserted)
		1	Stall (RDY pin deasserted)
18:16 (R/NW)	TFS	SPI_TFIFO Status. The SPI_STAT.TFS bits indicate the status of the SPI_TFIFO. The SPI uses this status when evaluating transmit watermark conditions.	
		0	Full TFIFO
		1	25% empty TFIFO
		2	50% empty TFIFO
		3	75% empty TFIFO
		4	Empty TFIFO
14:12 (R/NW)	RFS	SPI_RFIFO Status. The SPI_STAT.RFS bits indicate the status of the SPI_RFIFO. The SPI uses this status when evaluating receive watermark conditions.	
		0	Empty RFIFO
		1	25% full RFIFO
		2	50% full RFIFO
		3	75% full RFIFO
		4	Full RFIFO
11 (R/W1C)	TF	Transmit Finish Indication. The SPI_STAT.TF bit indicates that the SPI has detected the finish of a transmit burst transfer (the SPI_TWC count decrements to zero). This condition can only occur when SPI_TXCTL.TTI and SPI_TXCTL.TWCEN are enabled.	
		0	No status
		1	Transmit finish detected

Table 28-21: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W1C)	RF	Receive Finish Indication. The SPI_STAT.RF bit indicates that the SPI has detected the finish of a receive burst transfer (the SPI_RWC count decrements to zero). This condition can only occur when SPI_RXCTL.RTI and SPI_RXCTL.RWCEN are enabled.	
		0	No status
		1	Receive finish detected
9 (R/W1C)	TS	Transmit Start. The SPI_STAT.TS bit indicates that the SPI has detected the start of a transmit burst transfer. A transmit bursts starts with the load of SPI_TWC from the SPI_TWCR. This condition can only occur when SPI_TXCTL.TTI and SPI_TXCTL.TWCEN are enabled.	
		0	No status
		1	Transmit start detected
8 (R/W1C)	RS	Receive Start. The SPI_STAT.RS bit indicates that the SPI has detected the start of a receive burst transfer. A receive bursts starts with the load of SPI_RWC from the SPI_RWCR. This condition can only occur when SPI_RXCTL.RTI and SPI_RXCTL.RWCEN are enabled.	
		0	No status
		1	Receive start detected
7 (R/W1C)	MF	Mode Fault Indication. The SPI_STAT.MF bit, when SPI is a master and SPI_CTL.PSSE is enabled, indicates that multiple masters have asserted slave select inputs.	
		0	No status
		1	Mode fault occurred
6 (R/W1C)	TC	Transmit Collision Indication. The SPI_STAT.TC bit, when SPI is a slave, indicates that the load of data into the shift register has occurred too close to the first transmitting edge of the SPI clock.	
		0	No status
		1	Transmit collision occurred

Table 28-21: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	TUR	Transmit Underrun Indication. The SPI_STAT.TUR bit, when the transmit FIFO (SPI_TFIFO) is empty, indicates that the last word in the transmit FIFO has been re-sent as transmit data. Alternately, it indicates that zero has been sent as transmit data.	
		0	No status
		1	Transmit underrun occurred
4 (R/W1C)	ROR	Receive Overrun Indication. The SPI_STAT.ROR bit, when the receive FIFO (SPI_RFIFO) is full, indicates that a word in the receive FIFO has been overwritten with incoming receive data. Alternately, it indicates that incoming receive data has been discarded.	
		0	No status
		1	Receive overrun occurred
2 (R/NW)	TUWM	Transmit Urgent Watermark Breached. The SPI_STAT.TUWM bit indicates that the transmit urgent watermark (SPI_TXCTL.TUWM) has been reached. This condition is cleared when the transmit FIFO fills enough to reach the transmit regular watermark (SPI_TXCTL.TRWM).	
		0	Tx regular watermark reached
		1	Tx urgent watermark breached
1 (R/NW)	RUWM	Receive Urgent Watermark Breached. The SPI_STAT.RUWM bit indicates that the receive urgent watermark (SPI_RXCTL.RUWM) has been reached. This condition is cleared when the receive FIFO empties enough to reach the receive regular watermark (SPI_RXCTL.RRWM).	
		0	Rx regular watermark reached
		1	Rx urgent watermark breached
0 (R/NW)	SPIF	SPI Finished. The SPI_STAT.SPIF bit indicates that a single word transfer is complete.	
		0	No status
		1	Completed single word transfer

Transmit FIFO Data Register

The SPI_TFIFO register has an interface to the transmit shift register in the SPI and has an interface to the processor's data buses. The top level of the buffer is visible to programs as the 32-bit SPI_TFIFO register, but the size (number of word locations) of the transmit FIFO is actually flexible with transfer word size. The size of the transmit FIFO is 8 if the word size is 8-bit, or the size is 4 if the word size is 16-bit, or the size is 2 if the word size is 32-bit.

Both masters and slaves may stop or stall transmit transfers based on FIFO status. When the transmit FIFO is empty, the SPI master stops initiating new transfers on the SPI if SPI_TXCTL.TTI is enabled. A slave may stall the SPI interface when the content of the FIFO crosses the selected watermark. If data transmit requests continue after SPI_TFIFO is empty, the data sent from the transmit FIFO is invalid, and the SPI indicates this condition with transmit underrun (SPI_STAT.TUR). This condition is possible when SPI_TXCTL.TTI =0 and SPI_TXCTL.TEN =1 for a master, or for a slave that does not exercise flow control.

Note that the transmit FIFO is reset (cleared) when the SPI is disabled after being enabled.

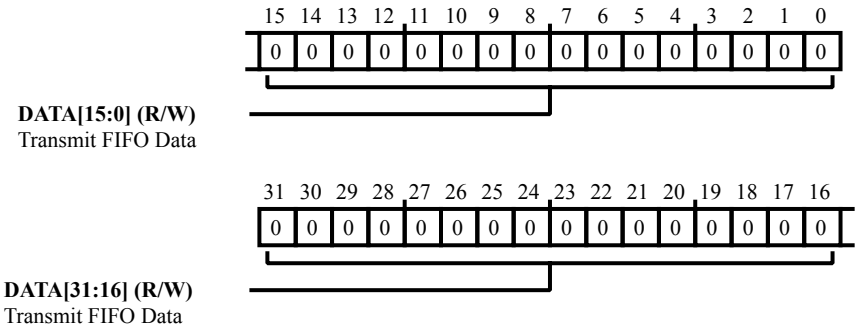


Figure 28-26: SPI_TFIFO Register Diagram

Table 28-22: SPI_TFIFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Transmit FIFO Data. The SPI_TFIFO.DATA bit field contains the FIFO transmit data.

Transmitted Word Count Register

The SPI_TWC register holds a count of the number of words remaining to be transmitted by the SPI. To start the decrement of the word count in SPI_TWC, enable the transmit word counter (SPI_TXCTL.TWCEN =1). The SPI uses the word count to control the duration of transfers and to signal the completion of a burst of transfers with the transmit finish interrupt request. In DMA mode, the SPI uses the SPI_TWC to ensure that the number of frames transmitted during a DMA transfer is equal to the number of words programmed in the DMA channel controller. The values programmed into the SPI_TWC registers should match the word count in the DMA configuration. The SPI_TWC maintains the number of frames to be transmitted in a transfer. The SPI_TWC should only be changed when the counter is disabled.

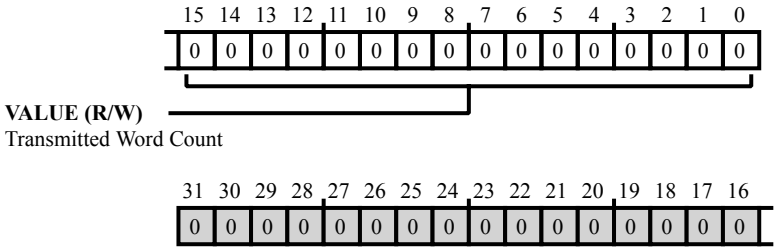


Figure 28-27: SPI_TWC Register Diagram

Table 28-23: SPI_TWC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count. The SPI_TWC.VALUE bits hold the transmit transfer word count.

Transmitted Word Count Reload Register

The SPI_TWCR register holds the transmit word count value that the SPI loads into the SPI_TWC register when the transfer count decrements to zero. To prevent the SPI from reloading the counter, use zero for the reload count value. The SPI_TWCR should only be changed when the counter is disabled.

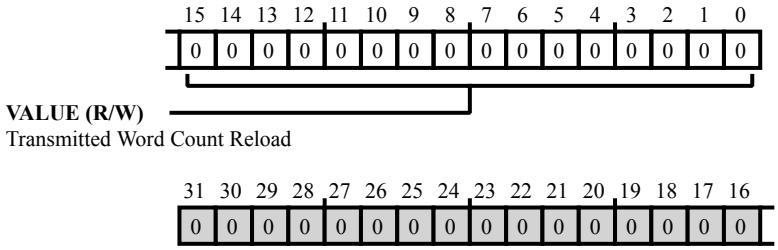


Figure 28-28: SPI_TWCR Register Diagram

Table 28-24: SPI_TWCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Transmitted Word Count Reload. The SPI_TWCR.VALUE bits hold the transmit transfer word count reload value.

Transmit Control Register

The SPI_TXCTL register enables the SPI transmit channel, initiates transmit transfers, and configures SPI_TFIFO buffer watermark settings.

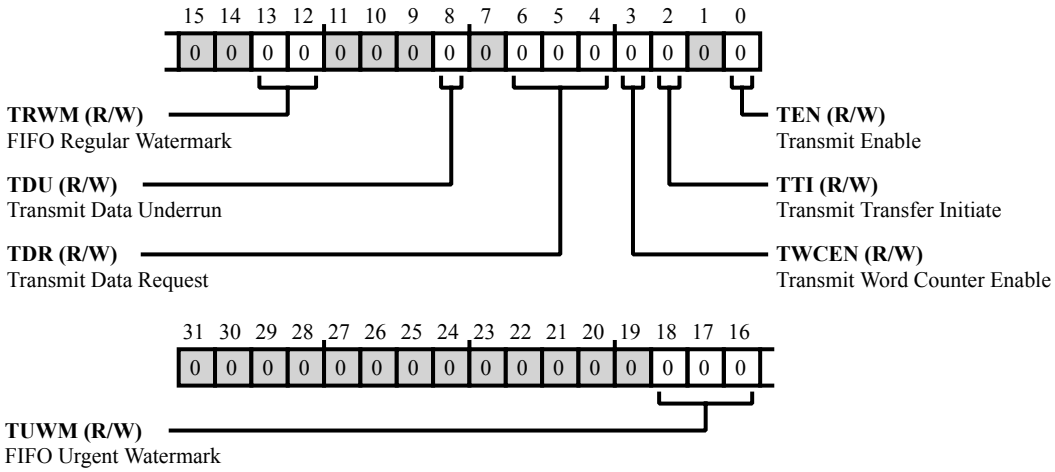


Figure 28-29: SPI_TXCTL Register Diagram

Table 28-25: SPI_TXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18:16 (R/W)	TUWM	FIFO Urgent Watermark. The SPI_TXCTL.TUWM bits select the transmit FIFO (SPI_TFIFO) watermark level for urgent data bus requests. The SPI also uses this watermark level for generation of the SPI_ILAT.TUWM interrupt request. When an urgent SPI_TFIFO watermark is enabled with SPI_TXCTL.TUWM, the SPI_TXCTL.TRWM selection is used as the deassertion condition for any SPI_ILAT.TUWM interrupt requests that are latched.	
		0	Disabled
		1	25% empty TFIFO
		2	50% empty TFIFO
		3	75% empty TFIFO
		4	Empty TFIFO

Table 28-25: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13:12 (R/W)	TRWM	FIFO Regular Watermark. The SPI_TXCTL.TRWM bits select the transmit FIFO (SPI_TFIFO) watermark level for regular data bus requests. When an urgent SPI_TFIFO watermark is enabled with SPI_TXCTL.TUWM, the SPI_TXCTL.TRWM selection is used as the deassertion condition for any SPI_ILAT.TUWM interrupt requests that are latched.	
		0	Full TFIFO
		1	TFIFO less than 25% empty
		2	TFIFO less than 50% empty
		3	TFIFO less than 75% empty
8 (R/W)	TDU	Transmit Data Underrun. The SPI_TXCTL.TDU bit selects handling for transmit data requests when the transmit buffer (SPI_TFIFO) is empty. If enabled and SPI_TFIFO is empty, the SPI transmits zero as data. If disabled and SPI_TFIFO is empty, the SPI transmits the last word in the buffer as data.	
		0	Send last word when SPI_TFIFO is empty
		1	Send zeros when SPI_TFIFO is empty
6:4 (R/W)	TDR	Transmit Data Request. The SPI_TXCTL.TDR bits select transmit FIFO (SPI_TFIFO) watermark conditions that direct the SPI to generate a transmit status interrupt request.	
		0	Disabled
		1	Not full TFIFO
		2	25% empty TFIFO
		3	50% empty TFIFO
		4	75% empty TFIFO
		5	Empty TFIFO
3 (R/W)	TWCEN	Transmit Word Counter Enable. The SPI_TXCTL.TWCEN bit enables the decrement of the transmit word count (SPI_TWC) register when the count is not zero and SPI_TXCTL.TTI is enabled. Enabling SPI_TXCTL.TWCEN prevents transmit underrun errors from occurring. The SPI_TXCTL.TWCEN bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable

Table 28-25: SPI_TXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	TTI	Transmit Transfer Initiate. The SPI_TXCTL.TTI bit enables initiation of transmit transfers if the transmit FIFO (SPI_TFIFO) is not empty. The bit also enables this initiation if SPI_TWC is not zero when SPI_TXCTL.TWCEN is enabled. Enabling SPI_TXCTL.TTI prevents transmit underrun errors from occurring. The SPI_TXCTL.TTI bit is valid only when the SPI is a master.	
		0	Disable
		1	Enable
0 (R/W)	TEN	Transmit Enable. The SPI_TXCTL.TEN bit enables SPI transmit channel operation.	
		0	Disable
		1	Enable

29 Serial Port (SPORT)

The programmable serial ports (SPORTs) support various protocols for serial data communication and provide a glueless hardware interface to many industry-standard data converters and codecs. They have high data rates and dual half-duplex datapaths and are ideal for establishing a direct serial connection among two or more processors in a multiprocessor system, as many processors provide compatible serial interfaces.

The SPORT top module consists of two half SPORTs with identical functionality and programming requirements. Each half SPORT can be independently configured as either a transmitter or receiver and can be coupled with the other half SPORT within the same SPORT top module. Further, each half SPORT provides two synchronous half-duplex data lines to double the total supported throughput. As such, a single SPORT top module can be used to provide up to four unidirctional or up to two full-duplex data streams. Further channels are possible as well, but utilization of multiple SPORT top modules is required, thus requiring external connections to provide a common time base.

Features

An individual SPORT top module consists of two independently configurable SPORT halves with identical functionality. These SPORT halves offer the following features:

- Up to two bidirectional data lines - each half SPORT supports up to two transmit or receive channels, thus allowing two unidirectional streams into or out of each half SPORT and providing greater flexibility for serial communications. If full-duplex functionality is desired, two SPORT halves can be combined to enable dual-stream bidirectional communication.
- Six operating modes:
    1. Standard DSP serial mode
    2. I^2S mode
    3. Left-Justified mode
    4. Right-Justified mode
    5. Multichannel (TDM) mode
    6. Packed mode

- Supports internally or externally generated clock.
- Support for both even and odd SCLK to SPORT clock (SPORT_CLK) ratios. If both data lines of a half SPORT are active, the maximum throughput is 2 x SPORT_CLK bps.
- Configurable rising or falling edge of the SPORT_CLK for driving and sampling data and frame syncs.
- Gated clock mode support for internallly or externally generated clocks in DSP serial mode and stereo modes (left-justified and I^2S mode).
- Supports frameless operation.
- Supports internally or externally generated frame sync signals.
- Programmable frame sync polarity.
- Programmable frame sync timing (synchronous to data or 1 SPORT clock in advance of it).
- Detection of prematurely received external frame syncs (with optional interrupt request generation).
- Programmable level-/edge-sensitivity for external frame syncs.
- Programmable (4–32-bit) data length, either in most significant bit (MSB) first or in least significant bit (LSB) first format, with optional sign-extension on received data.
- Optional 16-bit to 32-bit word packing (as receiver) and 32-bit to 16-bit word unpacking (as transmitter).
- Support for A-law and μ-law compression/decompression hardware companding, according to the G.711 specification, on transmitted/received words in all operating modes.
- Transmit underrun and receive overflow detection (with optional interrupt request generation).
- TDM mode transfers data on 128 contiguous channels from a stream of up to 1024 total channels (useful for H.100/H.110 and other telephony interfaces as a network communication scheme for multiple processors).
- Performs interrupt-driven, single word core transfers to and from on-chip or off-chip memory.
- Dedicated DMA channel for each SPORT half supporting autobuffer (for a repeated, identical range of transfers) and numerous descriptor-based (individual or repeated ranges of transfers with differing DMA parameters) modes.
- Master and slave trigger functionality.
- Unique transfer finish interrupt (TFI) signaling when the last transmit word is fully out of the transmit shift register.
- Multiplexer to internally connect critical timing signals between SPORT halves.

Signal Descriptions

Each half SPORT module has five dedicated signals, as described in the ***SPORT Signal Descriptions*** table. The actual pin name varies with different SPORT halves. Individual SPORT halves do not share any of its signals across

the pair that comprises the SPORT top module; however, it is possible to connect the clock and frame sync signals between the SPORT half pair, as explained in the Multiplexer Logic section.

All of the SPORT signals are multiplexed on the PORT pins, possibly sharing functionality with other peripherals on the device. By default, the PORT pins are in GPIO mode and must be reconfigured for SPORT functionality by setting the appropriate bits in the PORT_FER and PORT_MUX registers. Consult the processor datasheet for details regarding which ports the SPORT signals are available on, and be sure to configure the PORT_MUX register before the PORT_FER register.

Table 29-1: SPORT Signal Descriptions

Internal Node	Direction	Description
SPORTx_CLK	I/O	Transmit or receive serial clock. Data and frame syncs are driven or sampled on this clock's edges. This signal can be either internally or externally generated.
SPORTx_FS	I/O	Transmit or receive frame sync. The frame sync pulse initiates shifting of serial data. This signal is either internally or externally generated.
SPORTx_D0	I/O	Primary transmit or receive data channel. This signal can be configured as an output to transmit serial data or as an input to receive serial data.
SPORTx_D1	I/O	Secondary transmit or receive data channel. This signal can be configured as an output to transmit serial data or as an input to receive serial data.
SPORTx_TDV	O	Multichannel transmit data valid. This signal is only active in multichannel transmit mode and is asserted during enabled slots, as defined by the channel selection registers (SPORT_CS0_A through SPORT_CS3_B).

The data channel signals are transmit signals when the serial port is configured in transmit mode (SPORT_CTL_A.SPTRAN = 1). They are receive signals when the serial port is configured in receive mode (SPORT_CTL_A.SPTRAN = 0). The following sections further describe the SPORT signals.

NOTE: These sections explicitly refer to the registers associated with half SPORT A, but the same concepts also apply to half SPORT B.

Serial Clock

The serial port clock (SPORT_ACLK) is either a receive serial clock or a transmit serial clock, depending on the transfer direction (SPORT_CTL_A.SPTRAN), governing when the data bits are serially shifted into or out of the SPORT and when the frame sync signal is driven (in internal frame sync mode) or sampled (in external frame sync mode). It can be internally generated from the processor's system clock (SCLK) or externally provided. If the half SPORT is configured in internal clock mode (SPORT_CTL_A.ICLK= 1), then the SPORT_DIV_A.CLKDIV field specifies the divisor applied to SCLK to generate the SPORT clock. As it is a 16-bit divisor, a wide range of serial clock rates is possible. Use the following equation to calculate the serial clock frequency:

SPORT_ACLK = [SCLK ÷ (SPORT_DIV_A.CLKDIV + 1)]

From this, the following equation can be used to determine the value of SPORT_DIV_A.CLKDIV, given the SCLK frequency and the desired frequency of the SPORT clock:

SPORT_DIV_A.CLKDIV= [(SCLK ÷ SPORT_ACLK) - 1]

The half SPORT also supports a 1:1 SPORT_ACLK to SCLK ratio (per the equations above, program the clock divisor field to zero). In this case, the resulting SPORT clock frequency is equal to SCLK.

NOTE: Be careful not to exceed the maximum SPORT_ACLK frequency specified in the processor datasheet.

In certain operating modes, the SPORT can be configured to generate a gated clock, which is active only during valid data. In some applications, a SPORT uses it to generate a general-purpose clock in the system. In this case, enable the SPORT with the appropriate SPORT_DIV_A.CLKDIV divisor field in internal clock mode.

If a SPORT is configured in external clock mode (SPORT_CTL_A.ICLK= 0), the serial clock is an input signal, thus making the SPORT operate in slave mode. In this mode, the SPORT_DIV_A.CLKDIV is irrelevant and is ignored. The optional loopback capability provided by the internal SPORT multiplexer (SPMUX) block allows the slave SPORT to use the serial clock from the neighboring SPORT half in the same SPORT top module.

An externally-supplied serial clock does not need to be synchronous with the processor clocks. Further, the external clock can be a gated clock, but it must comply with the requirements described in the Gated Clock Mode section.

Refer to the product datasheet for exact AC timing specifications.

Frame Sync

The SPORT frame sync (SPORT_AFS) signal is either a receive frame sync or a transmit frame sync, depending on the transfer direction (SPORT_CTL_A.SPTRAN), which is used to determine the start of a new word or frame. When this signal goes active, the serial port starts serially shifting data into or out of the SPORT. The frame sync signal can be internally generated based on its serial clock (SPORT_ACLK) or externally provided, as configured by the SPORT_CTL_A.IFS bit.

If the half SPORT is configured to generate frame syncs (SPORT_CTL_A.IFS= 1), then the SPORT_DIV_A.FSDIV field specifies the divisor used to generate the periodic SPORT_AFS signal from the SPORT clock. As this is a 16-bit divisor, a wide range of frame sync rates to initiate periodic transfers is possible. Whether the serial clock is internally or externally generated, this divisor is a count of SPORT clock cycles between frame sync pulses, the formula for which is:

Number of serial clocks between frame syncs = (SPORT_DIV_A.FSDIV + 1)

From this, the following equation can be used to determine the value of SPORT_DIV_A.FSDIV, given the serial clock frequency and the desired frame sync frequency:

SPORT_DIV_A.FSDIV = [(SPORT_ACLK ÷ SPORT_AFS) – 1]

The frame sync is continuously active when SPORT_DIV_A.FSDIV= 0. The value of SPORT_DIV_A.FSDIV cannot be less than the serial word length minus one (the value of the SPORT_CTL_A.SLEN bit field). Failure to adhere to this guideline can cause an external device to abort the current operation or cause other unpredictable results.

NOTE: After enabling the SPORT, the first internal frame sync appears after a delay of SPORT_DIV_A.FSDIV + 3 SPORT clock cycles.

If a SPORT is configured for external frame syncs (SPORT_CTL_A.IFS = 0), then SPORT_AFS is an input signal and the SPORT_DIV_A.FSDIV field of the SPORT_DIV_A register is irrelevant and ignored. By default, this external signal is level-sensitive, but it can be configured as an edge-sensitive signal by setting the SPORT_CTL_A.FSED bit. The frame sync is expected to be synchronous with the serial clock. If not, it must meet the timing requirements that appear in the datasheet.

The SPORT can be used as a counter for dividing an external clock to generate periodic pulses or periodic interrupts. To do so, enable the SPORT with the appropriate SPORT_DIV_A.FSDIV divisor field with the SPORT configured for an external clock and internal data-independent frame syncs.

In some of the operating modes, the SPORT can be programmed to treat the frame sync signal as an optional signal by clearing the SPORT_CTL_A.FSR bit. Even with this bit cleared, the SPORT requires a single frame sync assertion to start the continuous transfers, after which it is ignored (for externally supplied frame syncs) or not generated (for internally-generated frame syncs). Characteristics of the frame sync depend on the settings in the SPORT control registers and the operating mode of the SPORT. For more information, refer to the SPORT_CTL_A register.

Data Signals

Each half SPORT has two bidirectional data lines known as the primary (SPORT_AD0) and secondary (SPORT_AD1) data channels. Both of the data lines can be configured as either transmitters or receivers using the SPORT_CTL_A.SPTRAN bit, thus permitting dual unidirectional data streams to increase the data throughput of the SPORT.

NOTE: Configuring one transmit data channel and one receive data channel on a single half SPORT is not supported.

The primary and secondary data lines can be individually enabled or disabled using the SPORT_CTL_A.SPENPRI and the SPORT_CTL_A.SPENSEC bits, respectively. However, if using both, enable or disable them concurrently. These data lines operate in a synchronous manner (sharing a clock and frame sync) but have separate datapaths with unique data buffers, shift registers and optional companding logic. All of the SPORT control settings are common for both channels, but the single DMA channel per half SPORT serves both the primary and secondary data channels.

When a SPORT is configured in multichannel transmit mode, the data pins three-state during inactive channel slots, thus allowing multiple transmitters to operate on the same bus with different active channels.

See the Architectural Concepts section for more details about data transfer operation.

Transmit Data Valid Signal

The transmit data valid (SPORT_ATDV) signal is available only in transmit multichannel modes (including packed mode). It is driven active during enabled multichannel slots, and it is driven inactive during the disabled channels. In other words, the SPORT_ATDV signal is active when data is being driven to the data pins and inactive when the

data pins are being three-stated. As such, the SPORT_ATDV signal can serve as an output-enable signal for the data transmit pin.

Functional Description

The following sections provide general information about the functionality of the processor's serial ports:

- Architectural Concepts
- Data Types and Companding
- Transmit Path
- Receive Path

CM41X_M4 SPORT Register List

The Serial Port (SPORT) controller, with its range of clock and frame synchronization options, supports a variety of serial communication protocols and provides a glueless hardware interface to many industry-standard data converters and CODECs. Each SPORT has two independent halves (A and B), and each half contains two channels (primary and secondary). A set of registers governs SPORT operations. For more information on SPORT functionality, see the SPORT register descriptions.

Table 29-2: CM41X_M4 SPORT Register List

Name	Description
SPORT_CS0_A	Half SPORT 'A' Multichannel 0-31 Select Register
SPORT_CS0_B	Half SPORT 'B' Multichannel 0-31 Select Register
SPORT_CS1_A	Half SPORT 'A' Multichannel 32-63 Select Register
SPORT_CS1_B	Half SPORT 'B' Multichannel 32-63 Select Register
SPORT_CS2_A	Half SPORT 'A' Multichannel 64-95 Select Register
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multichannel 96-127 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_CTL_B	Half SPORT 'B' Control Register
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_ERR_A	Half SPORT 'A' Error Register

Table 29-2: CM41X_M4 SPORT Register List (Continued)

Name	Description
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MCTL_A	Half SPORT 'A' Multichannel Control Register
SPORT_MCTL_B	Half SPORT 'B' Multichannel Control Register
SPORT_MSTAT_A	Half SPORT 'A' Multichannel Status Register
SPORT_MSTAT_B	Half SPORT 'B' Multichannel Status Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register

CM41X_M4 SPORT Interrupt List

Table 29-3: CM41X_M4 SPORT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
137	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
138	SPORT0_B_STAT	SPORT0 Channel B Status	Level	

29.3.3 SPORT Trigger List

CM41X_M4 SPORT DMA Channel List

Table 29-4: CM41X_M4 SPORT DMA Channel List

DMA ID	DMA Channel Name	Description
DMA10	SPORT0_A_DMA	SPORT0 Channel A DMA
DMA11	SPORT0_B_DMA	SPORT0 Channel B DMA

Block Diagram

Each SPORT top module consists of two separate blocks, known as half SPORTs (HSPORT) A and B, each with identical functionality and programming models. The ***Half Serial Port Block Diagram*** shows a detailed block diagram of a half SPORT.

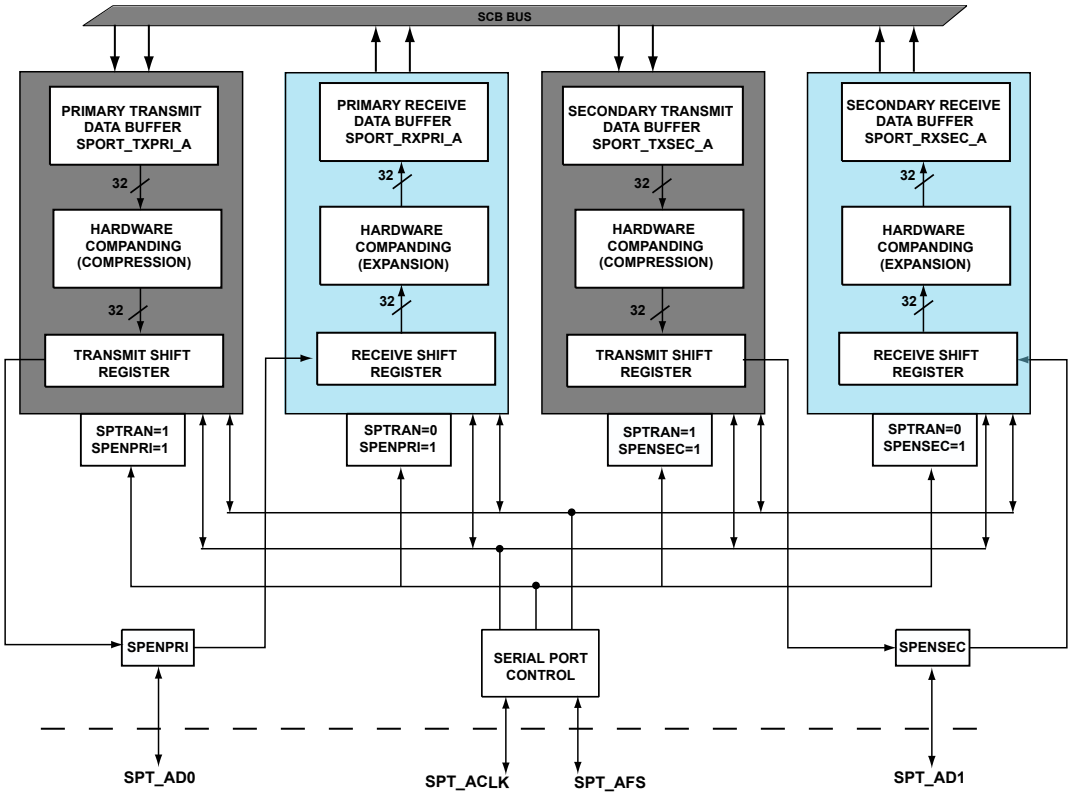


Figure 29-1: Half Serial Port Block Diagram

Architectural Concepts

Each half SPORT (HSPORT) block has its own set of control registers and data buffers, grouped per SPORT module. The HSPORT A and B blocks can be independently configured as either a transmitter or a receiver, with the option to be coupled together internally within the single SPORT top module. Each HSPORT also supports two synchronous bidirectional datapaths, referred to as the primary (`D0`) and secondary (`D1`) data lines, as shown in the ***Top-Level SPORT Diagram*** figure.

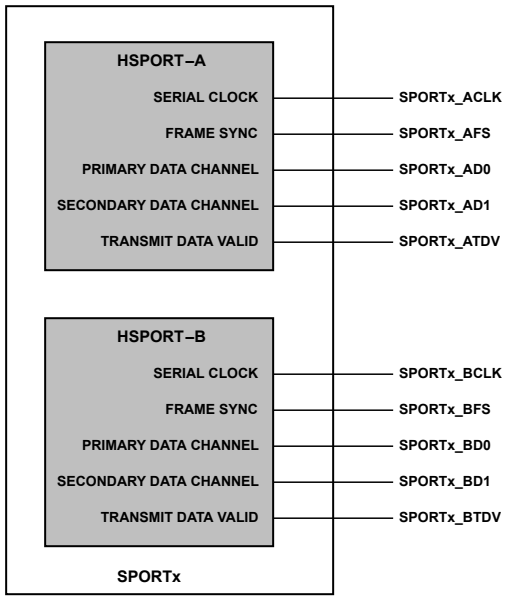


Figure 29-2: Top-Level SPORT Diagram

The SPORT_CTL_A.SPTRAN bit controls the direction for both datapaths of the HSPORT. Depending on whether the HSPORT is a transmitter or a receiver, the pair of data signals respectfully transmit or receive data bits synchronously. The dual data signals of each HSPORT cannot transmit and receive the data simultaneously in support of full-duplex operation, however, two HSPORTs can be combined to achieve this.

Serial communications are synchronized to the serial clock signal, where a valid clock pulse must accompany each data bit. Each HSPORT can take its clock from an external source or internally generate it from the processor's system clock using the SPORT_DIV_A.CLKDIV clock divisor bit field. Both primary and secondary data channels shift data based on the SPORT_CLK rate and the clock polarity defined by the SPORT_CTL_A.CKRE bit.

In addition to the serial clock signal, a frame synchronization signal is used to signify the beginning of an individual data word or a multichannel data stream (block of words). Each SPORT can take the frame sync signal from an external source or generate it (SPORT_FS), depending on the SPORT_CTL_A.IFS bit. An internally generated frame sync is derived from the SPORT clock using the SPORT_DIV_A.FSDIV divisor field. Both primary and secondary datapaths start shifting data either synchronous to or one serial clock in advance of detecting/generating a valid frame sync signal, as determined by the SPORT_CTL_A.LAFS bit. Various communication protocols for serial data can be emulated according to the frame sync format, and all frame sync options are available whether the signal is generated internally or externally.

NOTE: These SPORTs are not UARTs and cannot communicate with an RS-232 device or any other asynchronous communications protocol.

Multiplexer Logic

The SPORT multiplexing block (SPMUX) is situated between the SPORT hardware block and the processor's pin multiplexing logic. It allows the flexibility to route and share the clock and frame sync signals between the HSPORT A and B halves within each SPORT top module, which can double the data throughput (if both SPORT halves are transmitters or both are receivers) or provide full-duplex capabilities (if one HSPORT is a receiver and the other a

transmitter) without the need to allocate pins for the peripheral or make physical connections outside the processor. The SPORT_CTL2_A register is used to configure this loopback feature.

NOTE: Throughout this section, HSPORT A is used as a reference, but all concepts also apply to HSPORT B.

The multiplexing depends on the configuration of the SPORT_CTL_A.IFS and SPORT_CTL_A.ICLK bits, and the SPORT_CTL2_A.CKMUXSEL and SPORT_CTL2_A.FSMUXSEL bit settings control the multiplexing. The ***Frame Sync Combinations*** and ***Clock Combinations*** tables show the valid combinations for the bit settings.

NOTE: All other settings are illegal. However, hardware does not check or prevent the illegal settings. Ensure that programs use only legal combinations.

The column headers in the ***Frame Sync Combinations*** table are defined as follows:

- FS Combination = Frame sync combination, referenced in the notes that follow the ***Clock Combinations*** table.
- HSA_IFS = the setting of the SPORT_CTL_A.IFS configuration bit.
- HSB_IFS = the setting of the SPORT_CTL_B.IFS configuration bit.
- FSAMUX = the setting of the SPORT_CTL2_A.FSMUXSEL configuration bit.
- FSBMUX = the setting of the SPORT_CTL2_B.FSMUXSEL configuration bit.

The Routing column in the ***Frame Sync Combinations*** table defines how the signals are used between the SPORT halves and which pin is used for the frame sync (whether it is an input or an output). Within the column, the inequality characters (≤ and ≥) are used to show the direction of the signal, and the following abbreviations are used (where x = A or B):

- HSx_FI = Frame sync input signal, provided by an external device or the complementing HSPORT.
- HSx_FO = Frame sync output signal.
- SPx_FS = HSPORT's frame sync pin, where the signal is either:
  - provided by an external source and distributed to both HSPORT frame sync signals, or
  - internally generated by one HSPORT and routed to both the pin and to the complementary HSPORT frame sync signal.

Table 29-5: Frame Sync Combinations

FS Combination	HSA_IFS	HSB_IFS	FSAMUX	FSBMUX	Routing
1	0	0	0	0	Native FS Operation
2	0	1	0	0	Native FS Operation
3	1	0	0	0	Native FS Operation
4	1	1	0	0	Native FS Operation

Table 29-5: Frame Sync Combinations (Continued)

FS Combination	HSA_IFS	HSB_IFS	FSAMUX	FSBMUX	Routing
5	0	0	1	0	HSA_FI ≤ SPB_FS; HSB_FI ≤ SPB_FS
6	0	1	1	0	HSA_FI ≤ HSB_FO ≥ SPB_FS
7	0	0	0	1	HSB_FI ≤ SPA_FS; HSA_FI ≤ SPA_FS
8	1	0	0	1	HSB_FI ≤ HSA_FO ≥ SPA_FS

The column headers in the ***Clock Combinations*** table are defined as follows:

- CLK Combination = Clock combination, referenced in the notes that follow the table.
- HSA_ICLK = the setting of the `SPORT_CTL_A.ICLK` configuration bit.
- HSB_ICLK = the setting of the `SPORT_CTL_B.ICLK` configuration bit.
- CKAMUX = the setting of the `SPORT_CTL2_A.CKMUXSEL` configuration bit.
- CKBMUX = the setting of the `SPORT_CTL2_B.CKMUXSEL` configuration bit.

The Routing column in the ***Clock Combinations*** table defines how the signals are used between the SPORT halves and which pin is used for the serial clock (whether it is an input or an output). Within the column, the inequality characters (≤ and ≥) are used to show the direction of the signal, and the following abbreviations are used (x = A or B):

- HSx_CI = Serial clock input signal, provided by an external device or the complementing HSPORT.
- HSx_CO = Serial clock output signal.
- SPx_CLK = HSPORT's serial clock pin, where the signal is either:
  - provided by an external source and distributed to both HSPORT serial clock signals, or
  - internally generated by one HSPORT and routed to both the pin and to the complementary HSPORT serial clock signal.

Table 29-6: Clock Combinations

CLK Combination	HSA_ICLK	HSB_ICLK	CKAMUX	CKBMUX	Routing
9	0	0	0	0	Native CLK Operation
10	0	1	0	0	Native CLK Operation
11	1	0	0	0	Native CLK Operation
12	1	1	0	0	Native CLK Operation

Table 29-6: Clock Combinations (Continued)

CLK Combination	HSA_ICLK	HSB_ICLK	CKAMUX	CKBMUX	Routing
13	0	0	1	0	HSA_CI ≤ SPB_CLK; HSB_CI ≤ SPB_CLK
14	0	1	1	0	HSA_CI ≤ HSB_CO ≥ SPB_CLK
15	0	0	0	1	HSB_CI ≤ SPA_CLK; HSA_CI ≤ SPA_CLK
16	1	0	0	1	HSB_CI ≤ HSA_CO ≥ SPA_CLK

The following is a comprehensive list of the legal combinations for the above described frame sync and clock multiplexing configurations:

- FS Combinations 1–4 are compatible with all CLK Combinations (9–16)
- CLK Combinations 9–12 are compatible with all FS Combinations (1–8)
- FS Combination 5 is only compatible with CLK Combination 13 (and vice versa)
- FS Combination 6 is only compatible with CLK Combination 14 (and vice versa)
- FS Combination 7 is only compatible with CLK Combination 15 (and vice versa)
- FS Combination 8 is only compatible with CLK Combination 16 (and vice versa)

NOTE: Program only the `SPORT_CTL2` register of the HSPORT that is accepting the signal from the other HSPORT. However, be sure to set the `SPORT_CTL_A.CKRE` and `SPORT_CTL_A.LFS` polarity bits to have identical settings between the HSPORTs when making internal connections via the SPMUX block.

Data Types and Companding

The SPORT uses the data type select field `SPORT_CTL_A.DTYPE` bit to specify one of the four data formats supported by serial ports. These formats apply to any of the operating modes of serial port.

Table 29-7: Data Type Bit Field Settings

DTYPE field	SPORT Receiver	SPORT Transmitter
00	Right-justify, zero-fill unused most significant bits	Normal operation
01	Right-justify, sign-extend unused most significant bits	Reserved
10	Expand using μ-law	Compress using μ-law
11	Expand using A-law	Compress using A-law

These formats apply to data words loaded into the SPORT transmit or receive data buffers. The first two data formats (`00` and `01` values of `SPORT_CTL_A.DTYPE`) are applicable only when SPORT is configured as receiver.

When SPORT is configured as transmitter, only the significant bits are transmitted (per the field defined in control register). Therefore, the transmit data buffers are not actually zero-filled or sign-extended.

The other two data formats enable the companding logic on the transmit or receive path. Companding (compressing or expanding) is the process of logarithmically encoding and decoding data to minimize the number of bits sent. The SPORTs of the processor support the two most widely used companding algorithms, A-law, and µ-law. The algorithms are performed according to the CCITT G.711 specification.

If selected, companding applies to both the enabled data channels. When enabled as SPORT transmitter, writes to transmit buffer make the content compressed to 8 bits according to algorithm selected. (The content is zero filled to the width of the transmit word.) Similarly, if configured in receive mode, the 8 bits in the receive data buffers expand in right-justified, zero fill format per the algorithm selected. If companding is enabled in multichannel mode, it applies to all the active channels.

The compression for transmit data requires a minimum word length of 8 for proper function. If `SPORT_CTL_A.SLEN` is less than 7, then expansion does not work correctly. Also, if the data value is greater than 13-bit A-law or 14-bit µ-law maximum, it automatically compresses to the maximum value.

NOTE: The processor companding logic supports in-place companding feature. So, companding can be used for debug without enabling SPORT.

Companding as a Function

Since the values in the transmit and receive buffers are companded in place, the SPORT can use the companding hardware without transmitting or receiving data, which can be useful during testing or debugging. For companding to execute properly, program the SPORT registers prior to loading data values into the SPORT buffers.

To compress data in place without transmitting, use the following procedure:

1. Set the SPORT as a transmitter (`SPORT_CTL_A.SPTRAN` = 1) with both primary and secondary data channels disabled (`SPORT_CTL_A.SPENPRI` = `SPORT_CTL_A.SPENSEC`= 0).
2. Enable companding in the `SPORT_CTL_A.DTYPE` field.
3. Write a 32-bit data word to the transmit buffer.
4. Wait two system clock cycles to allow the SPORT companding hardware to reload the transmit buffer with the companded value. Any instructions that do not access the transmit buffer can be used to cause this delay.
5. Read the 8-bit compressed value from the transmit buffer.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer.

Transmit Path

When the `SPORT_CTL_A.SPTRAN` control bit is set, the HSPORT is in transmit mode. Primary and secondary transmit data paths are then enabled using the `SPORT_CTL_A.SPENPRI` and `SPORT_CTL_A.SPENSEC` bits, respectively. The primary and secondary datapaths are unique and identical, each including its own transmit data buffer, optional companding logic, and transmit shift register.

The data buffer on the primary transmit data path is SPORT_TXPRI_A, and the data buffer on the secondary transmit data path is SPORT_TXSEC_A. The transmit data buffer and output shift register form a FIFO type of structure. When packing is disabled (SPORT_CTL_A.PACK = 0), the SPORT can hold as many as three data words. If packing is enabled (SPORT_CTL_A.PACK = 1), the serial port can hold two packed data words at any time.

The transmit data for primary and secondary channels is written to the SPORT_TXPRI_A and SPORT_TXSEC_A transmit data buffers, respectively. The transmit data buffers can be accessed in core mode through the peripheral bus or in DMA mode through the DMA bus. When a SPORT is configured in transmit mode, the receive paths are deactivated and do not respond to serial clock or frame sync signals. Because the receive data buffers and receive shift registers are also deactivated, reading from an empty and inactive receive data buffer can cause the core to hang indefinitely.

NOTE: Be sure to avoid accesses to inactive data buffers. Such accesses can cause unpredictable SPORT behavior or a hang condition and are not reported in any status register.

This data can optionally be compressed in hardware according to the selected algorithm (μ-law or A-law) and then automatically transferred to the transmit shift register. The shift register, clocked by the SPORT_ACLK signal, then serially outputs this data on the SPORT_AD0 and/or SPORT_AD1 pins (if both are enabled, these output data bits are transmitted synchronously). If the SPORT uses a framing signal, the SPORT_AFS signal indicates the start of the serial word transmission.

When using DMA mode, a single DMA feeds the data buffers of the enabled channels (primary and/or secondary). When using both channels, interleave the data of these channels starting with the primary channel in the transmit buffer.

When the SPORT is configured in non-multichannel mode as a transmitter, the enabled SPORT data pins (SPORT_AD0 and/or SPORT_AD1) are always driven. When a SPORT channel is enabled, data from the transmit data buffer is loaded into the transmit shift register. The shift register then immediately latches the first bit of data (either the LSB or MSB, depending on the SPORT_CTL_A.LSBF configuration bit) and drives it to the respective data pin such that it is ready when the frame sync signal asserts. Similarly, if the frame sync period exceeds the serial word length, then the data pins are driven with the first bit of the next word for transmission immediately after the active word completes, and the outputs are held during the inactive serial clock cycles (clock cycles between frame sync pulses).

When the SPORT is configured in multichannel mode, the data pins are driven only during active transmit channels and are always three-stated during inactive channel slots.

The SPORT provides status of transmit data buffers and also error detection logic for transmit errors such as an underrun condition. See the Error Detection (Status) Interrupt section for more details.

Receive Path

When the SPORT_CTL_A.SPTRAN bit is cleared, the SPORT is in receive mode. Primary and/or secondary receive data paths can be enabled by setting the SPORT_CTL_A.SPENPRI and SPORT_CTL_A.SPENSEC configuration bits, respectively. These data paths are unique but identical, each with a receive shift register, optional companding logic, and a receive data buffer.

The data buffer on the primary receive path is SPORT_RXPRI_A, and the data buffer on the secondary receive path is SPORT_RXSEC_A. The receive data paths act like a three-deep (32-bit words) FIFO because they have two data registers plus an input shift register.

Upon enabling the SPORT data channels, the input shift register shifts in data bits on the SPORT_AD0 and/or SPORT_AD1 pins, synchronous to the SPORT clock signal. If the SPORT uses a framing signal, the SPORT_AFS signal indicates the beginning of the serial word (or frame) to be received. When an entire word is shifted into the primary and secondary channels, the data can be optionally expanded in hardware according to a selected algorithm (µ-law or A-law) and then automatically transferred to the SPORT_RXPRI_A and/or SPORT_RXSEC_A data buffers.

The receive data buffers can be read in core mode through the peripheral bus or in DMA mode through the DMA bus. When the SPORT uses DMA mode, a single DMA reads the data buffers of the enabled channels (primary and/or secondary) and interleaves them in memory beginning with the primary channel when both channels are enabled. When using both channels, software must de-interleave the data of these channels.

The SPORT provides the status of receive data buffers and also error detection logic for receive errors such as overflow. See the Error Detection (Status) Interrupt section for more details.

When a SPORT is configured in receive mode, the transmit paths are deactivated and do not respond to serial clock or frame sync signals. As the transmit data buffers and transmit shift registers in the data paths are also deactivated, programs must not try to access them.

NOTE: Be sure to avoid accesses to inactive data buffers. Such accesses can cause unpredictable SPORT behavior or a hang condition and are not reported in any status register.

Operating Modes and Options

The SPORT has a number of operating modes:

- Standard DSP Serial mode
- I^2S mode
- Left-Justified mode
- Right-Justified mode
- Multichannel (TDM) mode
- Packed I^2S mode

The SPORT halves within a SPORT top module can be independently configured in any of these operating modes unless they are coupled together using SPMUX logic, in which case they must be configured identically. Each half SPORT has its own set of control and data registers and is programmed similarly.

The ***Control Bits for SPORT Operating Modes*** table lists all the programmable configuration bits in the SPORT_CTL_A control register, which combine to determine the overall function and operating mode of the SPORT. The columns are arranged according to the setting of the SPORT_CTL_A.OPMODE bit that selects between standard DSP/multichannel modes and the various I²S modes, and the cell contents are defined as follows:

- Yes – bit is programmable for this mode of operation and may be written
- Reserved – bit is not programmable for this mode of operation and must not be written
- = value – bit must be set to this value to enable this mode of operation
- FUNCTION – indicates alternate function for this bit in this mode

NOTE: When changing operating modes, first clear the SPORT_CTL_A register before again writing the register with the new configuration settings.

Table 29-8: Control Bits for SPORT Operating Modes

Name (Bit #)	Standard DSP Serial Mode	I²S and Left-Justified Mode	Right-Justified Mode	Multichannel (TDM) Mode	Packed I²S Mode
SPENPRI (0)	Valid				
DTYPE (2:1)	Valid	Reserved		Valid	
LSBF (3)	Valid	Reserved		Valid	
SLEN (8:4)	Valid				
PACK (9)	Valid				
ICLK (10)	Valid				
OPMODE (11)	= 0	= 1	= 1	= 0	= 1
CKRE (12)	Valid				
FSR (13)	Valid	Reserved			
IFS (14)	Valid				
DIFS (15)	Valid			Reserved	
LFS (16)	Valid	L_FIRST/PLFS		Valid	L_FIRST/PLFS
LAFS (17)	Valid	OPMODE2	Valid	Reserved	
RJUST (18)	Reserved		= 1	Reserved	
FSED (19)	Valid	Reserved		Valid	Reserved
TFIEN (20)	Valid				
GCLKEN (21)	Valid		Reserved		

Table 29-8: Control Bits for SPORT Operating Modes (Continued)

Name (Bit #)	Standard DSP Serial Mode	I^2S and Left-Justified Mode	Right-Justified Mode	Multichannel (TDM) Mode	Packed I^2S Mode
SPENSEC (24)	Valid				
SPTRAN (25)	Valid				

Serial Word Length

The SPORT uses the `SPORT_CTL_A.SLEN` field to determine the word length of the serial data to transmit or receive. Each half SPORT can independently handle word lengths up to 32 bits, and the value that must be programmed to the `SPORT_CTL_A.SLEN` field is obtained from:

SLEN = Desired SPORT word length - 1

The minimal word length depends on the selected operating mode. Words smaller than 32 bits are right-justified in the transmit or receive buffers; however, data can be shifted in or out in MSB or LSB first format, as configured by the `SPORT_CTL_A.LSBF` bit. The received word can also be sign-extended or zero-filled when storing the data to processor memory, as governed by the `SPORT_CTL_A.DTYPE` bit.

The ***Data Lengths for SPORT Operating Modes*** table shows the range of valid word lengths for each of the supported SPORT operating modes.

Table 29-9: Data Lengths for SPORT Operating Modes

Mode	SPORT Word Length (SLEN+1)
Standard DSP Serial	4–32
I^2S	5–32
Left-Justified	5–32
Right-Justified	5–32
Multichannel (TDM)	5–32
Packed I^2S	5–32

NOTE: If the companding feature is enabled on the datapath, it limits the word length settings. See Data Types and Companding for more details about word lengths required for companding. If more than 32 bits per frame sync are required to transmit or receive, use the multichannel mode to spread the data across numerous continuous channels.

Clock Sample and Drive Edges

The SPORT uses two control signals to sample or drive the serial data:

1. Serial clock (`SPORT_ACLK`) - bit clock for the serial data.
2. Frame sync (`SPORT_AFS`) - divides the incoming data stream into frames.

These control signals can be internally generated or externally provided, as determined by the SPORT_CTL_A.ICLK and SPORT_CTL_A.IFS bit settings, respectively.

Data and frame syncs can be sampled on the rising or falling edges of the SPORT clock signal, as determined by the SPORT_CTL_A.CKRE bit. By default, the SPORT_CTL_A.CKRE = 0 setting configures the falling edge of the SPORT_ACLK signal as the sampling edge for receive data and externally supplied frame syncs. The receive data and frame syncs can be sampled on the rising edges of SPORT_ACLK when SPORT_CTL_A.CKRE = 1.

NOTE: The SPORT drives transmit data and internal frame sync signals on the opposite serial clock edge of the sampling edge. Be sure to select the same value for SPORT_CTL_A.CKRE for transmit and receive functions for any two HSPORTs that are connected together, and always verify the correct polarity for any external device connected to the SPORT.

The ***Frame Sync and Data Driven on Rising Edge*** figure provides an example of the drive and sample edges when two HSPORTs are connected together, each with SPORT_CTL_A.CKRE = 0. In this example, the HSPORT that is configured as the transmitter drives the serial clock and frame sync signals, and both HSPORTs are configured for early, active high frame syncs and a word length of eight bits.

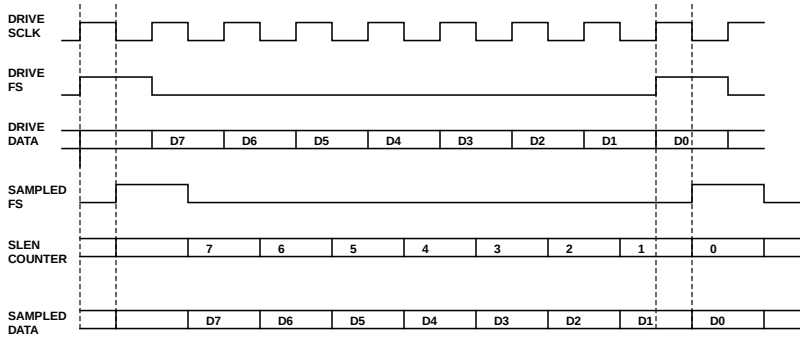


Figure 29-3: Frame Sync and Data Driven on Rising Edge

NOTE: The SCLK in the ***Frame Sync and Data Driven on Rising Edge*** figure is SCLK.

As shown, the transmitting HSPORT provides the clock and generates the frame sync. Because the HSPORTs are configured for early frame mode, the first bit of data is driven one serial clock later, with subsequent bits being driven on the following rising clock edges in the signal train. When the receiving HSPORT samples the frame sync signal (as indicated in the SAMPLED FS waveform), the SPORT_CTL_A.SLEN bit counter is loaded with the SPORT_CTL_A.SLEN setting, after which each SPORT_ACLK decrements the SPORT_CTL_A.SLEN counter until the full word is received. In this figure, the DRIVE FS and SAMPLED FS waveforms show the frame sync required for the next word in a continuous data stream. Note that it is legal for this frame sync to be sampled synchronous to the last bit of the previous data being sampled, as the early frame mode means that the data lags the frame sync by one serial clock cycle. If the frame sync were sampled as asserted before the D0 bit is sampled, the frame sync error is logged in the receiver's status register.

Since the transmitter drives the internally-generated frame sync and data on the rising edge of the serial clock, the receiver must use the falling edge to sample the externally-supplied frame sync and data.

Frame Sync Options

The following sections provide details regarding the programmable aspects of the SPORT frame sync signal. See the specific operating mode sections for additional information regarding frame sync requirements and behavior for each specific operating modes.

Data-Dependent versus Data-Independent Frame Syncs

By default, the generation of a frame sync signal is data-dependent:

- When the SPORT is configured as a transmitter (`SPORT_CTL_A.SPTRAN` = 1), an internally generated transmit frame sync is output when a new data word has been loaded into the channel transmit buffer of the SPORT (by either the core or the DMA engine).
- When the SPORT is configured as a receiver (`SPORT_CTL_A.SPTRAN` = 0), an internally-generated receive frame sync is output only when the receive data buffer is not full.

The data-independent frame sync option, enabled by setting the `SPORT_CTL_A.DIFS` bit, allows for the generation of a periodic framing signal, regardless of the status of the data buffers. When this bit is set, the frame sync output will be continuous and periodic, according to the setting of the `SPORT_DIV_A.FSDIV` field.

Support for Edge-Detected and Level-Sensitive Frame Syncs

The level-sensitive nature of frame sync signals operates well in a noise-free environment. However, if noise corrupts the signals coming into the SPORT, the internal logic can lose synchronization. For example, excessive noise on the frame sync signal may cause the frame sync to be sampled as inactive on the clock edge that it is intended to be synchronous to, but then be sampled at the correct active level one cycle later. Similarly, a noisy clock signal can cause an unintended clock edge, resulting in potential premature sampling of the frame sync signal being applied to the pin.

The ***Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync*** figure describes a scenario where an external frame sync signal is corrupted due to noise, causing the receiving SPORT module to incorrectly sample the signal. If the frame sync is driven on the rising edge of the serial clock at t_A, the SPORT would normally sample the signal on the falling edge of the serial clock at t_B. Due to the noise, however, the SPORT misses the first edge of the frame sync and instead samples it at t_C.

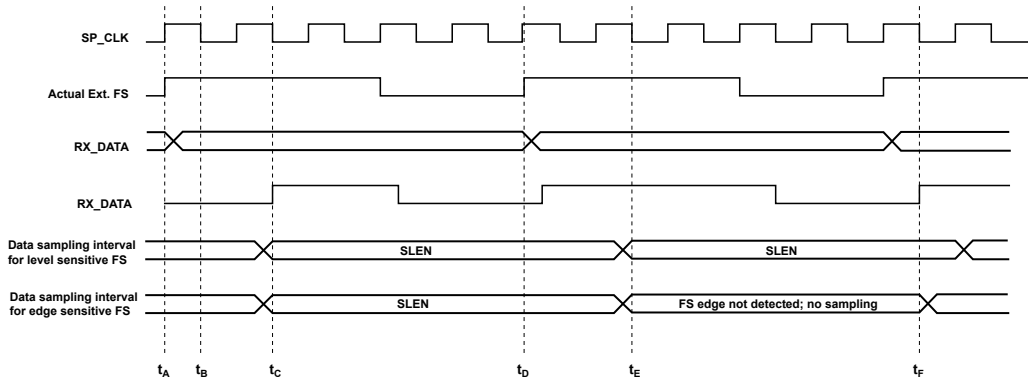


Figure 29-4: Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync

NOTE: SCLK in the ***Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync*** figure is SCLK.

When the above occurs, the internal word length counter runs for a period equal to the SPORT_CTL_A.SLEN field of the control register, but it erroneously expires at t_E rather than at the appropriate point at t_D, thus receiving incorrect data. Further, if a new level-sensitive frame sync edge arrives at time t_D, the SPORT samples this framing signal again at t_E. As such, the frame sync sampling continues to be misaligned with the external data.

To help address this, the SPORT module provides an option to configure the frame sync signal to instead be edge-sensitive via the SPORT_CTL_A.FSED configuration bit. When this bit is set with active high frame syncs enabled (SPORT_CTL_A.LFS = 0), the rising edge of the frame sync is valid. Conversely, when the frame sync is active low (SPORT_CTL_A.LFS= 1), the falling edge is defined to be valid.

NOTE: SPORT_CTL_A.FSED is valid only in external frame sync mode. In internal frame sync mode, the setting of this bit is irrelevant and ignored.

In the above example, an edge-sensitive frame sync signal is not detected at t_E because the edge of the framing signal already occurred in the previous cycle (t_D) and there is no new edge to detect at t_E. As a result, the internal word length counter remains idle for this frame, thus ignoring the incorrect data, and the counter correctly resumes operation at t_F when a new frame sync edge is detected.

This activity sets the SPORT_ERR_A.FSERRSTAT bit and optionally generates a premature frame sync error interrupt.

Frame sync edge detection is used by default for stereo modes. MCM mode and DSP serial mode choose between edge detection and normal mode of FS detection.

NOTE: When the SPORT is first enabled, an already active externally applied frame sync will not commence operation. The SPORT will wait for a valid change in the frame sync's state from inactive to active before operation begins.

Early versus Late Frame Syncs

Frame sync signals can occur in the same serial clock cycle as the first bit of the data word (late) or one serial clock cycle before the first bit (early), as controlled by the SPORT_CTL_A.LAFS bit.

By default, the frame sync signal is configured to be early (SPORT_CTL_A.LAFS = 0). The first bit of the transmit data word will be driven one serial clock cycle after the frame sync is asserted (whether sensed externally or internally provided), and the first bit of the receive data word is expected to lag the frame sync by one serial clock cycle. The frame sync is not checked again until the entire word has been transferred.

If data transmission is continuous in early framing mode, then an internally-generated frame sync signal will be asserted (pulsed active for one serial clock cycle) synchronous to the last data bit of the current transfer, as the first bit of the next transfer will be immediately driven in the next serial clock cycle (no clocks are wasted). This event is not a premature frame sync error, so the SPORT_ERR_A.FSERRSTAT bit is not set.

The frame sync can alternatively be configured as late (SPORT_CTL_A.LAFS = 1), in which case the first bit of the transmit data word is available in the same serial clock cycle that the frame sync is asserted (whether sensed

externally or internally provided), and the first bit of the receive data word is also latched in the same cycle. Serial clock edges latch the receive data bits, but the frame sync signal is checked only during the first bit of each word. Internally generated frame syncs remain asserted for the entire length of the data word in late framing mode.

The ***Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)*** figure illustrates these concepts.

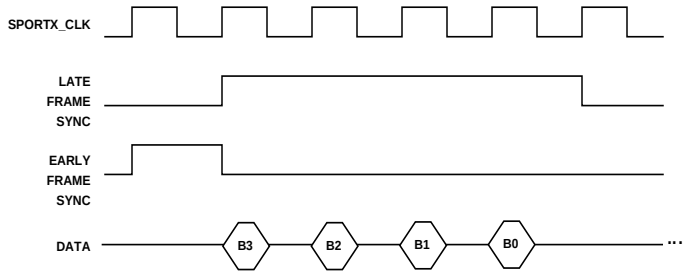


Figure 29-5: Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)

Framed versus Unframed Frame Syncs

The use of a frame sync signal is optional for SPORT operation, as controlled by the `SPORT_CTL_A.FSR` bit. When the frame sync is configured to be required (`SPORT_CTL_A.FSR` = 1), the data is defined to be framed (a frame sync signal must accompany every data word). To allow continuous transmission from the processor, ensure that a new data word is loaded into the transmit buffer before the ongoing transfer is completed (this is automatically cared for when DMA is used to transmit blocks of data).

Data words can be transferred continuously in what is referred to as unframed data mode, which is appropriate for continuous reception, by setting `SPORT_CTL_A.FSR` = 0. In this configuration, a single frame sync is still required to initiate communication, but it is subsequently unrequired once the communication begins. From that point onward, externally provided frame syncs are ignored and internally generated frame syncs are not driven. The ***Framed versus Unframed Data Stream*** figure shows the differences in SPORT operation between framed and unframed data modes with the frame sync configured to be early (`SPORT_CTL_A.LAFS` = 0).

NOTE: When DMA is enabled in a mode where frame syncs are not required, chaining can delay DMA requests. DMA requests are not always serviced frequently enough to guarantee continuous unframed data flow. Monitor status bits or check for a SPORT error interrupt to detect underflow or overflow of data.

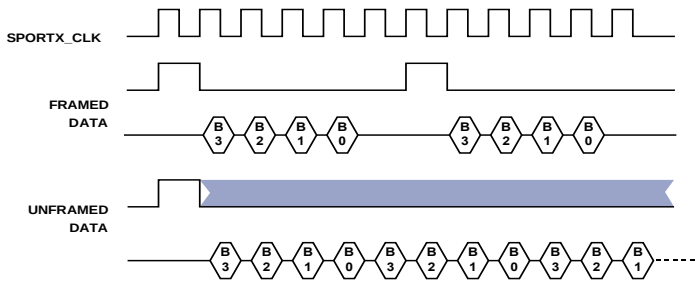


Figure 29-6: Framed versus Unframed Data Stream

Frame Sync Polarity

The framing signals can be active high or active low, as governed by the SPORT_CTL_A.LFS bit:

- When SPORT_CTL_A.LFS = 0, the corresponding frame sync signal is active high.
- When SPORT_CTL_A.LFS = 1, the corresponding frame sync signal is active low.

Active high is the default polarity of the frame sync signal.

Premature Frame Sync Error Detection

A SPORT framing signal is used to synchronize transmit or receive data. In external frame sync mode, any frame sync received during an active frame is premature and invalid. When this occurs, the SPORT_ERR_A.FSERRSTAT bit is set to indicate the framing error, and an optional error interrupt request can be generated for this event by setting the SPORT_ERR_A.FSERRMSK bit.

NOTE: The SPORT_ERR_A.FSERRSTAT bit is not set in the presence of uncleared underflow or overflow errors.

Refer to the ***Frame Sync Error Detection*** figure. The frame sync error bit gets set when an unexpected frame sync occurs during the ongoing data transfer (transmission or reception).

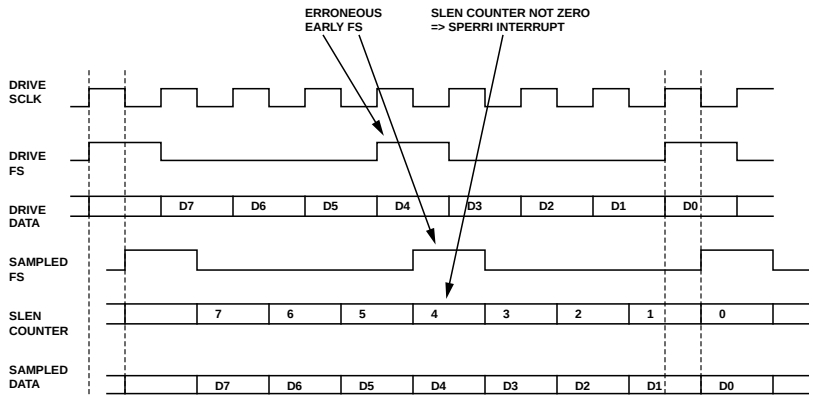


Figure 29-7: Frame Sync Error Detection

NOTE: SCLK in the ***Frame Sync Error Detection*** figure is SCLK.

Whether a SPORT is receiving or transmitting, its bit count is set to the programmed serial word length when the frame sync is sampled, which is then decremented every subsequent serial clock cycle until the transfer has completed. At this point, the bit count reaches zero and will be reset to the programmed serial length when the next frame sync is sampled. As such, the bit count value is non-zero during an active transfer, and the frame sync error is asserted if a frame sync is sampled when this count is non-zero.

Mode Selection

The SPORT's operating mode is configured in the SPORT_CTL_A and SPORT_MCTL_A registers. The ***SPORT Operating Modes*** table provides specific guidance to properly program these SPORT control registers for the desired mode of operation.

Table 29-10: SPORT Operating Modes

Operating Modes	SPORT_CTL_A.OPMODE	SPORT_CTL_A.LAFS	SPORT_CTL_A.RJUST	SPORT_MCTL_A.MCE
Standard DSP Serial	0	Programmable	Reserved	0
I^2S	1	0	Reserved	0
Left-Justified	1	1	Reserved	0
Right-Justified	1	1	1	0
Multichannel	0	Reserved	Reserved	1
Packed I^2S mode	1	Reserved	Reserved	1

The following sections provide detailed information for each of the supported SPORT modes of operation.

Standard DSP Serial Mode

The SPORT can be configured in standard DSP serial mode by clearing the SPORT_CTL_A.OPMODE and SPORT_MCTL_A.MCE bits. This mode provides great flexibility in terms of programmable options to configure the SPORTs to communicate with various serial devices such as serial data converters and audio codecs. In order to properly connect to such devices, various clocking, framing, and data formatting options are available.

Timing Control Bits

Several bits in the SPORT_CTL_A control register define the configuration of the SPORT in standard DSP serial mode:

- SLEN: serial word length (4–32 bits)
- LSBF: shift LSB or MSB first
- ICLK: internally generated or externally provided serial clock
- CKRE: sample on rising or falling edge of the serial clock
- IFS: internally generated or externally provided frame sync
- FSR: framed or continuous operation
- DIFS: data-dependent or data-independent frame sync
- LFS: active high or active low frame sync
- LAFS: frame sync synchronous to data or one clock cycle before it
- PACK: 16-bit to 32-bit packing option

- GCLKEN: free-running or gated clock

Clocking Options

In standard DSP serial mode, the SPORTs can either accept an external serial clock or generate one internally, as controlled by the SPORT_CTL_A.ICLK bit. For internally generated serial clocks (SPORT_CTL_A.ICLK = 1), the SPORT_DIV_A.CLKDIV field configures the serial clock rate from the system clock.

The SPORT clock can also be gated, where it is only valid during an active transfer, as controlled by the SPORT_CTL_A.GCLKEN bit.

The SPORT clock edge used for driving and sampling of serial data and frame syncs is configured using the SPORT_CTL_A.CKRE bit:

- If SPORT_CTL_A.CKRE = 0, input data and frame sync signals are sampled on the falling edge of the serial clock, and output data and frame sync signals are driven on the rising edge.
- If SPORT_CTL_A.CKRE = 1, input data and frame sync signals are sampled on the rising edge of the serial clock, and output data and frame sync signals are driven on the falling edge.

Stereo Modes

The SPORTs support three widely used stereo modes of operation:

- I^2S mode
- Left-Justified mode
- Right-Justified mode

In these modes, the serial data stream consists of left and right channels. The following sections describe these modes in more detail.

Channel Order

The active low frame sync (SPORT_CTL_A.LFS) bit is used to determine the polarity of the frame sync signal in the non-stereo modes of operation. For the stereo modes of operation, it instead controls whether the right or left channel is first in the data transfer. The ***Channel Order Bit Settings*** table shows which word is transmitted or received first, based on the setting of the SPORT_CTL_A.LFS bit.

Table 29-11: Channel Order Bit Settings

Mode	**SPORT_CTL_A.LFS=0**	**SPORT_CTL_A.LFS=1**
Left-Justified or Right-Justified	Left channel first	Right channel first
I^2S or Packed I^2S	Right channel first	Left channel first

I²S Mode

I²S mode is a commonly used stereo mode, where left and right channel data words are interleaved in the serial data stream and each transition of the frame sync signal is associated with one of the channels. The left channel data is transferred during the low segment of the frame sync signal, and the right channel data is transferred during the high segment of the frame sync signal. As such, the frame sync signal is considered to be a left-right (L/R) clock in this mode.

To set the SPORT up in I²S mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 1
- `SPORT_CTL_A.LFS` = 0
- `SPORT_MCTL_A.MCE` = 0

Protocol Configuration Options

Several bits in the `SPORT_CTL_A` control register must be configured to be compliant with the I²S standard, but they can be otherwise configured to support non-standard operation as well:

- SLEN: programmable (allowable word lengths are 5–32 bits)
- LSBF: set to 0 (MSB first)
- ICLK: programmable (serial bit clock can be internally generated or externally provided)
- IFS: programmable (serial L/R clock source must match serial bit clock source)
- LFS: set to 1 (left channel first)
- CKRE: set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK` = `SPORT_CTL_A.IFS` = 1), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The ***Word Select Timing in I²S Mode*** figure shows the SPORT timing in I²S mode. The data lags the L/R clock transition by one SCLK cycle, and the transfer begins with the left channel data word first.

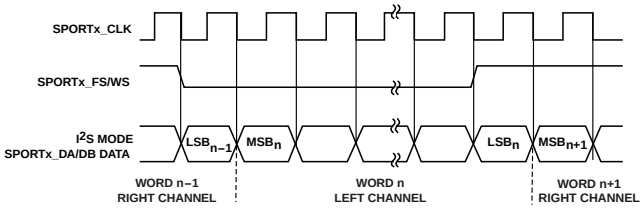


Figure 29-8: Word Select Timing in I²S Mode

Left-Justified Mode

Left-justified mode is a stereo mode subset of the I²S standard. As in I²S mode, the frame sync signal acts as a left-right clock (L/R clock), where left and right data samples are transferred each L/R clock period. The left channel is associated with the high segment of the frame sync, and the right channel aligns with the low segment of the frame sync. The difference between left-justified mode and standard I²S mode is that the channel data is driven in the same bit clock cycle as the L/R clock transition (rather than one bit clock cycle later), such that the MSB is synchronous with the leading edge of the frame sync transition.

To set the SPORT up in left-justified mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 1
- `SPORT_CTL_A.LAFS` = 1
- `SPORT_MCTL_A.MCE` = 0

Protocol Configuration Options

Several bits in the `SPORT_CTL_A` control register must be configured to operate the SPORT in left-justified mode, but they can be otherwise configured as well:

- SLEN: programmable (allowable word lengths are 5–32 bits)
- LSBF: set to 0 (MSB first)
- ICLK: programmable (serial bit clock can be internally generated or externally provided)
- IFS: programmable (serial L/R clock source must match serial bit clock source)
- LFS: set to 0 (left channel first)
- CKRE: set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK` = `SPORT_CTL_A.IFS` = 1), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The ***Word Select Timing in Left-Justified Mode*** figure shows the SPORT timing in left-justified mode. The start of a data sample is synchronous to the L/R clock transition, and the transfer begins with the left channel data word first.

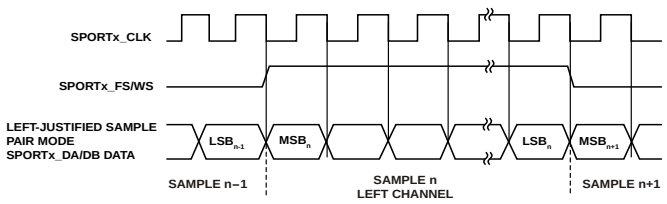


Figure 29-9: Word Select Timing in Left-Justified Mode

Right-Justified Mode

Right-justified mode is a stereo mode subset of the I²S standard. As in I²S mode and left-justified mode, the frame sync signal acts as a left-right clock (L/R clock), where left and right data samples are transferred each L/R clock period. The left channel is associated with the high segment of the frame sync, and the right channel aligns with the low segment of the frame sync. The difference between right-justified mode and standard I²S mode is that the LSB of the channel data ends at the point that the L/R clock transitions to frame the next sample (rather than one bit clock cycle after the L/R clock transition).

To set the SPORT up in right-justified mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 1
- `SPORT_CTL_A.RJUST` = 1
- `SPORT_MCTL_A.MCE` = 0

Timing Control Bits

Several bits in the `SPORT_CTL_A` control register must be configured to operate the SPORT in right-justified mode, but they can be otherwise configured as well:

- SLEN: programmable (allowable word lengths are 5–32 bits)
- LSBF: set to 0 (MSB first)
- ICLK: programmable (serial bit clock can be internally generated or externally provided)
- IFS: programmable (serial L/R clock source must match serial bit clock source)
- LFS: set to 0 (left channel first)
- CKRE: set to 1 (sample L/R clock and data on rising edge of bit clock)

Serial Bit Clock and L/R Clock Rates

If the SPORT is configured to generate the bit clock and the L/R clock (`SPORT_CTL_A.ICLK` = `SPORT_CTL_A.IFS` = 1), set the serial bit clock rate using the `SPORT_DIV_A.CLKDIV` bit field and the L/R clock rate using the `SPORT_DIV_A.FSDIV` bit field.

The ***Word Select Timing in Right-Justified Mode*** figure shows the SPORT timing in right-justified mode. The transmitter aligns the transmit data such that the last bit of the serial word is sent in the last clock cycle of the L/R clock (frame sync) signal marking the channels.

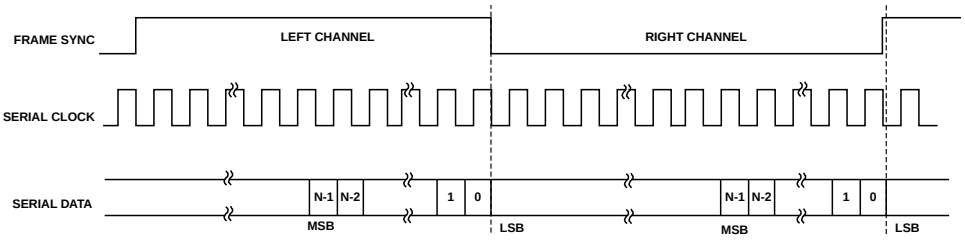


Figure 29-10: Word Select Timing in Right-Justified Mode

NOTE: For some SPORT-compatible ADCs or DACs such as the AD1871, right-justified mode is limited to commonly used ratios such as 64 FS and 128 FS. FS is the sampling frequency of ADCs and DACs, referrd to as the SPORT's L/R clock (frame sync) signal.

Consider the SPORT timing for right-justified mode, as shown in the ***Timing Comparison Between Different Stereo Modes*** figure. The frame sync width is limited to 32 SPORT clock periods (or 32 bits per channel) if:

- the SPORT's frame sync (L/R clock) runs at the FS rate, and
- the SPORT's serial bit clock runs at the 64 FS rate

The limitation applies to the frame sync width of either channel. If the data is confined to 24 bits, the SPORT introduces a 32–24=8-bit clock delay before it starts to transmit or capture data.

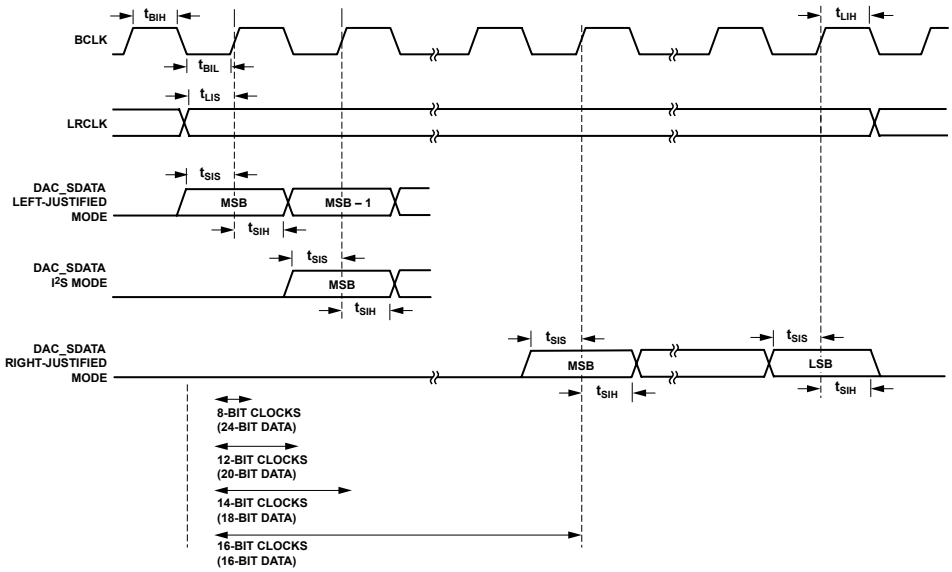


Figure 29-11: Timing Comparison Between Different Stereo Modes

Similarly, to support the 128 FS bit clock frequency, the frame sync width becomes 64 serial bit clock periods per channel. In this case, the delay can be a maximum of 59 bit clocks (64 – 5, which is the minimum serial data length in right-justified mode).

The starting point of the first bit is delayed so that the LSB of the serial data aligns properly with the end of the channel. A 6-bit counter is added for this purpose in the stereo mode counter, which is programmed by writing the least significant six bits of the `SPORT_MCTL_A.WOFFSET` field. Though this is a multichannel mode configuration register, the SPORT uses these bits in right-justified mode to configure the offset from the transition of the L/R

clock to where the first data bit must be driven in order to have the end of the last bit align properly with the next L/R clock transition. Software must program this register with the appropriate delay.

Multichannel (TDM) Mode

The multichannel mode of SPORT operation allows the SPORT to communicate as part of a time division multiplexed (TDM) serial system. In TDM communications, a large frame of streamed serial data words is defined to be a particular length. It consists of a specific number of channels, and each channel contains one serial data word of the defined data length. For example, a 24-word block of 24-bit data can be defined to be a window within a frame, having a duration of 576 bit clocks and comprised of 24 continuous channels. The SPORT is configured to transfer on specific channels within a defined window in this frame.

To set the SPORT up in multichannel mode, the following configuration is required:

- `SPORT_CTL_A.OPMODE` = 0
- `SPORT_MCTL_A.MCE` = 1

In multichannel mode, the SPORT can selectively transfer data on up to a maximum window size of 128 continuous channels out of a maximum 1024-channel frame while ignoring all the disabled channels within the window and all the channels outside the window. The SPORT can do any of the following on each channel:

- Transmit data (`SPORT_CTL_A.SPTRAN` = 1)
- Receive data (`SPORT_CTL_A.SPTRAN` = 0)
- Do nothing (during inactive channels)

Channel selection is configured in the half SPORT multichannel select registers (`SPORT_CS0_A` - `SPORT_CS3_A`) before enabling SPORT operation for multichannel mode. Programming of these registers is especially important in DMA data unpacked mode, since the SPORT data buffers begin operation immediately after the SPORT data lines are enabled. Be sure to enable multichannel operation (set the `SPORT_MCTL_A.MCE` bit) prior to enabling the SPORT itself.

Clocking Options

In multichannel mode, the SPORTs can either accept an external serial clock or generate one internally, as governed by the `SPORT_CTL_A.ICLK` bit. For an internally-generated serial clock (`SPORT_CTL_A.ICLK` = 1), the `SPORT_DIV_A.CLKDIV` bit field is used to configure the serial clock rate, as derived from the system clock.

The serial clock edges used to drive and sample data and frame syncs are also configurable using the `SPORT_CTL_A.CKRE` bit:

- If `SPORT_CTL_A.CKRE`= 0, input data and frame sync signals are sampled on the falling edge of the serial clock, and output data and frame sync signals are driven on the rising edge.
- If `SPORT_CTL_A.CKRE` = 1, input data and frame sync signals are sampled on the rising edge of the serial clock, and output data and frame sync signals are driven on the falling edge.

Frame Sync Options

The frame sync signal synchronizes the channels and restarts each multichannel sequence, starting with the channel 0 data word. For internally-generated frame syncs (SPORT_CTL_A.IFS = 1), the frame sync period in multichannel mode is defined as:

FS period = [(SPORT_CTL_A.SLEN + 1) × number of channels] - 1

The active level for the frame sync signal is also configurable by programming the SPORT_CTL_A.LFS bit. Set this bit to make the frame sync an active low signal, and clear it to make it active high.

In multichannel mode, frame sync timing resembles late framing mode (although the SPORT_CTL_A.LAFS bit is reserved in this mode). The first bit of the transmit data word is driven and the first bit of the receive data word is sampled in the same serial clock cycle as the frame sync, provided there is no programmed frame delay (SPORT_MCTL_A.MFD = 0).

Once the frame sync signal is asserted, word transfers are performed continuously for the duration of the active window, and no further frame syncs are required for different channels within the window. As such, internally-generated frame syncs are always data-independent, and the SPORT_CTL_A.DIFS bit is reserved.

Transmit Data Valid (TDV)

Each SPORT features a transmit data valid signal (SPORT_ATDV), which is driven high during enabled transmit channels. Because the SPORT output data signals are three-stated during inactive channels, the SPORT_ATDV signal signifies when the processor is actively driving the SPORT data outputs, thus serving as an output-enable signal for the data transmit pin(s).

Active Channel Selection Registers

In multichannel mode, the SPORT supports a window size of up to 128 channels for transmitting or receiving data, where it can selectively receive or transmit data in any of these 128 channels. Each channel can be individually enabled or disabled using the multichannel selection registers (SPORT_CS0_A to SPORT_CS3_A) to select the channels in which to transfer data during a multichannel communication stream. Data words associated with enabled channels are transmitted or received in the respective channels, while disabled channels cause a transmit SPORT to three-state the data output pins and a receive SPORT to ignore the data.

The four 32-bit multichannel selection registers combine to form up to a 128-bit meta-register to accommodate the maximum window size of 128 channels. Setting any bit within these registers enables the associated channel. The 128 channels are sequentially numbered from bit 0 in the SPORT_CS0_A register (corresponding to channel 0 of the window) to bit 31 of the SPORT_CS3_A register (corresponding to channel 127 of the window). For example, setting bit 13 of the SPORT_CS1_A register enables channel number 45 (add 32 for the channels in the SPORT_CS0_A). Likewise, setting bit 5 of the SPORT_CS3_A register enables channel number 101 (add 96 for the 32 channels in each of the SPORT_CS0_A, SPORT_CS1_A, and SPORT_CS2_A registers).

Multichannel Frame Delay (MFD)

The multichannel frame delay (SPORT_MCTL_A.MFD) field specifies the delay in serial bit clocks between the frame sync pulse and the first data bit in the frame. This configurability allows the processor to work with different types of telephony interface devices.

As SPORT_MCTL_A.MFD is a 4-bit field, the maximum value allowed for the frame delay is 15 serial clock cycles. When set to 0, the frame sync is concurrent with the first data bit. If SPORT_MCTL_A.MFD>0, a new frame sync can occur during the last channel(s) of a previous frame and still be valid (does not cause a frame sync error).

NOTE: If the required frame delay exceeds 15 serial clocks, use the window offset field (SPORT_MCTL_A.WOFFSET) to delay the start of channel 0 in increments of the serial word length, and then adjust SPORT_MCTL_A.MFDaccordingly. For example, if the serial word length is 12 bits and the desired frame delay is 16 serial clock cycles, set the SPORT_MCTL_A.WOFFSET to 1 to insert a 12-bit delay after the frame sync to where the channel 0 data begins, and then program SPORT_MCTL_A.MFD to 4 (i.e., 16 - 12).

Window Size (WSIZE)

Select the number of channels used in multichannel operation by programming the 7-bit SPORT_MCTL_A.WSIZE field. This field must be set to the actual number of channels minus one (SPORT_MCTL_A.WSIZE = Number of channels -1).

The 10-bit SPORT_MSTAT_A.CURCHAN field holds the channel number currently being serviced during multi-channel operation.

Window Offset (WOFFSET)

The window offset (SPORT_MCTL_A.WOFFSET) field specifies where in the 1024-channel frame to place the start of the active window (up to 128 channels long). A value of 0 specifies no channel offset from the frame sync (channel 0 immediately follows it). Any non-zero value indicates the number of channels that come between the frame sync and the start of channel 0 of the active frame, with 896 (for example, 1024–128) being the largest value that permits using all 128 channels.

As an example, a program could define an active window comprised of eight channels (SPORT_MCTL_A.WSIZE = 7) with a window offset of 93 (SPORT_MCTL_A.WOFFSET = 93). If configured in this fashion, the 8-channel window that the SPORT will transfer within resides in the channel range from 93 to 100 in the up-to-1024-channel frame.

Do not change the window offset or the number of multichannel slots (SPORT_MCTL_A.WSIZE) while the SPORT is enabled. If the combination of the window size and offset place any portion of the window out-of-range relative to the channel counter, none of the channels are enabled.

Companding Selection

Like the other operating modes, companding logic can optionally be applied to serial data (compression logic for transmit mode or expansion logic for receive mode). The two widely used companding algorithms, A-law and µ-law, are selectable using the SPORT_CTL_A.DTYPE field.

If companding is enabled, the companding algorithm is applied to both the primary and secondary datapaths. In multichannel mode, companding can be applied to either all or none of the enabled channels (companding cannot be selected on a per-channel basis).

Multichannel DMA Data Packing (MCPDE)

Multichannel DMA data packing and unpacking are enabled using the SPORT_MCTL_A.MCPDE bit.

When set, data is packed, and the SPORT expects the data in the DMA buffer to correspond only with enabled SPORT channels. For example, if only channels 1 and 9 are enabled in a 10-channel window (SPORT_MCTL_A.WSIZE = 9), the SPORT expects the buffer to be exactly two words in length, where channel 1 is associated with the first element in the buffer and channel 9 is associated with the second.

When cleared, data is unpacked, and the SPORT expects the DMA buffer to have a word for each of the channels in the active window, whether the channel is enabled or not. As such, the DMA buffer size must be exactly the size of the window. Using the same example as the packed case above, if only channels 1 and 9 are enabled in a 10-channel window (SPORT_MCTL_A.WSIZE = 9), then the DMA buffer size is ten words. The data at offsets 1 and 9 within the buffer are associated with the data transfers of channels 1 and 9, respectively. The rest of the words in the buffer are unused.

Packed I²S Mode

The SPORT supports a packed I²S mode, which can be used for audio codec communications using multiple channels. This mode allows applications to send more than the standard 32 bits per channel available through standard I²S mode. Packed mode is implemented using standard multichannel mode (and is therefore programmed similarly to multichannel mode).

To set the SPORT up in packed I²S mode, the following configuration is required:

- SPORT_CTL_A.OPMODE = 1
- SPORT_MCTL_A.MCE = 1

Like multichannel mode, packed I²S mode also supports a maximum of 128 channels, where up to 128 channels of data can be transferred for every transition of the frame sync signal acting as an L/R clock (for example, up to 128 left-channel words transfer during the high portion of the L/R clock, and up to 128 right-channel words transfer during the low portion).

As shown in the ***Packed I²S Mode 128 Operation*** figure, the packed waveforms are the same as those waveforms used in multichannel mode, except the frame sync is toggled for every frame and emulates I²S mode.

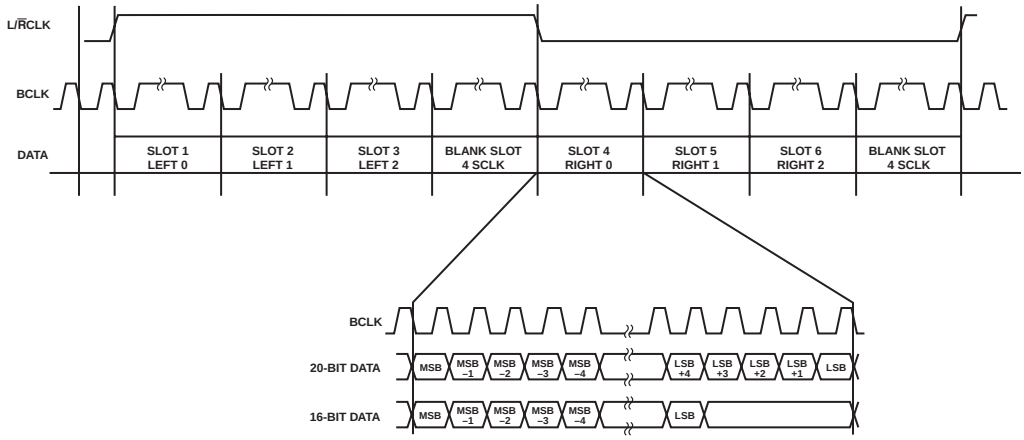


Figure 29-12: Packed I²S Mode 128 Operation

Serial Bit Clock Options

In packed I²S mode, the SPORTs can either accept an external serial bit clock or generate one internally, as governed by the `SPORT_CTL_A.ICLK` configuration bit. For an internally-generated serial bit clock (`SPORT_CTL_A.ICLK` = 1), use the `SPORT_DIV_A.CLKDIV` bit field to configure the serial bit clock rate from the system clock.

The serial bit clock edge that is used for sampling or driving data and frame syncs is programmable using the `SPORT_CTL_A.CKRE` bit.

L/R Clock (Frame Sync) Options

The frame sync period in packed I²S mode is defined as:

FS period = [(`SPORT_CTL_A.SLEN` + 1) × number of channels] - 1.

The L/R clock can be supplied externally or internally generated depending on the `SPORT_CTL_A.IFS` bit setting. The logic level of the L/R clock associated with the left and right channel data can be changed using the `SPORT_CTL_A.LFS` configuration bit.

Gated Clock Mode

Some system components such as ADCs and DACs utilize a SPI-compatible protocol for the interface. To communicate with such devices, the SPORT must support a gated clock, where the data valid information is embedded in the clock (for example, the clock only toggles when data is valid). This gated clock feature is enabled using the `SPORT_CTL_A.GCLKEN` bit.

To enable the gated clock mode of operation, program the SPORT to comply with the following requirements.

- Do not enable gated clock functionality in right-justified or multichannel mode
- Gated clock mode has the following requirements for other control bits:

  - The serial clock and frame sync signals must have the same source (SPORT_CTL_A.ICLK = SPORT_CTL_A.IFS)
  - Unframed mode is not supported (SPORT_CTL_A.FSR must be set)
  - Clear the SPORT_CTL_A.DIFS bit in transmit mode; set it in receive mode
- Satisfy the following necessary conditions when gated clock mode is enabled:
  - Seven serial clock cycles are required between enabling the SPORT and the first frame sync. If this requirement is not met, the SPORT can drop the first data (for subsequent data, this requirement is not applicable)
  - For externally-provided clock and frame sync, the frame sync must be inactive during clock synchronization after the SPORT has been enabled
  - For an edge-detected frame sync (SPORT_CTL_A.FSED = 1), the frame sync must transition back to the inactive state before the current word transfer is complete (or when the clock is still running). If this requirement is not met, the SPORT does not recognize the next valid frame sync and skips the channel. The SPORT continues to skip the frame syncs until the frame sync transitions back to an inactive state while the clock is active.

Data Transfers and Interrupts

SPORT data can be transferred to or from internal or external memory by two methods:

- Core-driven, single-word transfers
- DMA-driven, multiple-word transfers (optionally with multiple work units)

Core-driven transfers use SPORT interrupts to signal the processor core to perform MMR-based single-word transfers to or from the SPORT data buffers. DMA can be set up to automatically transfer a configurable number of serial words between the SPORT transmit/receive data buffers and memory, and then generate a data completion interrupt request when a work unit or a series of work units completes, thus signaling to the processor core that a block of data has been transferred.

The following sections provide information on core-driven and DMA-driven data transfers.

Data Buffers

When programming the serial port data channels (primary or secondary) as a transmitter by setting SPORT_CTL_A.SPTRAN =1, only the corresponding transmit data buffers (SPORT_TXPRI_A and SPORT_TXSEC_A) become active. The receive data buffers (SPORT_RXPRI_A and SPORT_RXSEC_A) remain inactive. Similarly, when the SPORT data channels are programmed for receive operation (SPORT_CTL_A.SPTRAN =0), then only corresponding receive data buffers (SPORT_RXPRI_A and SPORT_RXSEC_A) are active. Do not attempt to read or write inactive data buffers. If the processor operates on the inactive transmit or receive buffers while the SPORT is enabled, unpredictable results can occur.

Each of these buffers is 32-bit wide (corresponds to maximum serial data word length). When using word lengths less than 32 bits for SPORT operation, the data in these buffers is automatically right-justified. (The LSB bit of data is at the bit 0 location of the buffer). The upper unused bits can be zero-filled or sign-extended depending on SPORT_CTL_A.DTYPE field.

Transmit Data Buffers (SPORT_TXPRI_A and SPORT_TXSEC_A)

When enabled as a transmitter (SPORT_CTL_A.SPTRAN =1), each SPORT half has its own set of transmit data buffers. The primary (0) and secondary (1) datapaths of each SPORT half have separate data buffers, referred to as SPORT_TXPRI_A and SPORT_TXSEC_A respectively.

These transmit data buffers are 32 bits wide. Load these buffers with the data for transmission on the primary and secondary data channels. The DMA controller loads the data automatically. Or, the program running on the processor core loads the data manually.

Together with the output shift register, transmit data buffers act like a two-location FIFO. If data packing is disabled (SPORT_CTL_A.PACK =0), the transmit path can hold as many as three data words. If data packing is enabled (SPORT_CTL_A.PACK =1), it can hold two packed data words at any given time.

When the transmit shift register becomes empty (transfer out all the bits of previous word), data in the transmit data buffer is automatically loaded into it. An interrupt occurs when the output transmit shift register has been loaded, signifying that the transmit data buffer is empty and ready to accept the next word. This interrupt does not occur when serial port is operating in DMA mode or when the corresponding interrupt enable mask bit is set.

If only the primary datapath of a SPORT half is enabled, programs must not write to the inactive secondary transmit data buffer and conversely. If the core keeps writing to the inactive buffer, the status of that transmit buffer becomes full. This state can cause the core to hang indefinitely, since data is never transmitted to the output shift register.

Receive Data Buffers (SPORT_RXPRI_A and SPORT_RXSEC_A)

When enabled as receiver (SPORT_CTL_A.SPTRAN =0), each SPORT half has its own set of receive data buffers. The primary (0) and secondary (1) datapaths of each SPORT half have separate data buffers, referred as SPORT_RXPRI_A and SPORT_RXSEC_A respectively. Together with input shift register, the receive data buffers act like a three-location FIFO, as the receive path has two data registers.

These receive data buffers are the 32 bits wide. These buffers are automatically loaded from the receive shift register when a complete word has been received into it. An interrupt occurs when the receive data buffer is loaded, signifying that new data is available in the receive data buffer and is ready to read. This interrupt does not occur when the serial port is operating in DMA mode or when the corresponding interrupt enable mask bit is set.

If only the primary datapath of a SPORT half is enabled, programs must not read from the inactive secondary receive data buffer and conversely. If the core keeps reading from the inactive buffer, the status of that receive buffer becomes empty. This state can cause the core to hang indefinitely since new data is never received through the input shift register.

Data Buffer Status

The SPORT provides status information about its primary and secondary data buffers through the SPORT_CTL_A.DXSPRI and SPORT_CTL_A.DXSSEC bits, respectively. It also provides error status information through the corresponding SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DERRSEC bits, respectively. Depending on the SPORT_CTL_A.SPTRAN bit setting, these bits reflect the status of either the pair of transmit (SPORT_TXPRI_A and SPORT_TXSEC_A) or receive (SPORT_RXPRI_A and SPORT_RXSEC_A) buffers, indicating whether the buffer is full, partially full, or empty.

When attempting to read from an empty receive buffer or write to a full transmit buffer, the SPORT delays access until the buffer is ready, potentially resulting in excessive MMR bus response times. To avoid this when doing core-driven transfers, always check the buffer status to determine if the access can be made. The SPORT updates the status bits in the SPORT_CTL_A register during reads and writes by the core processor.

NOTE: These status bits are updated during reads and writes from the core processor even when the SPORT is disabled.

Two complete 32-bit words can be stored in the receive buffer while a third word shifts in. Therefore, almost three complete words can be received without the receive buffer being read before an overflow occurs. After receiving the third word completely, the shift register contents overwrite the second word, which will occur if the first word has not yet been read by the processor core or the DMA controller. This receive overflow condition is flagged through the error status bits of the SPORT_CTL_A register on the last bit of the third word.

Data Buffer Packing

When the SPORT is configured as a receiver with a serial data word length of 16 or less, the received data words can be packed into a 32-bit word. Similarly, if the SPORT is configured as a transmitter with a serial data word length of 16 or less, then 32-bit words being transmitted can be unpacked into 16-bit words. The SPORT_CTL_A.PACK bit is used to select this packing or unpacking feature.

When SPORT_CTL_A.PACK = 1, two consecutive received words are packed into a single 32-bit word, or each 32-bit word is unpacked and transmitted as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15–0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31–16. This packing method applies to both receive (packing) and transmit (unpacking) operations. In this case, the transmit and receive interrupt requests are generated for the 32-bit packed words, not for each 16-bit word.

NOTE: When 16-bit received data is packed into 32-bit words and stored in normal word space in the processor's internal memory, the 16-bit words can be read or written using short word space addressing.

Single-Word (Core) Transfers

The SPORTs can transmit or receive individual data words with interrupt requests occurring as each data word is transferred. When a SPORT is enabled with the corresponding DMA channel disabled, interrupt requests are generated when:

- a complete word has been received in the receive data buffer or

- the transmit data buffer is not full

When performing core transfers, be sure to access only those buffers that are associated with enabled datapaths, as governed by the transfer direction (SPORT_CTL_A.SPTRAN) bit and the primary/secondary (SPORT_CTL_A.SPENPRI/SPORT_CTL_A.SPENSEC) data enable bits. If inactive SPORT data buffers are read from or written to by the core while the SPORT is enabled, the core can hang. For example, if a half SPORT is programmed to be a transmitter and the core reads from one of the receive buffers associated with that half SPORT, the core can hang as if it were reading an empty buffer that is active and awaiting new data to arrive. Because this is a transmitting HSPORT, that data will never arrive, thus locking the core up until the SPORT is reset. To avoid such a situation, be sure to check the status of the appropriate data buffer before attempting a core access to it by interrogating the SPORT_CTL_A.DXSPRI or SPORT_CTL_A.DXSSEC status bits.

DMA Transfers

Direct memory access (DMA) provides a mechanism for transferring an entire block of serial data before an interrupt is generated. The processor's on-chip DMA controller automatically handles the DMA transfer, thus allowing the processor core to run in parallel until the entire block of data is transferred. When the interrupt request occurs, a service routine can then process the entire block of data (rather than react to single words), thus significantly reducing overhead.

Each half SPORT has a dedicated DMA channel that serves both the primary and secondary datapaths. When configured as a transmitter (SPORT_CTL_A.SPTRAN = 1) with both the primary and secondary datapaths enabled (SPORT_CTL_A.SPENPRI = SPORT_CTL_A.SPENSEC = 1), the DMA channel requires that the source DMA buffer interleave the data beginning with the primary channel, as it will alternately load to the primary and secondary transmit data buffers once it is enabled. The complementary operation is true in receive mode (SPORT_CTL_A.SPTRAN = 0) when both datapaths are enabled, as the DMA channel alternately reads from the primary and secondary receive data buffers and interleaves them in the destination DMA buffer. As such, software must de-interleave the data corresponding to the primary and secondary channels from the receive DMA buffer.

If the SPORT is configured in stereo mode, the same DMA channel handles both the left and right channels of both datapaths (primary and/or secondary). Therefore, for a transmit DMA with only one datapath enabled, the source buffer must be populated such that the left- and right-channel data is interleaved. If both datapaths are enabled, the DMA channel alternately loads to the primary and secondary transmit data buffers once it is enabled. As such, the interleaving requirement is for the primary left-channel data to be followed by the secondary left-channel data, then the primary right-channel data, and finally the secondary right-channel data. The complementary operation is true in receive mode, where the DMA channel alternately reads from the primary and secondary receive data buffers and interleave them in the destination DMA buffer. For the stereo modes of operation, the destination DMA buffer is interleaved as left-right data for a single data input. If both datapaths are enabled, the destination DMA buffer is written with the primary and secondary left-channel data followed by the primary and secondary right-channel data. As such, software must de-interleave the primary and secondary left- and right-channel data from the receive DMA buffer, as defined by this scheme.

Since both the primary and secondary datapaths share the single DMA channel, each half SPORT has a single interrupt request for data completion, as well as an error interrupt request. The DMA controller can generate an interrupt request at the end of a chain of DMA work units (when using multiple descriptors) or at the end of individual DMA work unit.

The SPORT DMA channels are assigned a higher priority than all the other DMA channels (for example, the SPI port). Having higher priority causes the SPORT DMA transfers to execute first when multiple DMA requests occur in the same cycle. The SPORT DMA channels are numbered and prioritized in the DMA channel list table in the DMA chapter.

Although the most efficient DMA transfers execute with 32-bit words, the SPORTs can handle word sizes from 4 to 32 bits (as defined by `SPORT_CTL_A.SLEN` field). If the serial data length is 16 bits or smaller, two pieces of data can be packed into 32-bit words for each DMA transfer, as selected by setting the `SPORT_CTL_A.PACK` bit. When this bit is set, the SPORT generates the transmit and receive interrupts for the 32-bit packed words, not for each 16-bit word. For more information, see the Data Buffer Status section.

NOTE: The SPORT DMA channel can access both internal memory and external memory of the processor without any core overhead.

Data Transfer Interrupt

Each half SPORT features a data transfer interrupt request that is shared by both the primary and secondary data channels in both transmit and receive modes. To determine the source of the data transfer interrupt request, applications can check the primary and secondary data buffer status bits (`SPORT_CTL_A.DXSPRI` and , `SPORT_CTL_A.DXSSEC` , respectively).

When using core-driven transfers, this interrupt's meaning depends on the direction of the SPORT:

- As transmitter (`SPORT_CTL_A.SPTRAN` = 1) - the transmit data buffer is empty
- As receiver (`SPORT_CTL_A.SPTRAN` = 0) - new data is available in the receive data buffer

NOTE: When data packing is enabled (`SPORT_CTL_A.PACK` = 1), the core-driven transmit and receive interrupt requests are generated for 32-bit packed words, not for each 16-bit word.

In both cases, the interrupt request can be used to signal the core that an individual transfer has completed. For transmit operations, it indicates that the transmit data buffer can be safely loaded (either the buffer is already empty or the last data has moved from the data buffer to the shift register). For receive operations, it indicates that new data has arrived and can be read (or must be read before a subsequent word overwrites it).

When the SPORT is configured to use DMA to move data between memory and the peripheral (the most generic way to use dedicated DMA for sport data transfers), the same data transfer interrupt request instead indicates the completion of the transfer of a block of serial data (rather than a single word). When DMA is used, the DMA count register must be initialized to specify the number of words to transfer. This count decrements after each DMA transfer on the channel, and the data transfer interrupt request signal is asserted when the word count reaches zero (for example, a DMA work unit has finished).

For transmit DMA, the interrupt request is raised when the last word in the DMA work unit is loaded from the source memory to the HSPORT FIFO. This interrupt request can signal to the core that a new DMA work unit can be configured or that other software threads can now run. The transmit interrupt request can optionally be deferred until the last word of the work unit has fully shifted out of the shift register (see the Transfer Finish Interrupt (TFI) section for details).

For receive DMA, the interrupt request is raised when the last word is loaded to the destination memory. In addition to that described for transmit DMA, this interrupt request also serves as an indication to the core that there is a buffer of newly acquired data that is ready to be processed.

See the DMA chapter for further details regarding enabling of the DMA interrupt requests associated with the various modes of DMA operation.

NOTE: As a single DMA channel services both the primary and secondary datapaths associated with the SPORT, there is a single DMA completion interrupt request.

Transfer Finish Interrupt (TFI)

When configured for transmit DMA (`SPORT_CTL_A.SPTRAN` =1), the data transfer interrupt request gets generated by the DMA engine itself when it decrements its count register upon loading the last element from memory to the HSPORT hardware. Alternately, the SPORT can use a Transmit Finish Interrupt (TFI) to signal the actual end of the transmission (for example, when the last bit of the last data word of the buffer has shifted out of the SPORT to the system) by setting the `SPORT_CTL_A.TFIEN` bit. When this bit is set, then DMA signal that would normally assert the data transfer interrupt request instead signals the SPORT that the DMA work unit is complete. The SPORT then waits until all the data in the FIFO is shifted out (including the transmit shift register) and asserts the TFI interrupt request upon completion.

NOTE: To enable this functionality in the DMA engine, be sure to configure the interrupt type field in the DMA configuration register for Peripheral interrupt. See the DMA chapter for further details.

The DMA muxing must also be configured for the SPORT or the interrupt is not be passed by DMA unit.

Error Detection (Status) Interrupt

In addition to the dedicated data transfer interrupt request, each half SPORT also features an optional error status interrupt request that can be triggered when error conditions occur relative to data or frame syncs associated with the half SPORT.

Data-related errors depend on the direction of the SPORT and reflect overflow or underflow conditions, which are depicted in the `SPORT_CTL_A` control register as read-only sticky bits `SPORT_CTL_A.DERRPRI` and `SPORT_CTL_A.DERRSEC` (for the primary and secondary channels, respectively).

- When the SPORT is configured as a transmitter, these bits provide transmit data buffer underflow status. When the frame sync signal occurs when the transmit data buffer is empty, the underflow bit corresponding with the offending transmit data buffer is set, as the SPORT will transmit data whenever it detects a valid frame sync signal, whether new data is present or not.

- When the SPORT is configured as a receiver, these bits provide receive overflow status. When a channel receives new data while the receive buffer is already full, the new data overwrites the existing data, thus causing an overflow. When this occurs, the overflow bit corresponding with the offending receive data buffer is set, as the SPORT receives data whenever it detects a valid frame sync signal, whether there is room in the receive buffer or not.

Each half SPORT also features an error register (SPORT_ERR_A), which is the source for the assertion of the described data-related error status bits. When a data-related error occurs on the primary or secondary datapaths, the error is logged in the SPORT_ERR_A.DERRPSTAT or SPORT_ERR_A.DERRSSTAT bits, respectively. To enable these status bits to generate the HSPORT status interrupt request in the SEC, the corresponding SPORT_ERR_A.DERRPMSK and SPORT_ERR_A.DERRSMSK bits must be set (for the primary and secondary datapaths, respectively).

The SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DERRSEC channel error status bits are sticky read-only bits that can be cleared in two ways:

- Reset the error detection logic by disabling the channel associated with the error condition (clear the SPORT_CTL_A.SPENPRI or SPORT_CTL_A.SPENSEC control bit).
- Clear the source of the interrupt by writing-1-to-clear the SPORT_ERR_A.FSERRSTAT, SPORT_ERR_A.DERRPSTAT, or SPORT_ERR_A.DERRSSTAT status bits.

In addition to data-related errors, SPORT_ERR_A also tracks frame sync errors in the SPORT_ERR_A.FSERRSTAT status bit. Similar to the data-related errors, the frame sync error can be enabled as a source for raising the error status interrupt request via the SEC by setting the SPORT_ERR_A.FSERRMSK bit. A frame sync error occurs when the frame sync is detected prematurely, as explained in the Premature Frame Sync Error Detection section.

A frame sync error is not detected in the following cases:

- When there is no active transmit or receive data, and the frame sync pulse occurs due to noise on the input signal – if there is no active transfer, a noise-induced frame sync pulse is valid.
- If there is an active underflow or overflow error – frame sync errors cannot be detected because the SPORT error logic does not run after one of the data errors has occurred and remains unserviced.
- When the frame sync pulse doesn't meet minimum timing requirements – if the frame sync pulse is shorter than a SPORT clock period, there is no guarantee that it gets sampled at all and may go unnoticed.

SPORT Programming Model

The following sections provide programming guidance for setting up the SPORTs for use in an application:

- Initializing Core-Driven (Non-MCM) Transfers
- Initializing Multichannel Transfers
- Using DMA for SPORT Transfers

- Using Companding as a Function

Initializing Core-Driven (Non-MCM) Transfers

The following programming model applies to all of Standard DSP Serial Mode, I²S Mode, Left-Justified Mode, and Right-Justified Mode for core-driven transfers. More steps are required to properly initialize the SEC to service the SPORT interrupts (see the SEC chapter for details).

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, this example also applies to half SPORT B.

1. Clear the `SPORT_CTL_A` and `SPORT_MCTL_A` configuration registers.

   ADDITIONAL INFORMATION: Clearing these registers ensures that the SPORT logic (including the multi-channel logic) is fully reset before attempting to reprogram it.

2. Optionally program the `SPORT_DIV_A` clock divisor register.

   ADDITIONAL INFORMATION: This step is only required for internally-generated timing signals. Configure the serial bit clock and/or frame sync (or L/R clock, for stereo modes) rates according to the guidance in the Serial Clock and Frame Sync sections.

3. Program the `SPORT_CTL_A` primary configuration register.

   ADDITIONAL INFORMATION: Set the SPORT operating mode along with the configurable clock, frame sync, word length, direction, and data format options (see the Operating Modes and Options section for details). Do not set the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` buffer enable bits in this step.

4. Optionally program the `SPORT_CTL2_A` secondary configuration register.

   ADDITIONAL INFORMATION: This step is required only if internal multiplexing logic must be enabled to share clock and frame sync signals between a top SPORT module's A and B halves (see the Multiplexer Logic section for details).

5. Optionally program the `SPORT_ERR_A` error register.

   ADDITIONAL INFORMATION: This step is required only if a separate SPORT error interrupt is desired (see the Error Detection (Status) Interrupt section for details).

6. For Right-Justified mode only, program the `SPORT_MCTL_A.WOFFSET` field.

   ADDITIONAL INFORMATION: In Right-Justified mode, this field serves as the delay count (`DCNT`) required to align the LSB of each stereo channel with the L/R clock transition and must be programmed manually (see the Right-Justified Mode section for details).

7. Enable the primary/secondary datapath(s) in the `SPORT_CTL_A` register.

   ADDITIONAL INFORMATION: This should be performed in a read-modify-write operation setting the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` bits, as appropriate.

8. Write data to be transmitted to the transmit buffer (SPORT_TXPRI_A and/or SPORT_TXSEC_A) *or* read data that has been received from the receive buffer (SPORT_RXPRI_A and/or SPORT_RXSEC_A).

   ADDITIONAL INFORMATION: These accesses are typically performed in the context of an interrupt service routine. See the SEC chapter for further information. Do not attempt to read or write inactive data buffers. If the core attempts to access inactive transmit or receive buffers while the SPORT is enabled, unpredictable results may occur.

Initializing Multichannel Transfers

When in Multichannel (TDM) Mode or Packed I^{2}S Mode, the SPORT is in a multichannel operational mode. Follow the steps below to properly initialize the SPORT for multichannel modes of operation. More steps are required to properly configure the system to service the SPORT interrupt requests (see the SEC chapter for details).

NOTE: This example uses half SPORT A registers. With appropriate register changes, this example also applies to half SPORT B.

1. Clear the SPORT_CTL_A and SPORT_MCTL_A registers.

   ADDITIONAL INFORMATION: Clearing these registers ensures that the SPORT logic (including the multichannel logic) is fully reset before attempting to reprogram it.

2. Optionally program the SPORT_DIV_A clock divisor register.

   ADDITIONAL INFORMATION: This step is only required for internally-generated timing signals. Configure the serial bit clock and/or frame sync (or L/R clock, for stereo modes) rates according to the guidance in the Serial Clock and SPORT_CTL2_A sections.

3. Program the SPORT_CS0_A - SPORT_CS3_A channel select registers.

4. Program the SPORT_MCTL_A multichannel configuration register.

   ADDITIONAL INFORMATION: The SPORT supports many multichannel options. For more information, see the Multichannel (TDM) Mode section. Do not set the SPORT_MCTL_A.MCE enable bit in this step.

5. Program the SPORT_CTL_A primary configuration register.

   ADDITIONAL INFORMATION: Set the SPORT operating mode along with the configurable clock, frame sync, word length, direction, and data format options (see the Operating Modes and Options section for details). Do not set the SPORT_CTL_A.SPENPRI and/or SPORT_CTL_A.SPENSEC buffer enable bits in this step.

6. Optionally program the SPORT_CTL2_A secondary configuration register.

   ADDITIONAL INFORMATION: This step is required only if internal multiplexing logic must be enabled to share clock and frame sync signals between a top SPORT module's A and B halves (see the Multiplexer Logic section for details).

7. Optionally program the SPORT_ERR_A error register.

ADDITIONAL INFORMATION: This step is required only if a separate SPORT error interrupt request is desired (see the Error Detection (Status) Interrupt section for details).

8. Set the `SPORT_MCTL_A.MCE` bit to enable multichannel mode.
9. Enable the primary/secondary datapath(s) in the `SPORT_CTL_A` register.

   ADDITIONAL INFORMATION: This should be performed in a read-modify-write operation setting the `SPORT_CTL_A.SPENPRI` and/or `SPORT_CTL_A.SPENSEC` bits, as appropriate. DMA mode is recommended for multichannel modes of operation. For more information, see the Using DMA for SPORT Transfers programming model.

Using DMA for SPORT Transfers

DMA is supported in all SPORT operating modes (Standard DSP Serial Mode, I^2S Mode, Left-Justified Mode, Right-Justified Mode, Multichannel (TDM) Mode or Packed I^2S Mode). To enable DMA operation with the SPORT, execute the steps described in this section after initializing and enabling the SPORT. Instead of using the single word read or write operations described in the referenced programming models, the DMA engine automates accesses to the enabled SPORT data buffers.

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, it also applies to half SPORT B.

1. Follow the guidance in the multichannel (Initializing Multichannel Transfers) or non-multichannel (Initializing Core-Driven (Non-MCM) Transfers) programming models to properly initialize and enable the SPORT hardware.
2. Prepare the data buffers in memory.

   ADDITIONAL INFORMATION: Ensure that the DMA buffer is defined according to the DMA Transfers section. For the multichannel modes of operation, be sure to also consider the setting of the `SPORT_MCTL_A.MCPDE` bit, as described in the Multichannel DMA Data Packing (MCPDE) section.
3. Initialize and enable the DMA channel allocated for the SPORT, as described in the Direct Memory Access (DMA) chapter.

Using Companding as a Function

The data in the transmit and receive buffers are actually companded in place. As such, the following programming model can be used to exercise the companding hardware without transferring data, which is useful for test/debug purposes.

NOTE: This example uses half SPORT A registers. With appropriate changes to register names, this example also applies to half SPORT B.

1. Configure the SPORT as a transmitter (`SPORT_CTL_A.SPTRAN=1`) with both the primary and secondary data channels disabled (`SPORT_CTL_A.SPENPRI=0` and `SPORT_CTL_A.SPENSEC=0`).

2. Enable the desired companding scheme in the SPORT_CTL_A.DTYPE field.
3. Write a 32-bit word to one of the transmit buffers.
4. Wait two system clock cycles.

   ADDITIONAL INFORMATION: This delay is required to allow the SPORT companding hardware to reload the transmit buffer with the companded result. Any instructions that do not access the transmit buffer can be used to cause this delay.
5. Read the 8-bit compressed value from the transmit buffer written above.

To expand data in place, use the same sequence of operations with the receive buffer instead of the transmit buffer. When expanding data in this way, set the appropriate serial word length (SPORT_CTL_A.SLEN).

CM41X_M4 SPORT Register Descriptions

Serial Port (SPORT) contains the following registers.

Table 29-12: CM41X_M4 SPORT Register List

Name	Description
SPORT_CS0_A	Half SPORT 'A' Multichannel 0-31 Select Register
SPORT_CS0_B	Half SPORT 'B' Multichannel 0-31 Select Register
SPORT_CS1_A	Half SPORT 'A' Multichannel 32-63 Select Register
SPORT_CS1_B	Half SPORT 'B' Multichannel 32-63 Select Register
SPORT_CS2_A	Half SPORT 'A' Multichannel 64-95 Select Register
SPORT_CS2_B	Half SPORT 'B' Multichannel 64-95 Select Register
SPORT_CS3_A	Half SPORT 'A' Multichannel 96-127 Select Register
SPORT_CS3_B	Half SPORT 'B' Multichannel 96-127 Select Register
SPORT_CTL2_A	Half SPORT 'A' Control 2 Register
SPORT_CTL2_B	Half SPORT 'B' Control 2 Register
SPORT_CTL_A	Half SPORT 'A' Control Register
SPORT_CTL_B	Half SPORT 'B' Control Register
SPORT_DIV_A	Half SPORT 'A' Divisor Register
SPORT_DIV_B	Half SPORT 'B' Divisor Register
SPORT_ERR_A	Half SPORT 'A' Error Register
SPORT_ERR_B	Half SPORT 'B' Error Register
SPORT_MCTL_A	Half SPORT 'A' Multichannel Control Register
SPORT_MCTL_B	Half SPORT 'B' Multichannel Control Register
SPORT_MSTAT_A	Half SPORT 'A' Multichannel Status Register

Table 29-12: CM41X_M4 SPORT Register List (Continued)

Name	Description
SPORT_MSTAT_B	Half SPORT 'B' Multichannel Status Register
SPORT_RXPRI_A	Half SPORT 'A' Rx Buffer (Primary) Register
SPORT_RXPRI_B	Half SPORT 'B' Rx Buffer (Primary) Register
SPORT_RXSEC_A	Half SPORT 'A' Rx Buffer (Secondary) Register
SPORT_RXSEC_B	Half SPORT 'B' Rx Buffer (Secondary) Register
SPORT_TXPRI_A	Half SPORT 'A' Tx Buffer (Primary) Register
SPORT_TXPRI_B	Half SPORT 'B' Tx Buffer (Primary) Register
SPORT_TXSEC_A	Half SPORT 'A' Tx Buffer (Secondary) Register
SPORT_TXSEC_B	Half SPORT 'B' Tx Buffer (Secondary) Register

Half SPORT 'A' Multichannel 0-31 Select Register

Each of the bits (when set, =1) of the SPORT_CS0_A register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

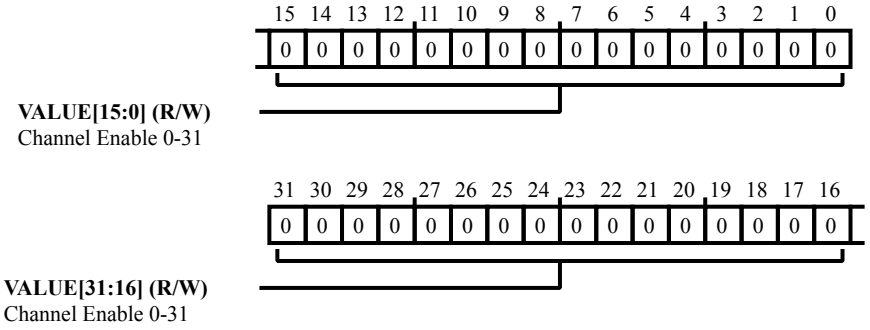


Figure 29-13: SPORT_CS0_A Register Diagram

Table 29-13: SPORT_CS0_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0-31.

Half SPORT 'B' Multichannel 0-31 Select Register

Each of the bits (when set, =1) of the SPORT_CS0_B register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

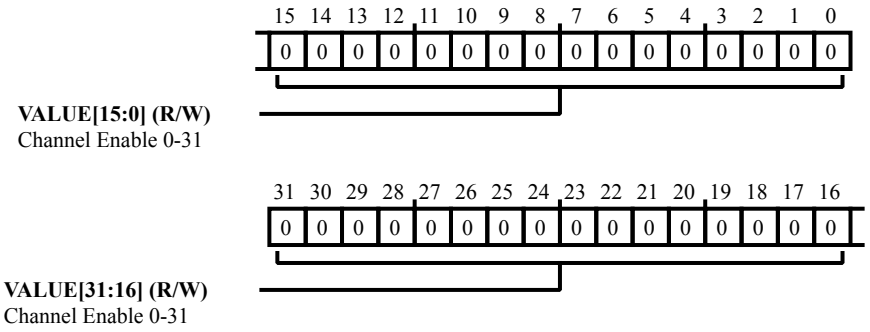


Figure 29-14: SPORT_CS0_B Register Diagram

Table 29-14: SPORT_CS0_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 0-31.

Half SPORT 'A' Multichannel 32-63 Select Register

Each of the bits (when set, =1) of the SPORT_CS1_A register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

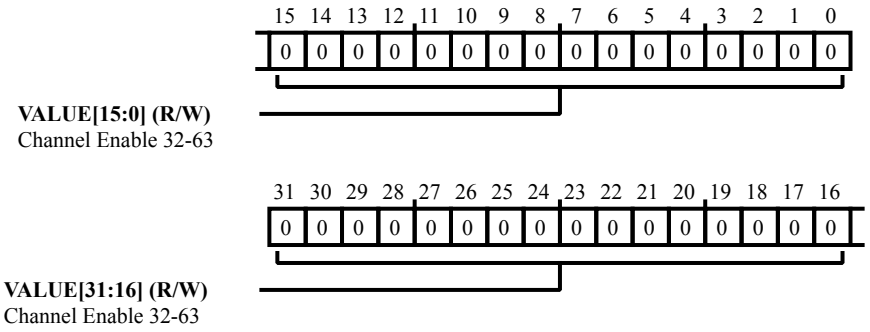


Figure 29-15: SPORT_CS1_A Register Diagram

Table 29-15: SPORT_CS1_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32-63.

Half SPORT 'B' Multichannel 32-63 Select Register

Each of the bits (when set, =1) of the SPORT_CS1_B register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

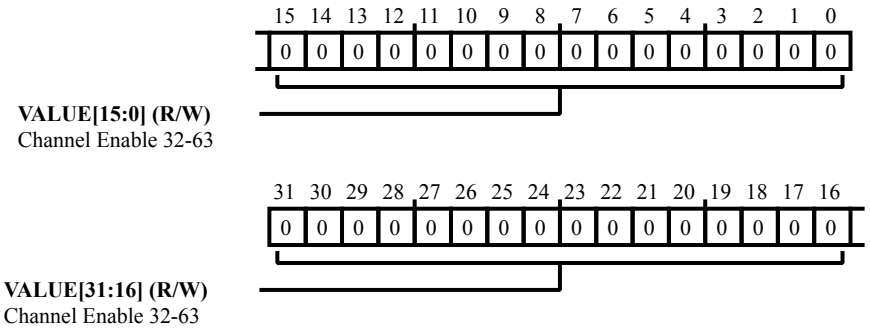


Figure 29-16: SPORT_CS1_B Register Diagram

Table 29-16: SPORT_CS1_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 32-63.

Half SPORT 'A' Multichannel 64-95 Select Register

Each of the bits (when set, =1) of the SPORT_CS2_A register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

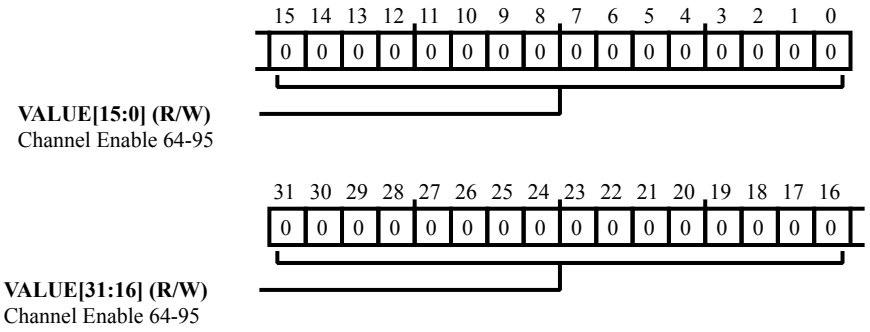


Figure 29-17: SPORT_CS2_A Register Diagram

Table 29-17: SPORT_CS2_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64-95.

Half SPORT 'B' Multichannel 64-95 Select Register

Each of the bits (when set, =1) of the SPORT_CS2_B register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

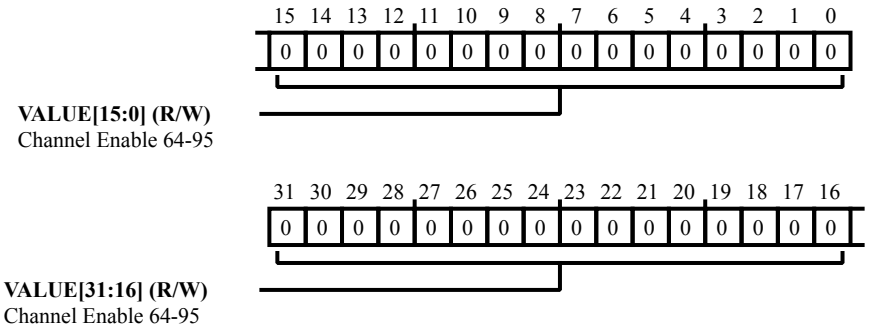


Figure 29-18: SPORT_CS2_B Register Diagram

Table 29-18: SPORT_CS2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 64-95.

Half SPORT 'A' Multichannel 96-127 Select Register

Each of the bits (when set, =1) of the SPORT_CS3_A register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

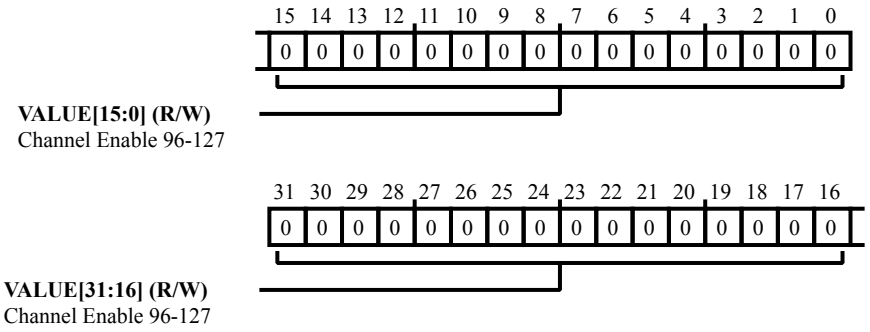


Figure 29-19: SPORT_CS3_A Register Diagram

Table 29-19: SPORT_CS3_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96-127.

Half SPORT 'B' Multichannel 96-127 Select Register

Each of the bits (when set, =1) of the SPORT_CS3_B register correspond to an active channel for the half SPORT in multichannel mode. When the register activates a channel (corresponding bit =1), the half SPORT transmits or receives the word in that channel's position of the data stream. When the register deactivates a channel (corresponding bit =0), the half SPORT either three-states its data transmit pin (during the channel's transmit time slot) or ignores incoming data (during the channel's receive time slot).

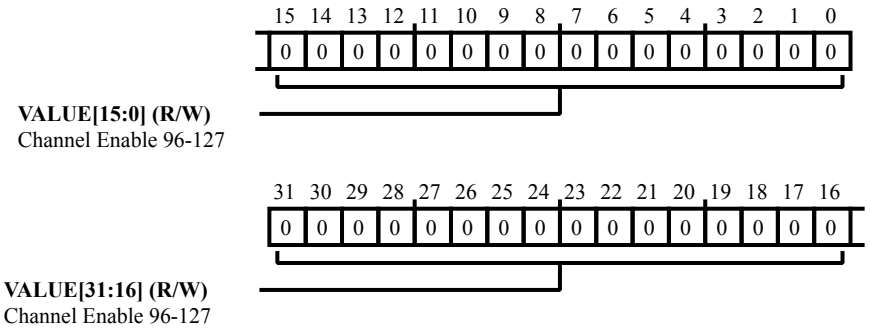


Figure 29-20: SPORT_CS3_B Register Diagram

Table 29-20: SPORT_CS3_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Channel Enable 96-127.

Half SPORT 'A' Control 2 Register

The SPORT_CTL2_A register controls multiplexing options for sharing serial clock and frame sync signals across the related half SPORTs.

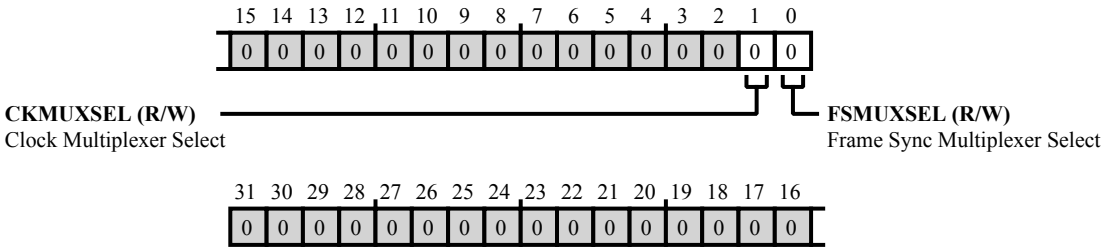


Figure 29-21: SPORT_CTL2_A Register Diagram

Table 29-21: SPORT_CTL2_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The SPORT_CTL2_A.CKMUXSEL bit enables multiplexing of the half SPORT' serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if the SPORT_CTL2_A.CKMUXSEL bit is enabled, half SPORT 'A' uses SPORT_BCLK instead of SPORT_ACLK.	
		0	Disable serial clock multiplexing
		1	Enable serial clock multiplexing
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The SPORT_CTL2_A.FSMUXSEL bit enables multiplexing of the half SPORT' frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if the SPORT_CTL2_A.FSMUXSEL bit is enabled, half SPORT 'A' uses SPORT_BFS instead of SPORT_AFS.	
		0	Disable frame sync multiplexing
		1	Enable frame sync multiplexing

Half SPORT 'B' Control 2 Register

The SPORT_CTL2_B register controls multiplexing options for sharing serial clock and frame sync signals across the related half SPORTs.

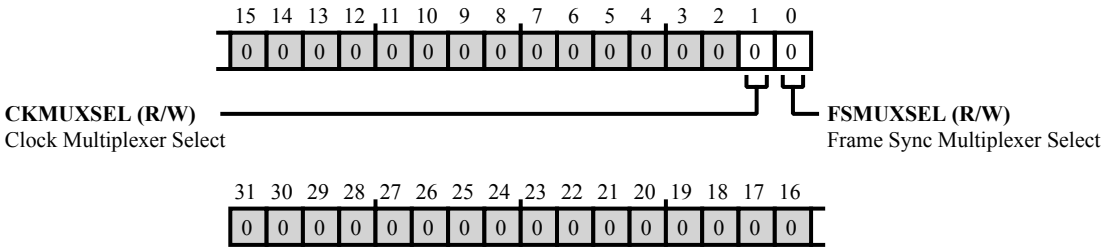


Figure 29-22: SPORT_CTL2_B Register Diagram

Table 29-22: SPORT_CTL2_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	CKMUXSEL	Clock Multiplexer Select. The SPORT_CTL2_B.CKMUXSEL bit enables multiplexing of the half SPORT' serial clock. In this mode, the serial clock of the related half SPORT is used instead of the half SPORT's own serial clock. For example, if the SPORT_CTL2_B.CKMUXSEL bit is enabled, half SPORT 'B' uses SPORT_ACLK instead of SPORT_BCLK.	
		0	Disable serial clock multiplexing
		1	Enable serial clock multiplexing
0 (R/W)	FSMUXSEL	Frame Sync Multiplexer Select. The SPORT_CTL2_B.FSMUXSEL bit enables multiplexing of the half SPORT' frame sync. In this mode, the frame sync of the related half SPORT is used instead of the half SPORT's own frame sync. For example, if the SPORT_CTL2_B.FSMUXSEL bit is enabled, half SPORT 'B' uses SPORT_AFS instead of SPORT_BFS.	
		0	Disable frame sync multiplexing
		1	Enable frame sync multiplexing

Half SPORT 'A' Control Register

The SPORT_CTL_A register contains transmit and receive control bits for SPORT half 'A', including serial port mode selection for the half SPORT's primary and secondary channels. The function of some bits in the SPORT_CTL_A register vary depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

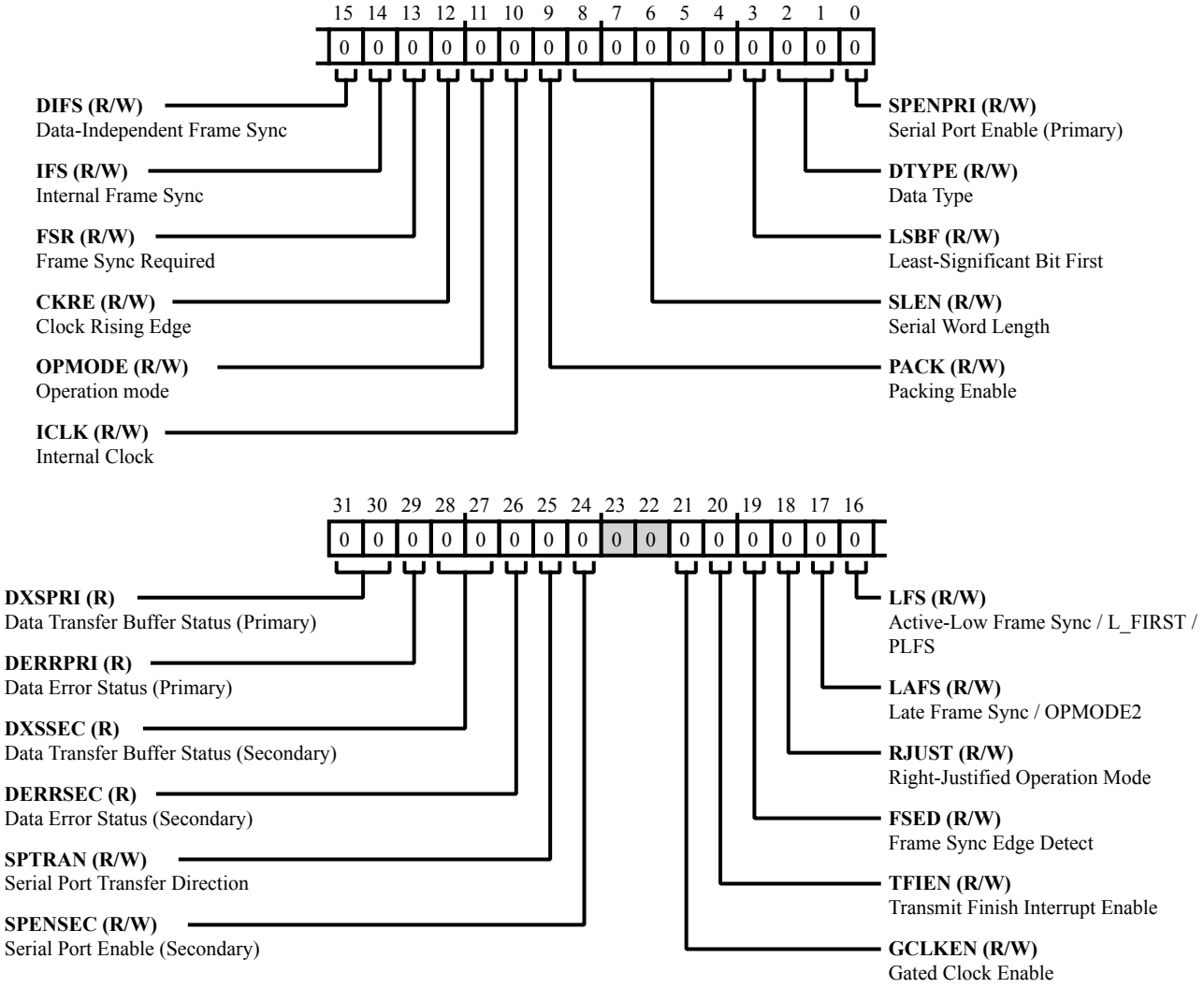


Figure 29-23: SPORT_CTL_A Register Diagram

Table 29-23: SPORT_CTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The SPORT_CTL_A.DXSPRI bit field indicates the status of the half SPORT's primary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full
29 (R/NW)	DERRPRI	Data Error Status (Primary). The SPORT_CTL_A.DERRPRI bit reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_A.FSR bit =1, the SPORT_CTL_A.DERRPRI bit indicates whether the SPORT_AFS signal (from an internal or external source) occurred while the SPORT_TXPRI_A data buffer was empty (during transmit) or the SPORT_RXPRI_A data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_AFS signal. It is important to note that, as a receiver, the SPORT_CTL_A.DERRPRI bit indicates when the channel has received new data while the SPORT_RXPRI_A receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_A.FSR bit =0, the SPORT_CTL_A.DERRPRI bit is set whenever the SPORT is required to transmit while the SPORT_TXPRI_A transmit buffer is empty. It is also set whenever the SPORT is required to receive while the SPORT_RXPRI_A receive buffer is full. The SPORT clears the SPORT_CTL_A.DERRPRI bit if the SPORT_ERR_A.DERRPSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The SPORT_CTL_A.DXSSEC bit field indicates the status of the half SPORT's secondary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full

Table 29-23: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The SPORT_CTL_A.DERRSEC bit reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_A.FSR bit =1, the SPORT_CTL_A.DERRSEC bit indicates whether the SPORT_AFS signal (from an internal or external source) occurred while the SPORT_TXSEC_A data buffer was empty (during transmit) or the SPORT_RXSEC_A data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_AFS signal. It is important to note that, as a receiver, the SPORT_CTL_A.DERRSEC bit indicates when the channel has received new data while the SPORT_RXSEC_A receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_A.FSR bit =0, the SPORT_CTL_A.DERRSEC bit is set whenever the SPORT is required to transmit while the SPORT_TXSEC_A transmit buffer is empty. It is also set whenever the SPORT is required to receive while the SPORT_RXSEC_A receive buffer is full. The SPORT clears the SPORT_CTL_A.DERRSEC bit if the SPORT_ERR_A.DERRSSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The SPORT_CTL_A.SPTRAN bit selects the transfer direction (receive or transmit) for the half SPORT's primary and secondary channels. When the direction is receive, the half SPORT activates the receive buffers, and the SPORT_ACLK and SPORT_AFS pins control the receive buffers. The transmit buffers are inactive when the half SPORT's transfer direction is receive. When the direction is transmit, the half SPORT activates the transmit buffers, and the SPORT_ACLK and SPORT_AFS pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit.	
		0	Receive
		1	Transmit
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The SPORT_CTL_A.SPENSEC bit enables the half SPORT's secondary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Table 29-23: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	GCLKEN	Gated Clock Enable. The SPORT_CTL_A.GCLKEN bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (SPORT_CTL_A.OPMODE = 0 or 1). This bit is ignored when the half SPORT is in right-justified mode (SPORT_CTL_A.RJUST =1) or multichannel mode (SPORT_MCTL_A.MCE =1). When the SPORT_CTL_A.GCLKEN bit is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).	
		0	Disable
		1	Enable
20 (R/W)	TFIEN	Transmit Finish Interrupt Enable. The SPORT_CTL_A.TFIEN bit selects when the half SPORT issues its transmission complete interrupt request, if a DMA complete interrupt request is enabled by the DMA_CFG.INT configuration. When enabled (SPORT_CTL_A.TFIEN =1), the DMA complete peripheral interrupt request is generated when the last bit of last word in the DMA is shifted out. When disabled (SPORT_CTL_A.TFIEN =0), the DMA interrupt request is generated when the DMA counter expires (the last word goes to the transmit buffer).	
		0	Last word sent (DMA count done) interrupt
		1	Last bit sent (Tx buffer done) interrupt
19 (R/W)	FSED	Frame Sync Edge Detect. The SPORT_CTL_A.FSED bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The SPORT_CTL_A.FSED bit may be enabled even during an active frame sync, and the half SPORT starts the transfer on the next valid rising or falling edge of external frame sync. If disabled (SPORT_CTL_A.FSED =0), the half SPORT operates in the standard level-sensitive detection mode for external frame sync.	
		0	Level detect frame sync
		1	Edge detect frame sync

Table 29-23: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	RJUST	Right-Justified Operation Mode. The SPORT_CTL_A.RJUST bit enables the half SPORT (if SPORT_CTL_A.OPMODE =1) to transfer data in right-justified operation mode. In this mode, the half SPORT aligns data to the end of the frame sync, rather than the start of the frame sync. When using right-justified mode, systems should program an appropriate delay count to introduce a clock delay before the half SPORT state machine starts to capture data. This value is set in the DCNT field (right-justified mode usage of the SPORT_MCTL_A.WOFFSET field). For information about appropriate delay selections, see the SPORT operating modes section.	
		0	Disable
		1	Enable
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (SPORT_CTL_A.OPMODE =0) or in right-justified mode (SPORT_CTL_A.RJUST =1), the SPORT_CTL_A.LAFS bit selects whether the half SPORT generates a late frame sync (SPORT_AFS during first data bit) or generates an early frame sync signal (SPORT_AFS during serial clock cycle before first data bit). When the half SPORT is in I^2S / left-justified mode (SPORT_CTL_A.OPMODE =1), the SPORT_CTL_A.LAFS bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I^2S mode. When the half SPORT is in multichannel mode (SPORT_MCTL_A.MCE =1), the SPORT_CTL_A.LAFS bit is reserved.	
		0	Early frame sync (or I^2S mode)
		1	Late frame sync (or left-justified mode)
16 (R/W)	LFS	Active-Low Frame Sync / L_FIRST / PLFS. When the half SPORT is in DSP standard mode and multichannel mode (SPORT_CTL_A.OPMODE =0), the SPORT_CTL_A.LFS bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I^2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1), the SPORT_CTL_A.LFS bit acts as L_FIRST, selecting whether the half SPORT transfers data first for the left or right channel.	
		0	Active high frame sync (DSP standard mode) or rising edge frame sync (multichannel mode) or right channel first (I^2S/packed mode) or left channel first (left-justified mode)
		1	Active low frame sync (DSP standard mode) or falling edge frame sync (multichannel mode) or left channel first (I^2S/packed mode) or right channel first (left-justified mode)

Table 29-23: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	DIFS	Data-Independent Frame Sync. The SPORT_CTL_A.DIFS bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by the SPORT_DIV_A.FSDIV bit. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full. Note that the SPORT_CTL_A.DIFS bit is automatically set when the half SPORT is in packed or multichannel modes.	
		0	Data-dependent frame sync
		1	Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The SPORT_CTL_A.IFS bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.	
		0	External frame sync
		1	Internal frame sync
13 (R/W)	FSR	Frame Sync Required. The SPORT_CTL_A.FSR bit selects whether or not the half SPORT requires frame sync for data transfer. This bit is automatically set when the half SPORT is in I^2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1) or is in multichannel mode (SPORT_MCTL_A.MCE =1).	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The SPORT_CTL_A.CKRE bit selects the rising or falling edge of the SPORT_ACLK clock for the half SPORT to sample receive data and frame sync. Note that the half SPORT changes the state of transmit data and frame sync signals on the non-selected edge of the SPORT_ACLK. Also, note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for the SPORT_CTL_A.CKRE bit. This programming drives the internally-generated signals on one edge of SPORT_ACLK and samples the received signals on the opposite edge.	
		0	Clock falling edge
		1	Clock rising edge

Table 29-23: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	OPMODE	Operation mode. The SPORT_CTL_A.OPMODE bit selects whether the half SPORT operates in DSP standard/multichannel mode or operates in I^2S/packed/left-justified mode. The mode selection affects the operation of the SPORT_CTL_A.LAFS and SPORT_CTL_A.LFS bits. Also, the SPORT_CTL_A.OPMODE bit enables or disables operation of the SPORT_CTL_A.GCLKEN, SPORT_CTL_A.FSED, SPORT_CTL_A.RJUST, SPORT_CTL_A.DIFS, SPORT_CTL_A.FSR, and SPORT_CTL_A.CKRE bits.	
		0	DSP standard/multichannel mode
		1	I^2S/packed/left-justified mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (SPORT_CTL_A.OPMODE =0), the SPORT_CTL_A.ICLK bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the SPORT_ACLK clock signal, and SPORT_ACLK is an output. The SPORT_DIV_A.CLKDIV serial clock divisor value determines the clock frequency. For internal clock disabled, the SPORT_ACLK clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.	
		0	External clock
		1	Internal clock
9 (R/W)	PACK	Packing Enable. The SPORT_CTL_A.PACK bit enables the half SPORT to perform 16- to 32-bit packing on received data and to perform 32- to 16-bit unpacking on transmitted data. The receive packing operation packs two successive received words into a single 32-bit word. The transmit unpacking operation unpacks each 32-bit word and transmits it as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15:0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31:16. This format applies to both receive (packing) and transmit (unpacking) operations. Companding may be used with word packing or unpacking. The half SPORT generates data transfer related interrupts when packing is enabled. The transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.	
		0	Disable
		1	Enable

Table 29-23: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8:4 (R/W)	SLEN	Serial Word Length. The SPORT_CTL_A.SLEN bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is: SPORT_CTL_A.SLEN = (serial word length in bits) - 1 For DSP standard mode (SPORT_CTL_A.OPMODE =0), use SPORT_CTL_A.SLEN of 3 to 31 bits. For I^2S / packed / left-justified mode (SPORT_CTL_A.OPMODE =1), use SPORT_CTL_A.SLEN of 4 to 31 bits.	
3 (R/W)	LSBF	Least-Significant Bit First. The SPORT_CTL_A.LSBF bit selects whether the half SPORT transmits or receives data LSB first or MSB first.	
		0	MSB first sent/received (big endian)
		1	LSB first sent/received (little endian)
2:1 (R/W)	DTYPE	Data Type. The SPORT_CTL_A.DTYPE bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (SPORT_CTL_A.OPMODE =0).	
		0	Right-justify data, zero-fill unused MSBs
		1	Right-justify data, sign-extend unused MSBs
		2	u-law compand data
		3	A-law compand data
0 (R/W)	SPENPRI	Serial Port Enable (Primary). The SPORT_CTL_A.SPENPRI bit enables the half SPORT's primary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Half SPORT 'B' Control Register

The SPORT_CTL_B register contains transmit and receive control bits for SPORT half 'B', including serial port mode selection for the half SPORT's primary and secondary channels. The function of some bits in the SPORT_CTL_B register vary, depending on the SPORT's operating mode. For more information, see the SPORT operating modes description. If reading reserved bits, the read value is the last written value to these bits or is the reset value of these bits.

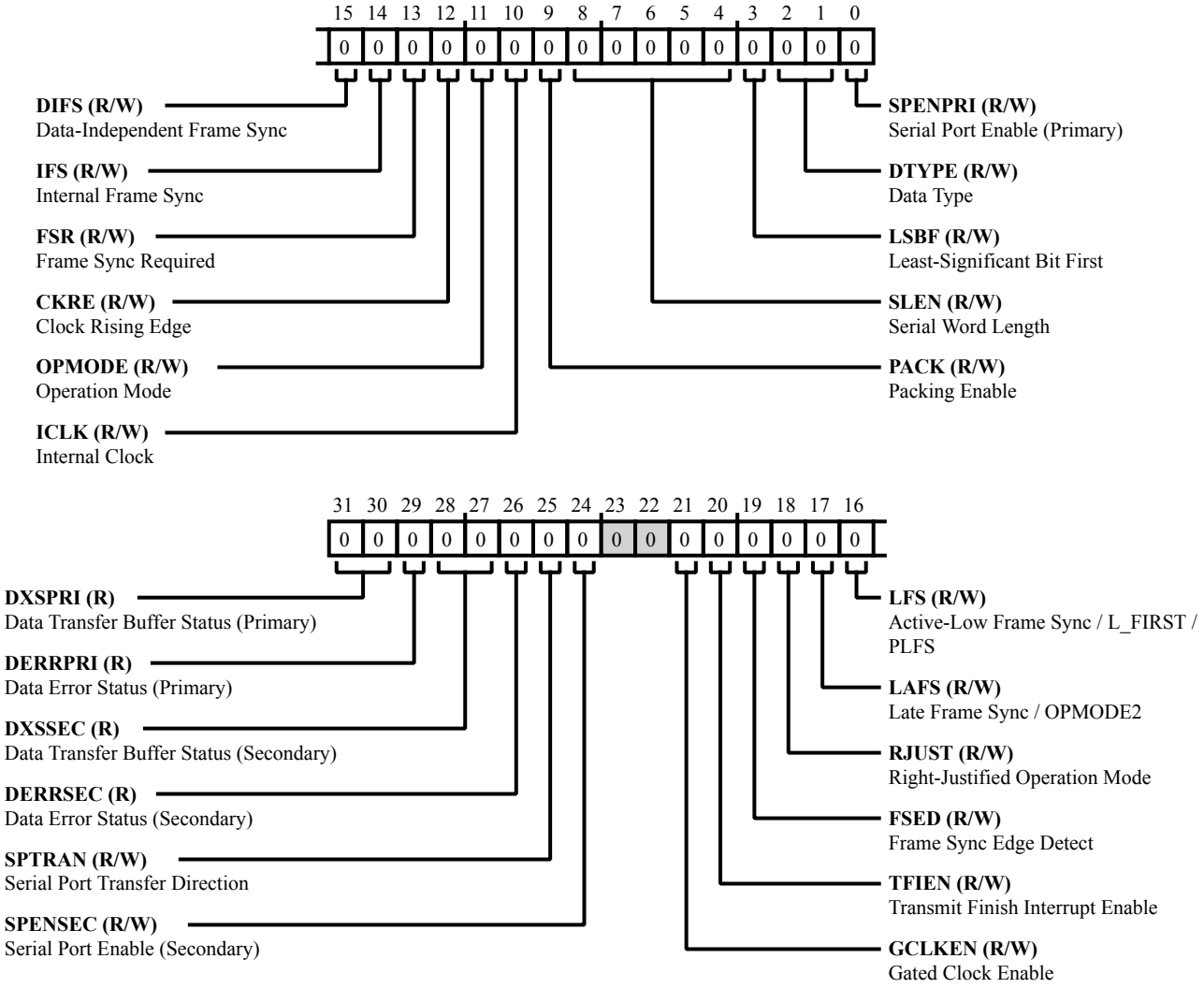


Figure 29-24: SPORT_CTL_B Register Diagram

Table 29-24: SPORT_CTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/NW)	DXSPRI	Data Transfer Buffer Status (Primary). The SPORT_CTL_B.DXSPRI bit field indicates the status of the half SPORT's primary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full
29 (R/NW)	DERRPRI	Data Error Status (Primary). The SPORT_CTL_B.DERRPRI bit reports the half SPORT's primary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_B.FSR bit =1, the SPORT_CTL_B.DERRPRI bit indicates whether the SPORT_BFS signal (from an internal or external source) occurred while the SPORT_TXPRI_B data buffer was empty (during transmit) or the SPORT_RXPRI_B data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_BFS signal. It is important to note that, as a receiver, the SPORT_CTL_B.DERRPRI bit indicates when the channel has received new data while the SPORT_RXPRI_B receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_B.FSR bit =0, the SPORT_CTL_B.DERRPRI bit is set whenever the SPORT is required to transmit while the SPORT_TXPRI_B transmit buffer is empty and is set whenever the SPORT is required to receive while the SPORT_RXPRI_B receive buffer is full. The SPORT clears the SPORT_CTL_B.DERRPRI bit if the SPORT_ERR_B.DERRPSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
28:27 (R/NW)	DXSSEC	Data Transfer Buffer Status (Secondary). The SPORT_CTL_B.DXSSEC bit field indicates the status of the half SPORT's secondary channel data buffer.	
		0	Empty
		1	Reserved
		2	Partially full
		3	Full

Table 29-24: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/NW)	DERRSEC	Data Error Status (Secondary). The SPORT_CTL_B.DERRSEC bit reports the half SPORT's secondary channel transmit underflow status or receive overflow status, depending on the SPORT transfer direction. If the SPORT_CTL_B.FSR bit =1, the SPORT_CTL_B.DERRSEC bit indicates whether the SPORT_BFS signal (from an internal or external source) occurred while the SPORT_TXSEC_B data buffer was empty (during transmit) or the SPORT_RXSEC_B data buffer was full (during receive). The SPORT transmits or receives data whenever it detects the SPORT_BFS signal. It is important to note that, as a receiver, the SPORT_CTL_B.DERRSEC bit indicates when the channel has received new data while the SPORT_RXSEC_B receive buffer is full. This new data overwrites existing data. If the SPORT_CTL_B.FSR bit =0, the SPORT_CTL_B.DERRSEC bit is set whenever the SPORT is required to transmit while the SPORT_TXSEC_B transmit buffer is empty. It is also set whenever the SPORT is required to receive while the SPORT_RXSEC_B receive buffer is full. The SPORT clears the SPORT_CTL_B.DERRSEC bit if the SPORT_ERR_B.DERRSSTAT bit is cleared.	
		0	No error
		1	Error (Tx underflow or Rx overflow)
25 (R/W)	SPTRAN	Serial Port Transfer Direction. The SPORT_CTL_B.SPTRAN bit selects the transfer direction (receive or transmit) for the half SPORT's primary and secondary channels. When the direction is receive, the half SPORT activates the receive buffers, and the SPORT_BCLK and SPORT_BFS pins control the receive buffers. The transmit buffers are inactive when the half SPORT's transfer direction is receive. When the direction is transmit, the half SPORT activates the transmit buffers, and the SPORT_BCLK and SPORT_BFS pins control the transmit shift registers. The receive buffers are inactive when the half SPORT's transfer direction is transmit.	
		0	Receive
		1	Transmit
24 (R/W)	SPENSEC	Serial Port Enable (Secondary). The SPORT_CTL_B.SPENSEC bit enables the half SPORT's secondary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Table 29-24: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	GCLKEN	Gated Clock Enable. The SPORT_CTL_B.GCLKEN bit enables gated clock operation for the half SPORT when in DSP serial mode or left-justified stereo modes (SPORT_CTL_B.OPMODE = 0 or 1). This bit is ignored when the half SPORT is in right-justified mode (SPORT_CTL_B.RJUST =1) or multichannel mode (SPORT_MCTL_B.MCE =1). When SPORT_CTL_B.GCLKEN is enabled, the SPORT clock is active when the SPORT is transferring data or when the frame sync changes (transitions to active state).	
		0	Disable
		1	Enable
20 (R/W)	TFIEN	Transmit Finish Interrupt Enable. The SPORT_CTL_B.TFIEN bit selects when the half SPORT issues its transmission complete interrupt request, if a DMA complete interrupt request is enabled by the DMA_CFG.INT configuration. When enabled (SPORT_CTL_B.TFIEN =1), the DMA complete peripheral interrupt request is generated when the last bit of last word in the DMA is shifted out. When disabled (SPORT_CTL_B.TFIEN =0), the DMA interrupt request is generated when the DMA counter expires (the last word goes to the transmit buffer).	
		0	Last word sent (DMA count done) interrupt
		1	Last bit sent (Tx buffer done) interrupt
19 (R/W)	FSED	Frame Sync Edge Detect. The SPORT_CTL_B.FSED bit enables the half SPORT to start transmitting or receiving after detecting an active edge of an external frame sync. The SPORT_CTL_B.FSED may be enabled even during an active frame sync, and the half SPORT starts the transfer on the next valid rising or falling edge of external frame sync. If disabled (SPORT_CTL_B.FSED =0), the half SPORT operates in the standard level-sensitive detection mode for external frame sync.	
		0	Level detect frame sync
		1	Edge detect frame sync

Table 29-24: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	RJUST	Right-Justified Operation Mode. The SPORT_CTL_B.RJUST bit enables the half SPORT (if SPORT_CTL_B.OPMODE =1) to transfer data in right-justified operation mode. In this mode, the half SPORT aligns data to the end of the frame sync, rather than the start of the frame sync. When using right-justified mode, systems should program an appropriate delay count to introduce a clock delay before the half SPORT state machine starts to capture data. This value is set in the DCNT field (right-justified mode usage of the SPORT_MCTL_B.WOFFSET field). For information about appropriate delay selections, see the SPORT operating modes section.	
		0	Disable
		1	Enable
17 (R/W)	LAFS	Late Frame Sync / OPMODE2. When the half SPORT is in DSP standard mode (SPORT_CTL_B.OPMODE =0) or in right-justified mode (SPORT_CTL_B.RJUST =1), the SPORT_CTL_B.LAFS bit selects whether the half SPORT generates a late frame sync (SPORT_BFS during first data bit) or generates an early frame sync signal (SPORT_BFS during serial clock cycle before first data bit). When the half SPORT is in I^2S / left-justified mode (SPORT_CTL_B.OPMODE =1), the SPORT_CTL_B.LAFS bit acts as OPMODE2, selecting whether the half SPORT is in left-justified mode or I^2S mode. When the half SPORT is in multichannel mode (SPORT_MCTL_B.MCE =1), the SPORT_CTL_B.LAFS bit is reserved.	
		0	Early frame sync (or I^2S mode)
		1	Late frame sync (or left-justified mode)

Table 29-24: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	LFS	Active-Low Frame Sync / L_FIRST / PLFS. When the half SPORT is in DSP standard mode and multichannel mode (SPORT_CTL_B.OPMODE =0), the SPORT_CTL_B.LFS bit selects whether the half SPORT uses active low or active high frame sync. When the half SPORT is in I^2S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1), the SPORT_CTL_B.LFS bit acts as L_FIRST, selecting whether the half SPORT transfers data first for the left or right channel.	
		0	Active high frame sync (DSP standard mode) or rising edge frame sync (multichannel mode) or right channel first (I^2S/packed mode) or left channel first (left-justified mode)
		1	Active low frame sync (DSP standard mode) or falling edge frame sync (multichannel mode) or left channel first (I^2S/packed mode) or right channel first (left-justified mode)
15 (R/W)	DIFS	Data-Independent Frame Sync. The SPORT_CTL_B.DIFS bit selects whether the half SPORT uses a data-independent or data-dependent frame sync. When using a data-independent frame sync, the half SPORT generates the sync at the interval selected by SPORT_DIV_B.FSDIV. When using a data-dependent frame sync, the half SPORT generates the sync on the selected interval when the transmit buffer is not empty or when the receive buffer is not full. Note that the SPORT_CTL_B.DIFS bit is automatically set when the half SPORT is in packed or multichannel modes.	
		0	Data-dependent frame sync
		1	Data-independent frame sync
14 (R/W)	IFS	Internal Frame Sync. The SPORT_CTL_B.IFS bit selects whether the half SPORT uses an internal frame sync or uses an external frame sync. Note that the externally-generated frame sync does not need to be synchronous with the processor's system clock.	
		0	External frame sync
		1	Internal frame sync

Table 29-24: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	FSR	Frame Sync Required. The SPORT_CTL_B.FSR selects whether or not the half SPORT requires frame sync for data transfer. This bit is automatically set when the half SPORT is in I^2S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1) or is in multichannel mode (SPORT_MCTL_B.MCE =1).	
		0	No frame sync required
		1	Frame sync required
12 (R/W)	CKRE	Clock Rising Edge. The SPORT_CTL_B.CKRE selects the rising or falling edge of the SPORT_BCLK clock for the half SPORT to sample receive data and frame sync. Note that the half SPORT changes the state of transmit data and frame sync signals on the non-selected edge of the SPORT_BCLK. Also note that the transmit and receive related SPORT halves (A and B) should be programmed with the same value for SPORT_CTL_B.CKRE. This programming drives the internally-generated signals on one edge of SPORT_BCLK and samples the received signals on the opposite edge.	
		0	Clock falling edge
		1	Clock rising edge
11 (R/W)	OPMODE	Operation Mode. The SPORT_CTL_B.OPMODE bit selects whether the half SPORT operates in DSP standard / multichannel mode or operates in I^2S / packed / left-justified mode. The mode selection affects the operation of the SPORT_CTL_B.LAFS and SPORT_CTL_B.LFS bits. Also, the SPORT_CTL_B.OPMODE bit enables or disables operation of the SPORT_CTL_B.GCLKEN, SPORT_CTL_B.FSED, SPORT_CTL_B.RJUST, SPORT_CTL_B.DIFS, SPORT_CTL_B.FSR, and SPORT_CTL_B.CKRE bits.	
		0	DSP standard/multichannel mode
		1	I^2S/packed/left-justified mode
10 (R/W)	ICLK	Internal Clock. When the half SPORT is in DSP standard mode (SPORT_CTL_B.OPMODE =0), the SPORT_CTL_B.ICLK bit selects whether the half SPORT uses an internal or external clock. For internal clock enabled, the half SPORT generates the SPORT_BCLK clock signal, and the SPORT_BCLK is an output. The SPORT_DIV_B.CLKDIV serial clock divisor value determines the clock frequency. For internal clock disabled, the SPORT_BCLK clock signal is an input, and the serial clock divisor is ignored. Note that the externally-generated serial clock does not need to be synchronous with the processor's system clock.	
		0	External clock
		1	Internal clock

Table 29-24: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	PACK	Packing Enable. The SPORT_CTL_B.PACK bit enables the half SPORT to perform 16- to 32-bit packing on received data and to perform 32- to 16-bit unpacking on transmitted data. The receive packing operation packs two successive received words into a single 32-bit word. The transmit unpacking operation unpacks each 32-bit word and transmits it as two 16-bit words. The first 16-bit (or smaller) word is right-justified in bits 15:0 of the packed word, and the second 16-bit (or smaller) word is right-justified in bits 31:16. This format applies to both receive (packing) and transmit (unpacking) operations. Companding may be used with word packing or unpacking. The half SPORT generates data transfer related interrupts when packing is enabled. The transmit and receive interrupts are generated for the 32-bit packed words, not for each 16-bit word.	
		0	Disable
		1	Enable
8:4 (R/W)	SLEN	Serial Word Length. The SPORT_CTL_B.SLEN bits selects word length in bits for the half SPORT's data transfers. Word may be from 4- to 32-bits in length. The formula for selecting the word length in bits is: SPORT_CTL_B.SLEN = (serial word length in bits) - 1 For DSP standard mode (SPORT_CTL_B.OPMODE =0), use SPORT_CTL_B.SLEN of 3 to 31 bits. For I^2S / packed / left-justified mode (SPORT_CTL_B.OPMODE =1), use SPORT_CTL_B.SLEN of 4 to 31 bits.	
3 (R/W)	LSBF	Least-Significant Bit First. The SPORT_CTL_B.LSBF bit selects whether the half SPORT transmits or receives data LSB first or MSB first.	
		0	MSB first sent/received (big endian)
		1	LSB first sent/received (little endian)
2:1 (R/W)	DTYPE	Data Type. The SPORT_CTL_B.DTYPE bits selects the data type formatting for the half SPORT's data transfers in DSP standard mode (SPORT_CTL_B.OPMODE =0).	
		0	Right-justify data, zero-fill unused MSBs
		1	Right-justify data, sign-extend unused MSBs
		2	u-law compand data
		3	A-law compand data

Table 29-24: SPORT_CTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	SPENPRI	Serial Port Enable (Primary). The `SPORT_CTL_B.SPENPRI` bit enables the half SPORT's primary channel. When this bit is cleared (changes from =1 to =0), the half SPORT automatically flushes the channel's data buffers.	
		0	Disable
		1	Enable

Half SPORT 'A' Divisor Register

The SPORT_DIV_A register contains divisor values that determine frequencies of internally-generated clocks and frame syncs for half SPORT 'A'.

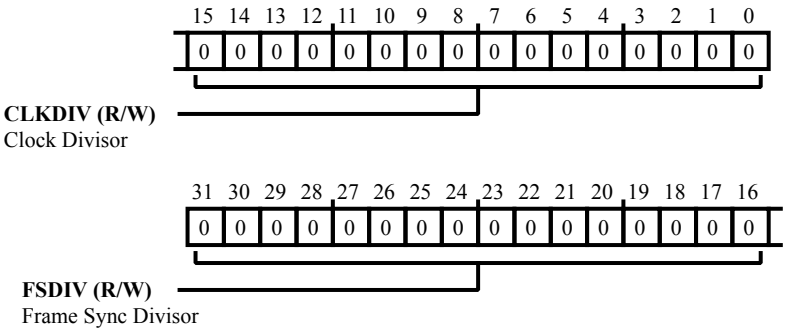


Figure 29-25: SPORT_DIV_A Register Diagram

Table 29-25: SPORT_DIV_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	Frame Sync Divisor. The SPORT_DIV_A.FSDIV bits select the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. The formula relating SPORT_DIV_A.FSDIV to the number of cycles between frame sync pulses is: SPORT_DIV_A.FSDIV = (number of serial clocks between frame syncs) - 1 Use the following equation to determine the value of SPORT_DIV_A.FSDIV, given the serial clock frequency and desired frame sync frequency: FSDIV = (SPORT_ACLK / SPORT_AFS) - 1 Note that the frame sync is continuously active when SPORT_DIV_A.FSDIV = 0. The value of SPORT_DIV_A.FSDIV should not be less than the serial word length (SPORT_CTL_A.SLEN), as this may cause an external device to abort the current operation or cause other unpredictable results.
15:0 (R/W)	CLKDIV	Clock Divisor. The SPORT_DIV_A.CLKDIV bits select the divisor that the half SPORT uses to calculate the serial clock (SPORT_ACLK) from the processor system clock (SCLK). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (SPORT_CTL_A.ICLK =1), legal SPORT_DIV_A.CLKDIV values are 0 to 65535. Given the processor system clock frequency and desired serial clock frequency, use the following formula to calculate the value of SPORT_DIV_A.CLKDIV: CLKDIV = (SCLK / SPORT_ACLK) - 1 For the maximum serial clock frequency, see the processor data sheet.

Half SPORT 'B' Divisor Register

The SPORT_DIV_B contains divisor values that determine frequencies of internally-generated clocks and frame syncs for SPORT half 'B'.

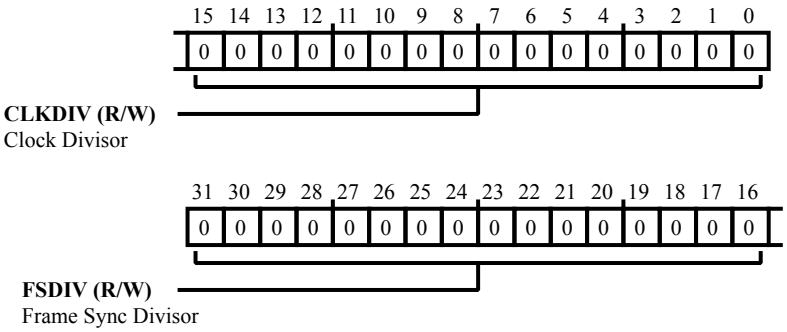


Figure 29-26: SPORT_DIV_B Register Diagram

Table 29-26: SPORT_DIV_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSDIV	Frame Sync Divisor. The SPORT_DIV_B.FSDIV bits select the number of transmit or receive clock cycles that the half SPORT counts before generating a frame sync pulse. The half SPORT counts serial clock cycles whether these are from an internally- or an externally-generated serial clock. The formula relating SPORT_DIV_B.FSDIV to the number of cycles between frame sync pulses is: SPORT_DIV_B.FSDIV = (number of serial clocks between frame syncs) - 1 Use the following equation to determine the value of SPORT_DIV_B.FSDIV, given the serial clock frequency and desired frame sync frequency: FSDIV = (SPORT_BCLK / SPORT_BFS) - 1 Note that the frame sync is continuously active when SPORT_DIV_B.FSDIV = 0. The value of SPORT_DIV_B.FSDIV should not be less than the serial word length (SPORT_CTL_B.SLEN), as this may cause an external device to abort the current operation or cause other unpredictable results.
15:0 (R/W)	CLKDIV	Clock Divisor. The SPORT_DIV_B.CLKDIV bits select the divisor that the half SPORT uses to calculate the serial clock (SPORT_BCLK) from the processor system clock (SCLK). The divisor is a 16-bit value, allowing a wide range of serial clock rates. When configured for internal clock (SPORT_CTL_B.ICLK =1), legal SPORT_DIV_B.CLKDIV values are 0 to 65535. Given the processor system clock frequency and desired serial clock frequency, use the following formula to calculate the value of SPORT_DIV_B.CLKDIV: CLKDIV = (SCLK SPORT_BCLK) - 1 For the maximum serial clock frequency, see the processor data sheet.

Half SPORT 'A' Error Register

The SPORT_ERR_A register contains error status and error interrupt mask bits for SPORT half 'A', including error handling bits for the half SPORT's primary and secondary channels and frame sync. Detected errors are frame sync violations or buffer over/underflow conditions.

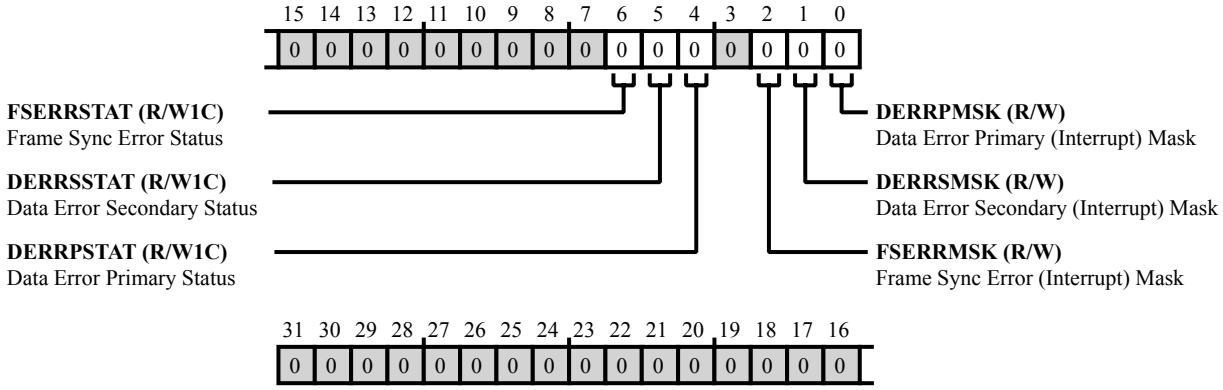


Figure 29-27: SPORT_ERR_A Register Diagram

Table 29-27: SPORT_ERR_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	FSERRSTAT	Frame Sync Error Status. The SPORT_ERR_A.FSERRSTAT bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero. When a half SPORT is receiving or transmitting, its bit count is set to a word length (for example, SPORT_CTL_A.SLEN = 31). After each serial clock edge, the half SPORT decrements the transfer's bit count. After the word is received or transmitted, the transfer's bit count reaches zero, and the half SPORT resets it (for example, to 32) on next frame sync. Normal SPORT data transfers always have a non-zero bit count value when active transmission or reception is occurring. Normal SPORT frame syncs occur after the bit count becomes zero.	
		0	No error
		1	Error (non-zero bit count at frame sync)

Table 29-27: SPORT_ERR_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	DERRSSTAT	Data Error Secondary Status. The SPORT_ERR_A.DERRSSTAT bit indicates the error status for the half SPORT's secondary channel data buffers. During transmit (SPORT_CTL_A.SPTRAN =1), the SPORT_ERR_A.DERRSSTAT bit indicates the transmit underflow status. During receive (SPORT_CTL_A.SPTRAN =0), the SPORT_ERR_A.DERRSSTAT bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a secondary data error. This bit can also be used to clear the read-only SPORT_CTL_A.DERRSEC status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
4 (R/W1C)	DERRPSTAT	Data Error Primary Status. The SPORT_ERR_A.DERRPSTAT bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (SPORT_CTL_A.SPTRAN =1), the SPORT_ERR_A.DERRPSTAT bit indicates the transmit underflow status. During receive (SPORT_CTL_A.SPTRAN =0), the SPORT_ERR_A.DERRPSTAT bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a primary data error. This bit can also be used to clear the read-only SPORT_CTL_A.DERRPRI status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The SPORT_ERR_A.FSERRMSK unmasks (enables) the half SPORT to generate the frame sync error interrupt request.	
		0	Mask (disable)
		1	Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The SPORT_ERR_A.DERRSMSK unmasks (enables) the half SPORT to generate the data error interrupt request for the secondary channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The SPORT_ERR_A.DERRPMSK unmasks (enables) the half SPORT to generate the data error interrupt request for the primary channel.	
		0	Mask (disable)
		1	Unmask (enable)

Half SPORT 'B' Error Register

The SPORT_ERR_B register contains error status and error interrupt mask bits for SPORT half 'B', including error handling bits for the half SPORT's primary and secondary channels and frame sync. Detected errors are frame sync violations or buffer over/underflow conditions.

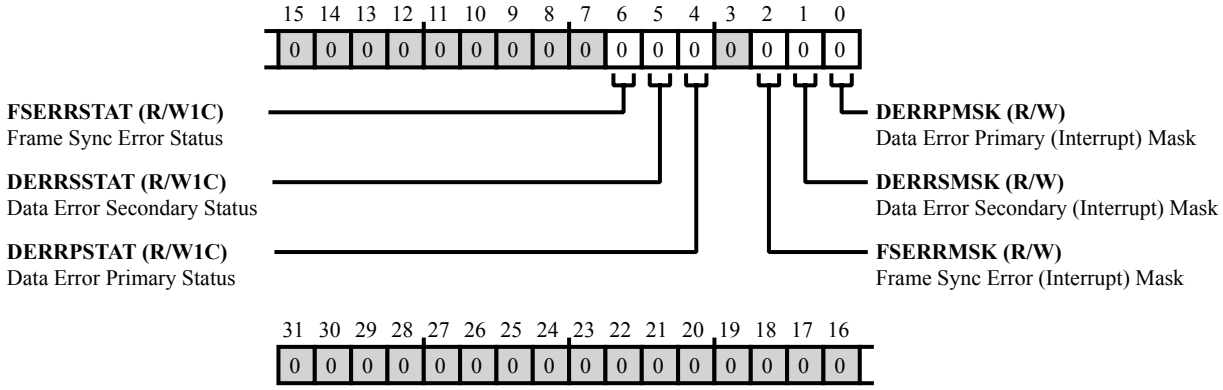


Figure 29-28: SPORT_ERR_B Register Diagram

Table 29-28: SPORT_ERR_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	FSERRSTAT	Frame Sync Error Status. The SPORT_ERR_B.FSERRSTAT bit indicates that the half SPORT has detected a frame sync when the bit count (bits remaining in the frame) is non-zero. When a half SPORT is receiving or transmitting, its bit count is set to a word length (for example, SPORT_CTL_B.SLEN = 31). After each serial clock edge, the half SPORT decrements the transfer's bit count. After the word is received or transmitted, the transfer's bit count reaches zero, and the half SPORT resets it (for example, to 32) on next frame sync. Normal SPORT data transfers always have a non-zero bit count value when active transmission or reception is occurring. Normal SPORT frame syncs occur after the bit count becomes zero.	
		0	No error
		1	Error (non-zero bit count at frame sync)

Table 29-28: SPORT_ERR_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	DERRSSTAT	Data Error Secondary Status. The SPORT_ERR_B.DERRSSTAT bit indicates the error status for the half SPORT's secondary channel data buffers. During transmit (SPORT_CTL_B.SPTRAN =1), SPORT_ERR_B.DERRSSTAT indicates the transmit underflow status. During receive (SPORT_CTL_B.SPTRAN =0), SPORT_ERR_B.DERRSSTAT indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a secondary data error. This bit can also be used to clear the read-only SPORT_CTL_B.DERRSEC status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
4 (R/W1C)	DERRPSTAT	Data Error Primary Status. The SPORT_ERR_B.DERRPSTAT bit indicates the error status for the half SPORT's primary channel data buffers. During transmit (SPORT_CTL_B.SPTRAN =1), the SPORT_ERR_B.DERRPSTAT bit indicates the transmit underflow status. During receive (SPORT_CTL_B.SPTRAN =0), the SPORT_ERR_B.DERRPSTAT bit indicates the receive overflow status. This bit is used to clear the latch of SPORT status interrupt request when triggered by a primary data error. This bit can also be used to clear the read-only SPORT_CTL_B.DERRPRI status bit.	
		0	No error
		1	Error (transmit underflow or receive overflow)
2 (R/W)	FSERRMSK	Frame Sync Error (Interrupt) Mask. The SPORT_ERR_B.FSERRMSK unmasks (enables) the half SPORT to generate the frame sync error interrupt request.	
		0	Mask (disable)
		1	Unmask (enable)
1 (R/W)	DERRSMSK	Data Error Secondary (Interrupt) Mask. The SPORT_ERR_B.DERRSMSK unmasks (enables) the half SPORT to generate the data error interrupt request for the secondary channel.	
		0	Mask (disable)
		1	Unmask (enable)
0 (R/W)	DERRPMSK	Data Error Primary (Interrupt) Mask. The SPORT_ERR_B.DERRPMSK unmasks (enables) the half SPORT to generate the data error interrupt request for the primary channel.	
		0	Mask (disable)
		1	Unmask (enable)

Half SPORT 'A' Multichannel Control Register

The SPORT_MCTL_A register controls the half SPORT's multichannel operations. This register enables multichannel operation, enables multichannel data packing, selects the multichannel frame delay, selects the number of multichannel slots, and selects the multichannel window offset size.

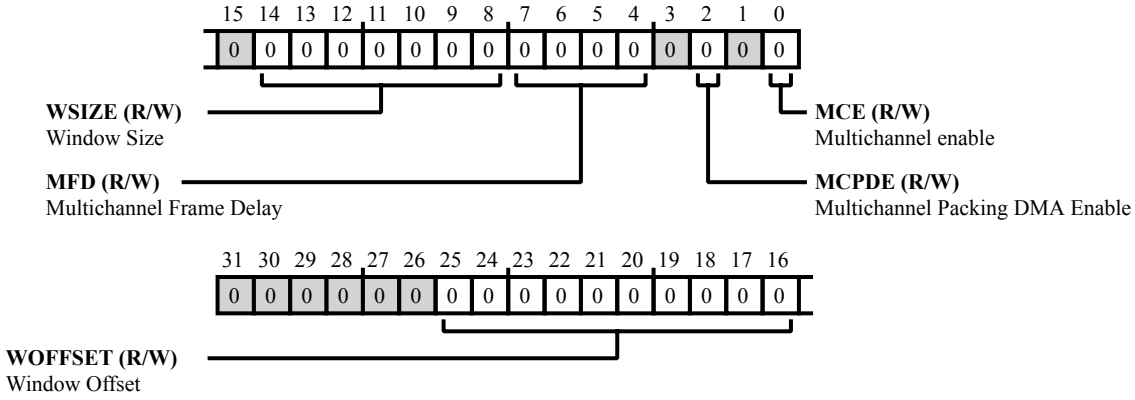


Figure 29-29: SPORT_MCTL_A Register Diagram

Table 29-29: SPORT_MCTL_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The SPORT_MCTL_A.WOFFSET bits select the start location for the half SPORT's active window of channels within the 1024-channel range. A value of 0 specifies no offset and 896 is the largest value that permits using all 128 channels. When multichannel mode is disabled (SPORT_MCTL_A.MCE =0) and the right-justified mode is enabled (SPORT_CTL_A.RJUST =1), the least significant 6 bits of SPORT_MCTL_A.WOFFSET serve as the delay count (DCNT) field. These bits introduce a clock delay before the half SPORT state machine starts to capture data. For information about appropriate delay selections, see the SPORT operating modes section.
14:8 (R/W)	WSIZE	Window Size. The SPORT_MCTL_A.WSIZE bits select the window size for the half SPORT's active window of channels. Use the following formula to calculate the window size value: SPORT_MCTL_A.WSIZE = (number of channel slots) -1
7:4 (R/W)	MFD	Multichannel Frame Delay. The SPORT_MCTL_A.MFD bits select the delay (in serial clock cycles) between the half SPORT's multichannel frame sync pulse and channel 0. The 4-bit field allows selecting multichannel frame delay of 0-15 serial clocks.

Table 29-29: SPORT_MCTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MCPDE	Multichannel Packing DMA Enable. The SPORT_MCTL_A.MCPDE bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multichannel data transfers.	
		0	Disable
		1	Enable
0 (R/W)	MCE	Multichannel enable. The SPORT_MCTL_A.MCE bit enables multichannel operations for the half SPORT. The half SPORT is configured in normal multichannel mode if SPORT_CTL_A.OPMODE=0; while it is configured in packed mode if SPORT_CTL_A.OPMODE=1. When configuring in these modes, the multichannel enable bit (SPORT_MCTL_A.MCE) should be set before enabling the SPORT data channel enable bits (SPORT_CTL_A.SPENPRI and/or SPORT_CTL_A.SPENSEC). When these channel bits transition from 1 to 0, note that the half SPORT's data transfer buffers are cleared, and the SPORT_CTL_A.DERRPRI and SPORT_CTL_A.DERRSEC bits are cleared.	
		0	Disable
		1	Enable

Half SPORT 'B' Multichannel Control Register

The SPORT_MCTL_B register controls the half SPORT's multichannel operations. This register enables multichannel operation, enables multichannel data packing, selects the multichannel frame delay, selects the number of multichannel slots, and selects the multichannel window offset size.

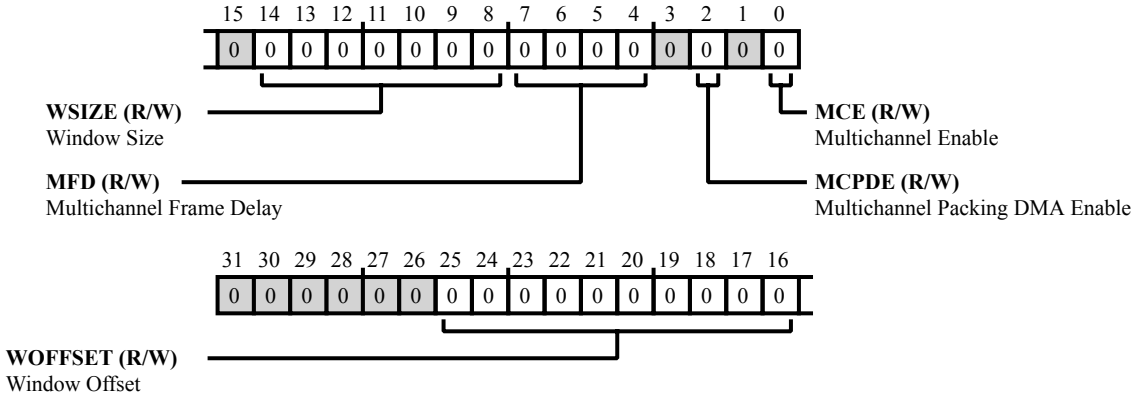


Figure 29-30: SPORT_MCTL_B Register Diagram

Table 29-30: SPORT_MCTL_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/W)	WOFFSET	Window Offset. The SPORT_MCTL_B.WOFFSET bits select the start location for the half SPORT's active window of channels within the 1024-channel range. A value of 0 specifies no offset and 896 is the largest value that permits using all 128 channels. When multichannel mode is disabled (SPORT_MCTL_B.MCE =0) and right-justified mode is enabled (SPORT_CTL_B.RJUST =1), the least significant 6 bits of SPORT_MCTL_B.WOFFSET serve as the delay count (DCNT) field. These bits introduce a clock delay before the half SPORT state machine starts to capture data. For information about appropriate delay selections, see the SPORT operating modes section.
14:8 (R/W)	WSIZE	Window Size. The SPORT_MCTL_B.WSIZE bits select the window size for the half SPORT's active window of channels. Use the following formula to calculate the window size value: SPORT_MCTL_B.WSIZE = (number of channel slots) -1
7:4 (R/W)	MFD	Multichannel Frame Delay. The SPORT_MCTL_B.MFD bits select the delay (in serial clock cycles) between the half SPORT's multichannel frame sync pulse and channel 0. The 4-bit field allows selecting a multichannel frame delay of 0-15 serial clocks.

Table 29-30: SPORT_MCTL_B Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	MCPDE	Multichannel Packing DMA Enable. The SPORT_MCTL_B.MCPDE bit enables DMA data packing for transmit and enables DMA data unpacking for the half SPORT's multichannel data transfers.	
		0	Disable
		1	Enable
0 (R/W)	MCE	Multichannel Enable. The SPORT_MCTL_B.MCE bit enables multichannel operations for the half SPORT. The half SPORT is configured in normal multichannel mode if SPORT_CTL_B.OPMODE=0; while it is configured in packed mode if SPORT_CTL_B.OPMODE=1. When configuring in these modes, the multichannel enable bit (SPORT_MCTL_B.MCE) should be set before enabling SPORT data channel enable bits (SPORT_CTL_B.SPENPRI and/or SPORT_CTL_B.SPENSEC). When these channel bits transition from 1 to 0, note that the half SPORT's data transfer buffers are cleared, and the SPORT_CTL_B.DERRPRI and SPORT_CTL_B.DERRSEC bits are cleared.	
		0	Disable
		1	Enable

Half SPORT 'A' Multichannel Status Register

The SPORT_MSTAT_A register indicates the current multichannel being serviced among the half SPORT's active channels in multichannel mode. The half SPORT increments the value by one in this register as each channel is serviced. The value in the SPORT_MSTAT_A register restarts at 0 at each frame sync.

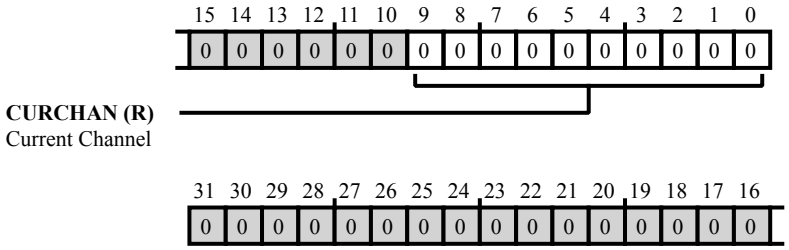


Figure 29-31: SPORT_MSTAT_A Register Diagram

Table 29-31: SPORT_MSTAT_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The SPORT_MSTAT_A.CURCHAN bits indicate the half SPORT's current channel being serviced in multichannel mode.

Half SPORT 'B' Multichannel Status Register

The SPORT_MSTAT_B register indicates the current multichannel being serviced among the half SPORT's active channels in multichannel mode. The half SPORT increments the value by one in this register as each channel is serviced. The value in the SPORT_MSTAT_B register restarts at 0 at each frame sync.

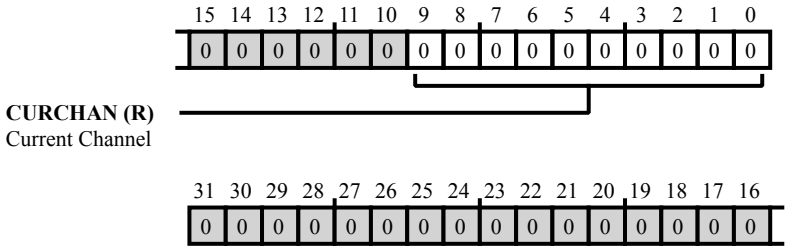


Figure 29-32: SPORT_MSTAT_B Register Diagram

Table 29-32: SPORT_MSTAT_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (R/NW)	CURCHAN	Current Channel. The SPORT_MSTAT_B.CURCHAN bits indicate the half SPORT's current channel being serviced in multichannel mode.

Half SPORT 'A' Rx Buffer (Primary) Register

The SPORT_RXPRI_A register buffers the half SPORT's primary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the primary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXPRI_A register. This data can be read in core mode (interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXPRI_A register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

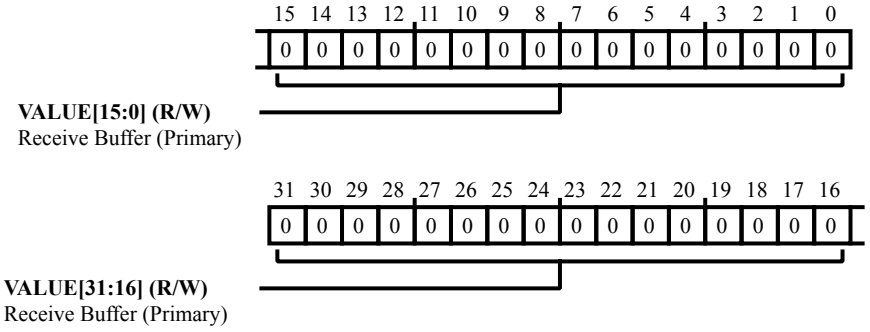


Figure 29-33: SPORT_RXPRI_A Register Diagram

Table 29-33: SPORT_RXPRI_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The SPORT_RXPRI_A.VALUE bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Rx Buffer (Primary) Register

The SPORT_RXPRI_B register buffers the half SPORT's primary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the primary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXPRI_B register. This data can be read in core mode (interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXPRI_B register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

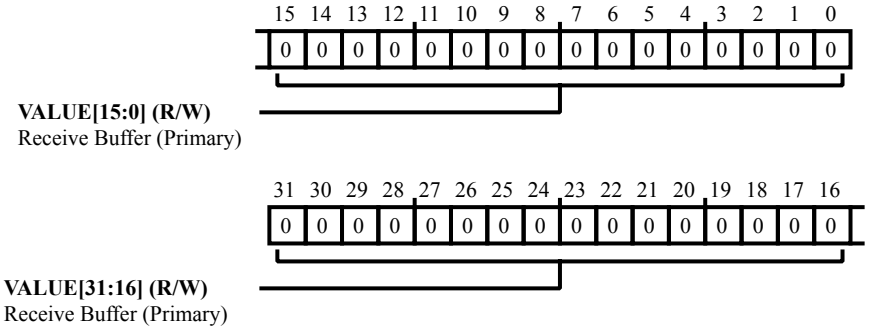


Figure 29-34: SPORT_RXPRI_B Register Diagram

Table 29-34: SPORT_RXPRI_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Primary). The SPORT_RXPRI_B.VALUE bits hold the half SPORT's primary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

Half SPORT 'A' Rx Buffer (Secondary) Register

The SPORT_RXSEC_A register buffers the half SPORT's secondary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the secondary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXSEC_A register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXSEC_A register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

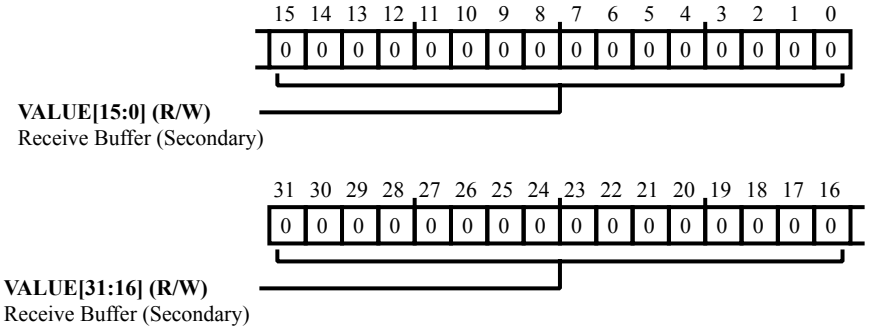


Figure 29-35: SPORT_RXSEC_A Register Diagram

Table 29-35: SPORT_RXSEC_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The SPORT_RXSEC_A.VALUE bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Rx Buffer (Secondary) Register

The SPORT_RXSEC_B register buffers the half SPORT's secondary channel receive data. This buffer becomes active when the half SPORT is configured to receive data on the secondary channel. After a complete word has been received in the receive shifter, it is placed into the SPORT_RXSEC_B register. This data can be read in core mode (in interrupt-based or polling-based mechanism) or directly transferred into processor memory using the DMA controller. With a data buffer and an input shift register, the SPORT_RXSEC_B register acts as a two-location buffer. So, the SPORT can keep a maximum of two 32-bit received words at any given time (independent of the SPORT_CTL_A.PACK bit setting).

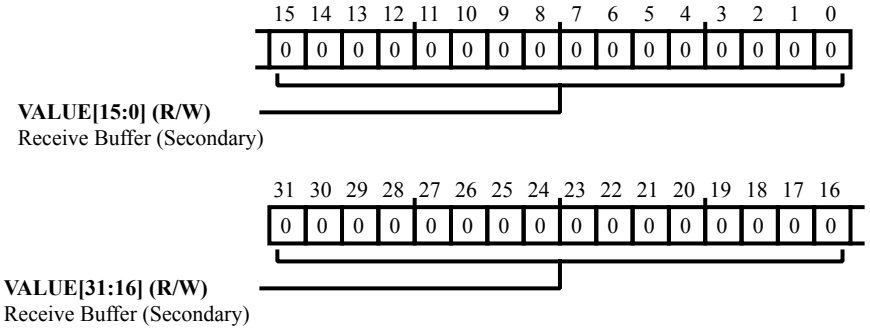


Figure 29-36: SPORT_RXSEC_B Register Diagram

Table 29-36: SPORT_RXSEC_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Receive Buffer (Secondary). The SPORT_RXSEC_B.VALUE bits hold the half SPORT's secondary channel receive data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

Half SPORT 'A' Tx Buffer (Primary) Register

The SPORT_TXPRI_A register buffers the half SPORT's primary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the primary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXPRI_A register acts as a three-location buffer if SPORT data packing is disabled (SPORT_CTL_A.PACK =0); while it acts as a two-location buffer when packing is enabled (SPORT_CTL_A.PACK =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXPRI_A register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

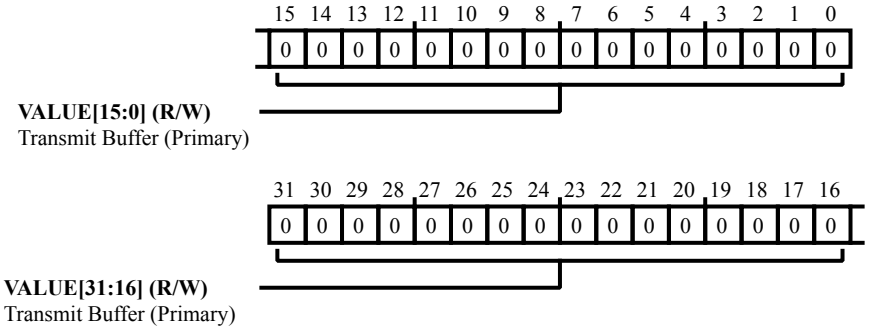


Figure 29-37: SPORT_TXPRI_A Register Diagram

Table 29-37: SPORT_TXPRI_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The SPORT_TXPRI_A.VALUE bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Tx Buffer (Primary) Register

The SPORT_TXPRI_B register buffers the half SPORT's primary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the primary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXPRI_B register acts as a three-location buffer if SPORT data packing is disabled (SPORT_CTL_B.PACK =0); while it acts as a two-location buffer when packing is enabled (SPORT_CTL_B.PACK =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXPRI_B register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

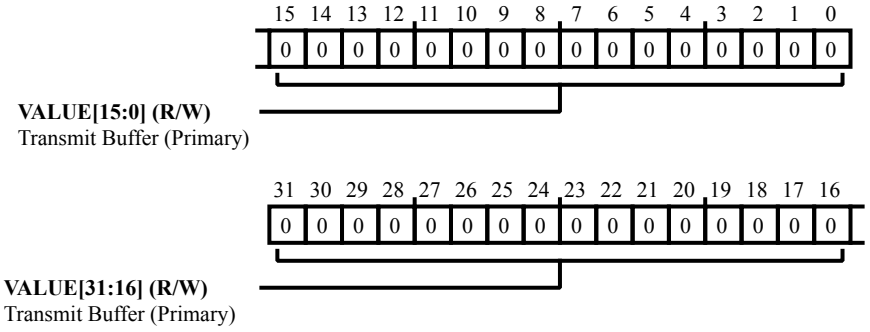


Figure 29-38: SPORT_TXPRI_B Register Diagram

Table 29-38: SPORT_TXPRI_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Primary). The SPORT_TXPRI_B.VALUE bits hold the half SPORT's primary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

Half SPORT 'A' Tx Buffer (Secondary) Register

The SPORT_TXSEC_A register buffers the half SPORT's secondary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the secondary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXSEC_A register acts as a three-location buffer if SPORT data packing is disabled (SPORT_CTL_A.PACK =0); while it acts as a two-location buffer when packing is enabled (SPORT_CTL_A.PACK =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXSEC_A register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

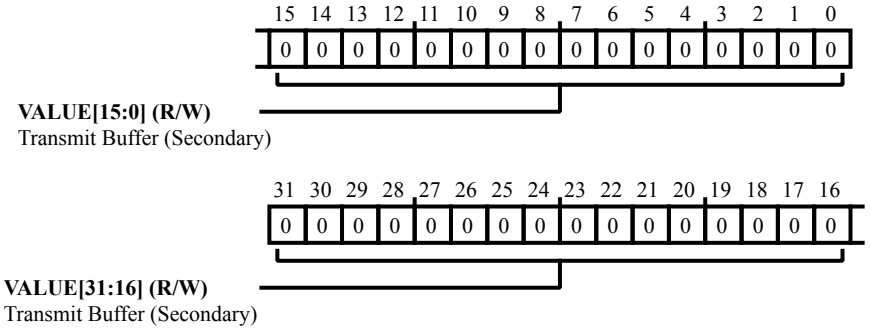


Figure 29-39: SPORT_TXSEC_A Register Diagram

Table 29-39: SPORT_TXSEC_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The SPORT_TXSEC_A.VALUE bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_A.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_A and SPORT_MCTL_A register descriptions.

Half SPORT 'B' Tx Buffer (Secondary) Register

The SPORT_TXSEC_B register buffers the half SPORT's secondary channel transmit data. This register must be loaded with the data to be transmitted if the half SPORT is configured to transmit on the secondary channel. Either a program running on the processor core loads the data into the buffer (word-by-word process) or the DMA controller automatically loads the data into the buffer (DMA process).

The SPORT_TXSEC_B register acts as a three-location buffer if SPORT data packing is disabled (SPORT_CTL_B.PACK =0); while it acts as two-location buffer when packing is enabled (SPORT_CTL_B.PACK =1). So, depending on the PACK bit setting, two 32-bit words or three 32-bit words can be stored in the transmit queue at any time. When the transmit register is loaded and any previous word has been transmitted, the SPORT_TXSEC_B register contents are automatically loaded into the output shifter. The half SPORT can issue an interrupt request (transmit buffer is not full) when it has loaded the output transmit shifter, signifying that the transmit buffer is ready to accept the next word. This interrupt request does not occur when the half SPORT is executing a DMA-based transfer.

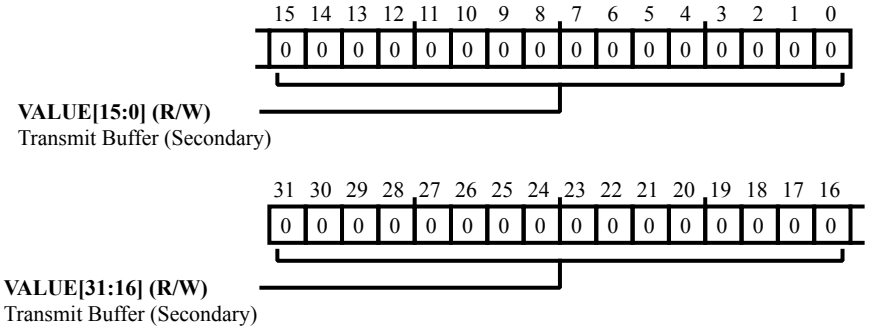


Figure 29-40: SPORT_TXSEC_B Register Diagram

Table 29-40: SPORT_TXSEC_B Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Transmit Buffer (Secondary). The SPORT_TXSEC_B.VALUE bits hold the half SPORT's secondary channel transmit data. Note that changes to the half SPORT operation mode (for example, toggling the SPORT_MCTL_B.MCE) empty the contents of this data buffer. For more information, see the SPORT_CTL_B and SPORT_MCTL_B register descriptions.

30 Analog-to-Digital Converter Controller (ADCC)

The ADCC provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). The processor or its peripheral infrastructure initiate the analog-to-digital conversions, based on either external or internal events, by giving the triggers to ADCC module.

NOTE: The ADCC and DACC chapters describe the control and data interface to the AFE die. For information about the analog portion (I/O pins and electrical specifications) of the AFE die, see the product data sheet.

The ADCC hardware initiates the ADC sampling (based on some processor or its peripheral events) by providing sampling signals with required timings in real time. Using the ADCC permits flexible scheduling of sampling instants and provides precise control execution of timing and analog sampling events on the ADCs. The ADCC saves both processor MIPS and provides precise controllability for ADC sampling time.

The converted ADC data from the ADCs is directly available in ADCC registers. The data can be read in core mode to store data into processor memory or can be directly routed through a dedicated DMA.

NOTE: In this chapter, the terms *Chip Select* and *Convert Start* are interchangeably used.

ADCC Features

The ADCC provides many architecture-based features (basic to the design) and mode-selectable features (usage is optional or configurable).

Architecture-based features of the ADCC include:

- One ADC interface (ADCC0) to control one ADC channel (ADC0) and a second ADC interface (ADCC1) to control two ADC channels (ADC1 and ADC2).
- Automated ADC sampling with ADC control hardware for sending the control word, executing conversion cycles, and reading the converted data with programmable timing.
- Up to six trigger inputs which permit the ADCC to initiate the ADC precise event sampling.

  Either the processor core or its peripheral infrastructure generates the trigger inputs internally.
- Up to 32 ADC sampling events per valid trigger received.

Each event must be assigned to either of the ADCC timers and assigned to one of the ADCs (and its channel). Each event is independently programmable to specify when to initiate ADC sampling based on the trigger input.

- 32 event status registers (one per each event), which indicate the amount of delay before the start of event handling after the event match occurs.
- Two independent 32-bit ADCC timers for controlling ADCC operation from two different sources.

  Any of the 32 events must be assigned to any ADCC timer. The ADCC automatically stops these timers after completion of associated events to save power. The trigger input to ADCC timer can be enabled or masked on-the-fly to temporarily disable all the events related to a trigger source.
- Separate eight-deep pending FIFOs for each ADC interface to queue the active events when the corresponding ADC is busy.
- Automated ADC sampling process, placing the converted data from the ADC in a directly available ADCC event data register.

  This data can be read in core mode to store the data in required memory space, or the data can be directly DMA transferred.
- Internally generated ADC clock from the processor system clock.

  This clock is gated (for example, it is active only when controlling the ADC) to provide excellent noise immunity during conversion process. The clock polarity (for example, the first edge after the convert start signal is asserted) is configurable.
- Serial clock, chip select, control signals, and data signals to control the ADC operations during control, conversion, and data read phases.
- A built-in DMA unit.

  There is a DMA unit with a channel for each ADCC timer for transferring the converted data of the associated events. The DMA unit supports an optional mechanism for circular buffering.
- Error detection capabilities, including support for the detection of an event miss, event collision, ADCC timer trigger overrun, bandwidth monitor error when using DMA and memory write response error conditions.
- Trigger master capability provides two trigger signals to the TRU unit on the completion of each ADCC timer frame.

Mode-selectable features of the ADCC include:

- Bandwidth monitoring option (when using DMA mode of data read operation), which identifies whether the timer count has gone beyond an expected limit when receiving an ADC data frame.

  This option also provides a status bit to indicate whether a DMA is pending reception of data from the ADC.
- Status indication (status bits) and optional interrupt request generation (when using core mode of data read operation) to the core on the completion of each event.

- An interrupt request can be optionally generated at the end of each ADCC timer frame completion or on ADCC error detection.

ADCC Functional Description

The ADCC controller provides a means to automate the ADC sampling sequence precisely. It reduces the core overhead required (compared to sampling methods where no dedicated ADC controller is present) and simplifies the ADC accesses. Two independent ADC interfaces in the ADCC synchronize the controls between the processor and the two on-chip ADCs. These features permit flexible scheduling of sampling instants and provide precise control execution of timing and analog sampling events on the ADCs.

NOTE: Refer to the following sections for more information about ADCC functionality:

- ADCC Signal Descriptions
- ADCC Block Diagram
- ADCC Architectural Concepts

CM41X_M4 ADCC Register List

The ADC Controller (ADCC) automates the ADC sampling process and simplifies ADC accesses. The ADCC provides an interface that synchronizes the controls between the processor and an analog-to-digital converter (ADC). A set of registers governs ADCC operations. For more information on ADCC functionality, see the ADCC register descriptions.

Table 30-1: CM41X_M4 ADCC Register List

Name	Description
ADCC_ADCRW0	ADC2 Interface RW Access Register
ADCC_ADCRW1	ADC2 Interface RW Access Register
ADCC_BPTR0	Base Pointer 0 Register
ADCC_BPTR1	DMA Base Pointer 1 Register
ADCC_BWMON0	Bandwidth Monitor 0 Register
ADCC_BWMON1	Bandwidth Monitor 1 Register
ADCC_CBNUM0	Timer0 Circular Buffer DMA Wrap Number Register
ADCC_CBNUM1	Timer1 Circular Buffer DMA Wrap Number Register
ADCC_CBSIZ0	Circular Buffer Size 0 Register
ADCC_CBSIZ1	Circular Buffer Size 1 Register
ADCC_CTL	Control Register
ADCC_DATOVF	Data Overflow Indication Register
ADCC_ECOL	Event Collision Status Register

Table 30-1: CM41X_M4 ADCC Register List (Continued)

Name	Description
ADCC_EIMSK	Event Interrupt Mask Register
ADCC_EIMSK_CLR	Event Interrupt Mask Clear Register
ADCC_EIMSK_SET	Event Interrupt Mask Set Register
ADCC_EISTAT	Event Interrupt Status Register
ADCC_EMISS	Event Miss Status Register
ADCC_EPND	Pending Events Status Register
ADCC_ERRMSK	Error Mask Register
ADCC_ERRMSK_CLR	Error Mask Clear Register
ADCC_ERRMSK_SET	Error Mask Set Register
ADCC_ERRSTAT	Error Status Register
ADCC_EVCTL[nn]	Event n Control Register
ADCC_EVDAT[nn]	Event n Data Register
ADCC_EVSTAT[nn]	Event n Status Register
ADCC_EVTEN	Event Enable Register
ADCC_EVTEN_CLR	Event Enable Clear Register
ADCC_EVTEN_SET	Event Enable Set Register
ADCC_EVT[nn]	Event n Time Register
ADCC_FIMSK	Frame Interrupt Mask Register
ADCC_FIMSK_CLR	Frame Interrupt Mask Clear Register
ADCC_FIMSK_SET	Frame Interrupt Mask Set Register
ADCC_FISTAT	Frame Interrupt Status Register
ADCC_FRINC0	Frame Increment 0 Register
ADCC_FRINC1	Frame Increment 1 Register
ADCC_NUMFRAM0	Timer0 Frame Limit Count Register
ADCC_NUMFRAM1	Timer1 Frame Limit Count Register
ADCC_T0STAT	Timer 0 Status Register
ADCC_T1STAT	Timer 1 Status Register
ADCC_TCA0	Timing Control A (ADC0) Register
ADCC_TCA1	Timing Control A (ADC1) Register
ADCC_TCB0	Timing Control B (ADC0) Register
ADCC_TCB1	Timing Control B (ADC1) Register

Table 30-1: CM41X_M4 ADCC Register List (Continued)

Name	Description
ADCC_TMR0	Timer 0 Current Count Register
ADCC_TMR1	Timer 1 Current Count Register
ADCC_TRGCNT0	Trigger Count TIMER0 Register
ADCC_TRGCNT1	Trigger Count TIMER1 Register

CM41X_M0 ADCC Interrupt List

Table 30-2: CM41X_M0 ADCC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
13	ADCC0_ERR	ADCC0 ADC Error	Level	
23	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level	
24	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level	

CM41X_M4 ADCC Interrupt List

Table 30-3: CM41X_M4 ADCC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
62	ADCC1_ERR	ADCC1 ADC Error	Level	
64	ADCC0_ERR	ADCC0 ADC Error	Level	
129	ADCC1_TMR0_EVT	ADCC1 Timer 0 Event Complete	Level	
130	ADCC1_TMR1_EVT	ADCC1 Timer 1 Event Complete	Level	
131	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level	
132	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level	

CM41X_M0 ADCC Trigger List

Table 30-4: CM41X_M0 ADCC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
13	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level
14	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level

Table 30-5: CM41X_M0 ADCC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
20	ADCC0_SLV0	ADCC0 Trigger Slave 0	Pulse
21	ADCC0_SLV1	ADCC0 Trigger Slave 1	Pulse
22	ADCC0_SLV2	ADCC0 Trigger Slave 2	Pulse
23	ADCC0_SLV3	ADCC0 Trigger Slave 3	Pulse
24	ADCC0_SLV4	ADCC0 Trigger Slave 4	Pulse
25	ADCC0_SLV5	ADCC0 Trigger Slave 5	Pulse

CM41X_M4 ADCC Trigger List

Table 30-6: CM41X_M4 ADCC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
29	ADCC0_TMR0_EVT	ADCC0 Timer 0 Event Complete	Level
30	ADCC0_TMR1_EVT	ADCC0 Timer 1 Event Complete	Level
31	ADCC1_TMR0_EVT	ADCC1 Timer 0 Event Complete	Level
32	ADCC1_TMR1_EVT	ADCC1 Timer 1 Event Complete	Level

Table 30-7: CM41X_M4 ADCC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
28	ADCC1_SLV0	ADCC1 Trigger Slave 0	Pulse
29	ADCC1_SLV1	ADCC1 Trigger Slave 1	Pulse
30	ADCC1_SLV2	ADCC1 Trigger Slave 2	Pulse
31	ADCC1_SLV3	ADCC1 Trigger Slave 3	Pulse
32	ADCC1_SLV4	ADCC1 Trigger Slave 4	Pulse
33	ADCC1_SLV5	ADCC1 Trigger Slave 5	Pulse

ADC to ADCC Interface

There are two ADCCs for each of the cortex core platforms in the system. Each ADCC works independently from the other.

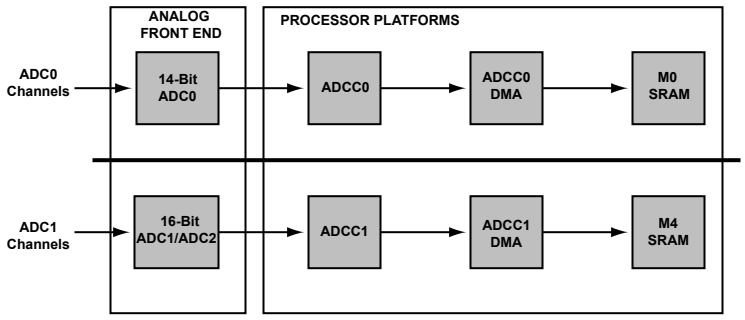


Figure 30-1: ADC to ADCC Interface

ADC Multiplexed Input Scheme

The ADC0 supports 7 input channels (D0 through D6) that are multiplexed as 7:1 input to ADC0 for sampling. ADC0 can support sampling only one channel at a time. This channel is chosen while sending the control word to the ADC from the ADCC0.

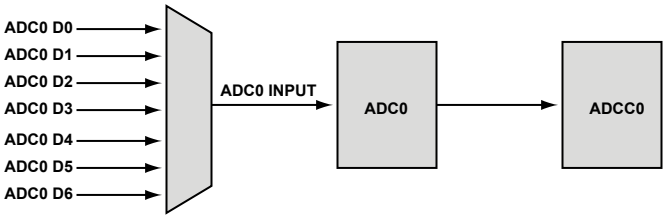


Figure 30-2: ADC0 Multiplexed Input

ADC1 and ADC2 support a total of 12 channels each as shown in the multiplex diagram below. ADC1 and ADC2 can both support sampling multiple channels at same time. This is achieved through the Track and Hold circuits (T/H) in the input of the ADCs. Each ADC can support simultaneous sampling of three input channels at a time, via three T/H circuits. Note that data is sent to ADCC one after the other serially, even if the sampling is done simultaneously.

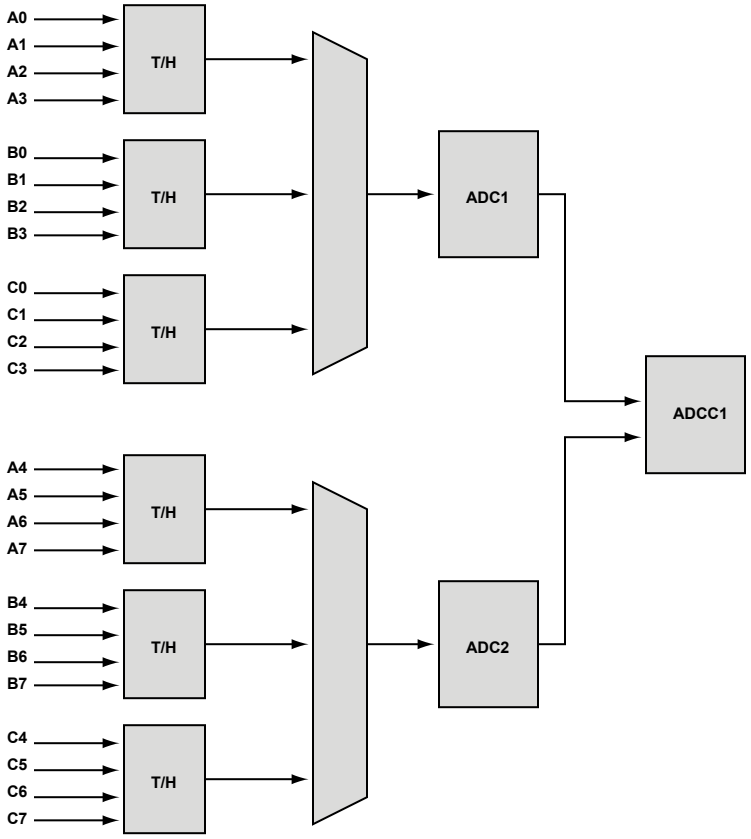


Figure 30-3: ADC1, ADC2 Multiplexed Input

ADCC Block Diagram

The ADCC controller consists of a trigger input selection multiplexer, two independent 32-bit timers, 32 event register bank, 32 event comparators, a pending event FIFO, and a timing generation unit for two ADC interfaces. Also, the ADCC incorporates two in-built DMA units, one for each ADCC timer, to store ADC samples directly in required memory space.

The following figures are block diagrams of ADCC0 and ADCC1. See ADCC Signal Descriptions for more details on the various input and output signals specified in the block diagrams. The digital interfaces of ADCC0 and ADCC1 are exactly alike.The differences in the ADC Timing and Control Unit interface level as follows:

- ADCC0 has only one ADC interface (for ADC0) and only one ADC unit (Unit A)
- ADCC1 has two ADC interfaces (ADC1 and ADC2) and two ADC units exist (Units A and B).

The ADCC interface to the digital side includes core and DMA access, interrupts and triggers. DMA is the preferred way to operate the ADCC because it allows the cores to perfrom other tasks. It is possible to synchronize the operation of ADC sampling and other peripherals. For instance, a PWM sync trigger maybe used to trigger an ADC sampling by interconnecting a PWM sync trigger and a ADCC input trigger via the Trigger Routing Unit. The ADCC also has multiple ways to interrupt the core, in order to process the data or to let the core take an action based on sampling completion.

NOTE: ADCC0 runs on SCLK and ADCC1 runs on SYSCLK. This chapter uses a common system clock and must be interpreted as SYSCLK/SCLK0, as appropriate for programming.

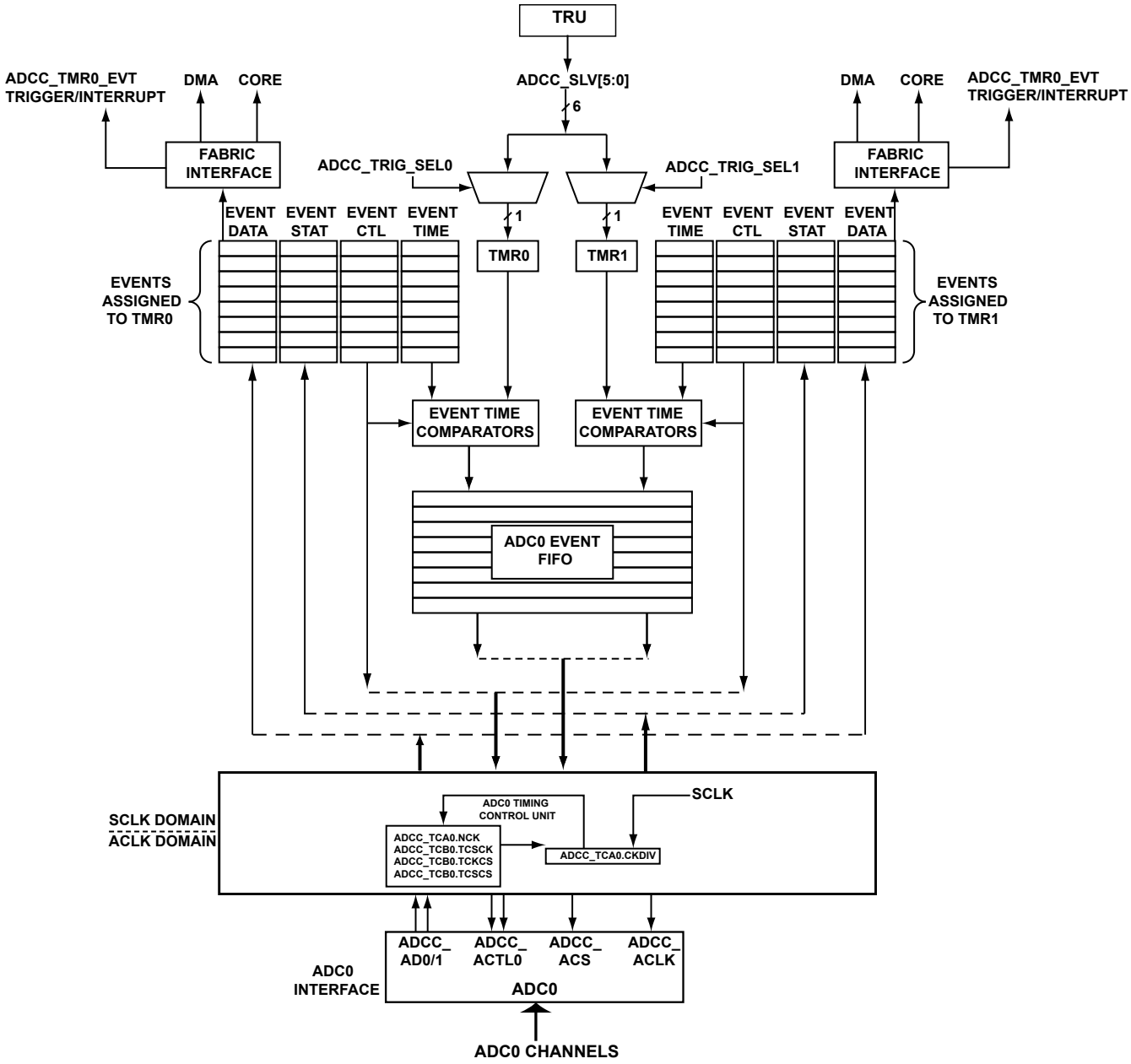


Figure 30-4: ADCC0 Block Diagram

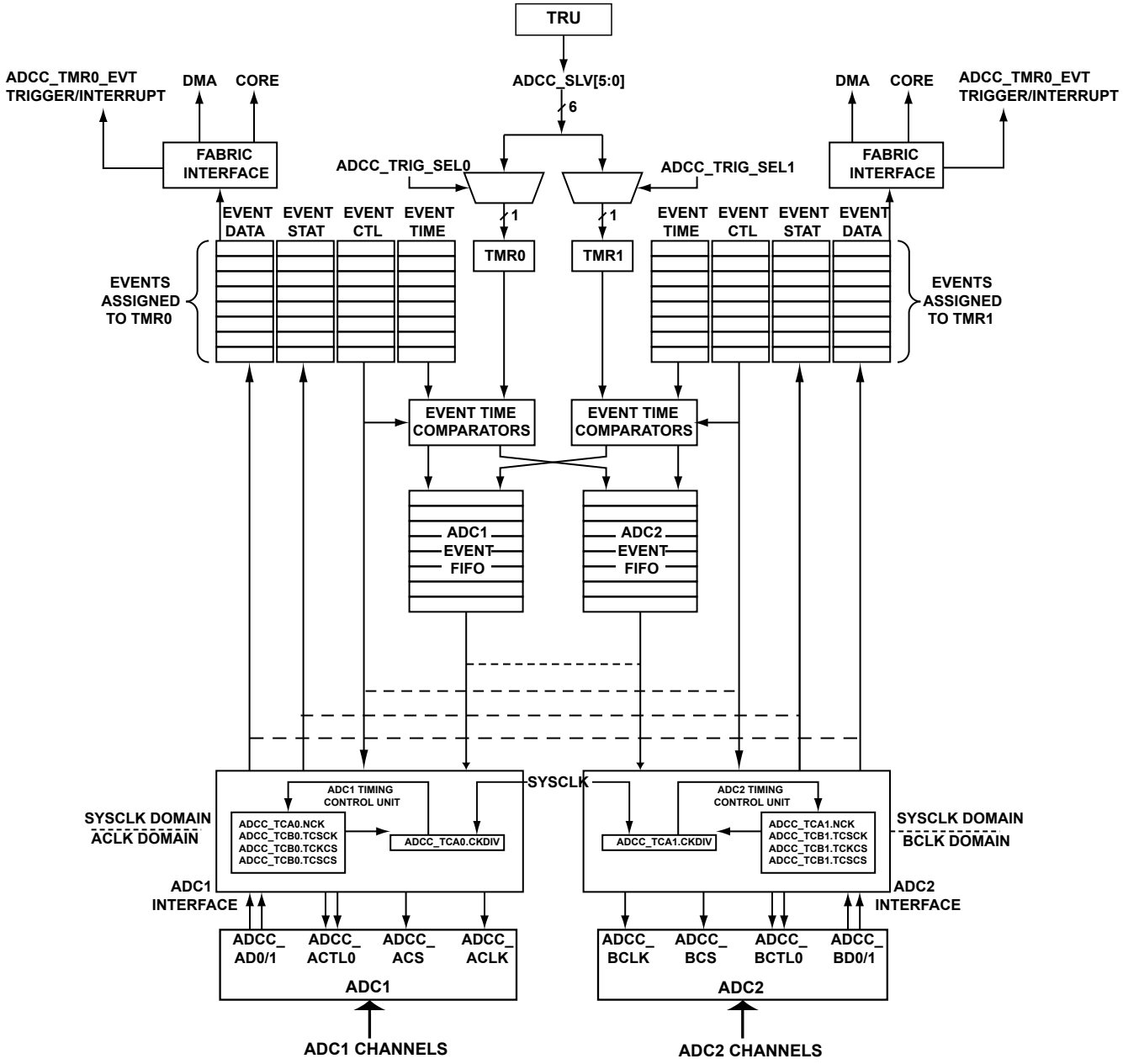


Figure 30-5: ADCC1 Block Diagram

NOTE: For more information about each part of the ADCC, see ADCC Architectural Concepts.

NOTE: ADCC1 and ADCC0 have the same function and design, even though they connect to a different set of ADCs.

NOTE: The hard reset or system reset signals that go to the processor and digital units such as ADCCs are not connected to the Analog Front end.

ADCC Signal Descriptions

The ADCC controls the operation of internal ADCs, based on the settings for enabled events and trigger input. As the ADCs are internal, none of the ADCC signals are available as external package pins.

ADC Interface Signals

A number of signals connect the ADCC to the ADCs for providing ADC clock, chip select, and control. Using these signals, the ADCC regulates the ADC sampling sequence and reads the converted data. These signals appear in the ***ADCC-to-ADC1/2 Signal Descriptions*** table.

Table 30-8: ADCC-to-ADC1/2 Signal Descriptions

Name	I/O	Description
$\overline{\texttt{ADCC\_ACS}}$ or $\overline{\texttt{ADCC\_BCS}}$	O	ADCC chip select (or start-of-conversion) for the ADC
ADCC_ACLK or ADCC_BCLK	O	ADCC clock for ADC communication
ADCC_ACTL0 or ADCC_BCTL0	O	ADCC control signal for sending control word
ADCC_ACTL1 or ADCC_BCTL1	O	ADCC ADC0/1 (A) control 1 for sending control word
ADCC_AD0 and ADCC_AD1 or ADCC_BD0 and ADCC_BD1	I	ADCC data lines for reading converted data from the ADC

ADCC_SLV[n] Trigger Slaves

The ADCC can accept up to six trigger slave inputs, which the trigger routing unit (TRU) of the processorroutes internally. Only two of these trigger slaves can be accepted at a time. The ADCC trigger selection field (for example, ADCC_CTL.TRGSEL0 bit field) assigns one trigger for each ADCC timer, which can be same or different.

Table 30-9: ADCC0 Triggers

ADCC0_SLV0
ADCC0_SLV1
ADCC0_SLV2
ADCC0_SLV3
ADCC0_SLV4
ADCC0_SLV5

Table 30-10: ADCC1 Triggers

ADCC1_SLV0
ADCC1_SLV1
ADCC1_SLV2
ADCC1_SLV3

Table 30-10: ADCC1 Triggers
(Continued)

ADCC1_SLV4
ADCC1_SLV5

Two 32-bit internal timers (TMR0/TMR1) in the ADC can be independently configured to start up on being triggered by a trigger master (for example, PWM).

When the ADCC detects a valid trigger, the corresponding ADCC timer starts counting at the rate of the processor system clock (SCLK). The ADCC treats trigger inputs as edge-sensitive.

If the trigger input appears before completion of an ADCC frame, the trigger overrun status bit is set and the ADCC (optionally) generates an error interrupt. As each ADCC timer can initiate the ADCC frame, the ADCC provides two trigger overrun bits, one for each timer.

ADCC_TMRn_EVT Trigger Masters

The ADCC can act as a trigger master, providing one of two trigger signals to the TRU on completion of each of the ADCC timer frames. The triggers can be selectively enabled (for example, using the `ADCC_CTL.TRGOE0` bit).

ADCC_ACLK, ADCC_BCLK

The ADCC clock provides the clock source for communicating with ADCs, which the ADCs also use for the conversionprocess. The ADCC provides this clock in gated format (for example, active only when controlling the ADCs) to ensure excellent noise immunity during the conversion process.

The ADC clock is internally generated from the system clock of the processor. For example, the `ADCC_TCA0.CKDIV` bit field specifies the divider to generate the ADC clock signal from SCLK. This divisor is a 16-bit field, allowing a wide range of clock rates for ADC operation.

Use the following equation to calculate the ADC clock frequency:

`ADCC_ACLK` = (SYSCLK) ÷ (`ADCC_TCA0.CKDIV` + 1)

Alternatively, calculate the clock divisor required for the ADC0 clock frequency as:

`ADCC_TCA0.CKDIV` = (SYSCLK ÷ `ADCC_ACLK`) – 1

The minimum and default value of `ADCC_TCA0.CKDIV` field is 0. Do not set the divisor field to zero when SCLK is greater than the maximum clock frequency the ADC supports.

Both the ADC and the ADCC use this clock to drive the output signals and sample the incoming signals.

NOTE: ADCC0 runs on SCLK and ADCC1 runs on SYSCLK. The above programming is common for ADCC0/SCLK as well.

ADCC_ACS, ADCC_BCS

The ADCC provides a chip select signal to select the ADC for communication. It asserts the signal while sending the control word, during conversion period, or while readingconverted data. These three phases form the ADC sampling sequence, which starts based on a trigger input and the settings of one or more enabled events.

The width and period of the chip select signal is configurable based on ADCC timing register settings.

- Time-CS-to-CK-setup (for example, `ADCC_TCB0.TCSCK`) is the minimum delay between the assertion edge of the chip select and the first edge of the ADC clock
- Number-of-clocks (for example, `ADCC_TCA0.NCK`) is the number of ADC clock cycles to be given in each chip select pulse.
- Time-CK-to-CS-hold (for example, `ADCC_TCB0.TCKCS`) is the minimum delay between last edge of the ADC clock and the de-assertion edge of the chip select signal
- Time-CS-to-CS-delay (for example, `ADCC_TCB0.TCSCS`) is the minimum delay between the completion of one phase and starting of another phase of the ADC sampling sequence. (For example, the delay is the time between the de-assertion edge of one chip select signal to the assertion edge of the next chip select signal).

All parameters are specified in terms of ADC clock cycles.

For example, on ADC, the width of the chip select signal is configurable based on the `ADCC_TCA0.NCK`, `ADCC_TCB0.TCSCK`, and bit fields. The active duration of ADC clock select is calcula- ted as (in terms of ADC clock cycles):

$\overline{\texttt{ADCC\_ACS}}$ Active Duration = (`ADCC_TCB0.TCSCK` + `ADCC_TCA0.NCK`)

The ADCC uses the `ADCC_TCB0.TCSCS` field to control the inactive period of the ADC chip select.

Total Period (TCSP) = $\overline{\texttt{ADCC\_ACS}}$ Active Duration + `ADCC_TCB0.TCSCS`)

The figure shows the timing relationships provided through the timing register settings.

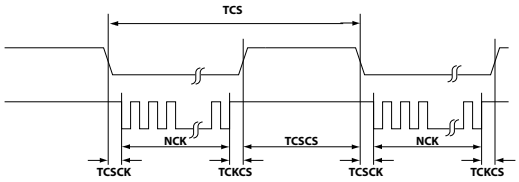


Figure 30-6: ADCC Clock Signal and Chip Select Signal Description

ADCC_CS Active Duration = `ADCC_TCB0.TCSCK` + `ADCC_TCA0.NCK` + `ADCC_TCB0.TCKCS`

ADCC_CS Inactive Duration = `ADCC_TCB0.TCSCS`

Total ADCC_CS period = TCS = ADCC_CS Active Duration + ADCC_CS Inactive Duration

ADC1 and ADC2 Timing Calculation

Substituting the values for the ADSP-CM41x, with an example TCSP as 340 ns for ADC1 and ADC2, the following timing can be configured. Refer to the product data sheet for timing specifications.

NCK (Number of Clocks) is fixed as 8, TCSCK (Time CS to CK Setup) is fixed as 1 TCKCS (Time CK to CS Hold) is fixed as 0.

To achieve 340 ns, with the requirements above, the TCSCS value can be derived as 8 as given: TCSCK + TCKCS + NCK + TCSCS = 1 + 0 + 8 + 8 = 17 ADCC_ACK clocks. With a System clock (SYSCLK) of 100 MHz and an ADCC_ACK divider of 2, the ADCs can achieve 50 MHz of ADC clock. With an ADCC_ACK = 50 MHz, the period = 20 nsec and the final configuration achieves 340 nsec.

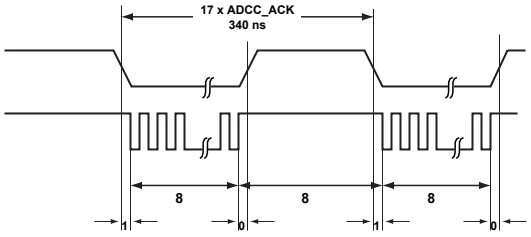


Figure 30-7: ADC1 and ADC2 Timing Calculation

ADC0 Timing Calculation

Substituting the values for the ADSP-CM41x, with an example TCSP as 500 ns for ADC0, the following timing can be configured. Refer to the product data sheet for timing specifications.

NCK (Number of Clocks) is fixed as 8, TCSCK (Time CS to CK Setup) is fixed as 1 TCKCS (Time CK to CS Hold) is fixed as 0.

To achieve 500 ns, with the requirements above, the TCSCS value can be derived as 16 as given: TCSCK + TCKCS + NCK + TCSCS = 1 + 0 + 8 + 16 = 25 ADCC_ACK clocks. With a System clock (SYSCLK) of 100 MHz and an ADCC_ACK divider of 2, the ADCs can achieve 50 MHz of ADC clock. With an ADCC_ACK = 50 MHz, the period = 20 nsec and the final configuration achieves 500 nsec.

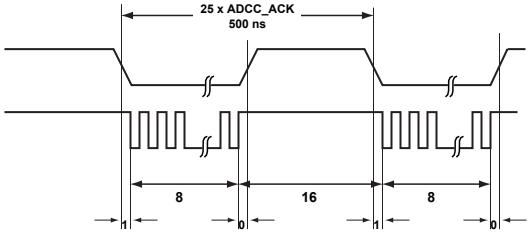


Figure 30-8: ADC0 Timing Calculation

Timing and Control Unit

The ADCC Timing and Control Unit provides two ADC interfaces. Each interface can independently control one ADC with the required timing and interface protocol.

When an event becomes ready to handle, the ADC interface unit initiates the ADC sampling sequence, according to the ADCC settings for that particular event. The ADC sampling sequence is divided into phases:

- Control phase – the control word is sent to ADC
- Conversion phase – the conversion pulse is provided
- Data phase – the converted data from the ADC is read

The ADC interface provides an ADC clock, an ADC chip select, control lines for sending the control word, and data lines for reading converted data.

The ADCC timing registers determine the timing of the ADCC_ACLK, ADCC_ACS, ADCC_BCLK, and ADCC_BCS signals. The ADC clock signals are gated and are provided only while controlling the ADC. The chip select signal has programmable timing in terms of the ADC clock. The ADCC uses the bits of the ADCC_CTL register to determine the timings and protocol of ADC control lines and data lines.

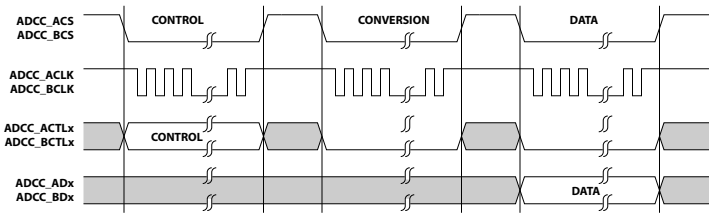


Figure 30-9: Control Unit Timing

Control Conversion and data phase can be pipelined to achieve high throughput. Refer to the Programming Concepts on how ADC sampling timing can be arrived at, for high throughput cases. It is application's responsibility to arrange the Event timing in such a way that the three phases can be pipelined, within the timing restrictions.

NOTE: The application does not have direct access to the ADC interface lines clock, data and chip select mentioned above and these are embedded internally in the design. These signals are automatically driven whenever ADCC performs an access to ADC, based on the sampling requirement programmed as per Event Registers.

General Operation

The processor or its peripheral infrastructure (based on either external or internal conditions) provides trigger signals to the ADCC module to initiate the ADC conversions. The ADCC can simultaneously accept up to six trigger inputs from different sources (for example, from trigger masters of the TRU of the processor).

Two 32-bit counters, known as ADCC timers, are central to ADCC operation. These timers each independently accept one of the trigger inputs and start counting at the system clock rate of the processor. The counting begins when a valid edge is detected on the selected trigger input.

The ADCC provides 32 events that can be independently programmed to occur at the specified time after the trigger signal. This time is configured in the time bit field, ADCC_EVT[nn].TIME of the corresponding event time register. The event can be assigned to either to ADCC TMR0 or ADCC TMR1, by configuring the timer select bit field, ADCC_EVCTL[nn].TMRSEL of the corresponding event control register. As the ADCC timers count at

SCLK rate, the event comparator unit compares the event time of enabled events with the current count of the assigned ADCC timer. If the count matches, the particular event is active.

The event control register of each event also specifies which ADC interface executes the sampling sequence of the event. The `ADCC_EVCTL[nn].ADCSEL` bit field selects either ADC1 interface or ADC2 interface. Also, the event control register stores a control word, which (when sent to an ADC) selects the analog input channel that the ADC uses for conversion. When an event becomes active, the comparators signal the activity to the timing and control unit of the corresponding ADC interface.

If the ADC interface is ready to initiate an ADC sampling sequence, the interface starts the sequence for the current active event. When the ADC interface is busy, the new active event is stored in an 8-deep pending event FIFO.(For example, the ADC interface is busy when an event becomes active while it is already executing the sampling sequence of another event). The interface executes events from the pending FIFO (in chronologically order) when the ADC interface is free.

Introduction to Events

A sample to be taken from an ADC is referred to as an event and events can also be inserted as NOP (No Operation/Sampling) instructions. The events can be programmed to occur at specified times after a trigger input to the ADCC. The sample can be from any ADC channel within the ADCC domain, either in asynchronous mode or synchronous mode. An event is complete when the data reception (for the event) completes. The ADCC can handle a total of 32 events which can be independently configured and enabled. Each event can be assigned to an ADCC timer and can instruct the ADCC to start a sampling sequence for ADCs.

A typical waveform is shown in the following figure.

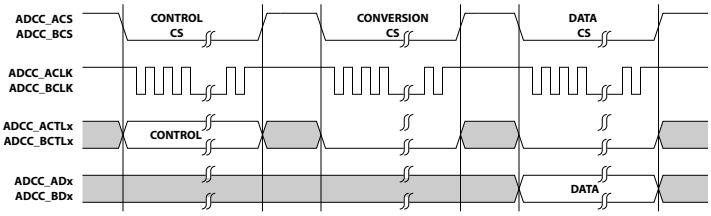


Figure 30-10: ADCC Operation Example

The ADCC hardware automates this sampling process, and the converted data is directly available to read in the data register (`ADCC_EVDAT[nn]`) of that event. When the ADCC starts to execute an event, the conversion data is only available during the third frame being sent. If another event becomes active during this time, service for the newly arrived frame is delayed. To reduce the latency of service for an event in collision events, the ADCC can decide to pipeline the ADC sampling sequence of different events. (An event is *in collision* if it becomes active while the ADCC is busy executing a previously received event).

The ADCC can handle a total of 32events which can be independently configured and enabled. Each event can be assigned to an ADCC timer and can instruct the ADCC to start a sampling sequence for either ADC1 and ADC2 OR ADC0.

Each event consists of a four register bank, including an event control register (ADCC_EVCTL[nn]), an event time register (ADCC_EVT[nn]), an event status register (ADCC_EVSTAT[nn]), and an event data register (ADCC_EVDAT[nn]). The ADCC_EVTEN register contains event enable bits.

The event control register determines the following:

- Event assignment to TMR0 or TMR1
- Event execution on the ADC0 interface when using ADCC0 OR Event execution on the ADC1/2 interface when using ADCC1
- ADC channel selection for sampling
- Requirement for simultaneous sampling with another ADC
- Memory offset for event data storage for reading ADC data in DMA mode

The event time register specifies the time offset from the corresponding ADCC timer trigger input to the start of that particular event. (For example, the time offset is when the event must happen based on the trigger input). This time offset is specified in terms of system clock of the processor.

The event status register provides the delay in number of system clock cycles after the event match occurred. This information is useful for debugging purpose.

The result of ADC sampling (for example, converted data from the ADC) is directly stored into the event data register. The data can be accessed in core mode or can be directly DMA transferred into processor memory.

An ADCC timer frame completes when all the events associated with that ADCC timer complete after detecting a valid trigger. At least one event per ADCC timer must be enabled for the ADCC to execute an ADC sampling sequence.

Event Examples

The following example illustrates the trigger, event, sample, and data transfer (channel) process. The ***ADCC Operation Example - Relating All Signals*** figure provides an overview of ADCC operations.

In the example the application samples two analog inputs. One input is from each ADC interface, at every PWM period. Another input is comprised of two samples from different analog inputs, one from each ADC interface, whenever there is some activity on a general-purpose input pin.

The application requires a total of four ADCC events. Two are triggered every PWM period, and the other two are triggered when ever there is an edge on the general-purpose input pin. This example also assumes that events 0 through 3 (EVT-0 through EVT-3) are selected for this purpose.

- The pulse-width modulator (PWM) triggers EVT-0 and EVT-1.

  EVT-0 initiates ADC1 sampling for the ADC1_CHN0 channel, while EVT-1 initiates ADC2 sampling for the ADC2_CHN2 channel.

- A general-purpose pin input triggers EVT-2 and EVT-3.

EVT-2 initiates ADC1 sampling for the ADC1_CHN2 channel, while EVT-3 initiates ADC2 sampling for the ADC2_CHN3 channel.

Because there are two trigger sources in the application, both ADCC timers (TMR0 and TMR1) are enabled. One of the timers (in the example, TMR0) accepts a trigger from the PWM master, while the other timer (in the example, TMR1) accepts a trigger from the general-purpose pin input. The PWM_SYNC signal is used to generate the trigger signal in each cycle. The PINT block is used to provide the trigger signal from the edges that appear on the general-purpose input pin. The TRU of the processor routes the trigger signals from the trigger masters (in the example, PWM and PINT) to the trigger slave (in the example, TMR0 and TMR1).

The activation of events after the trigger input is configurable in the `ADCC_EVT[nn]` (Event Time) register. The ***ADCC Operation Example - Event Information*** table provides a list of event information.

Table 30-11: ADCC Operation Example - Event Information

Event	ADC Interface	ADC Channel	ADCC Timer	Triggered by	Event Time
EVT-0	1	ADC1_CHN0	0	PWM Sync	EVT0 < EVT1
EVT-1	2	ADC2_CHN1	0	PWM Sync	EVT0 < EVT1
EVT-2	1	ADC1_CHN2	1	GP Input	EVT3 < EVT2
EVT-3	2	ADC2_CHN3	1	GP Input	EVT3 < EVT2

In this case, the ADCC signals can look as shown in ***ADCC Operation Example - Relating All Signals***.

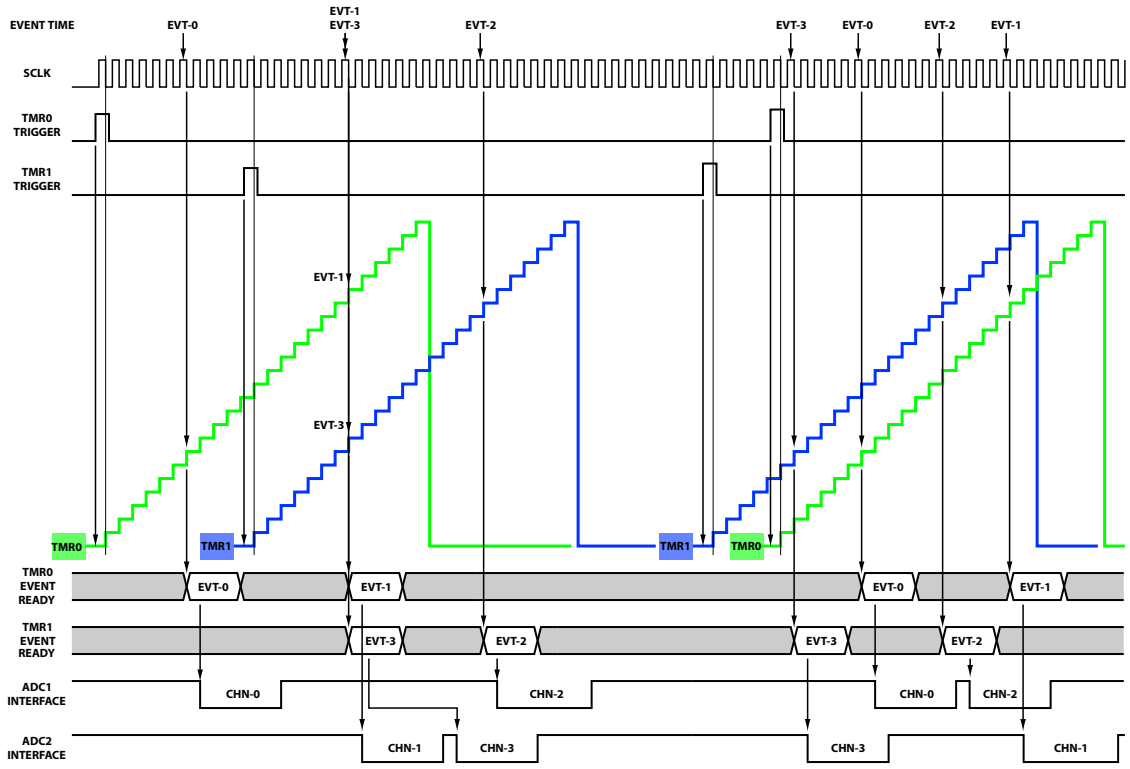


Figure 30-11: ADCC Operation Example - Relating All Signals

Note the following key points from the ***ADCC Operation Example - Relating All Signals*** figure:

- The ADCC timers start counting at the SCLK rate when a valid edge on a selected trigger is detected. The event comparators compare the timer count with the event time register of associated events. The comparators signal that the event is Ready on a count match.
- Each timer continues to run until all the associated events complete. The exact time depends on whether the core or DMA operating modes are used for reading the ADC data. After all the events complete, the ADCC timer count is reset to zero.
- The trigger inputs of both of the timers are asynchronous to each other. Events associated with the trigger inputs can overlap.

  In the example, note that EVT-1 and EVT-3haveoccurred at the same time. (For example, event comparators have signaled EVT-1 of TMR0 and EVT-3 of TMR1 as ready in the same ADCC_ACLK cycle). The priority in this case goes to the event associated with TMR0, and EVT-1 is serviced first. EVT-3 is placed in the pending event FIFO, and the event collision error bit is set.
- For simplicity, the ***ADCC Operation Example - Relating All Signals*** figure shows the collided events as separately serviced, one after the other. The ADC interface can decide during real-world operation to pipeline the ADC sampling. The ***ADCC Operation Example - Event Tightly Pipelined*** figure describes this operation.
- The pulse on the ADC interface line shows ADC sampling sequence symbolically. This presentation does not detail the exact sampling sequence and does not show the ADCC signals used to control the ADCs.
- There is no restriction for assigning events to ADCC timer 0 or timer 1. There is also no restriction for assigning a particular event to the ADC1 interface or ADC2 interface. All the events can be independently programmed.

When an event becomes active, the ADC sampling sequence related to it starts when the ADC interface is ready to initiate the sampling process. The ADC sampling sequence is divided into phases:

- Control phase – the ADCC sends the control word to select the ADC channel for conversion. Typically, one ADC provides multiple input channels, one of which is selected for conversion by specifying the channel-ID while sending the control word. The control word for each event is stored in the `ADCC_EVCTL[nn].CTLWD` field. For more information about the ADC control word, see Control Word Format.
- Conversion phase – the ADCC sends a conversion pulse to initiate the conversion for the selected ADC channel.
- Data phase – the ADCC reads the converted data from the ADC. The ADCC provides the chip select or start of conversion signal for each of these phases.

ADCC Operation Example - Event Overlapped

Consider a case in which event 1 becomes active while the ADC interface is providing the conversion pulse for the event 0 ADC sampling sequence. The ADCC starts the sampling sequence for event 1 in the next phase. For example, the ADCC sends the control word (using the `ADCC_EVCTL[nn].CTLWD` bit field) for event 1 while receiving data for the event 0 sample. Refer to the ***ADCC Operation Example - Event Overlapped*** figure.

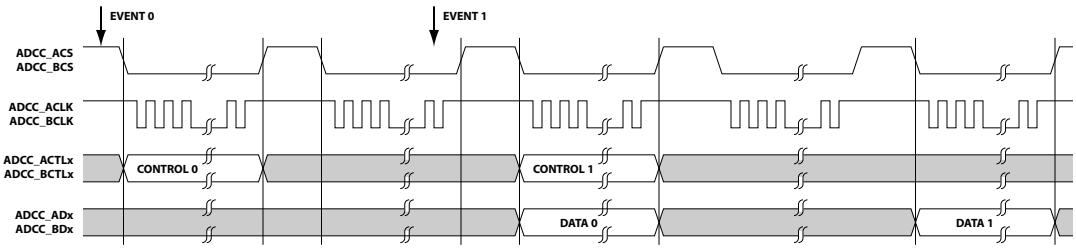


Figure 30-12: ADCC Operation Example - Event Overlapped

ADCC Operation Example - Event Tightly Pipelined

The ADC interface can tightly pack the ADC sampling sequence, in such a way that events can be pipelined in a tight manner. This helps to achieve maximum throughput from the ADC interface as well as to achieve the maximum sampling rate. When the ADC interface sends the control word for an event, the interface provides a conversion pulse to the previous event and reads the converted data of the previous event.

ADCC0 Operation Example

This example assumes that event timing is configured for maximum throughput for ADC0.

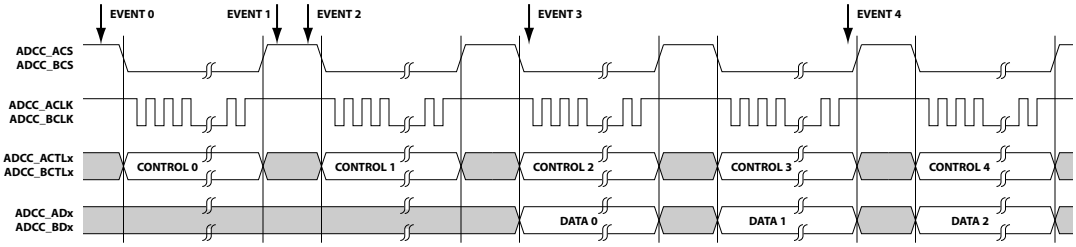


Figure 30-13: ADC0 Tight Sampling

ADC0 can support pipelining events in such a way that, sampling and reading of samples can be done back to back on every event operation as shown above. It is a case of true tight pipelining because CONTROL, CONVERSION and DATA cycles pertaining to subsequent pipelined events can all occur at same time.

ADCC1 Operation Example

This example assumes that event timings are configured for maximum throughput with single channel/MUX sampling.

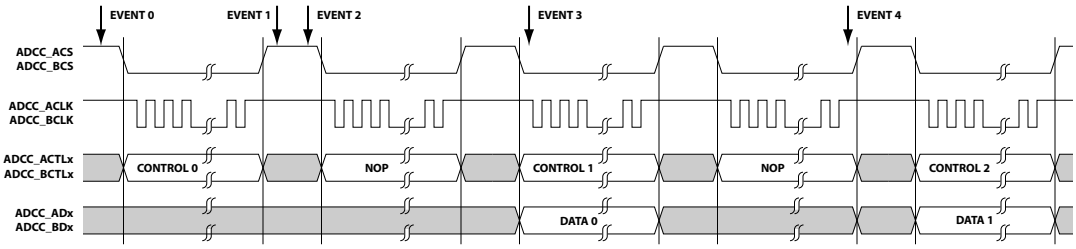


Figure 30-14: ADC1 and ADC2 1x Tight Sampling

ADC1 and ADC2 use track-and-hold circuits to sample input channels. Multiplexers control these circuits, supporting simultaneous sampling of three channels. When sampling from the same multiplexer back-to-back, the program needs to insert a NOP. The NOP ensures that no control word is sent.

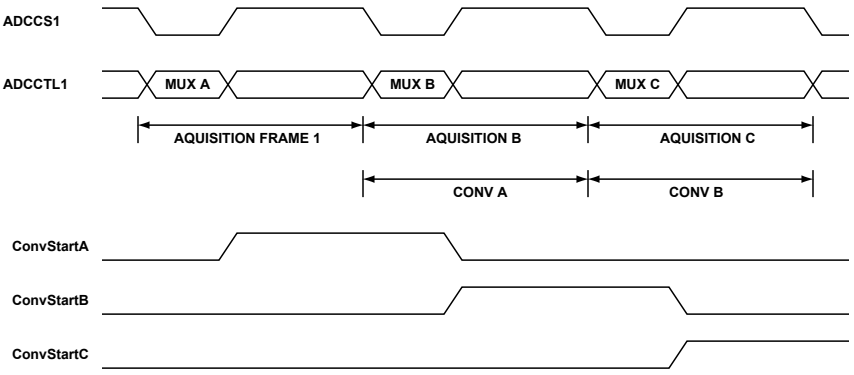


Figure 30-15: Consecutive Channel Sampling, Maximum Throughput

As shown in the figure, the NOP is only required when the same multiplexer is used for consecutive sampling. If a different multiplexer (and in turn a different input channel) is used, analog data can be sampled consecutively. For more information, see Simultaneous Sampling Mode.

ADCC Architectural Concepts

The following sections describe the blocks appearing in the ADCC Block Diagram.

- Core and DMA Interfaces
- Trigger Inputs
- Timers
- Event Register Banks
- Event Comparators
- Pending Event FIFO
- Timing and Control Unit

Core and DMA Interfaces

The ADCC has a 32-bit core interface through which the core programs the ADCC control registers and reads the ADCC status registers. The ADCC can also use this interface for reading the 16-bit converted ADC data stored in event data registers in core mode.

To minimize the core overhead, the ADCC provides a 16-bit DMA interface for directly transferring converted ADC data from the event data registers to the required memory space. This interface consists of two built-in DMA units, one for each ADCC timer.

Trigger Inputs

The ADCC can accept up to six trigger inputs, based on which ADCC timers start running at processor system clock rate (SCLK). Also, the ADCC contains two 32-bit internal timers (TMR0 and TMR1). Each timer can be independently configured to use one of the trigger inputs. The `ADCC_CTL.TRGSEL0` bit field selects the trigger input for TMR0, and the `ADCC_CTL.TRGSEL1` bit field selects the trigger input for TMR1. When both ADCC timers are enabled for different trigger inputs, the ADCC uses two trigger inputs. The ADCC uses one trigger input when both ADCC timers are enabled for same trigger input or if only one ADCC timer is enabled. The ADCC ignores non-selected trigger inputs.

The TRU provides system-level sequence control without core intervention. Configure the slave-select (SSR) field of the selected trigger input to the receive triggers from a specific trigger master. In this way, the ADCC slave trigger can accept triggers asserted by that particular trigger master or asserted through software. Software writes the ID of that trigger master to one of the fields in the `TRU_MTR` register. The trigger-out response from the selected master is internally routed to the ADCC trigger input. For more information about the TRU and trigger slaves or masters, see the Trigger Routing Unit (TRU) chapter.

Timers

The ADCC provides two independent 32-bit timers (TMR0 and TMR1). TMR0 is enabled by default when the controller is enabled. TMR1 is enabled optionally using the `ADCC_CTL.TMR1EN` bit.

These timers start counting at system clock rate (SCLK) when a valid edge is detected on the selected trigger input. The timer only stops counting under one of the following conditions:

- A timer rollover occurs.
- All the events associated with the trigger have completed.

NOTE: The ADCC supports continuous data sampling, in which timer-driven sampling schedule frames abut back-to-back. In this situation, the timer runs continuously, and cycles back to zero on each new timer trigger to start a new timer frame.

The *timer-rollover case* can never happen unless the event time register of an event is programmed at some point after the trigger occurs. This programming is a practice that is contrary to the guidelines provided in the Programming Model.

In the *all-events-completed case*, the exact time at which the timer stops counting depends on the core mode or DMA mode of the ADC data read operation. In DMA mode, the timer stops counting only when all ADC data related to ADCC frame are written into processor memory and the memory write response returns successfully.

The ADCC supports continuous data sampling, in which timer-driven sampling schedule frames abut back-to-back. In this case, the timer runs continuously, and cycles back to zero upon each new timer trigger to start a new timer frame.

When an ADCC timer is disabled or the ADCC controller is disabled, the timer resets to zero.

Event Register Banks

Typically, a sample to be taken from an ADC is referred to as an event. The events can be programmed to occur at specified times after a trigger input to the ADCC. The sample can be from any ADC channel within the ADCC domain, either in asynchronous mode or synchronous mode. An event is complete when the data reception (for the event) completes.

The ADCC can handle a total of 32 events which can be independently configured and enabled. Each event can be assigned to an ADCC timer and can instruct the ADCC to start a sampling sequence for ADCs.

Each event consists of a four register bank, including an event control register (`ADCC_EVCTL[nn]`), an event time register (`ADCC_EVT[nn]`), an event status register (`ADCC_EVSTAT[nn]`), and an event data register (`ADCC_EVDAT[nn]`). The `ADCC_EVTEN` register contains event enable bits.

The event control register determines the following:

- Event assignment to TMR0 or TMR1
- Event execution on the ADC0 interface when using ADCC0 OR Event execution on the ADC1/2 interface when using ADCC1
- ADC channel selection for sampling.
- ADC MUX selection for ADC1 and ADC2.
- Requirement for simultaneous sampling with another ADC.
- Memory offset for event data storage for reading ADC data in DMA mode

The event time register specifies the time offset from the corresponding ADCC timer trigger input to the start of that particular event.(For example, the time offset is when the event must happen based on the trigger input). This time offset is specified in terms of system clock of the processor.

The event status register provides the delay in number of system clock cycles after the event match occurred. This information is useful for debugging purpose.

The result of ADC sampling (for example, converted data from the ADC) is directly stored into the event data register. The data can be accessed in core mode or can be directly DMA transferred into processor memory.

An ADCC timer frame completes when all the events associated with that ADCC timer complete after detecting a valid trigger. At least one event per ADCC timer must be enabled for the ADCC to execute an ADC sampling sequence.

Event Comparators

The event comparator block consists of 32 event time comparators, which determine when an enabled event is ready for handling. After detecting a valid edge on a selected trigger input, the ADCC timer starts running at the system clock rate. The comparators compare the ADCC timer count with the event time specified in the `ADCC_EVT[nn].TIME` bit field of the enabled event. If the time value matches, the comparators signal the timing and control unit, indicating that an event has become active and is ready to handle. In response, the timing and control unit starts the ADC sampling sequence (if the ADC interface is available).

Ifmore than one event is active during the same system clock cycle, the ADCC processes only the highest priority event. (The event in this case is associated with an ADCC timer and is assigned to the same ADC interface). All other eventsare missed (even if there was space available in the pending event FIFO). When an event is missed, the corre- sponding event miss bit is set in the `ADCC_EMISS` register.

The priority of events is fixed, with the event with the lowest event ID having higher priority than other compared events. The priority of the registers from highest to lowest is `ADCC_EVT0>ADCC_EVT1>... >ADCC_EVT22>ADCC_EVT31`. The events assigned to TMR0 have higher priority than events assigned to TMR1. If the events from both ADCC timers to the same ADC interface become active in the same system clock cycles, the event triggeredby TMR0 receives higher priority.

Pending Event FIFO

The ADCC provides a separate, 8-deep FIFO for each ADC interface. If an event becomes ready when another event is ongoing on the same ADC interface, the occurred event is stored in the pending event FIFO. The stored event is an event in collision, and the corresponding event collision bit is set in the `ADCC_ECOL` register.

If the pending event FIFO is full when an event becomes active, the event is missed, and the corresponding event miss bit is set in the `ADCC_EMISS` register.

After the ADC interface unit is free to start the new ADC sampling sequence, the event from the FIFO is shifted to the timing and control unit.

On disabling the ADCC, all pending events in the pending event FIFO are flushed.

ADCC Operating Modes

Data Transfer Modes

The ADC interface of ADCC has a data acquisition capability in which the converted data from ADC is directly stored into the data register of the corresponding event. The event setup instructs the ADCC as to the ADC to use and the channel to sample. When the ADC interface handles the events, it initiates the ADC sampling sequence by providing the control word and conversion pulse to ADC. Then, the ADCC reads the converted ADC data and collects the data into the data register of that event, `ADCC_EVDAT[nn]`.

The ADCC can read the newly received data in the ADCC_EVSTAT[nn] register either in core mode or in DMA mode, which permits storing the data in the required memory space. The ADCC supports the following methods for reading ADC samples:

- Core-driven single data reception for each ADC sample
- DMA-driven data read for multiple words transfers with minimal intervention of core.

The ADCC_CTL.DMAEN bit enables the DMA-driven mode of ADCC operation. When disabled, the ADCC uses core-driven mode. Core-driven transfers use ADCC data interrupts to signal the processor core to perform single word read from the event data register, which is ready. DMA transfers can be set up to transfer a configurable number of ADC samples to internal or external memory of processor without core intervention.

For more information, see Core-Driven Data Read Mode and DMA-Driven Data Read Mode.

Core-Driven Data Read Mode

The ADCC provides 32 event data registers (one for each ADCC event) for storing sampled ADC data associated with that event. These registers, ADCC_EVDAT[nn], are accessible in core mode. Typically, when an ADC sampling sequence completes, the newly available read data is stored in the data register of that event. The ADCC signals the data ready interrupt to the core. In the interrupt service routine, the core checks the ADCC event interrupt status register, ADCC_EISTAT, to determine which event is complete and reads the ADC sample from its data register.

The ADCC event interrupt mask register, ADCC_EIMSK, provides bits to unmask (enable) or mask (disable) the data read interrupt requests for individual events. If there are multiple events in an ADCC frame, to reduce the interrupt requests for every event in that frame, the ADCC provides a frame completion interrupt. The interrupt request is triggered once in a frame after completion of all the events. The individual data requests from events can be masked, then their data can be read based on the ADCC timer frame completion interrupt request.

In core mode, the “end of the ADCC timer frame” is when all the event data related to that ADCC timer has been received from the ADC. The ADCC timer stops at the end of frame and is reset to zero.

DMA-Driven Data Read Mode

The ADCC includes two built-in DMA units for reading the ADC samples received in event data registers. The controller provides a 16-bit DMA interface to store these ADC samples directly in processor memory without core intervention. One DMA unit handles data read requests for the events associated with TMR0, and the other DMA unit handles data requests for the events associated with TMR1.

Each DMA unit provides a set of registers to configure the DMA work-unit:

- Base pointer register (for example, ADCC_BPTR0) contains the base address of memory space for storing ADC samples.
- Frame increment register (for example, ADCC_FRINC0) contains the address increment applicable to the DMA base pointer register between the ADCC timer frames.
- Circular buffer size register (for example,ADCC_CBSIZ0) holds number of ADCC timer frames to be received.

Apart from these registers, the DMA unit uses the event offset field of control register of the respective event to store the ADC samples related to it. The ADCC_EVCTL[nn].EVTOFS field specifies the memory offset in the frame block defined by the base pointer and frame increment registers for each ADCC frame.

Each ADC timer frame can consist of many ADCC events. The ADCC_BPTR0 or ADCC_BPTR1 registers hold the base-address used for placing the event data of the first frame into memory. The event data are each placed in a location calculated from the base pointer and the event offset (for example, ADCC_BPTR0 + ADCC_EVCTL[nn].EVTOFS). The base address of the second frame is calculated from the base pointer and the frame increment (for example, ADCC_BPTR0 + ADCC_FRINC0). The base address of the third frame includes a multiple of the frame increment (for example, ADCC_BPTR0 + 2 ADCC_FRINC0). This *linear buffering mode* calculation pattern continues through the series of frames. The calculation of the address for placing the received data of an event in the n^{th} frame applies all of these factors. For example:

Location = ADCC_BPTR0 + (n1) ADCC_FRINC0 + ADCC_EVCTL[nn].EVTOFS

When the circular buffer size register (for example, ADCC_CBSIZ0) is programmed to zero, the DMA operates (as previously described) in linear buffering mode.

When the circular buffer size register is programmed to a non-zero value, the DMA operates in *circular buffering mode*. In this mode, the DMA initially follows the same pattern as given for the linear buffering. But, after a specified number of frames, the frame base pointer is reset to the base pointer address (for example, ADCC_BPTR0). It is reset to the address instead of calculating the value from the base pointer plus frame increment. The ADCC uses the circular buffer size register (for example, ADCC_CBSIZ0) to set the number of frames after which circular buffer wraparound (resetting to the base pointer) occurs. After receiving the selected number of frames, the circular buffer wraparound occurs, and the DMA begins overwriting the data from previous frames.

NOTE: Circular buffer wraparound is sometimes referred to as *circular buffer loopback*.

Considering operation in both DMA modes, the base pointer register holds the address for the following frames:

- The first frame after the ADCC is enabled (both DMA modes)
- The first frame after the core writes to the base pointer register (in linear buffering mode)
- The first frame after a circular buffer wraparound occurs (in circular buffering mode)

NOTE: There is no DMA burst feature, and each event data is written separately after completion of all the events in that frame.

In DMA mode, the end of an ADCC timer frame occurs when all the event data related to that ADCC timer is written into memory and responses have been received. The ADCC timer stops at the end of frame and is reset to zero. When there are multiple pending events to write into memory, the priority is from the highest priority for event 0 to the lowest for event 31.

Because the ADC data is 16 bits wide, bit 0 of the base pointer register and the frame increment register is non-writable and reads as 0. This read-no-write access for bit 0 is to avoid address alignment error, by ensuring 16-bit alignment. Do not attempt to access the event data registers directly while in DMA mode.

If the ADCC is disabled while a DMA transfer is in progress, the DMA completes (gracefully). It writes all data received (up to the point the ADCC was disabled) into memory. The DMA pending status for the corresponding timer (for example, `ADCC_T0STAT.DPND`) remains high until all such transactions complete on the DMA bus, including the write response signaling.

DMA Bandwidth Monitoring

In DMA mode, the ADCC provides a DMA bandwidth monitoring feature, which permits identifying whether the ADCC timer count has gone beyond an expected limit while completing a frame. The ADCC provides a bandwidth monitor register (for example, `ADCC_BWMON0`) for each ADCC timer to use bandwidth monitoring for their frames.

For example, consider a case where each frame of an ADCC timer can complete much earlier than *Q* number of SCLK cycles after receiving the trigger. The program can use the DMA bandwidth monitoring feature to check whether all the events and their DMA transfers in each ADCC timer frame complete within that time. Program the register of the related ADCC timer with a count of *Q* in this case.

The ADCC raises the error interrupt when the corresponding ADCC timer crosses this count while completing a frame. The ADCC timers count until the DMA transfer of all events completes.

If the bandwidth monitoring feature is not needed, program the bandwidth monitor register to zero, disabling this feature.

Simultaneous Sampling Mode

The simultaneous sampling feature in ADC1 and ADC2 of ADSP-CM41x permits reading samples from multiple MUXs as well as from both the ADCs in a simultaneous synchronous fashion. Thus the processor ADC system can support the following modes of simultaneous operation:

- 3x channel OR 2x channel sampling from any of MUXs (A OR B OR C) in an ADC.
- 6x channel OR 4x channel sampling from any of MUXs (A OR B OR C) between both the ADCs.
    - Example 1: 3x channel on ADC1 and 3x channel on ADC2, making it 6x together.
    - Example 2: 2x channel on ADC1 and 2x channel on ADC2, making it 4x together.

3x OR 2x Simultaneous Sampling

When 2 or all the 3 MUXs are selected in the `ADCC_EVCTL[nn].CTLWD` bit field, then the ADC is in simultaneous sampling mode. If this mode is enabled in first frame then the conversion of all subsequent MUXs happens in the next frames automatically, and data transfer to ADCC in subsequent cycles. Data is available after each conversion cycle for each of the MUX.

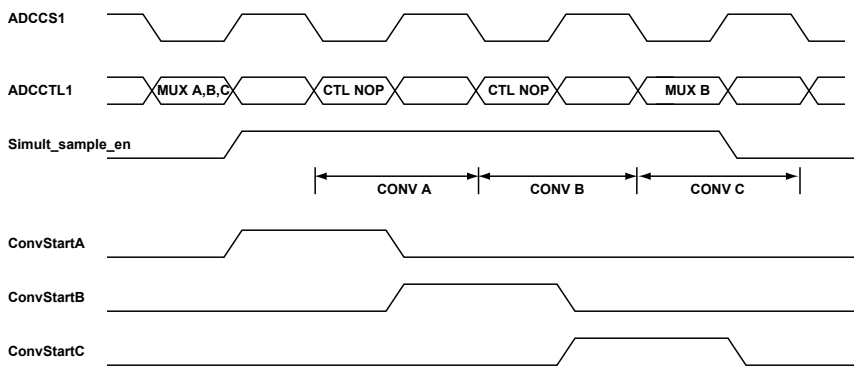


Figure 30-16: 3x Simultaneous Sampling

For simultaneous sampling to take place, the ADCC has to send NOP frames (where no MUX is enabled), after the first frame for the number of subsequent MUXs enabled in simultaneous mode. For example, if 3x sampling is enabled, the following control word programmed from the `ADCC_EVCTL[nn].CTLWD` bit field must be sent to ADC from ADCC:

- First Frame – Enable MUX A, B and C
- Second Frame – NOP Frame
- Third Frame – NOP Frame

The data is presented sequentially in the order of A, B and C. Corresponding Event numbers can be used to specify the locations to place the data.

If 2x sampling is enabled, the following control word programmed from the `ADCC_EVCTL[nn].CTLWD` bit field must be sent to ADC from ADCC:

- First Frame – Enable MUX A, B
- Second Frame – NOP Frame

The data is presented sequentially in the order of A and B. Corresponding Event numbers can be used to specify the locations to place the data.

NOTE: No consecutive use of the same MUX is allowed in any of the sampling conditions (1x/2x/3x/6x, in a maximum throughput case), while an active conversion is on-going. For instance, if A, B and C MUXs are simultaneously selected, conversion phases of all three MUXs must complete before another set of simultaneous sampling can be started with a new CTL word. However, programs can send a start conversion for another MUX. For example, for the case of 2x where A and B MUXs are selected, the program can send a control word (using the `ADCC_EVCTL[nn].CTLWD` bit field) to start a conversion for MUX B, while the MUX C conversion is still on-going.

Ideally simultaneous sampling sequences (MUX A+B+C) can't be interrupted and the sequence is followed as is, but special care must be taken if a conversion is externally triggered and can't be predicted. This operation can be called an *asynchronous timer operation*, where the two timers are completely uncorrelated. This can cause an unpredicted

event to intrude in a MUX A–B–C sequence, or to violate the requirement of NOP frames. To prevent violating these ADCC constraints, use one of the following programming models.

Synchronized, pre-planned timer schedules.

- Both timer frame periods run at integer multiples of each other (length of timer schedule).
- All event times in both timers are all placed at multiples of the same 'slot time' or ADC sample period.
- All slot usage is scheduled in advance by the user to avoid collisions and to avoid back-to-back MUX rule violations.
- Advantage is that both timers can freely use both ADCs.

Partition of resources between unsynchronized timers.

- ADCs are partitioned between timers, so they don't share.
- Disadvantage is that simultaneous sampling between ADCs is not available.
- Also, this requires partitioning the ADC input pins 50/50 between timers.
- Advantage is that both timers can run fully asynchronously from one another and no collisions between timers are possible.

6x or 4x Simultaneous Sampling

The simultaneous sampling feature ensures that the samples on two ADC channels are taken simultaneously, using a shared chip select. (For example, one sample is from ADC1 and another from ADC2). The simultaneous sampling sequence on both ADCs is initiated only after all the pending events are handled on the respective ADC interface. This functionality ensures that the two ADC interfaces are synchronized.

The ***Simultaneous Sampling with Two Events*** figure shows a case of simultaneous sampling. Event 0 is configured to initiate sampling of channel-m of ADC1, and event 1 is configured to initiate sampling of channel-n of ADC2.

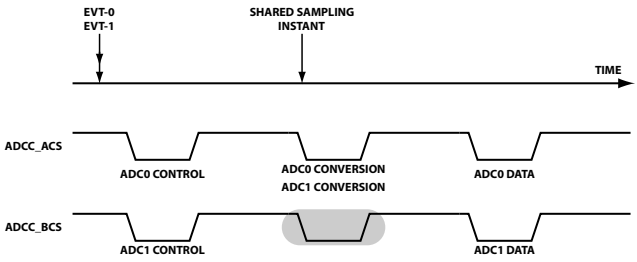


Figure 30-17: Simultaneous Sampling with Two Events

Event 0 and event 1 are triggered at the same time, because both events are assigned to same ADCC timer and the event time registers of both events are same. As there are no pending events on both ADC interfaces, the ADCC immediately chooses to start the ADC sampling sequence on both ADC interfaces synchronously.

If pending events are present on either of the ADCs, simultaneous sampling happens only after both ADCs are available. The ***Simultaneous Sample with Many Events*** figure shows a case in which events 0 and 1 sample on the

ADC1 channels. Events 2, 3, 4, 5, and 10 sample on the ADC2 channels so that event 1 and event 10 are a simultaneous sampling event pair. Assume that the ADC1 interface has only event 0 before simultaneous sampling; while the ADC2 interface has events 2, 3, 4, and 5 before simultaneous sampling. This case demonstrates the wait implemented by the ADC interface to manage pending events until it processes the simultaneous sampling pair.

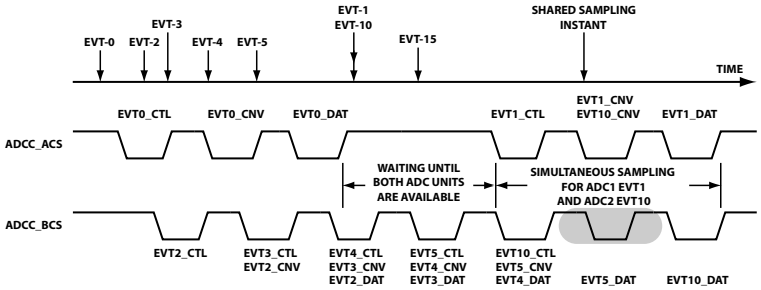


Figure 30-18: Simultaneous Sample with Many Events

From the figure, it is important to observe that when the simultaneous sampling event pair becomes active, the ADC2 interface is idle. The interface does not initiate the sampling sequence for event 1 associated with it. The ADC2 interface is busy servicing pending events activated before the paired event (event-10) is ready. The ADC1 interface waits until the ADC2 interface becomes available to handle the paired event. Also, note that the ADC interface treats simultaneous sampling event pairs as individual (non-paired or normal) events based on prioritization. There is no priority assigned to paired events over normal events.

Due to the inherent nature of the ADC interface to handle the events in serial, the event triggered first is handled first. The ADC interface handles subsequent triggered events only after all other triggered pending events. If another event (for example, event 15) is triggered on ADC1 interface, the event is stored in pending event FIFO, even if ADC1 interface is not handling any event. Event 15 is handled only after execution of the pending events triggered before it (for example, after executing event 1).

Refer to the ***Simultaneous Sampling with Two Events*** figure and the ***Simultaneous Sample with Many Events*** figure. While executing simultaneous sampling events, the conversion pulse to ADC2 in the ADC sampling sequence is not given from ADC2 interface, but is shared from the ADC1 interface.

Diagnostic Mode

The processor has a special feature to verify the internal supply and signals for the user in a diagnostic mode. In this mode, the ADC1 OR ADC0 captures data from following signals instead of the actual external ADC pins. It is also possible to route the internal 12-bit DAC output to ADCs as given below. This diagnostic feature is enabled by setting the `DIAG_ADC_CHN_EN` in the `AFE Registers (AFE_Regs.h)` register.

Table 30-12: ADC1 Multiplex and Channels

Multiplex/Channel	2	1	0
A	DAC	AVDD0	REGCAPA0
B	DAC	AVCOMP	REGCAPD

Table 30-12: ADC1 Multiplex and Channels (Continued)

Multiplex/Channel	2	1	0
C	DAC	REFBUFOUT0	AVDD0

Table 30-13: ADC0 Channels

Channel	Input
0	REGCAPA0
1	REGCAPA1
2	REFBUFOUT2
3	REFBUFOUT1
4	DAC
5	AVDD1

NOTE: Please consult the product specific datasheet for the specifications of all of these signals except for the DAC output signal.

ADCC Event Control (SEC and TRU Related)

For the ADCC, the term *event* typically refers to ADC-related sampling and data conversion. The processor considers events as interrupt requests (related to the NVIC/SEC) and triggers (related to the TRU).

The ADCC provides status and error bits through different registers to signal the core about its state and various error conditions that occurred during its operation.

The ADCC provides a primary error status register (ADCC_ERRSTAT) and two supplementary error status registers (ADCC_ECOL and ADCC_EMISS). The supplementary registers provide more information about the errors flagged in ADCC_ERRSTAT register. Optionally, the ADCC can generate an error interrupt request based on these conditions.

Also, the ADCC provides an event completion status register (ADCC_EISTAT), which is used in core mode to service the ADC data read requests. A frame interrupt status register (ADCC_FISTAT) provides timer frame completion status, which indicates whether all the events related to a frame have completed successfully. Optionally, the ADCC data interrupt request can be generated based on these conditions.

The ADCC provides a timer status register (for example, ADCC_T0STAT) for each ADCC timer. This status register indicates the current frame number the ADCC is handling. Also, this register indicates whether DMA is pending reception of data from the ADC. An event pending register (ADCC_EPND) indicates which events are yet to start after the trigger.

More information about ADCC features relating to processor events (NVIC/SEC interrupts and TRU triggers) is available in the following sections:

- Interrupt Status

- Error Status
- Pending, Frame, and Delay Status
- Event Handling Latency

Interrupt Status

The ADCC provides a data interrupt channel for each ADCC timer. ADCC_TMR0_EVT carries the TMR0 related interrupts, and ADCC_TMR1_EVT carries the TMR1 related interrupts. The ADCC uses the same interrupt channel whether executing data reads in core mode or in DMA mode.

When operating in core mode, the ADCC uses the same data interrupt request to service the data read requests from each event. The event completion bit (ADCC_EISTAT.EVT[nn]) indicates that data for the corresponding event is pending and ready for the core to read. Based on the selection in the timers select bit (ADCC_EVCTL[nn].TMRSEL) of the event, the corresponding event interrupt requests are sent on either the ADCC_TMR0_EVT or on the ADCC_TMR1_EVT channel. Optionally, this interrupt request from each event can be individually masked off from being signaled, using the bits in the ADCC_EIMSK register. The event data ready bits in ADCC_EISTAT register are sticky. Clear these bits with a W1C operation.

The ADCC supports Last frame completion status (LFINT) as well as per Frame completion status (FNIT) described below.

Frame Completion Status

The frame completion status bits for each ADCC timer (ADCC_FISTAT.FINT0 and ADCC_FISTAT.FINT1) are applicable when using either core or DMA mode. In core mode, this bit is set when data corresponding to all events (which were not missed) in a frame are received from the ADC. In DMA mode, this bit is set when:

- the data corresponding to all the events of a frame are received
- the data is DMA-transferred into processor memory, and
- are successfully received from the SCB

Clear the frame completion status bits to acknowledge before the next trigger appears. Otherwise, the trigger is ignored, and the trigger overrun error condition is flagged. Clear these sticky status bits with a W1C operation.

Last Frame Completion Status

The last frame completion status bits for each ADCC timer (ADCC_FISTAT.LFINT0 and ADCC_FISTAT.LFINT1) are applicable when using either core or DMA mode, and if total number of frames (ADCC_NUMFRAM0/ADCC_NUMFRAM1) in the ADCC is a non-zero value. In core mode, this bits get set when all data corresponding to events (which were not missed) of all the frames are successfully received by ADCC from ADC. In DMA mode, this bit is set when:

- the data corresponding to all events of all the frames in total number of frames are received
- the data is DMA-transferred in to processor memory, and

- the memory write responses are successfully received from the SCB.

Umasking the Status Interrupts

The frame interrupt mask register (ADCC_FIMSK) unmasks (enables) OR masks (disables) the frame completion interrupts (Last Frame Interrupt and Frame Interrupt) from each ADCC timer.

NOTE: It is possible to perform back to back continuous sampling without having to clear the FINTn/LFINTn status bits. For this, the corresponding bits in the ADCC_FIMSK register must be used to mask these interrupts. In such a case, ADCCs can continue to do further sampling without any core intervention at all. Additionally no trigger over run would be registered.

If the status interrupts as unmasked: Clear the frame completion status bits to acknowledge before the next trigger appears. Otherwise, the incoming trigger is ignored, and the trigger overrun error condition is flagged which can lead to data loss. Clear these sticky status bits with a W1C operation. This operation can be done inside the interrupt handler, and optimally done by using a Memory DMA approach, where the MDMA write operation is automatically triggered from an ADCC trigger output.

ADCC Interrupts

There are two interrupt handlers for the ADCC: ADCC_ERR for all error interrupts ADCC_TMR_EVT for event interrupts and frame status interrupts.

Error Status

The ADCC can signal error conditions during its operation. It reports these conditions in an error status register (ADCC_ERRSTAT), which holds the error status for the following:

- Trigger overrun error
- Event miss in frame error
- Event collision in frame error
- DMA bandwidth monitoring error
- Memory write response error

The trigger overrun error, DMA bandwidth monitoring error, and memory write response errors for ADCC timers are individually flagged through separate bits.

The event miss bit is flagged when any one of the events are missed during the ADCC frame. The ADCC provides a separate event miss register (ADCC_EMISS), which indicates which event was missed.

Similarly, the event collision bit is flagged when any one of the events collided with an active event. For example, this flagging occurs when an event becomes active while the ADC interface is already handling another event. The collided event is placed in pending event FIFO, and its handling is delayed. The ADCC provides a separate event collision register (ADCC_ECOL), which indicates which event collided.

The ADCC provides an error interrupt channel (ADCC_ERR) to signal these error conditions to the core. This interrupt request carries error-related interrupts for both ADCC timers and for both ADC interfaces. By default, all the error conditions generate interrupt requests. But, the error conditions can be individually masked from appearing on the ADCC_ERR signal by programming the error mask register (ADCC_ERRMSK).

The following sections provide additional information about the most common ADCC error conditions.

Trigger Overrun Error Status

Trigger overrun is a condition in which the ADCC timer detects a valid trigger edge on the selected trigger input before a previously triggered frame completes.

The ADCC timer starts counting when it detects a valid edge on its trigger input. Then, the ADCC handles the events associated with the trigger by initiating ADC sampling sequences at the required time offsets. After all the events complete, the frame is completed, and the ADCC sets the frame completion status bit (for example, ADCC_FISTAT.FINT0). In DMA mode, the frame completion status bit is set when:

- the data corresponding to all the events of a frame are received
- then DMA is transferred into processor memory, and
- the successful memory write responses are received from the SCB.

If the status interrupts (FINT/LFINT) are unmasked via the ADCC_FIMSK regiater, then the frame completion conditions must be acknowledgedby clearing these bits in the software. After the bit is cleared, the ADCC timer is ready to accept new trigger, initiating a new frame. If the new trigger pulse is receivedbefore this time, it is treated as prematuretrigger, and the trigger overrun condition is flagged.

When this condition occurs, the ADCC ignores this trigger, sets the respective trigger overrun status bit (for example, ADCC_ERRSTAT.TRGOV0), generates the error interrupt request (if enabled), and continues its operation normally.

A trigger overrun condition also occurs when a new trigger is detected while ADC programming is in-progress (indicated by the ADCC_ADCRW0.PND bit). This case can be avoided if the timer trigger enable (for example, the ADCC_CTL.TRGIE0 bit) is disabled before ADC programming then re-enabled at the end of ADC programming.

Event Miss Error Status

An ADCC event is considered missed when the ADC sampling sequence corresponding to that event is not executing on the required ADC interface. When an event is missed, the ADCC_ERRSTAT.EMIS bit is flagged and in the ADCC_EMISS.EVT[nn] bit corresponding to that event is set to provide more information to the application.

In the following scenarios, a programmed event can be missed:

- *FIFO overrun*

  An event is missed if the event pending FIFO is full when the event match occurs.

- *Event time conflict*

  One or more events is missed if:

  - two or more events requiring sampling at the same ADC interface are assigned to same ADCC timer, and
  - the events have identical event time values in their event time registers.

  Only the higher priority event is forwarded, and the lower priority events are missed.

- *Non-paired simultaneous sampling*

  An event is missed if it is enabled for simultaneous sampling (`ADCC_EVCTL[nn].SIMSAMP` bit) on one ADC interface, but there is no paired simultaneous sampling event programmed for the other ADC interface.

- *ADC interface conflict for simultaneous sampling pair*

  An event is missed if both events in a simultaneous sampling event pair are configured for same the ADC interface. The event pair in this case is considered as two normal events (not a simultaneously sampled pair), and the lower priority event is missed.

- *Missed half of simultaneous sampling pair*

  Both events are missed if one event in a simultaneous sampling pair is missed at one ADC interface due to any event miss reasons previously mentioned.

The priority of event is based on its event number ID. For example, the event with lowest event ID has higher priority compared to other events, such that EVT0>EVT1>…>EVT22>EVT31. The priority does not depend on whether it is assigned to the ADC1 interface or to the ADC2 interface. The priority does depend on whether the event is assigned to TMR0 or assigned to TMR1. The events assigned to TMR0 have higher priority than events assigned to TMR1.

This priority is checked only at the event comparator block, when more than two events become active at the same system clock cycle. After the events are queued, the ADC interface does not check the priority, and events are handled in sequence as they become active.

As mentioned in *event time conflict*, low priority events are missed when two or more events are assigned to same ADCC timer with identical event time register value. (Events are assigned to same ADC interface). Consider a case in which two events (EVT1 and EVT5) occur simultaneously.

- If both events are assigned to same ADCC timer and route to the same ADC interface (ADC1 or ADC2):
  - the EVT5 event (being the lower priority event) is missed, and
  - the EVT1 event is forwarded to the ADC interface for handling.

Do not assign the same event time for multiple normal events of same ADCC timer and send them to the same ADC interface. It is the responsibility of the programmer to ensure that the values in the event time registers do not lead to event misses.

- If one of the events is assigned to TMR0 and the other to TMR1 (both route to same ADC interface), the event assigned to TMR0 is forwarded first to the ADC interface. Then, the event assigned to TMR1 is forwarded. Typically, the event is stored in the event pending FIFO when the FIFO is not full. This case appears when sources that are not synchronized trigger TMR0 and TMR1. And, the event time register is not always the same for both events. It is important to consider the possibility of events occurring either simultaneously or being missed when enabling events on two asynchronously triggered timers. Sufficient time must space the events, so the pending event FIFO does not operate at its maximum capacity.
- If both events are assigned to the same ADCC timer, both events are forwarded to the respective ADC interfaces for handling, assuming there is no pending event. (In this case, one event routes to the ADC1 interface and the other to ADC2 interface). A simultaneous event pair is assigned to same ADCC timer with same event time but going to different ADC interfaces. If one of the events is missed for some reason, the paired event also is missed. (For example, the pending event FIFO on the particular ADC interface is full or if another high priority event from same timer becomes active in the same system clock cycle)

NOTE: To reduce the event miss condition resulting from pending event FIFO full conditions, the ADCC provides an 8-deep event FIFO for each ADC interface. But, the programmer must ensure that the events are sufficiently spaced apart from each other (event handling in pipelined manner can also be considered). The programmer must ensure that event miss conditions do not lead to FIFO overrun.

Event Collision Error Status

If event times are not sufficiently spaced apart, an event could occur while a previous event is underway. An ADCC event is considered *in collision* if:

- that event becomes active (the event comparator unit signaled it as ready-to-handle) when the ADC interface is busy handling a previous event, or
- there are already some events in the pending event FIFO waiting to be handled

In such cases, the newly active event cannot be issued to the ADC at the earliest possible time because of some previous events are not complete yet.

The ADCC provides an 8-deep FIFO for queuing collided events. The `ADCC_ERRSTAT.ECOL` bit flags the collision condition, and the `ADCC_EVT[nn]` bit corresponding to that event is set to provide more information to the application. The ADCC also provides a means for the application to know (by how many system clock cycles) the event was delayed. It indicates this time in the `ADCC_EVSTAT[nn].DLYCNT` field of that event, which is updated only when the control word (`ADCC_EVCTL[nn].CTLWD`) for the event is sent to the ADC.

If one event of a simultaneous sampling pair undergoes collision, the other event (in the simultaneous sampling pair) also is indicated as undergoing collision.

Due to inherent nature of ADCC to queue the ready events, the collided event can be handled before completion of all the pending events. For example, this previous event (for instance, $(n-1)^{th}$) is completed only after the reception of its data. When a collision occurs, it indicates that the n^{th} event is sent to the ADC only after some delay. But, it is not after the completion of $(n-1)^{th}$ event. When a chip select pulse of the conversion phase or data phase of $(n-1)^{th}$ word must be sent to the ADC, the ADCC state-machine sends the control word for n^{th} event with the earliest chip select pulse.

NOTE: The event collision is an error condition in which the expected sampling time of the ADC for the collided event can be delayed. The event is not missed.

DMA Bandwidth Monitoring Error Status

The ADCC can be configured to store the ADC samples related to enabled events directly in the processor memory through the built-in DMA unit. When using this mode, the ADCC provides a DMA bandwidth monitoring feature, which identifies whether the ADCC timer count has gone beyond an expected limit while completing its frame. The bandwidth monitor register (for example, `ADCC_BWMON0`) specifies this count.

When the ADCC timer count crosses the count specified in bandwidth monitor register while completing a frame, the bandwidth error bit flags the DMA bandwidth error for the respective DMA unit. (An example of the bandwidth error bit is `ADCC_ERRSTAT.BWERR0`).Clear this sticky error bit with a W1C operation. Optionally, the ADCC can assert the error interrupt request to the system on a bandwidth error. Optionally, the ADCC can use an error interrupt to notify the core of this error condition.

Memory Write Response Error Status

If the ADCC detects a memory write error response that corresponds to a data write issued for an ADCC timer event, the memory response error bit flags the condition. (An example of the memory response error bit is `ADCC_ERRSTAT.MERR0`). Clear this sticky error bit with a W1C operation. Optionally, the ADCC can use an error interrupt to notify the core of this error condition.

Pending, Frame, and Delay Status

In addition to data request status and error bits, the ADCC provides bits for DMA pending status, event pending status, current frame number, and event delay status. These conditions operate as follows:

DMA Pending Status

If the ADCC is disabled while a DMA transaction is in-process, the DMA finishes (gracefully). It does not shut down until the data corresponding to all completed events is sent to memory and responses are received. The DMA pending bit (for example, `ADCC_T0STAT.DPND`) indicates the DMA activity associated with an

ADCC timer. This status bit is useful for cases in which the ADCC is later re-enabled. The application typically polls this bit before re-enabling the ADC controller.

If the ADCC is disabled while a DMA transaction is in-process, the DMA finishes (gracefully). It does not shut down until the data corresponding to all completed events is sent to memory and responses are received. The DMA pending bit (for example, `ADCC_T0STAT.DPND`) indicates the DMA activity associated with an ADCC timer. This status bit is useful for cases in which the ADCC is later re-enabled. The application typically polls this bit before re-enabling the ADC controller.

Event Pending Status

The ADCC provides an event pending register (`ADCC_EPND`) to indicate the events within the current frames that are pending execution. This register contains a dedicated bit corresponding to each ADCC event. At the trigger pulse, when a frame corresponding to an ADCC timer is initiated, all the bits corresponding to enabled events in that frame are set. The bits are cleared upon receiving the data from the ADC for the corresponding events. Because the bits in this register indicate whether the event is pending, if an event is missed, the respective bit also is cleared.

.

Current Frame Status (Number)

In DMA mode, the application can require knowledge of the number of frames the ADCC handled. The current frame count bit field (for example, `ADCC_T0STAT.CURFR`) of the associated ADCC timer provides this information. The field indicates the current frame number in chronological order after the ADCC was enabled (in linear buffer mode). Or, the field indicates the frame number after the last wraparound (in circular buffer mode). This field wraps over after 0xFFFF frames.

Event Delay Status

When an event is in collision, the application can need knowledge of the amount of time the event was delayed (in system clock cycles). The ADCC uses the delay count bit field (`ADCC_EVSTAT[nn].DLYCNT`) of the event to indicate this information. The ADCC updates the field only when it sends the control word of the event to the ADC.

Event Handling Latency

The event handling latency of the ADC interface is the time between the internal occurrence of the event and start of the ADC sampling sequence for the event. The latency is the time between:

- the event time value matching the ADCC timer count value, and
- the assertion of chip-select for the control phase of the ADC sampling sequence

If an event becomes active when the ADC interface of ADCC is idle, the timing and control unit immediately starts handling that event. (The event has a predictable latency of 4–5 system clock cycles).

If the event becomes active when the ADC interface is busy handling a previous sampling event, the new event is held in the pending event FIFO. The latency increases by the duration that the new event is held in the pending event FIFO. To reduce the latency in servicing this new event, the ADC interface can decide to queue the sampling sequence for this event with the sampling sequence of an ongoing event. For more information about this operation, see the ADCC Functional Description.

For a simultaneous sampling event pair, the latency can depend on the activities on other the ADC interface. This event pair is executed only when both ADC interfaces are ready.

Sometimes there can be few cycles of latency at the trigger routing unit (TRU) of processor when providing trigger inputs to the ADCC from different trigger master sources. This situation is especially prone to occur when an asynchronous external trigger input is selected as the trigger input of ADCC. The time needed for synchronization to this external signal can lead to few system clock cycles of delay.

The ADCC provides an event delay count status field (`ADCC_EVSTAT[nn].DLYCNT`) for each event to convey the latency that occurred in handling an event. This count indicates the number of system clock cycles the event was delayed, after the event match occurred. This field gets updated in every ADCC timer frame, when the ADC convert start signal goes active to transmit the control word to the ADC corresponding to that event.

The ADCC provides an event delay count status field (`ADCC_EVSTAT[nn].DLYCNT`) for each event to convey the latency that occurred in handling an event. This count indicates the number of system clock cycles the event was delayed, after the event match occurred. This field gets updated in every ADCC timer frame, when the ADC convert start signal goes active to transmit the control word (`ADCC_EVCTL[nn].CTLWD`) to the ADC corresponding to that event.

ADCC Programming Concepts

There are general programming concepts for using the ADCC when it is present on any processor.

NOTE: There can also be processor-specific guidelines for programming the ADCC.

Fully configure the ADCC including trigger setup, ADCC events, and DMA programming (if necessary) before enabling the controller by setting `ADCC_CTL.EN` bit. Set the trigger input bits (for example, `ADCC_CTL.TRGIE0`) just before enabling the ADCC.

Do not configure two or more events assigned to the same timer and going to the same ADC interface with the same event time. If the ADCC has this configuration, other event miss error conditions can occur.

If different sources that are not synchronized trigger the ADCC timers, it is important to consider the possibility of events occurring either simultaneously or being missed. Space apart the events with sufficient time, so the pending event FIFO does not operate at (or beyond) its maximum capacity.

The frame completion status bit must be acknowledged for every frame. If they are not acknowledged, trigger overrun conditions can occur.

There is no restriction on dividing the events among ADCC timers and among ADC interfaces.

When configuring a simultaneous sampling event pair, certain restrictions apply in terms of configuring event attributes. See Simultaneous Sampling Mode for more details.

The event registers can be modified after the ADC controller is enabled; these registers include the event time (ADCC_EVT[nn]), event control (ADCC_EVCTL[nn]), and event enable (ADCC_EVTEN). Do not modify these registers when the frame is in-progress. If necessary, modify these registers only after completion of the frame associated with the ADCC timer to which the event is assigned. Modify them before clearing the frame completion interrupt status bit. These registers must be stable when the trigger arrives.

In linear DMA mode, the base pointer register (for example, ADCC_BPTR0) and frame increment register (for example,ADCC_FRINC0) can be modified after enabling the ADC controller. Do not modify the base pointer register in circular buffer DMA mode. The changes in DMA registers are considered only in the next frame.

Do not disable the ADC controller while a frame is in-progress (for example, disable the ADCC only after the frame interrupt). This protocol prevents unfinished transfers on ADC interface. Before re-enabling the ADCC, poll the DMA pending bit (for example, ADCC_T0STAT.DPND) to confirm that there is no DMA activity. Before reconfiguring the ADCC_CTL register, disable the ADCC. The only write allowed into an enabled ADCC control register is to disable it.

The ADCC status registers and counter-values are retained upon disabling ADCC (for debug purposes) and are cleared on re-enabling the controller (a 0-to-1 transition on the ADCC_CTL.EN bit). Error status bits are not cleared with this transition. Clear errors with a W1C operation.

ADCC Control and Timing Registers

The ADCC block is a common design extended for other products. Some of the interface programming (using ADCC Control and Timing registers) are specific to the ADSP-CM41xF processor and should be programmed according to the specification given below. Alternate values are not applicable for ADSP-CM41xF.

BIT name in ADCC_CTL	Value
MODE	1
CSIZE	0
DSIZE	1
TRGPOL0	1
TRGPOL1	1
CKPOL0	0
CKPOL1	0
CSPOL0	0
CSPOL1	0
TIDLE0	0
TIDLE1	0
DSWP0	0

BIT name in ADCC_CTL	Value
DSWP1	0
LSBF0	0
LSBF1	0

BIT name in ADCC_TCAy	Value
CKTGRE	1
NCK	8

BIT name in ADCC_TCBy	Value
TCSCK	1
TCSCS	0

Control Word Format

The following tables present the controlword format programmed using the `ADCC_EVCTL[nn].CTLWD` bit field. The actual Control Word furnished below is 8-bit and is LSB-aligned in the 16-bit `ADCC_EVCTL[nn].CTLWD` field.

Table 30-14: ADC1 and ADC2 Control (CTL) Word Format

Bit Position	Field	Entries	Program as
[7:5]	(ADC_CHAN) ADC Channel Number, there are up to 8 channels.	7 = CHAN 7 6 = CHAN 6 5 = CHAN 5 4 = CHAN 4 3 = CHAN 3 2 = CHAN 2 1 = CHAN 1 0 = CHAN 0	ADC Channel Number
[4:2]	(ADC_MUX) ADC Channel Mux, Multiple selections allowed for first event of single channel 2x/3x simultaneous sampling	4 = MUX A 3 = MUX B 2 = MUX C	Select any or all muxes
[1]	Share CS (Share chip select)		Program as 1 if simultaneous sampling is necessary
[0]	RESERVED		Program as 0

Table 30-15: ADC0 Control (CTL) Word format

Bit Position	Field	Entries	Program as
[7]	RESERVED		Program as 0
[6:4]	(ADC_CHAN) ADC Channel Number, there are up to 7 channels.	6 = CHAN 6 5 = CHAN 5 4 = CHAN 4 3 = CHAN 3 2 = CHAN 2 1 = CHAN 1 0 = CHAN 0	ADC Channel Number
[3:0]			Progam as 0

ADCC Continuous Sampling Sequence

The following table shows a method to understand the back-to-back sampling sequence when a minimum chip select for the three ADCs is assumed. The table can be interpreted as control word and timing sequences for achieving maximum throughput scenarios.

ADC CS Period (TCS)		340ns	340ns	340ns	340ns	340ns
ADC CS Period Count		1	2	3	4	5
ADC 1 OR ADC2 3x sim-sampling	Control Phase	CTLWD: MUX ABC & CHNL	CTLWD: NOP (No Mux)	CTLWD: NOP (No Mux)	NOP (delay)	CTLWD: MUX ABC & CHNL
	Conversion Phase		CNV A	CNV B	CNV C	
	Data Phase			DATA A	DATA B	DATA C
ADC CS Period Count		1	2	3	4	
ADC 1 OR ADC2 2x sim-sampling	Control Phase	CTLWD: MUX AB & CHNL	CTLWD: NOP (No Mux)	NOP (delay)	CTLWD: MUX AB & CHNL	
	Conversion Phase		CNV A	CNV B	NOP (delay)	
	Data Phase			DATA A	DATA B	
ADC CS Period Count		1	2	3	4	

ADC 1 OR ADC2 1x sampling	Control Phase	CTLWD: MUX A & CHNL	NOP (delay)	CTLWD: MUX A & CHNL	NOP (delay)	CTLWD: MUX A & CHN
	Conversion Phase		CNV A	NOP (delay)	CNV A	
	Data Phase			DATA A	NOP (delay)	DATA A
ADC CS Period (TCS)		500ns	500ns	500ns	500ns	500ns
ADC CS Period Count		1	2	3	4	5
ADC0 1x sampling	Control Phase	CTLWD: CHNL	CTLWD: CHNL	CTLWD:CHNL	CTLWD:CHNL	CTLWD:CHNL
	Conversion Phase		CNV	CNV	CNV	CNV
	Data Phase			DATA	DATA	DATA

Initializing the AFE

The ADCs, DACs and FOCP internal registers must be initialized in order to use AFE components. This is accomplished using a programming sequence through ADCC0. In addition to these internal registers, read and write access is available for the following registers.

- `AFE Registers (AFE_Regs.h)` – Performs initialization and turning on additional features
- AFE Registers (AFE_Regs.h) – Status register reflecting status of FOCP
- `AFE Registers (AFE_Regs.h)` – Status register reflecting status of ECC of Internal registers

Internal Register Error Correction

As mentioned above, there are 10 banks of internal registers that are initialized during the initialization phase of the AFE. Refer to AFE Initialization for details on how to initialize AFE. Single Bit Error Correction is available for these registers, while Double Bit Errors are indicated in the `AFE Registers (AFE_Regs.h)` register. The ECC check function is always turned on for these registers.

RESET for AFE

A processor hard reset and system reset are not connected to the Analog Front End including the ADCs. When these reset signals are asserted, only their digital counterpart such as ADCCs or DACCs are reset. The USERCONFIGREG0 registers that are mentioned above for AFE Initialization are read from internal AFE hardware, and they typically do not need to be reset when the digital domain is reset. However, if desired all the AFE registers maybe reset by doing a power recycle only of the AFE domain. Consult the data sheet for information about the AFE domain power supplies.

AFEOK Signal

The AFE (Analog Front End that comprises of ADCs) provides a signal to the MCU indicating that it is operating properly. Deassertion of this signal indicates that the ADC data being provided to the MCU may no longer be within specification and/or the AFE the functioning of the AFE is not correct. For safety purposes, the system designer may elect to cause deassertion of this signal to cause a processor interrupt and/or immediate hardware responses.

AFEOK is a safety signal and if it is not asserted, and the issue can be anywhere in AFE domain. Programs need to verify all required hardware and software related to AFE operation. Since the verification is perfromed in the initialization stage, there is no need to verify the sampling part of software with ADCC registers. Typical scenarios of failure includes power supply issues with the AFE or issues with internal hardware, or internal LDOs.

The AFE_OK signal is active high, and is initially expected to be deasserted (low) at reset. The state of the AFEOK is visible in the `SYSBLK_SYSSTAT.AFE_OK` register bit in both SYSBLK0 and SYSBLK1.

The AFEOK signal is provided with the following event connections. The interrupts show an inverted state. AFE_NOTOK interrupt is asserted when the AFEOK signal is deasserted.

Interrupts

- The AFE_NOTOK interrupt to the Cortex-M4 core
- The AFE_NOTOK interrupt to the Cortex-M0 core

Trigger Masters

- The AFE_NOTOK trigger master in the Cortex-M4 TRU1 unit

Since the AFEOK is initially (default state after reset) deasserted, the program has to initialize the AFE (as given in the programming guidelines) before initializing the ADCC for conversion or before initializing interrupts.

6x Sampling

While setting up individual event control and timing remains almost the same for the 6x sampling case, the following bits must also be included in Control Register:

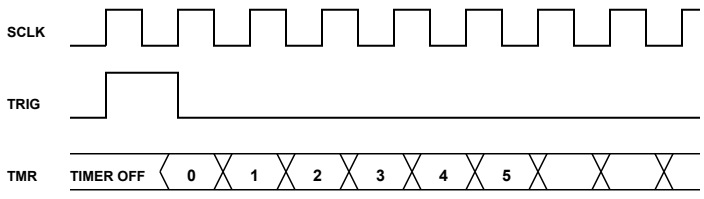
- Enable the `ADCC_EVCTL[nn].SIMSAMP` bit.
- Enable all MUXA, MUXB, MUXC in the `ADCC_EVCTL[nn].CTLWD` bit field.
- Bit [1] of the `ADCC_EVCTL[nn].CTLWD` bit field can optionally be set to 1 so that a shared chip select is used between ADCs. This is only required for more precise synchronization.
- If simultaneous sampling is selected, the same channel number is used. For instance, if Channel 0 is selected for A and B MUX, then same channel number, 0 is used for both A and B.

Precise Event Timing

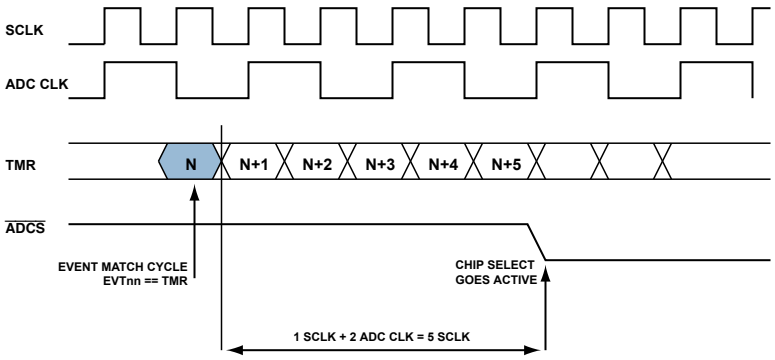
This section describes the various Programming guidelines for achieving precise sampling times at the ADCs. Wherever two ADCs are not mentioned, the information is applicable for all ADCs including ADC0 and the clock can be either SCLK or SYSCLK based on the domain.

1. To access two ADCs using a timer's events, the clock division ratios of both ADCs must be equal. Various timing parameters in the ADCC_TCA0/ADCC_TCA1/ADCC_TCB0/ADCC_TCB1 registers should also be programmed with same values for both ADCs.
2. To access two ADCs using a timer's events, the SYNCCK feature has to be enabled in the ADCC_TCA0/ ADCC_TCA1 registers and both ADCs should select the same trigger input for synchronizing their clocks. Avoiding the SYNCCK feature can bring in an additional uncertain delay of up to N–1 system clock cycles (where 1:N is clock division ratio) between the triggers to samples.
3. Each timers' triggers should be given after mN number of system clock cycles after the synchronizing trigger input given to ADC(s), where 1:N is the clock division ratio and m is any positive integer.
4. For a clock division ratio of 1:N, all the event times programmed in the ADCC_EVT[nn] registers must be multiples of N. (for example 0 , kN etcetera where k is a positive integer)
5. An event is considered NON-PIPELINED if its event time EVT(n) is such that there are no previous events with event times in the range [EVT(n) – 3 × t_{CSP} × N, EVT(n)]. A NON-PIPELINED event has a delay of {1 × SYSCLK + 2 × ADCC_ACLK} after the event matches the SYSCLK cycle until the control chip select is sent out. For 1:2 ratio of SYSCLK: ADCC_ACLK, this is SYSCLK periods. Here precise timing of NON-PIPE-LINED events is possible.
6. For PIPELINED events, precise pipe-lining is possible if the difference between event times is always maintained to be an integer multiple of t_{CSP} × N (1:N being the clock division ratio). If this guideline is followed, each PIPELINED event will have a delay of:

   {1 × SYSCLK + 2 × ADC clocks} after the event match SYSCLK cycle until the control chip select is sent out. For 1:2 ratio of SYSCLK: ADC clock = 5 SYCLK periods.
7. If multiple frame's events are getting pipelined (they do not satisfy condition 6 above), the triggers which initiate the frames must be spaced apart by multiples of t_{CSP} × N number of system clocks. (1:N being the clock division ratio).
8. The TIMER counts start one cycle after a trigger input. Thus the TIMER count's relation to trigger input is:



   An event programmed at EVT=0 goes out 2 SYSCLKs + 2 ADCC_ACLK after the SYSCLK period with the trigger pulse active.
9. If the precise timing guidelines as mentioned above are followed for all events, the delay from the event match SYSCLK cycle till the Chip select going active is fixed to be 1 SYSCLK + 2ADC clock cycles. This is explained for the case of 1:2 clock division ratio in the figure below.



General Programming Guidelines

The discussion in the following sections provides the general programming model of the ADCC.

- All Mask, Set or Clear registers can be modified while ADC controller is in operation.
- Status clearing (by Write-1-to-Clear) is allowed while ADCC is in operation.
- Apart from the above, the ADCC_BPTR0/ADCC_BPTR1, ADCC_EVT[nn], ADCC_FRINC0/ADCC_FRINC1, ADCC_EVCTL[nn] and ADCC_EVTEN registers are the only ADC control registers that may be modified after the ADC controller is enabled.
- In the circular buffering mode of operation, the ADCC_BPTR0/ADCC_BPTR1 registers can be modified after the number of frames programmed in the ADCC_CBSIZ0/ADCC_CBSIZ1 registers are completed, provided that the value programmed in the ADCC_NUMFRAM0/ADCC_NUMFRAM1 registers has been used to generate the LFINT interrupt after ADCC_CBSIZ0/ADCC_CBSIZ1 frames are completed.

  Write the ADCC_BPTR0/ADCC_BPTR1 registers prior to the reception of the next trigger. (The new write can be done using an interrupt service routine initiated by LFINT if the ADCC_NUMFRAM0/ADCC_NUMFRAM1 registers are programmed to generate the interrupt after the number of frames configured in the ADCC_CBSIZ0/ADCC_CBSIZ1 registers).
- Do not modify the ADCC_EVCTL[nn] and ADCC_EVT[nn] registers for enabled events when the frame (for which the events were enabled) is in progress. For other events which are not enabled in any of the currently progressing frames, the ADCC_EVCTL[nn] and ADCC_EVT[nn] registers may be modified. Enable the ADCC_EVTEN register only after the modifications are done.
- The configuration for the ADCC_EVTEN register for a particular frame should be stable when the trigger arrives. Further changes in the ADCC_EVTEN register are considered only for the subsequent frame. The same holds for the ADCC_BPTR0/ADCC_BPTR1 registers.
- Do not disable the ADC controller while a frame is in progress. Disable the ADCC only after the Frame interrupt/Last Frame Interrupt for the ADCC_NUMFRAM0/ADCC_NUMFRAM1 registers is non zero. This prevents unfinished transfers on an ADC interface and associated anomalies, if any. If stopped in the middle of a transaction on an ADC, the transaction may be dropped midway. Not adhering to this guideline may lead to wrong data on re-enabling the ADC controller.

- For debug purposes, the MMR status and counter values are retained when the ADCC is disabled.
- MMR registers such as counters and other status are cleared on a 0 to 1 transition on the enable bit. Error status is not cleared with this 0 to 1 transition. Clear errors using a W1C operation.
- All MMR register changes which are required between frames should be done when the ADCC_NUMFRAM0/ ADCC_NUMFRAM1 registers =0, after the frame interrupt is generated and before software clears the frame interrupt bit in AFC_AISTAT register.
- If the ADCC_NUMFRAM0/ADCC_NUMFRAM1 registers = non-zero, all MMR register changes must be done after the LFINT (last frame interrupt) is generated (after N + 1 frames) and before software clears the interrupt. The next trigger is accepted only after clearing the LFINTx bit.
- Disable the ADCC_CTL register before reconfiguring it. The only write allowed into an enabled ADCC_CTL register is to disable it. Enable the ADCC only after ensuring that the ADCC_T0STAT.DPND/ ADCC_T1STAT.DPND bits are zero.
- For ADC1 and ADC2, no two consecutive samples can be taken from the same channel MUX (A / B / C). Use Event programming to resolve this restriction. This implies that if event times of EVT(n) and EVT(n – 1) are such that: EVT(n – 1)≤ EVT(n) ≤ EVT(n – 1) + t_{CSP} × N

  then EVT(n) and EVT(n – 1) should not have the same channel mux selected (t_{CSP} is the chip select period in terms of ADC clock periods and 1:N is the clock division ratio).

6x Sampling

For 6x event control and timing, the following bits must also be configured in the Event Control register.

- Enable the ADCC_EVCTL[nn].SIMSAMP bit
- Enable all MUXA, MUXB, MUXC in the ADCC_EVCTL[nn].CTLWD bit field.
- Bit [1] in the ADCC_EVCTL[nn].CTLWD bit field must be set to 1, so that the shared chip select is used between ADCs. This is required for more precise synchronization.

Event and Sample Timing Relationships

There are several According to the relative timings of successive events, various scenarios can be seen in the ADCC operation. Only non-simultaneous sampling scenarios are considered in this section. Various cases and their handling in the ADCC's protocol is as follows:

NOTE: t_{CSP} is the chip select period and 1:N is the clock division ratio between SYSCLK and ADCC_ACLK.

NOTE: The examples use SYSCLK which is assigned to ADCC1 however they are applicable for SCLK0/ADCC0 as well.

Case1: Events are Spaced Apart in Time

Let EVT(n – 1) and EVT(n) be the two events shown in the figure. To have no pipelining among the two events, the guideline to be followed is: EVT(n – 1) + 3 × t_{CSP} × N < EVT(n)

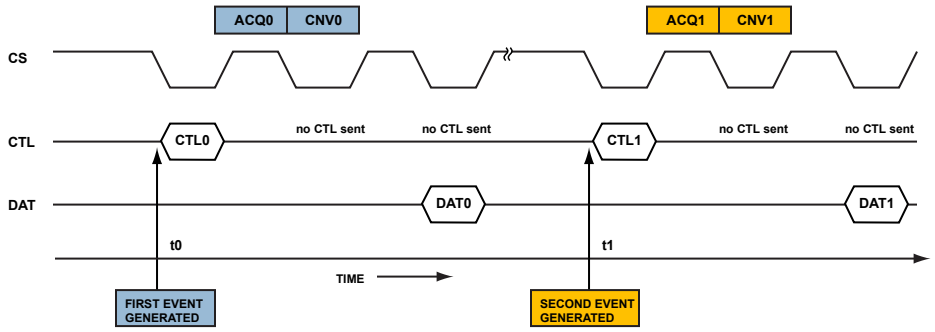


Figure 30-19: Handling of Spaced Events

Case 2: Second Event Timed at the Slot for the Chip Select After the Data Chip Select of the Previous Event.

In this case, the control transmission of EVT(n) happens only after the data reception of EVT(n – 1). However, the timing of the control chip select is decided by the previous CS pulse, since the chip select pulse width needs to be at least t_{CSP} × N SYSCLK periods.

Let EVT(n – 1) and EVT(n) be the two events shown in the figure. For EVT(n) CTL to be transmitted with the fourth CS pulse, EVT(n – 1) + 2t_{CSP} × N < EVT(n) ≤ EVT(n – 1) + 3 × t_{CSP} × N

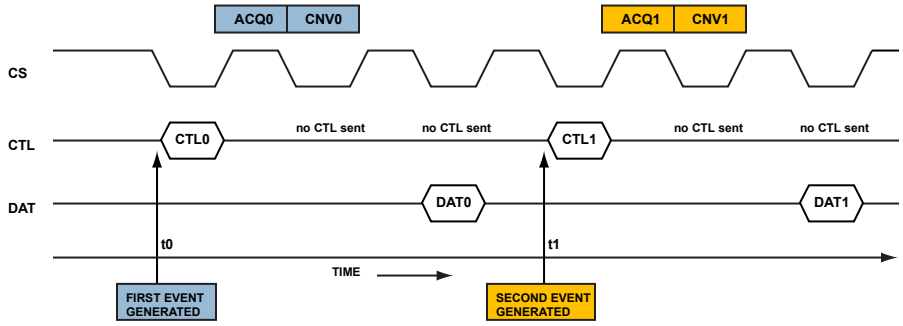


Figure 30-20: Pipelining with Fourth Chip Select

Case 3: The Second Event Pipelined with the Data Chip Select of the Previous Event

Let EVT(n – 1) and EVT(n) be the two events shown in the figure. To pipeline EVT(n) with the data chip select of EVT(n – 1), the guideline to be followed is EVT + t_{CSP} × N < EVT(n) ≤ EVT(n – 1) + 2 × t_{CSP} × N

NOTE: This is the maximum throughput condition for ADCC1 (ADC1 an ADC2).

NOTE: In this case of maximum throughput, if ADC1 or ADC2 is used for sampling, then back to back sampling on the same MUX is allowed only if no CTL word is sent for the same MUX, while its conversion is ongoing.

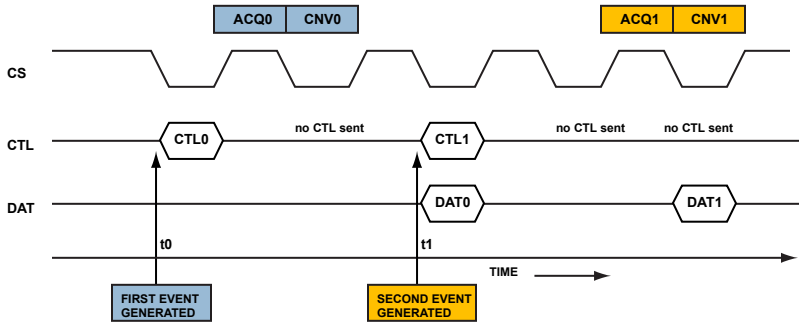


Figure 30-21: Handling Overlapping Events

Case 4 : Subsequent (n+1th) Events Pipelined with the Conversion Chip Select for the Previous(nth) Event

Let EVT(n – 1) and EVT(n) be two consecutive events shown in the figure. To pipeline EVT(n) with the CNV chip select of EVT(n – 1), the guideline to be followed is EVT(n – 1) < EVT(n) ≤ EVT(n – 1) + t_{CSP} × N

NOTE: This approach is only allowed for ADCC0

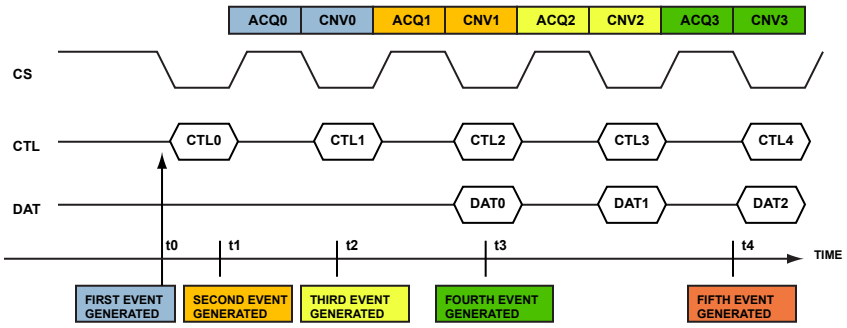


Figure 30-22: Tight Pipelining

Delay from Event firing to control transmission

The delay after the event match cycle to control transmission is captured in the `ADCC_EVSTAT[nn].DLYCNT` bit field. If the guidelines in the sections above are followed, this is {1 SYSCLK period + 2 ADC clock periods}.

Delay from Control Transmission to Sampling

The delay from control transmission to event sampling is:

Max {Conversion_time,control_transmission time} + Acquisition_time

For operating at the maximum possibe ADC throughput, the control transmission needs to be less than the conversion time. Thus, delay from control transmission to sampling is:

Conversion_time + Acquisition_time. This is equivalent to the period of Chip Select.

Programming Model

The discussion in the following sections provides the detailed programming model of the ADCC.

AFE Register Access Programming Model

There are three registers in the AFE domain that needs to be accessed via ADCC, but they are not visible in the ADCC Memory Map. These registers (detailed in AFE_Regs.h) are listed below and are read or written via ADCC0/ADC0

- USERCONFIGREG0
- STATUSREG0
- STATUSREG1

These registers are needed for multiple situations which include:

- AFE Initialization of its internal hardware configuration and other registers.
- Enabling diagnostic mode.
- Enabling clock fault check on FOCP.
- Verifying which Comparator has triggered FOCP interrupt.
- Selecting the DAC for programming.

NOTE: Only AFE initialization requires the ADC clock to be 1 MHz. These registers can be accessed in any frequency within the maximum range specified in the datasheet.

Control Word for Register Access

Accessing AFE registers are done via ADCC0 and are issued with a special control word explained below. This is an 8-bit field and is written as LSB-aligned in the 16-bit ADCC_EVCTL[nn].CTLWD field. See references in ADCC Initialization, FOCP and DACC chapters on how the control words are used.

Bit Position	Field	Entries	Program as
[7]	RESERVED		Program as 1
[6]	WNR (Select between Write and Read)		= 1 for Write = 0 for Read
[5]	RESERVED		Program as 0
[4-1]	REGADDRESS (Register Address)		Register Address
[0]	RESERVED		Program as 0

NOTE: Accessing the AFE registers implies using the ADC0. User software must ensure that ADC0 is used when it is completely idle and no event is active at that time. The second option is to turn off ADC0 by writing to the control register before accessing these registers and re-enabling ADC0 for actual ADC sampling. This ensures that ADC0 is used only to access the registers during that period.

AFE Registers (AFE_Regs.h)

The AFE Registers are used to enable and verify the status of various specific settings in the Analog Front End. This includes the ADCC, FOCP, and DACC modules.

NOTE: These registers are reset after a power supply recycle, but not after a hard reset assertion of the processor.

```
/
*=============================================================================
=================
   USER CONFIG and STATUS Registers
=============================================================================
===============*/
#define REG_AFE0_USERCFG    0x00001000   /*  AFE0 AFE User Configuration Register
*/
#define REG_AFE0_STATUSREG0 0x00001001   /*  AFE0 AFE Status Register 0 */
#define REG_AFE0_STATUSREG1 0x00001002   /*  AFE0 AFE Status Register 1 */
```

Table 30-16: USER CONFIG and STATUS Registers

Register_Bit	Bit	Description	Value
#define BITP_AFE_USERCFG_HWCFG_LOAD	22	Enables the Hardware Configuration Load	0x00400000
#define BITP_AFE_USERCFG_HWCFG_LD_BLK_SEL	18	Selects Blocks 0-9 for Loading the Hardware Configuration during Initialization	0x003C0000
#define BITP_AFE_USERCFG_DAC_EN_ILIM	16	Current Limit Enable for DAC Output Buffer	0x00010000
#define BITP_AFE_USERCFG_DAC_SEL	15-14	Enables Selection of DAC. 00 = 12 Bit DAC 01 = DAC_A 10 = DAC_B 11 = Not Used	0x0000C000
#define BITP_AFE_USERCFG_DIAG_FOCP_EN	13	Enables Diagnostic Mode for FOCP, a Use Config Mode Bit	0x00002000
#define BITP_AFE_USERCFG_DIAG_ADC_CHN_EN	12	Enable Diagnostic Mode for ADC, a User Config Mode Bit	0x00001000

Table 30-16: USER CONFIG and STATUS Registers (Continued)

Register_Bit	Bit	Description	Value
#define BITP_AFE_USERCFG_FOCP_CLK_CHECK_EN	11	Enable the FOCP Fault Check on FOCP_CLK	0x00000800
#define BITP_AFE_USERCFG_MON_AUX-BUFF_BYPASS	10	Bypass AUx Buff for Monitor ADC, a User Config Bit	0x00000400
#define BITP_AFE_USERCFG_PRI1_AUX-BUFF_BYPASS	9	Bypass Auxbuff for Primary ADC, a User Config Mode	0x00000200
#define BITP_AFE_USERCFG_PRI2_AUX-BUFF_BYPASS	8	Enable Auxbuff Bypass for Pri2 ADC, a User Config Bit	0x00000100
#define ENUM_AFE_USERCFG_EN000_16		DAC_SEL: selects 12 bit DAC	0x00000000
#define ENUM_AFE_USERCFG_EN001_17		DAC_SEL: selects DAC A (over voltage DAC)	0x00004000
#define ENUM_AFE_USERCFG_EN002_18		DAC_SEL: selects DAC B (under voltage DAC)	0x00008000

Programming Sequence

The typical programming sequence consists of several steps which are accomplished in the following order.

1. AFE Initialization
2. Verify AFE_OK signal is asserted.
3. Timing and Configuration initialization of ADCs
4. Configuration of Events for Frame capturing, Event Control (Frame Setup)
5. Initialize ADCC DMAs
6. Configure Interrupts
7. Enable ADCC

The final setup includes initializing the ADCC itself (`ADCC_CTL.EN`) and then waiting for interrupts for complete frame capture by DMA and process the buffer by the core.

AFE Initialization

Initialization code must be executed before the ADCs can be used. This initializes the Analog part of the entire AFE, including the ADCs, (ADC1, ADC2 and ADC0). A single interface (ADCC0) is used to initialize all ADCs. During initialization, the ADC clock (`ADCC_ACLK`) must be set to 1 MHz. The following code sequence is mandatory

and is typically performed in the ARM Cortex-M4 application code. Once the initialization is performed, a mandatory delay of 10 msec must be added before disabling the ADCC. (The ADCC must be disable at the end, in order to reset the whole state machine for further programming for sampling and data acquisition.)

```
void Init_AFE(void)
{

/*Initialize clocks including Core Clock and System clock*/
Initialize_CGU();

/*Timing Control A reg, ADC_CLK should be 1MHz. We assume 100 MHz System Clock.*/
*pREG_ADCC0_TCA0 =  (((99 << BITP_ADCC_TCA0_CKDIV ) & BITM_ADCC_TCA0_CKDIV ) |
                     ((8 << BITP_ADCC_TCA0_NCK ) & BITM_ADCC_TCA0_NCK ) );
__DSB();

/*Timing Control B reg */
*pREG_ADCC0_TCB0 = (((1 << BITP_ADCC_TCB0_TCSCK ) & BITM_ADCC_TCB0_TCSCK ) |
                   ((0 << BITP_ADCC_TCB0_TCKCS ) & BITM_ADCC_TCB0_TCKCS ) |
                    16 << BITP_ADCC_TCB0_TCSCS ) & BITM_ADCC_TCB0_TCSCS ) );
__DSB();

/*Enable ADCC*/
*pREG_ADCC0_CTL = (BITM_ADCC_CTL_EN |
                   BITM_ADCC_CTL_MODE|
                   BITM_ADCC_CTL_DSIZE );
__DSB();

/* Following initializes 10 internal registers in the AFE. */
#define BITM_AFE_USERCFG_HWCFG_LOAD 0x00400000
#define BITP_AFE_USERCFG_HWCFG_LD_BLK_SEL     18
#define BITM_AFE_USERCFG_HWCFG_LD_BLK_SEL     0x003C0000

#define CTLWD_AFEINIT 0xC0

/*Bit[7] - RESERVED = 1 */
//Bit[6] - WNR = 1
//Bit[5] - RESERVED = 0
//Bit[4-1] - REGADDRESS = 0000
//Bit[0] - RESERVED = 0

for (int AFEregno=0; AFEregno <10; AFEregno ++)
{
   unsigned int USER_CFG0 = ((BITM_AFE_USERCFG_HWCFG_LOAD)|
                             ((AFEregno << BITP_AFE_USERCFG_HWCFG_LD_BLK_SEL) &
                               BITM_AFE_USERCFG_HWCFG_LD_BLK_SEL));

   /*AFE Register access Control Word */
    *pREG_ADCC0_ADCRW0 =
          (((CTLWD_AFEINIT << BITP_ADCC_ADCRW0_CTLW) & BITM_ADCC_ADCRW0_CTLW ) |
```

```
            ((USER_CFG0 << BITP_ADCC_ADCRW0_DAT) & BITM_ADCC_ADCRW0_DAT ));

   /*wait until ADCC operation completed*/
     while((*pREG_ADCC0_ADCRW0 & BITM_ADCC_ADCRW0_PND))
     __NOP();
}

//wait for exactly 10 msec. Below is a dummy function.
delay_ms(10);

//disable ADCC
*pREG_ADCC0_CTL = 0x0000;
//disable ADCC
__DSB();

}
```

Verifying AFE_OK Signal

The SYS_AFE_NOTOK signal is asserted when the part comes out of reset and the program needs to perform the AFE Initialization before turning on the AFE_NOTOK interrupts.

```
if(!(*pREG_SYSBLK1_SYSSTAT  & BITM_SYSBLK_SYSSTAT_AFE_OK))
printf(”AFE is not initialized properly”);
```

Initializing ADC Timing and Configuration

There are general programming concepts for using the ADCC when it is present on any processor. For control, the ADCC provides two ADC interfaces (in this example, ADC1 and ADC2), which can be independently configured for the ADC sampling sequence timing. The ADC expects an 8-bit control word on a single control line (for example, `ADCC_ACTL0`), while the controller drives the converted 16-bit data on two data lines (for example, `ADCC_AD0` and `ADCC_AD1`).

Timing Initialization

For each interface (ADC1 and ADC2) set up the following.

1. Clock Divisor (`ADCC_TCA0.CKDIV`/`ADCC_TCA1.CKDIV`)
2. Program all timing configuration. The programming must meet the minimum conditions explained earlier. A snippet is given below for ADC1 and ADC2.
   - Program number of clocks for Conversion Pulse (`ADCC_TCA0.NCK`/`ADCC_TCA1.NCK`).
   - Timing CS to CS Delay (`ADCC_TCB0.TCSCS`/`ADCC_TCB1.TCSCS`)
   - Time CK to CS Hold (`ADCC_TCB0.TCKCS`/`ADCC_TCB1.TCKCS`)
   - Time CS to CK Setup (`ADCC_TCB0.TCSCK`/`ADCC_TCB1.TCSCK`)

```
*pREG_ADCC1_TCA0 = (((1 << BITP_ADCC_TCA0_CKDIV ) & BITM_ADCC_TCA0_CKDIV ) |
                   ((8 << BITP_ADCC_TCA0_NCK ) & BITM_ADCC_TCA0_NCK ) );

*pREG_ADCC1_TCB0 = (((1 << BITP_ADCC_TCB0_TCSCK ) & BITM_ADCC_TCB0_TCSCK ) |
                   ((0 << BITP_ADCC_TCB0_TCKCS ) & BITM_ADCC_TCB0_TCKCS ) |
                   ((8 << BITP_ADCC_TCB0_TCSCS ) & BITM_ADCC_TCB0_TCSCS ) );

*pREG_ADCC1_TCA1 = (((1 << BITP_ADCC_TCA1_CKDIV ) & BITM_ADCC_TCA1_CKDIV ) |
                   ((8 << BITP_ADCC_TCA1_NCK ) & BITM_ADCC_TCA1_NCK ) );

*pREG_ADCC1_TCB1 = (((1 << BITP_ADCC_TCB0_TCSCK ) & BITM_ADCC_TCB0_TCSCK ) |
                   ((0 << BITP_ADCC_TCB0_TCKCS ) & BITM_ADCC_TCB0_TCKCS ) |
                   ((8 << BITP_ADCC_TCB0_TCSCS ) & BITM_ADCC_TCB0_TCSCS ) );
```

Control Register Initialization

The following is an example for setting up the ADC Control Register.

For each interface (ADC1 and ADC2 programmer to select the configuration)

1. Program the ADCC_CTL register. Several parameters are fixed, see “ADCC Control & Timing Registers” in Programming Concepts.
2. Trigger Slave: There are up 6 timer slave triggers to select two active connections (`ADCC_CTL.TRGSEL0`/ `ADCC_CTL.TRGSEL1`).
3. Enable Timer 1 if needed (`ADCC_CTL.TMR1EN`).

```
#define  ADCC1_CTL      (((1 << BITP_ADCC_CTL_TRGSEL0) &
BITM_ADCC_CTL_TRGSEL0)     |\
                        ((2 << BITP_ADCC_CTL_TRGSEL1) &
BITM_ADCC_CTL_TRGSEL1)      |\
                        ((0 << BITP_ADCC_CTL_DSWP0 ) &
BITM_ADCC_CTL_DSWP0 )        |\
                        ((0 << BITP_ADCC_CTL_DSWP1 ) &
BITM_ADCC_CTL_DSWP1 )        |\
                        ((0 << BITP_ADCC_CTL_LSBF0 ) &
BITM_ADCC_CTL_LSBF0 )        |\
                        ((0 << BITP_ADCC_CTL_LSBF1 ) &
BITM_ADCC_CTL_LSBF1 )        |\
                        ((1 << BITP_ADCC_CTL_TRGPOL0  ) &
BITM_ADCC_CTL_TRGPOL0 )   |\
                        ((1 << BITP_ADCC_CTL_TRGPOL1  ) &
BITM_ADCC_CTL_TRGPOL1 )   |\
                ((1 << BITP_ADCC_CTL_TMR1EN ) & BITM_ADCC_CTL_TMR1EN )    |\
                        ((0 << BITP_ADCC_CTL_TRGOE0 ) &
BITM_ADCC_CTL_TRGOE0 )      |\
                        ((0 << BITP_ADCC_CTL_TRGOE1 ) &
BITM_ADCC_CTL_TRGOE1 )      |\
                ((1 << BITP_ADCC_CTL_DSIZE  ) & BITM_ADCC_CTL_DSIZE  )    |\
```

```
                          ((0 << BITP_ADCC_CTL_CSIZE  ) &
BITM_ADCC_CTL_CSIZE  )       |\
                          ((1 << BITP_ADCC_CTL_MODE  ) &
BITM_ADCC_CTL_MODE  )         |\
                          ((1 << BITP_ADCC_CTL_DMAEN  ) & BITM_ADCC_CTL_DMAEN  ))
```

NOTE: If an application requires changing the ADC voltage reference (from the default) then this change step must be performed prior to any other ADCC programming.

Configuring DMA

To use DMA with the ADCC, the following registers need to be configured.

```
/*Example Register setup for frame configuration*/

*pREG_ADCC1_BPTR0 = (unsigned int) &ADC_Data[0];
*pREG_ADCC1_FRINC0 = 32 *2;    //32 events enabled, each 16 bit
*pREG_ADCC1_CBSIZ0 = 2048;     //2048 is total number of frames
*pREG_ADCC1_BWMON0 = 0;        //turn off bandwidth monitor
*pREG_ADCC1_NUMFRAM0 = 2048;   //interrupt after 2048 frames
```

Configuring Interrupts

There are multiple interrupts for the ADCC. A typical setup includes setting the Last Frame Interrupt and the Error Interrupt as follows.

```
*pREG_ADCC1_FIMSK  &= ~(BITM_ADCC_FIMSK_SET_LFINT0 ) ;
*pREG_ADCC1_FIMSK  |= (BITM_ADCC_FIMSK_SET_FINT0 ) ;
*pREG_ADCC1_ERRMSK  = ~(BITM_ADCC_ERRMSK_EMIS | BITM_ADCC_ERRMSK_TRGOV1|
BITM_ADCC_ERRMSK_TRGOV0);
```

Frame Setup 1x Sampling

The following steps are used to program 1x sampling.

1. To program Events, use the ADCC_EVT[nn] and ADCC_EVCTL[nn] registers.
2. Enable the events in the ADCC_EVTEN_SET register.
3. To program the event timer register, be aware of the timing requirements. Consult the descriptions on various sampling modes to understand how Chip Select timing must be programmed. The ADSP-CM41x data sheet contains the minimum requirements for example:

   *pADCC_REG_Ptr = (EVT_ID × tCSp × (2 × (t_{CSCS} + 8+ 1)));

   - EVT_ID specifies the EVT Number.
   - t_{CSp} is the Chip Select to Chip Select Gap, if it is a case of 1x at maximum throughput, it must be 2.

- t_{CSCS} can be programmable
- 8 implies the 8 clocks for NCK, which is for 16 bits, 2 data lines.
- 1 implies the 1 clock for t_{CSCK}.

4. To program the event control register, select:
   - `ADCC_EVCTL[nn].TMRSEL` is the Timer 0/1 selection
   - `ADCC_EVCTL[nn].EVTOFS` is the Event Offset in memory placement, usually 2 bytes
   - `ADCC_EVCTL[nn].SIMSAMP` is the If simultaneous sampling enabled
   - `ADCC_EVCTL[nn].ADCSEL` is the ADC1/ADC2
   - `ADCC_EVCTL[nn].CTLWD` is the Control Word

```
/*
-    Timer0 only
-    1x sampling
-    2 bytes of data
-    MUXB enabled
*/

*pREG_ADCC1_EVCTLx =
(((0    << BITP_ADCC_EVCTL_TMRSEL) & BITM_ADCC_EVCTL_TMRSEL ) |
((ADC_SEL<< BITP_ADCC_EVCTL_ADCSEL) & BITM_ADCC_EVCTL_ADCSEL ) |
((0     << BITP_ADCC_EVCTL_SIMSAMP) & BITM_ADCC_EVCTL_SIMSAMP ) |
(((2*EVTID)<< BITP_ADCC_EVCTL_EVTOFS) & BITM_ADCC_EVCTL_EVTOFS ) |
((ADC_CHNL << 5) & 0xE0 ) |MUXB );
```

Frame Capture

The following steps provide an overview of the program flow for using the ADCC for setting up frames and receiving ADC data.

1. Write configuration data (including event control data) to the ADCC memory mapped registers.
2. Configure the ADCC_CTL register.
3. Enable the ADCC (`ADCC_CTL.EN` =1). The ADCC waits for the trigger input.
4. A trigger input starts the ADCC timer at the time of an enabled event.

   ADDITIONAL INFORMATION: When the data is ready, DMA transfers and the core can read data in parallel from the ADCC.

   The ADCC transfers control words the ADC and receives data from the ADC.
5. The ADCC generates a last frame complete interrupt request (`ADCC_FISTAT.LFINT0`) when the data corresponding to all events in all the frame are received.

ADDITIONAL INFORMATION: The ADCC can be re-configuring; if necessary, changing the base pointer (for example, ADCC_BPTR0).

6. Clear the frame complete interrupt (for example, ADCC_FISTAT.LFINT0 =0).

   If more frames are expected, the ADCC resumes waiting for the trigger input.

   If no more frames are expected, the ADCC can be disabled (ADCC_CTL.EN =0).

CM41X_M4 ADCC Register Descriptions

ADC Controller (ADCC) contains the following registers.

Table 30-17: CM41X_M4 ADCC Register List

Name	Description
ADCC_ADCRW0	ADC2 Interface RW Access Register
ADCC_ADCRW1	ADC2 Interface RW Access Register
ADCC_BPTR0	Base Pointer 0 Register
ADCC_BPTR1	DMA Base Pointer 1 Register
ADCC_BWMON0	Bandwidth Monitor 0 Register
ADCC_BWMON1	Bandwidth Monitor 1 Register
ADCC_CBNUM0	Timer0 Circular Buffer DMA Wrap Number Register
ADCC_CBNUM1	Timer1 Circular Buffer DMA Wrap Number Register
ADCC_CBSIZ0	Circular Buffer Size 0 Register
ADCC_CBSIZ1	Circular Buffer Size 1 Register
ADCC_CTL	Control Register
ADCC_DATOVF	Data Overflow Indication Register
ADCC_ECOL	Event Collision Status Register
ADCC_EIMSK	Event Interrupt Mask Register
ADCC_EIMSK_CLR	Event Interrupt Mask Clear Register
ADCC_EIMSK_SET	Event Interrupt Mask Set Register
ADCC_EISTAT	Event Interrupt Status Register
ADCC_EMISS	Event Miss Status Register
ADCC_EPND	Pending Events Status Register
ADCC_ERRMSK	Error Mask Register
ADCC_ERRMSK_CLR	Error Mask Clear Register
ADCC_ERRMSK_SET	Error Mask Set Register

Table 30-17: CM41X_M4 ADCC Register List (Continued)

Name	Description
ADCC_ERRSTAT	Error Status Register
ADCC_EVCTL[nn]	Event n Control Register
ADCC_EVDAT[nn]	Event n Data Register
ADCC_EVSTAT[nn]	Event n Status Register
ADCC_EVTEN	Event Enable Register
ADCC_EVTEN_CLR	Event Enable Clear Register
ADCC_EVTEN_SET	Event Enable Set Register
ADCC_EVT[nn]	Event n Time Register
ADCC_FIMSK	Frame Interrupt Mask Register
ADCC_FIMSK_CLR	Frame Interrupt Mask Clear Register
ADCC_FIMSK_SET	Frame Interrupt Mask Set Register
ADCC_FISTAT	Frame Interrupt Status Register
ADCC_FRINC0	Frame Increment 0 Register
ADCC_FRINC1	Frame Increment 1 Register
ADCC_NUMFRAM0	Timer0 Frame Limit Count Register
ADCC_NUMFRAM1	Timer1 Frame Limit Count Register
ADCC_T0STAT	Timer 0 Status Register
ADCC_T1STAT	Timer 1 Status Register
ADCC_TCA0	Timing Control A (ADC0) Register
ADCC_TCA1	Timing Control A (ADC1) Register
ADCC_TCB0	Timing Control B (ADC0) Register
ADCC_TCB1	Timing Control B (ADC1) Register
ADCC_TMR0	Timer 0 Current Count Register
ADCC_TMR1	Timer 1 Current Count Register
ADCC_TRGCNT0	Trigger Count TIMER0 Register
ADCC_TRGCNT1	Trigger Count TIMER1 Register

ADC2 Interface RW Access Register

Read/Write access register for ADC1/ADC0 interface

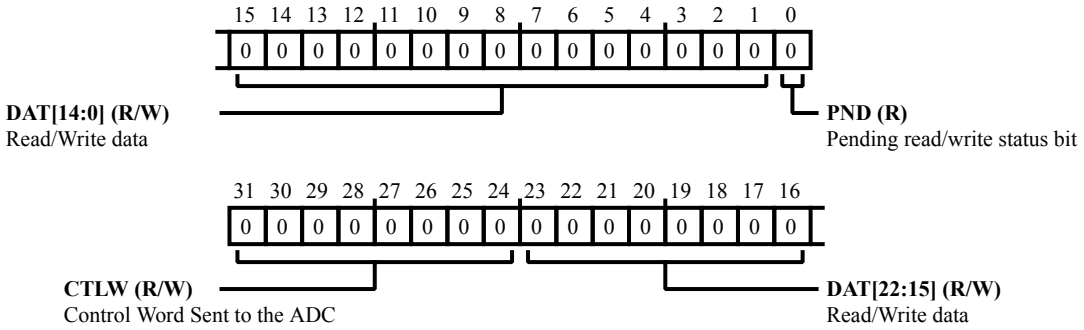


Figure 30-23: ADCC_ADCRW0 Register Diagram

Table 30-18: ADCC_ADCRW0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	CTLW	Control Word Sent to the ADC.
23:1 (R/W)	DAT	Read/Write data.
0 (R/NW)	PND	Pending read/write status bit.

ADC2 Interface RW Access Register

Read/Write access register for ADC2 interface.

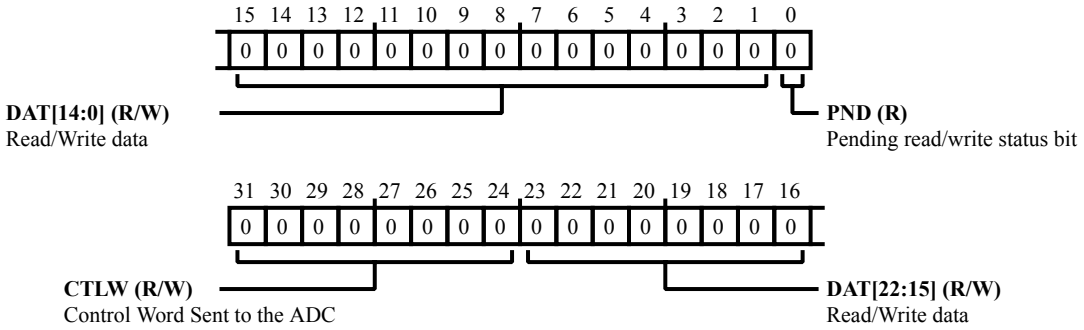


Figure 30-24: ADCC_ADCRW1 Register Diagram

Table 30-19: ADCC_ADCRW1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	CTLW	Control Word Sent to the ADC.
23:1 (R/W)	DAT	Read/Write data.
0 (R/NW)	PND	Pending read/write status bit.

Base Pointer 0 Register

The ADCC_BPTR0 register provides the base pointer (address) used for placing the first Timer 0 frame's event-data into memory through DMA. The value of base address (pointer) in the ADCC_BPTR0 register at the time of the trigger pulse (start of frame) corresponds to one of the following Timer 0 related frames:

- The first frame after ADCC is enabled
- The first frame after the ADCC_BPTR0 register is written by core (in linear DMA mode)
- The first frame after a loop back occurs in circular buffering mode

The data from the first frame's events is placed in memory, starting at the address indicated with the value of the frame base address. Each of the event data are placed in a location (ADCC_BPTR0 + ADCC_EVCTL[nn].EVTOFS), where ADCC_EVCTL[nn].EVTOFS is a programmable offset for each event.

The second frame's base address (pointer) is calculated as (ADCC_BPTR0 + ADCC_FRINC0).

The third frame's base address (pointer) is calculated as (ADCC_BPTR0 + (2 x ADCC_FRINC0)).

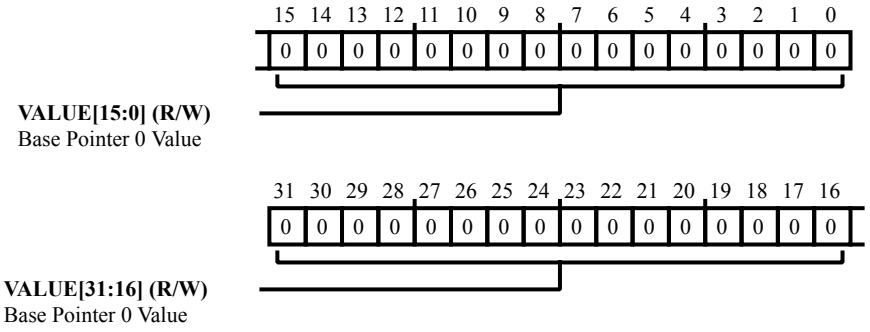


Figure 30-25: ADCC_BPTR0 Register Diagram

Table 30-20: ADCC_BPTR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Base Pointer 0 Value. The ADCC_BPTR0.VALUE bits hold the base pointer (address) for the first frame of the DMA. Note that bit 0 must be written as 0 to ensure correct address alignment.

DMA Base Pointer 1 Register

The ADCC_BPTR1 register provides the base pointer (address) used for placing the first Timer 1 frame's event-data into memory through DMA. The value of base address (pointer) in the ADCC_BPTR1 register at the time of the trigger pulse (start of frame) corresponds to one of the following Timer 1 related frames:

- The first frame after ADCC is enabled
- The first frame after the ADCC_BPTR1 register is written by core (in linear DMA mode)
- The first frame after a loop back occurs in circular buffering mode

The data from the first frame's events is placed in memory, starting at the address indicated with the value of the frame base address. Each of the event data are placed in a location (ADCC_BPTR1 + ADCC_EVCTL[nn].EVTOFS), where ADCC_EVCTL[nn].EVTOFS is a programmable offset for each event.

The second frame's base address (pointer) is calculated as (ADCC_BPTR1 + ADCC_FRINC0).

The third frame's base address (pointer) is calculated as (ADCC_BPTR1 + (2 x ADCC_FRINC0)).

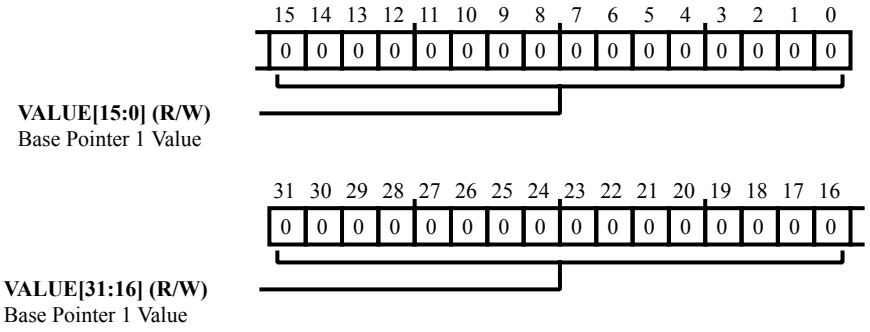


Figure 30-26: ADCC_BPTR1 Register Diagram

Table 30-21: ADCC_BPTR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Base Pointer 1 Value. The ADCC_BPTR1.VALUE bits hold the base pointer (address) for the first frame of the DMA. Note that bit 0 must be written as 0 to ensure correct address align-ment.The ADCC_BPTR1.VALUE bits hold the base pointer (address) for the first frame of the DMA. Note that bit 0 must be written as 0 to ensure correct address alignment.

Bandwidth Monitor 0 Register

The ADCC_BWMON0 register monitors the Timer 0 count and can be used to signal an error if the count exceeds the ADCC_BWMON0.CNT value in DMA mode. This checks whether the DMA transfer of all events in the frame are completed before a certain count of the timer. To monitor for completion of all events and their DMA before n SCLK cycles after the trigger, program the value n in the ADCC_BWMON0.CNT field. When the timer count exceeds n, the ADCC issues an error interrupt and indicates the status with the ADCC_ERRSTAT.BWERR0 bit. The Timer stops only when all DMA transfers are complete and responses received, in DMA mode.

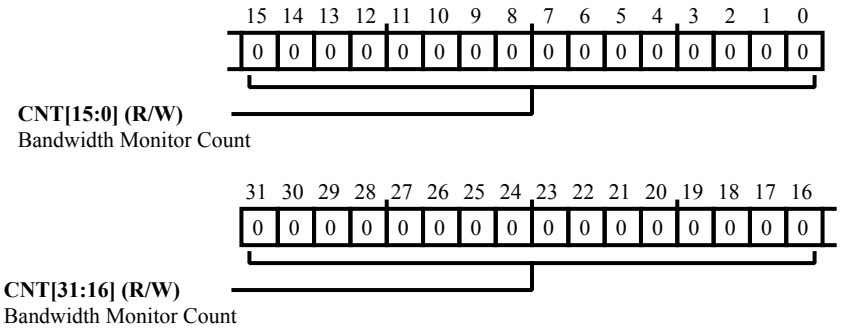


Figure 30-27: ADCC_BWMON0 Register Diagram

Table 30-22: ADCC_BWMON0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CNT	Bandwidth Monitor Count. The ADCC_BWMON0.CNT bits hold the maximum expected Timer 0 count for a frame. If programmed as 0, bandwidth monitoring capability is disabled.

Bandwidth Monitor 1 Register

The ADCC_BWMON1 register monitors the Timer 1 count and can be used to signal an error if the count exceeds the ADCC_BWMON1.CNT value in DMA mode. This checks whether the DMA transfer of all events in the frame are completed before a certain count of the timer. To monitor for completion of all events and their DMA before n SCLK cycles after the trigger, program the value n in the ADCC_BWMON1.CNT field. When the timer count exceeds n, the ADCC issues an error interrupt request and indicates the status with the ADCC_ERRSTAT.BWERR1 bit. The Timer stops only when all DMA transfers are complete and responses received, in DMA mode.

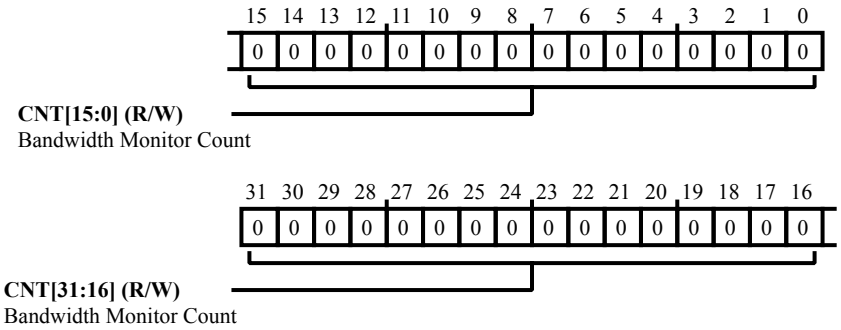


Figure 30-28: ADCC_BWMON1 Register Diagram

Table 30-23: ADCC_BWMON1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	CNT	Bandwidth Monitor Count. The ADCC_BWMON1.CNT bits hold the maximum expected Timer 1 count for a frame. If programmed as 0, bandwidth monitoring capability is disabled.

Timer0 Circular Buffer DMA Wrap Number Register

The ADCC_CBNUM0 register holds the number of wraps completed by TIMER0 in circular buffer DMA.

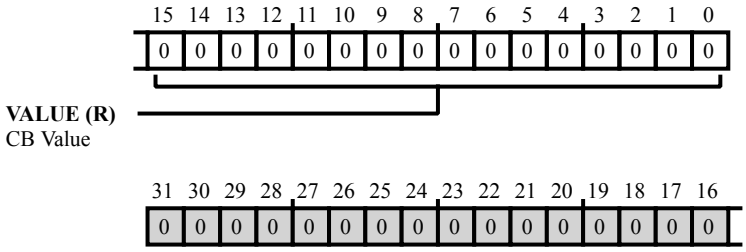


Figure 30-29: ADCC_CBNUM0 Register Diagram

Table 30-24: ADCC_CBNUM0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	CB Value. The ADCC_CBNUM0.VALUE bit field indicates how many wraparounds have been completed in circular buffer DMA for the TIMER, after the ADCC was enabled.

Timer1 Circular Buffer DMA Wrap Number Register

The ADCC_CBNUM1 register holds the number of wraps completed by TIMER1 in circular buffer DMA.

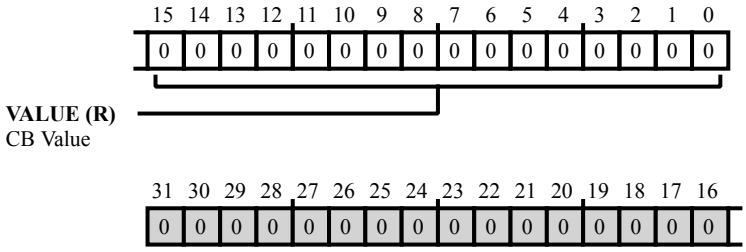


Figure 30-30: ADCC_CBNUM1 Register Diagram

Table 30-25: ADCC_CBNUM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	CB Value. The ADCC_CBNUM1.VALUE bit field indicates how many wraparounds have been completed in circular buffer DMA for the TIMER, after the ADCC was enabled.

Circular Buffer Size 0 Register

The ADCC_CBSIZ0 register holds the circular buffer size to be used for the Timer 0 frames of DMA. If the ADCC_CBSIZ0.VALUE bit field is programmed as 0, the ADCC performs only linear buffering is for the Timer 0 frames. For more information about ADCC linear and circular buffering, see the ADCC functional description.

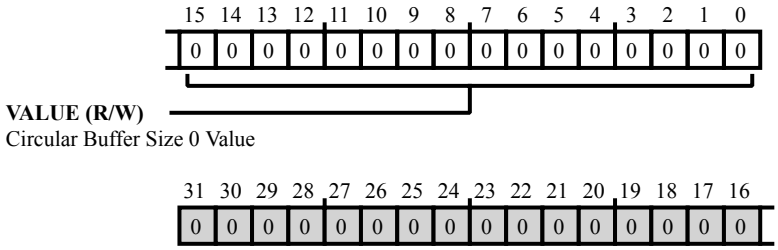


Figure 30-31: ADCC_CBSIZ0 Register Diagram

Table 30-26: ADCC_CBSIZ0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Circular Buffer Size 0 Value. The ADCC_CBSIZ0.VALUE bits select the number of Timer 0 frames in a DMA circular buffer (number of frames after which to wrap back); if =0, linear mode is used instead of circular DMA mode.

Circular Buffer Size 1 Register

The ADCC_CBSIZ1 register holds the circular buffer size to be used for the Timer 1 frames of DMA. If the ADCC_CBSIZ1.VALUE bit field is programmed as 0, the ADCC performs only linear buffering is for the Timer 1 frames. For more information about ADCC linear and circular buffering, see the ADCC functional description.

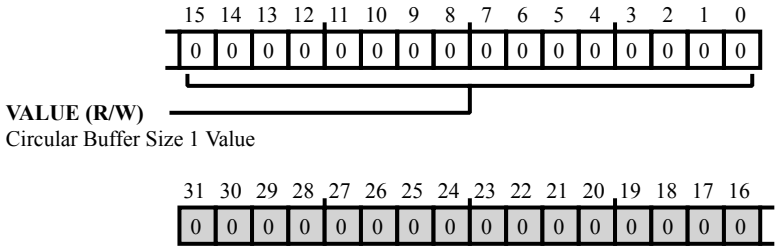


Figure 30-32: ADCC_CBSIZ1 Register Diagram

Table 30-27: ADCC_CBSIZ1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Circular Buffer Size 1 Value. The ADCC_CBSIZ1.VALUE bits select the number of Timer 1 frames in a DMA circular buffer (number of frames after which to wrap back); if =0, linear mode is used instead of circular DMA mode.

Control Register

The ADCC_CTL register enables ADCC operation and configures a number of ADC1/ADC2 and ADC0 interface features.

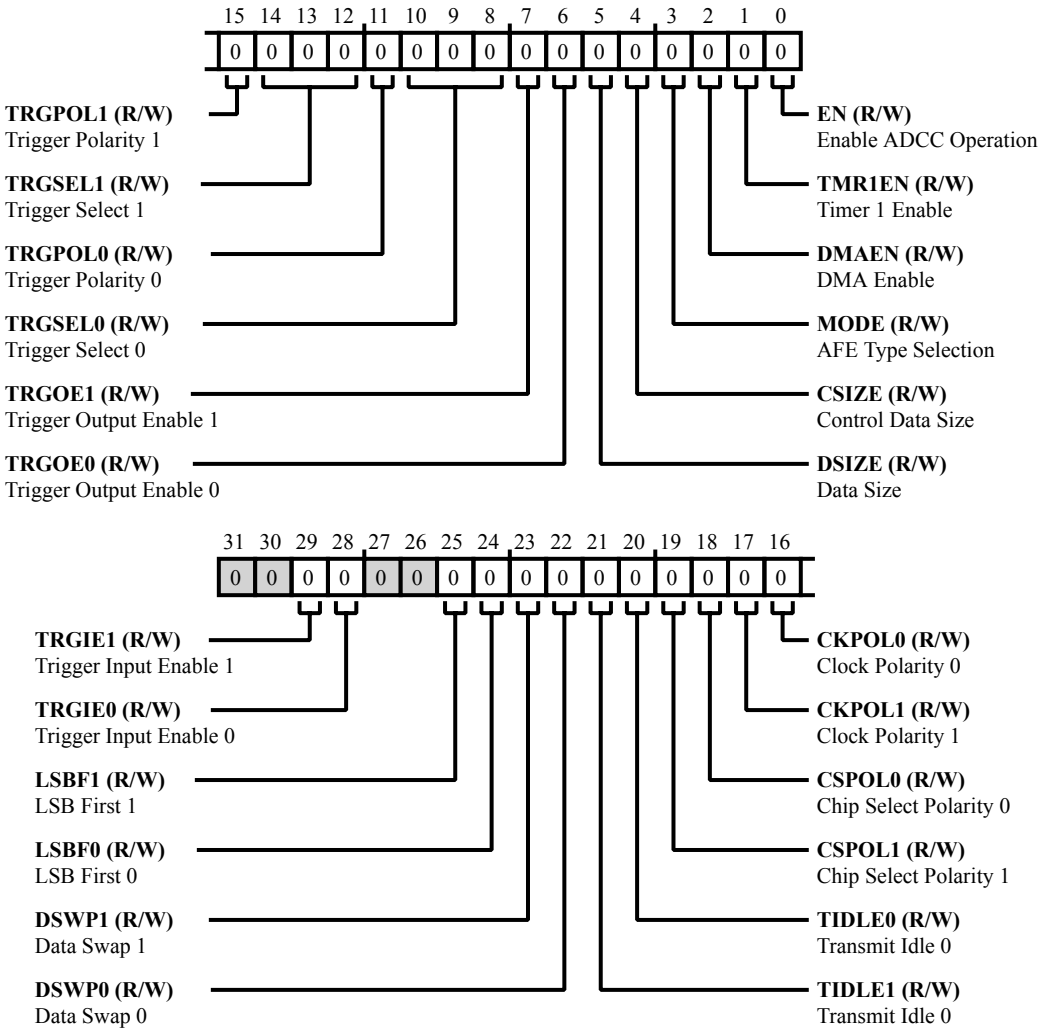


Figure 30-33: ADCC_CTL Register Diagram

Table 30-28: ADCC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	TRGIE1	Trigger Input Enable 1. The ADCC_CTL.TRGIE1 bit enables recognition of trigger input as valid to initiate Timer 1 frames.	
		0	Disable Recognition of Trigger Input
		1	Enable Recognition of Trigger Input

Table 30-28: ADCC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	TRGIE0	Trigger Input Enable 0. The `ADCC_CTL.TRGIE0` bit enables recognition of trigger input as valid to initiate Timer 0 frames.	
		0	Disable Recognition of Trigger Input
		1	Enable Recognition of Trigger Input
25 (R/W)	LSBF1	LSB First 1. The `ADCC_CTL.LSBF1` bit selects LSB or MSB first mode for ADC2 interface operations. For more information about LSB/MSB first mode operations, see the ADCC functional description.	
		0	MSB First Mode
		1	LSB First Mode
24 (R/W)	LSBF0	LSB First 0. The `ADCC_CTL.LSBF0` bit selects LSB or MSB first mode for ADC1 or ADC0 interface operations. For more information about LSB/MSB first mode operations, see the ADCC functional description.	
		0	MSB First Mode
		1	LSB First Mode
23 (R/W)	DSWP1	Data Swap 1. The `ADCC_CTL.DSWP1` bit enables data swap operations on ADC2 for dual-bit (2-bit wide) control and data. For more information about data swap operations, see the ADCC functional description.	
		0	Disable Data Swap
		1	Enable Data Swap
22 (R/W)	DSWP0	Data Swap 0. The `ADCC_CTL.DSWP0` bit enables data swap operations on ADC1 or ADC2 for dual-bit (2-bit wide) control and data. For more information about data swap operations, see the ADCC functional description.	
		0	Disable Data Swap
		1	Enable Data Swap

Table 30-28: ADCC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	TIDLE1	Transmit Idle 0. The ADCC_CTL.TIDLE1 bit selects the value to hold on the ADCC_BCTL0 pin for ADC1 or ADC0 when idle (not transmitting a valid control register). The first chip select after enabling the ADCC is a control chip select. The ADCC_CTL.TIDLE1 value is held on the control pins after this chip select, unless a valid control transmission takes place.	
		0	Hold Idle Control Pin Low
		1	Hold Idle Control Pin High
20 (R/W)	TIDLE0	Transmit Idle 0. The ADCC_CTL.TIDLE0 bit selects the value to hold on the ADCC_ACTL0 pin for ADC1 or ADC0 when idle (not transmitting a valid control register). The first chip select after enabling the ADCC is a control chip select. The ADCC_CTL.TIDLE0 value is held on the control pins after this chip select, unless a valid control transmission takes place.	
		0	Hold Idle Control Pin Low
		1	Hold Idle Control Pin High
19 (R/W)	CSPOL1	Chip Select Polarity 1. The ADCC_CTL.CSPOL1 bit selects the polarity (active high or low) for ADCC_BCS pin (ADC2 interface).	
		0	Active Low CS
		1	Active High CS
18 (R/W)	CSPOL0	Chip Select Polarity 0. The ADCC_CTL.CSPOL0 bit selects the polarity (active high or low) for ADCC_ACS pin (ADC1 or ADC0 interface).	
		0	Active Low CS
		1	Active High CS
17 (R/W)	CKPOL1	Clock Polarity 1. The ADCC_CTL.CKPOL1 bit selects the clock polarity for the ADC2 interface. This selection chooses the clock edge (rising or falling) driven after the ADCC_BCS pin is asserted for sampling the first bit of control and data.	
		0	Sample on Falling Edge
		1	Sample on Rising Edge

Table 30-28: ADCC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	CKPOL0	Clock Polarity 0. The ADCC_CTL.CKPOL0 bit selects the clock polarity for the ADC1 or ADC0 interface. This selection chooses the clock edge (rising or falling) driven after the ADCC_ACS pin is asserted for sampling the first bit of control and data.	
		0	Sample on Falling Edge
		1	Sample on Rising Edge
15 (R/W)	TRGPOL1	Trigger Polarity 1. The ADCC_CTL.TRGPOL1 bit select the polarity for trigger associated with Timer 1.	
		0	Trigger on Falling Edge
		1	Trigger on Rising Edge
14:12 (R/W)	TRGSEL1	Trigger Select 1. The ADCC_CTL.TRGSEL1 bits selects the Timer 1 initiate trigger from among the ADCC_TMR0_EVT and ADCC_TMR1_EVT triggers from =0 for trigger 0 to =n for trigger n.	
11 (R/W)	TRGPOL0	Trigger Polarity 0. The ADCC_CTL.TRGPOL0 bit select the polarity for trigger associated with Timer 0.	
		0	Trigger on Falling Edge
		1	Trigger on Rising Edge
10:8 (R/W)	TRGSEL0	Trigger Select 0. The ADCC_CTL.TRGSEL0 bits selects the Timer 0 initiate trigger from among the ADCC_TMR0_EVT and ADCC_TMR1_EVT triggers from =0 for trigger 0 to =n for trigger n.	
7 (R/W)	TRGOE1	Trigger Output Enable 1. The ADCC_CTL.TRGOE1 bit enables the trigger output generation for Timer 1 events.	
		0	Disable
		1	Enable
6 (R/W)	TRGOE0	Trigger Output Enable 0. The ADCC_CTL.TRGOE0 bit enables the trigger output generation for Timer 0 events.	
		0	Disable
		1	Enable

Table 30-28: ADCC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	DSIZE	Data Size. The ADCC_CTL.DSIZE bit selects the interface width (single- or dual-bit) for receiving sample data from the ADC.	
		0	1-Bit Wide Interface
		1	2-Bit Wide Interface
4 (R/W)	CSIZE	Control Data Size. The ADCC_CTL.CSIZE bit selects the interface width (single- or dual-bit) for transmitting control data to the ADC.	
		0	1-Bit Wide Interface
		1	2-Bit Wide Interface
3 (R/W)	MODE	AFE Type Selection. The ADCC_CTL.MODE bit selects the processor model that the ADCC is connected to.	
		0	None
		1	ADSP-CM41x
2 (R/W)	DMAEN	DMA Enable. The ADCC_CTL.DMAEN bit enables ADCC DMA operation, selecting the mode for sampled data transfer from the ADC to memory. If DMA is disabled (=0), the core should read data from the ADCC_EVDAT[nn] registers. If DMA is enabled, the ADCC transfers data using DMA, and th linear or circular buffer mode is chosen by the ADCC_CBSIZ0 or ADCC_CBSIZ1 register.	
		0	Disable
		1	Enable
1 (R/W)	TMR1EN	Timer 1 Enable. The ADCC_CTL.TMR1EN bit enables Timer 1. Note that Timer 0 is always enabled when the ADCC is enabled.	
		0	Disable
		1	Enable
0 (R/W)	EN	Enable ADCC Operation. The ADCC_CTL.EN bit enables ADCC operation (global enable for module).	
		0	Disable
		1	Enable

Data Overflow Indication Register

The ADCC_DATOVF register indicates the overflow status of various events

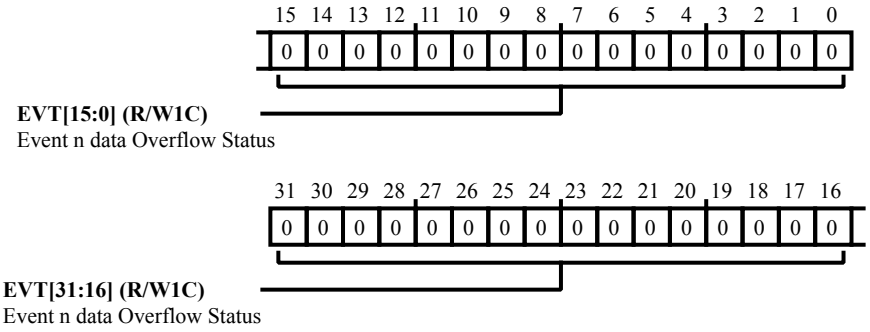


Figure 30-34: ADCC_DATOVF Register Diagram

Table 30-29: ADCC_DATOVF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W1C)	EVT	Event n data Overflow Status. The ADCC_DATOVF.EVT bits each correspond to an ADCC event that had data overflow. When any bit in this register is set, the ADCC_ERRSTAT.DATOVF bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin. The function of each bit is shown in the enumerations below.	
		0	No Status
		4294967295	Event Collision Occurred

Event Collision Status Register

The ADCC_ECOL register indicates the collision status for ADCC events. When any bit in this register is set, the ADCC_ERRSTAT.ECOL bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin.

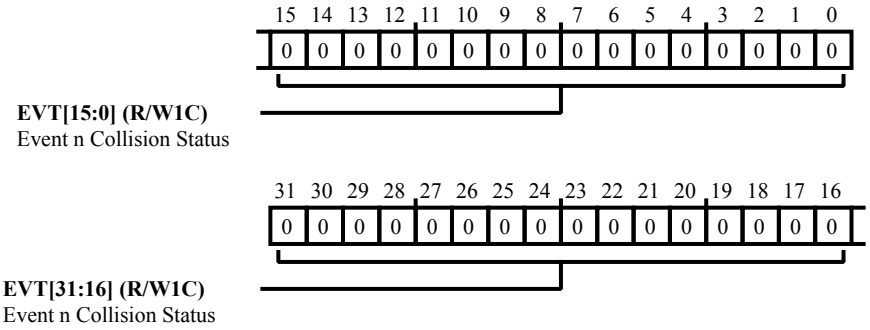


Figure 30-35: ADCC_ECOL Register Diagram

Table 30-30: ADCC_ECOL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W1C)	EVT	Event n Collision Status. The ADCC_ECOL.EVT bits each correspond to an ADCC event that underwent collision. When any bit in this register is set, the ADCC_ERRSTAT.ECOL bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin. The function of each bit is shown in the enumerations below.	
		0	No Status
		4294967295	Event Collision Occurred

Event Interrupt Mask Register

The ADCC_EIMSK register masks (disables) generation of ADCC_TMR0_EVT or ADCC_TMR1_EVT event interrupt requests based on status in the ADCC_EISTAT register.

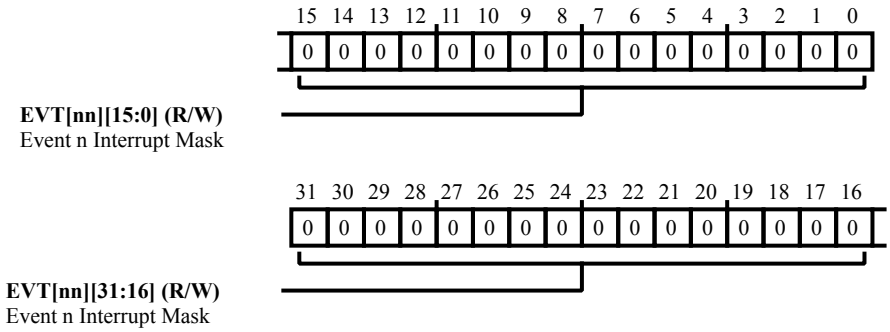


Figure 30-36: ADCC_EIMSK Register Diagram

Table 30-31: ADCC_EIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W)	EVT[nn]	Event n Interrupt Mask. The ADCC_EIMSK.EVT[nn] bits mask (disable) each correspond to data interrupt requests from ADCC events (n from 23 to 0). The function of each bit is shown in the enumerations below.	
		0	Unmask (Enable) Event Interrupt
		4294967295	Mask (Disable) Event Interrupt

Event Interrupt Mask Clear Register

The ADCC_EIMSK_CLR register can be used to selectively clear bits in the ADCC_EIMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_EIMSK_CLR clears the corresponding bit in ADCC_EIMSK. Reading the ADCC_EIMSK_CLR register returns the data present in the ADCC_EIMSK register.

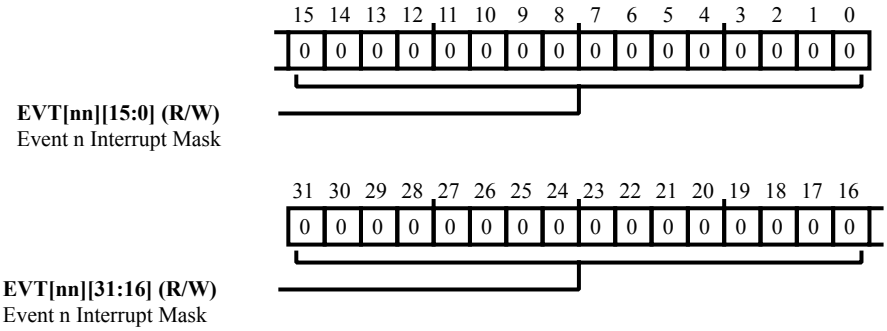


Figure 30-37: ADCC_EIMSK_CLR Register Diagram

Table 30-32: ADCC_EIMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1C)	EVT[nn]	Event n Interrupt Mask Clear. The ADCC_EIMSK_CLR.EVT[nn] bits permit clearing individual bits in the ADCC_EIMSK register without affecting other bits in the register. Write 1 to individual ADCC_EIMSK_CLR.EVT[nn] bits to clear the corresponding bit in ADCC_EIMSK.

Event Interrupt Mask Set Register

The ADCC_EIMSK_SET register can be used to selectively set bits in the ADCC_EIMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_EIMSK_SET sets the corresponding bit in ADCC_EIMSK. Reading the ADCC_EIMSK_SET register returns the data present in the ADCC_EIMSK register.

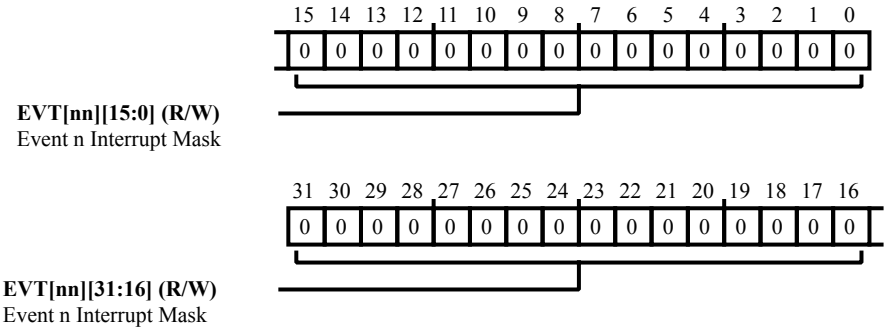


Figure 30-38: ADCC_EIMSK_SET Register Diagram

Table 30-33: ADCC_EIMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1S)	EVT[nn]	Event n Interrupt Mask Set. The ADCC_EIMSK_SET.EVT[nn] bits permit setting individual bits in the ADCC_EIMSK register without affecting other bits in the register. Write 1 to individual ADCC_EIMSK_SET.EVT[nn] bits to set the corresponding bit in ADCC_EIMSK.

Event Interrupt Status Register

The ADCC_EISTAT register indicates the interrupt status bits for ADCC events. Based on the ADCC_EVCTL[nn].TMRSEL bit selection, the ADCC generates the corresponding ADCC_TMR0_EVT or ADCC_TMR1_EVT event interrupt requests. Each of these interrupt requests can be optionally masked (disable interrupt output) using the bits in ADCC_EIMSK register.

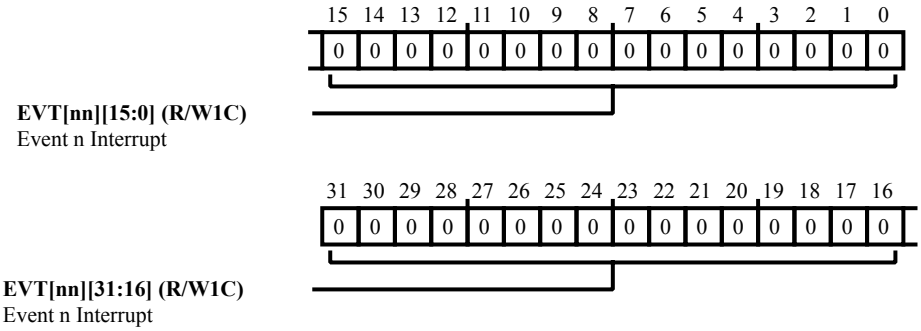


Figure 30-39: ADCC_EISTAT Register Diagram

Table 30-34: ADCC_EISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W1C)	EVT[nn]	Event n Interrupt. The ADCC_EISTAT.EVT[nn] bits each correspond to data interrupts from ADCC events (n from 23 to 0). The function of each bit is shown in the enumerations below. Note that data interrupt requests from events are applicable only in non-DMA mode.	
		0	No Data Pending for Core Read
		4294967295	Event Data Pending for Core Read

Event Miss Status Register

The ADCC_EMISS register indicates the miss status for ADCC events. When any bit in this register is set, the ADCC_ERRSTAT.EMIS bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin.

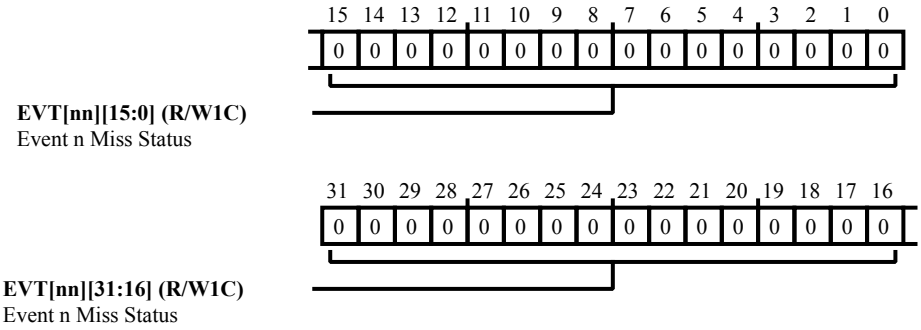


Figure 30-40: ADCC_EMISS Register Diagram

Table 30-35: ADCC_EMISS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W1C)	EVT[nn]	Event n Miss Status. The ADCC_EMISS.EVT[nn] bits each correspond to an ADCC event that was missed. When any bit in this register is set, the ADCC_ERRSTAT.EMIS bit is set, which may optionally cause the ADCC to generate output on the ADCC_ERR pin.The function of each bit is shown in the enumerations below.	
		0	No Status
		4294967295	Event Miss Occurred

Pending Events Status Register

The ADCC_EPND register indicates which events within the current frames are pending (waiting for data transfer). Each of the ADCC_EPND.EVT[nn] bits from 0 to 31 corresponds to an events from 0 to 31. When the trigger pulse initiating a frame (corresponding to a Timer) is detected, all the status pending bits corresponding to enabled events in that frame are set. The ADCC clears each status bit on receiving the data from ADC for the corresponding event. If a pending event is missed, the ADCC clears the corresponding status bit.

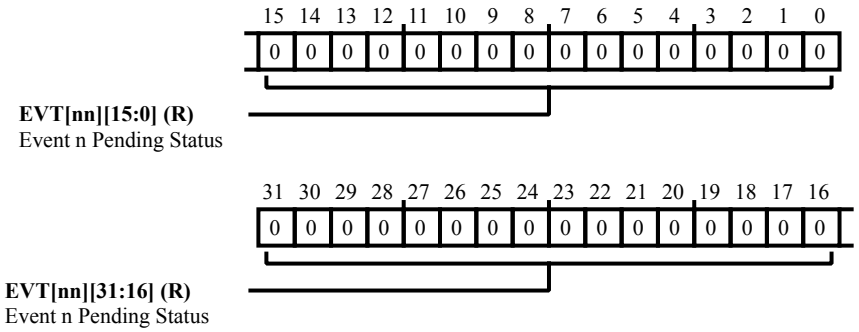


Figure 30-41: ADCC_EPND Register Diagram

Table 30-36: ADCC_EPND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	EVT[nn]	Event n Pending Status. The ADCC_EPND.EVT[nn] bits each correspond to an ADCC event (n from 31 to 0) that are pending (waiting for data transfer). (1 << n) Event n pending 0x80000000 = (1 << 31) Event 31 pending --- --- 0x00000001 = (1 << 0) Event 0 pending

Error Mask Register

The ADCC_ERRMSK register masks (disables) or unmasks (enables) reporting of ADC related errors.

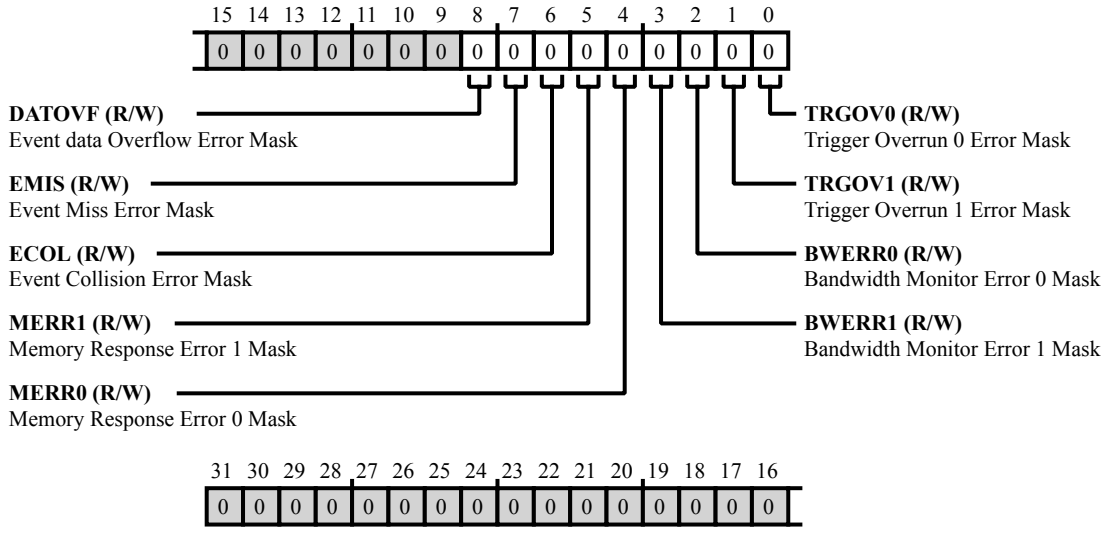


Figure 30-42: ADCC_ERRMSK Register Diagram

Table 30-37: ADCC_ERRMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	DATOVF	Event data Overflow Error Mask. The ADCC_ERRMSK.DATOVF bit masks (disables) generating an error interrupt request on an event overflow error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
7 (R/W)	EMIS	Event Miss Error Mask. The ADCC_ERRMSK.EMIS bit masks (disables) generating an error interrupt request on an event miss error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
6 (R/W)	ECOL	Event Collision Error Mask. The ADCC_ERRMSK.ECOL bit masks (disables) generating an error interrupt request on an event collision error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)

Table 30-37: ADCC_ERRMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	MERR1	Memory Response Error 1 Mask. The ADCC_ERRMSK.MERR1 bit masks (disables) generating an error interrupt request on a memory response error related to Timer 1.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
4 (R/W)	MERR0	Memory Response Error 0 Mask. The ADCC_ERRMSK.MERR0 bit masks (disables) generating an error interrupt request on a memory response error related to Timer 0.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
3 (R/W)	BWERR1	Bandwidth Monitor Error 1 Mask. The ADCC_ERRMSK.BWERR1 bit masks (disables) generating an error interrupt request on a bandwidth monitor error related to Timer 1.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
2 (R/W)	BWERR0	Bandwidth Monitor Error 0 Mask. The ADCC_ERRMSK.BWERR0 bit masks (disables) generating an error interrupt request on a bandwidth monitor error related to Timer 0.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
1 (R/W)	TRGOV1	Trigger Overrun 1 Error Mask. The ADCC_ERRMSK.TRGOV1 bit masks (disables) generating an error interrupt request on a trigger overrun for the trigger associated with Timer 1.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
0 (R/W)	TRGOV0	Trigger Overrun 0 Error Mask. The ADCC_ERRMSK.TRGOV0 bit masks (disables) generating an error interrupt request on a trigger overrun for the trigger associated with Timer 0.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)

Error Mask Clear Register

The ADCC_ERRMSK_CLR register can be used to selectively clear bits in the ADCC_ERRMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_ERRMSK_CLR clears the corresponding bit in ADCC_ERRMSK. Reading the ADCC_ERRMSK_CLR register returns the data present in the ADCC_ERRMSK register.

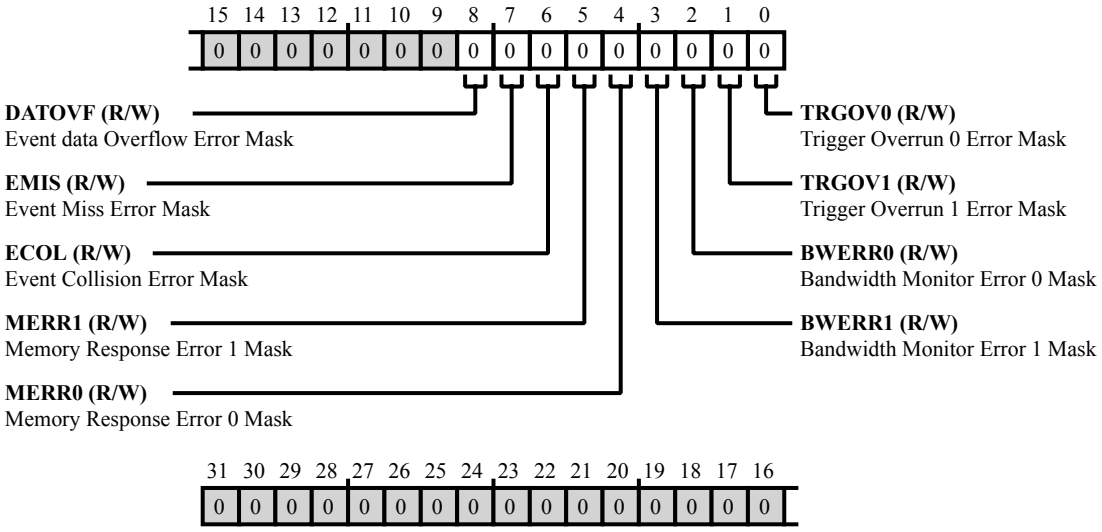


Figure 30-43: ADCC_ERRMSK_CLR Register Diagram

Table 30-38: ADCC_ERRMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	DATOVF	Event data OVerflow Error Mask Clear. Write 1 to ADCC_ERRMSK_CLR.DATOVF to clear the corresponding bit in ADCC_ERRMSK.
7 (R/W1C)	EMIS	Event Miss Error Mask Clear. Write 1 to ADCC_ERRMSK_CLR.EMIS to clear the corresponding bit in ADCC_ERRMSK.
6 (R/W1C)	ECOL	Event Collision Error Mask Clear. Write 1 to ADCC_ERRMSK_CLR.ECOL to clear the corresponding bit in ADCC_ERRMSK.
5 (R/W1C)	MERR1	Memory Response Error 1 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.MERR1 to clear the corresponding bit in ADCC_ERRMSK.
4 (R/W1C)	MERR0	Memory Response Error 0 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.MERR0 to clear the corresponding bit in ADCC_ERRMSK.

Table 30-38: ADCC_ERRMSK_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W1C)	BWERR1	Bandwidth Monitor Error 1 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.BWERR1 to clear the corresponding bit in ADCC_ERRMSK.
2 (R/W1C)	BWERR0	Bandwidth Monitor Error 0 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.BWERR0 to clear the corresponding bit in ADCC_ERRMSK.
1 (R/W1C)	TRGOV1	Trigger Overrun 1 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.TRGOV1 to clear the corresponding bit in ADCC_ERRMSK.
0 (R/W1C)	TRGOV0	Trigger Overrun 0 Mask Clear. Write 1 to ADCC_ERRMSK_CLR.TRGOV0 to clear the corresponding bit in ADCC_ERRMSK.

Error Mask Set Register

The ADCC_ERRMSK_SET register can be used to selectively set bits in the ADCC_ERRMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_ERRMSK_SET sets the corresponding bit in ADCC_ERRMSK. Reading the ADCC_ERRMSK_SET register returns the data present in the ADCC_ERRMSK register.

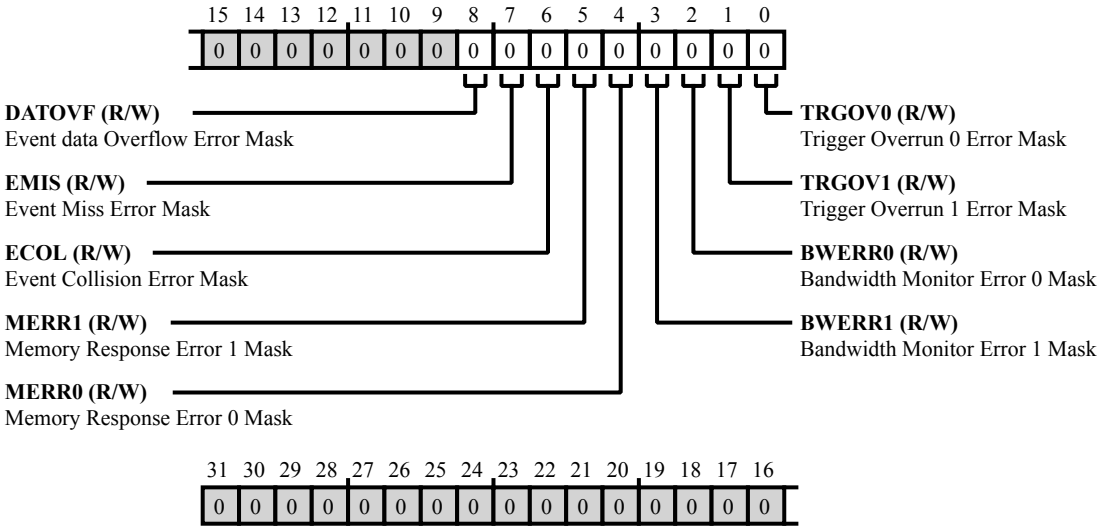


Figure 30-44: ADCC_ERRMSK_SET Register Diagram

Table 30-39: ADCC_ERRMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1S)	DATOVF	Event data Overflow Error Mask Set. Write 1 to ADCC_ERRMSK_SET.DATOVF to set the corresponding bit in ADCC_ERRMSK.
7 (R/W1S)	EMIS	Event Miss Error Mask Set. Write 1 to ADCC_ERRMSK_SET.EMIS to set the corresponding bit in ADCC_ERRMSK.
6 (R/W1S)	ECOL	Event Collision Error Mask Set. Write 1 to ADCC_ERRMSK_SET.ECOL to set the corresponding bit in ADCC_ERRMSK.
5 (R/W1S)	MERR1	Memory Response Error 1 Mask Set. Write 1 to ADCC_ERRMSK_SET.MERR1 to set the corresponding bit in ADCC_ERRMSK.
4 (R/W1S)	MERR0	Memory Response Error 0 Mask Set. Write 1 to ADCC_ERRMSK_SET.MERR0 to set the corresponding bit in ADCC_ERRMSK.

Table 30-39: ADCC_ERRMSK_SET Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W1S)	BWERR1	Bandwidth Monitor Error 1 Mask Set. Write 1 to ADCC_ERRMSK_SET.BWERR1 to set the corresponding bit in ADCC_ERRMSK.
2 (R/W1S)	BWERR0	Bandwidth Monitor Error 0 Mask Set. Write 1 to ADCC_ERRMSK_SET.BWERR0 to set the corresponding bit in ADCC_ERRMSK.
1 (R/W1S)	TRGOV1	Trigger Overrun 1 Mask Set. Write 1 to ADCC_ERRMSK_SET.TRGOV1 to set the corresponding bit in ADCC_ERRMSK.
0 (R/W1S)	TRGOV0	Trigger Overrun 0 Mask Set. Write 1 to ADCC_ERRMSK_SET.TRGOV0 to set the corresponding bit in ADCC_ERRMSK.

Error Status Register

The ADCC_ERRSTAT register indicates status for errors relating to trigger overruns, bandwidth monitoring, memory responses, and events.

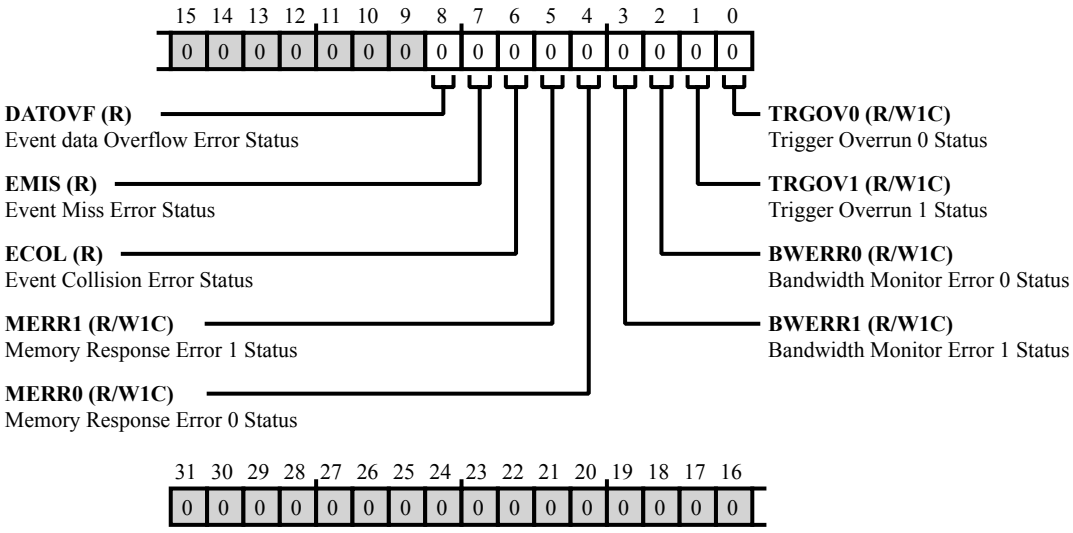


Figure 30-45: ADCC_ERRSTAT Register Diagram

Table 30-40: ADCC_ERRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	DATOVF	Event data Overflow Error Status. The ADCC_ERRSTAT.DATOVF bit indicates whether an event underwent overflow. To identify which the events had overflow, read the value from the ADCC_ERRSTAT register. To clear this error, clear the cause from the ADCC_ERRSTAT register.	
		0	No Status
		1	Event Miss Error
7 (R/NW)	EMIS	Event Miss Error Status. The ADCC_ERRSTAT.EMIS bit indicates whether an event was missed. To identify which the events were missed, read the value from the ADCC_EMISS register. For more information about missed events, see the ADCC functional description. To clear this error, clear the cause from the ADCC_EMISS register.	
		0	No Status
		1	Event Miss Error

Table 30-40: ADCC_ERRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/NW)	ECOL	Event Collision Error Status. The ADCC_ERRSTAT.ECOL bit indicates whether a collision has occurred for any event. To identify which events underwent collision, read the value from the ADCC_ECOL register. For more information about event collisions, see the ADCC functional description. To clear this error, clear the cause from the ADCC_ECOL register.	
		0	No Status
		1	Event Collision Error
5 (R/W1C)	MERR1	Memory Response Error 1 Status. The ADCC_ERRSTAT.MERR1 bit indicates whether a memory write error response on data write issued for an event corresponding to Timer 1 has occurred.	
		0	No Status
		1	Memory Response Error
4 (R/W1C)	MERR0	Memory Response Error 0 Status. The ADCC_ERRSTAT.MERR0 bit indicates whether a memory write error response on data write issued for an event corresponding to Timer 0 has occurred.	
		0	No Status
		1	Memory Response Error
3 (R/W1C)	BWERR1	Bandwidth Monitor Error 1 Status. The ADCC_ERRSTAT.BWERR1 bit indicates whether a bandwidth monitor error (while monitoring is enabled, the timer count crossed the value written in ADCC_BWMON1) for Timer 1 has occurred.	
		0	No Status
		1	Bandwidth Monitor Error
2 (R/W1C)	BWERR0	Bandwidth Monitor Error 0 Status. The ADCC_ERRSTAT.BWERR0 bit indicates whether a bandwidth monitor error (while monitoring is enabled, the timer count crossed the value written in ADCC_BWMON0) for Timer 0 has occurred.	
		0	No Status
		1	Bandwidth Monitor Error

Table 30-40: ADCC_ERRSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	TRGOV1	Trigger Overrun 1 Status. The `ADCC_ERRSTAT.TRGOV1` bit indicates whether a trigger overrun (a trigger input detected while a frame or ADC programming is in progress) for Timer 1 has occurred. The trigger input is ignored.	
		0	No Status
		1	Trigger Overrun
0 (R/W1C)	TRGOV0	Trigger Overrun 0 Status. The `ADCC_ERRSTAT.TRGOV0` bit indicates whether a trigger overrun (a trigger input detected while a frame or ADC programming is in progress) for Timer 0 has occurred. The trigger input is ignored.	
		0	No Status
		1	Trigger Overrun

Event n Control Register

The `ADCC_EVCTL[nn]` register controls programmable features of the corresponding event.

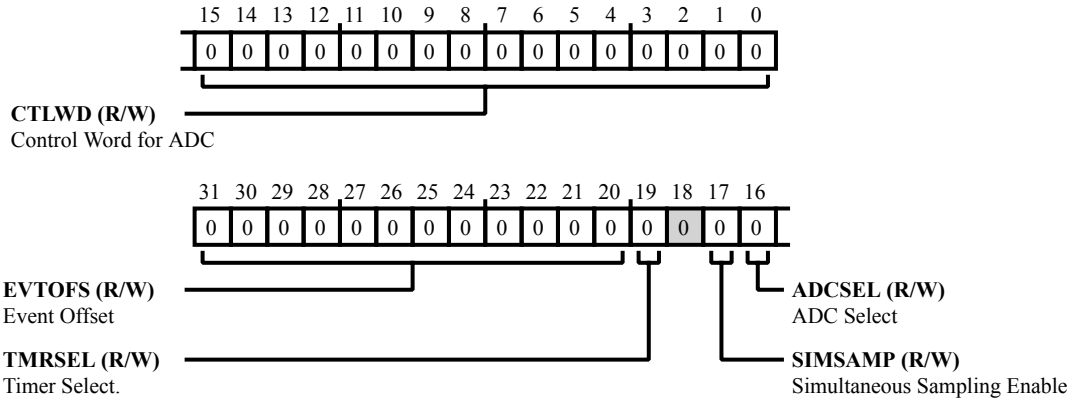


Figure 30-46: ADCC_EVCTL[nn] Register Diagram

Table 30-41: ADCC_EVCTL[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:20 (R/W)	EVTOFS	Event Offset. The `ADCC_EVCTL[nn].EVTOFS` bits hold the memory address offset at which the event's data is stored. This offset is added to the frame's base pointer to produce the final memory address. `ADCC_EVCTL[nn].EVTOFS`'s LSB must =0 for correct address alignment.	
19 (R/W)	TMRSEL	Timer Select.. The `ADCC_EVCTL[nn].TMRSEL` bit selects whether the event corresponds to Timer 0 or 1.	
		0	Event on Timer 0
		1	Event on Timer 1
17 (R/W)	SIMSAMP	Simultaneous Sampling Enable. The `ADCC_EVCTL[nn].SIMSAMP` bit enables simultaneous sampling operation. See "Event Miss Error Status" in the chapter for more information.	
		0	Disable
		1	Enable
16 (R/W)	ADCSEL	ADC Select. The `ADCC_EVCTL[nn].ADCSEL` bit select whether the event executes on the ADC0 interface or the ADC1 interface.	
		0	Event on ADC0
		1	Event on ADC1

Table 30-41: ADCC_EVCTL[nn] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	CTLWD	Control Word for ADC. The `ADCC_EVCTL[nn].CTLWD` bits hold the control word to be sent to ADC. These same 16 bits (or lesser, depending on the corresponding value of `ADCC_TCA0.NCK` or `ADCC_TCA1.NCK`) are sent to ADC. Program the control word LSB aligned in this field. For more information about control word content, see the ADCC programming guidelines.

Event n Data Register

The ADCC_EVDAT[nn] register holds the data sampled from the ADC channel. The data from this register can be read by the core or transferred through DMA. If the data sample received is less than 16 bits wide (depends on value of corresponding ADCC_TCA0.NCK or ADCC_TCA1.NCK), the data is MSB aligned in the ADCC_EVDAT[nn].VALUE field, and the lower bits are zero filled.

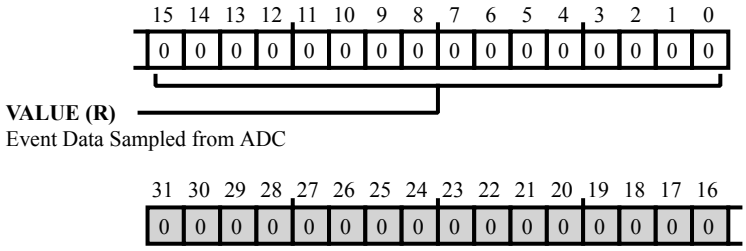


Figure 30-47: ADCC_EVDAT[nn] Register Diagram

Table 30-42: ADCC_EVDAT[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	VALUE	Event Data Sampled from ADC. The ADCC_EVDAT[nn].VALUE bits hold the data of the event sampled from the ADC.

Event n Status Register

The ADCC_EVSTAT[nn] register indicates event status for event n.

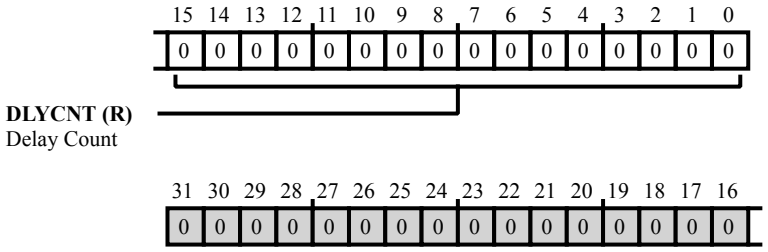


Figure 30-48: ADCC_EVSTAT[nn] Register Diagram

Table 30-43: ADCC_EVSTAT[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	DLYCNT	Delay Count. The ADCC_EVSTAT[nn].DLYCNT bits indicate the number of SCLK cycles by which the event was delayed after the event match occurred. The ADCC updates this field when the corresponding ADCC_ACS or ADCC_BCS is asserted to transmit the control word corresponding to the event to the ADC.

Event Enable Register

The ADCC_EVTEN register enables detection of individual ADCC events. Based on the ADCC_EVCTL[nn].TMRSEL bit selection, the ADCC generates the corresponding ADCC_TMR0_EVT or ADCC_TMR0_EVT event interrupt requests. Note that the events corresponding to a frame must be enabled before the trigger corresponding to the frame arrives.

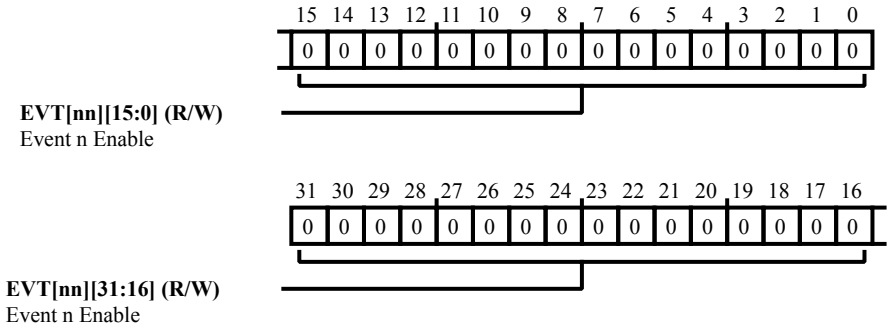


Figure 30-49: ADCC_EVTEN Register Diagram

Table 30-44: ADCC_EVTEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:0 (R/W)	EVT[nn]	Event n Enable. The ADCC_EVTEN.EVT[nn] bits each correspond to an ADCC event (n from 23 to 0). The function of each bit is shown in the enumerations below.	
		0	Disable Event Detection
		4294967295	Enable Event Detection

Event Enable Clear Register

The ADCC_EVTEN_CLR register can be used to selectively clear bits in the ADCC_EVTEN register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_EVTEN_CLR clears the corresponding bit in ADCC_EVTEN. Reading the ADCC_EVTEN_CLR register returns the data present in the ADCC_EVTEN register.

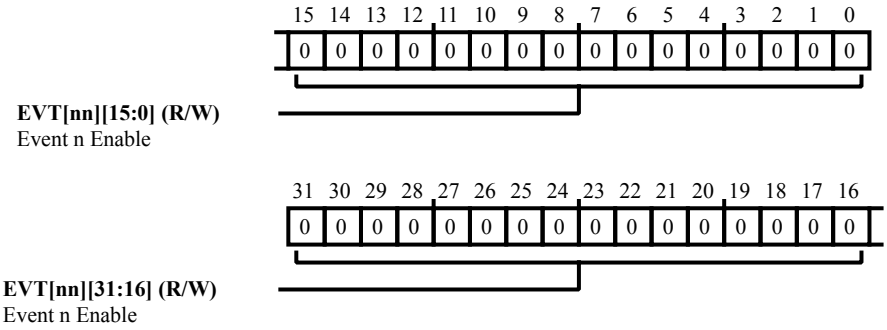


Figure 30-50: ADCC_EVTEN_CLR Register Diagram

Table 30-45: ADCC_EVTEN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1C)	EVT[nn]	Event n Enable Clear. The ADCC_EVTEN_CLR.EVT[nn] bits permit clearing individual bits in the ADCC_EVTEN register without affecting other bits in the register. Write 1 to individual ADCC_EVTEN_CLR.EVT[nn] bits to clear the corresponding bit in ADCC_EVTEN.

Event Enable Set Register

The ADCC_EVTEN_SET register can be used to selectively set bits in the ADCC_EVTEN register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_EVTEN_SET sets the corresponding bit in ADCC_EVTEN. Reading the ADCC_EVTEN_SET register returns the data present in the ADCC_EVTEN register.

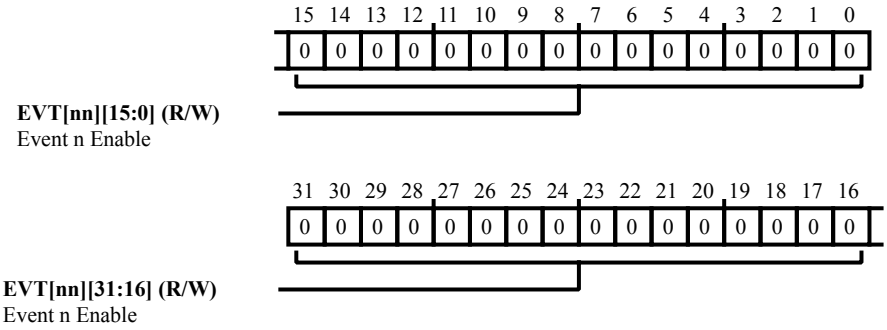


Figure 30-51: ADCC_EVTEN_SET Register Diagram

Table 30-46: ADCC_EVTEN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1S)	EVT[nn]	Event n Enable Set. The ADCC_EVTEN_SET.EVT[nn] bits permit setting individual bits in the ADCC_EVTEN register without affecting other bits in the register. Write 1 to individual ADCC_EVTEN_SET.EVT[nn] bits to set the corresponding bit in ADCC_EVTEN.

Event n Time Register

The ADCC_EVT[nn] register holds the time (in SCLK cycles) at which to sample (the event) from ADC.

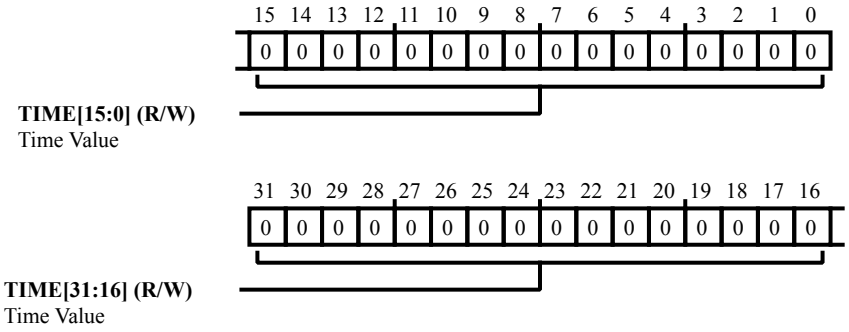


Figure 30-52: ADCC_EVT[nn] Register Diagram

Table 30-47: ADCC_EVT[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TIME	Time Value. The ADCC_EVT[nn].TIME bits holds the time value (in terms SCLK cycles) at which to sample (the event) from ADC.

Frame Interrupt Mask Register

The ADCC_FIMSK register masks (disables) generation of ADCC_TMR0_EVT or ADCC_TMR1_EVT event interrupt requests based on status in the ADCC_FISTAT register.

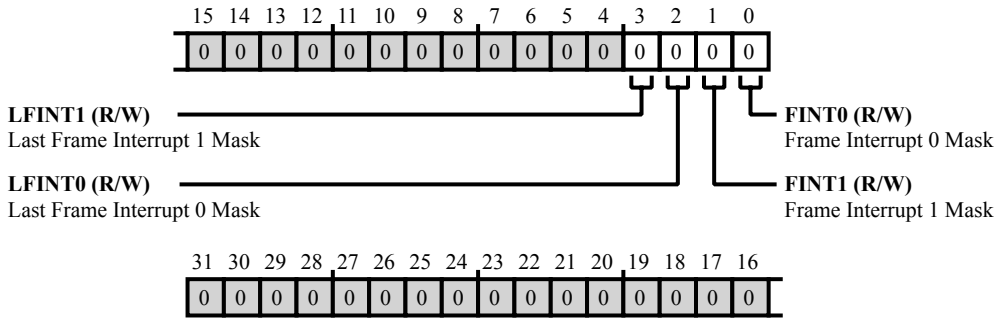


Figure 30-53: ADCC_FIMSK Register Diagram

Table 30-48: ADCC_FIMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	LFINT1	Last Frame Interrupt 1 Mask. The ADCC_FIMSK.LFINT1 bits mask (disable) the ADCC_FISTAT.LFINT1 frame complete interrupt requests for data transfer related to Timer 1.	
2 (R/W)	LFINT0	Last Frame Interrupt 0 Mask. The ADCC_FIMSK.LFINT0 bits mask (disable) the ADCC_FISTAT.LFINT0 frame complete interrupt requests for data transfer related to Timer 0.	
1 (R/W)	FINT1	Frame Interrupt 1 Mask. The ADCC_FIMSK.FINT1 bits mask (disable) the ADCC_FISTAT.FINT1 frame complete interrupt requests for data transfer related to Timer 1.	
		0	Unmask (Enable) Frame Interrupt
		1	Mask (Disable) Frame Interrupt
0 (R/W)	FINT0	Frame Interrupt 0 Mask. The ADCC_FIMSK.FINT0 bits mask (disable) the ADCC_FISTAT.FINT0 frame complete interrupt requests for data transfer related to Timer 0.	
		0	Unmask (Enable) Frame Interrupt
		1	Mask (Disable) Frame Interrupt

Frame Interrupt Mask Clear Register

The ADCC_FIMSK_CLR register can be used to selectively clear bits in the ADCC_FIMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_FIMSK_CLR clears the corresponding bit in ADCC_FIMSK. Reading the ADCC_FIMSK_CLR register returns the data present in the ADCC_FIMSK register.

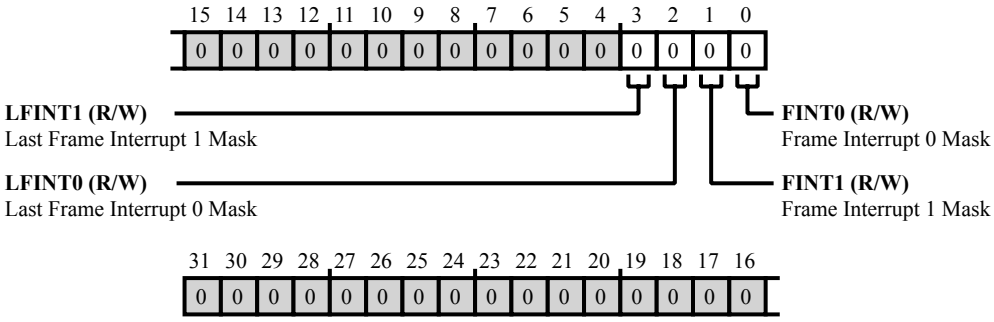


Figure 30-54: ADCC_FIMSK_CLR Register Diagram

Table 30-49: ADCC_FIMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W1C)	LFINT1	Last Frame Interrupt 1 Mask. The ADCC_FIMSK_CLR.LFINT1 bits permit clearing the ADCC_FIMSK.LFINT1 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_CLR.LFINT1 bit to clear the ADCC_FIMSK.LFINT1 bit.
2 (R/W1C)	LFINT0	Last Frame Interrupt 0 Mask. The ADCC_FIMSK_CLR.LFINT0 bits permit clearing the ADCC_FIMSK.LFINT0 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_CLR.LFINT0 bit to clear the ADCC_FIMSK.LFINT0 bit.
1 (R/W1C)	FINT1	Frame Interrupt 1 Mask Clear. The ADCC_FIMSK_CLR.FINT1 bits permit clearing the ADCC_FIMSK.FINT1 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_CLR.FINT1 bit to clear the ADCC_FIMSK.FINT1 bit.
0 (R/W1C)	FINT0	Frame Interrupt 0 Mask Clear. The ADCC_FIMSK_CLR.FINT0 bits permit clearing the ADCC_FIMSK.FINT0 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_CLR.FINT0 bit to clear the ADCC_FIMSK.FINT0 bit.

Frame Interrupt Mask Set Register

The ADCC_FIMSK_SET register can be used to selectively set bits in the ADCC_FIMSK register without affecting other bits in the register. Writing a 1 to any bit position in ADCC_FIMSK_SET sets the corresponding bit in ADCC_FIMSK. Reading the ADCC_FIMSK_SET register returns the data present in the ADCC_FIMSK register.

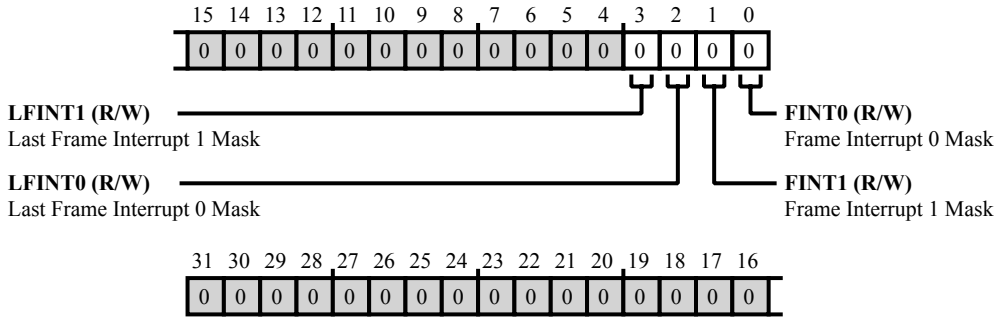


Figure 30-55: ADCC_FIMSK_SET Register Diagram

Table 30-50: ADCC_FIMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W1S)	LFINT1	Last Frame Interrupt 1 Mask. The ADCC_FIMSK_SET.LFINT1 bits permit setting the ADCC_FIMSK.LFINT1 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_SET.LFINT1 bit to set the ADCC_FIMSK.LFINT1 bit.
2 (R/W1S)	LFINT0	Last Frame Interrupt 0 Mask. The ADCC_FIMSK_SET.LFINT0 bits permit setting the ADCC_FIMSK.LFINT0 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_SET.LFINT0 bit to set the ADCC_FIMSK.LFINT0 bit.
1 (R/W1S)	FINT1	Frame Interrupt 1 Mask Set. The ADCC_FIMSK_SET.FINT1 bits permit setting the ADCC_FIMSK.FINT1 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_SET.FINT1 bit to set the ADCC_FIMSK.FINT1 bit.
0 (R/W1S)	FINT0	Frame Interrupt 0 Mask Set. The ADCC_FIMSK_SET.FINT0 bits permit setting the ADCC_FIMSK.FINT0 bit without affecting other bits in the register. Write 1 to the ADCC_FIMSK_SET.FINT0 bit to set the ADCC_FIMSK.FINT0 bit.

Frame Interrupt Status Register

The ADCC_FISTAT register indicates frame interrupt status. The ADCC generates interrupt requests corresponding to Timer 0 on the ADCC_TMR0_EVT output and generates the interrupt requests corresponding to Timer 1 on the ADCC_TMR1_EVT output.

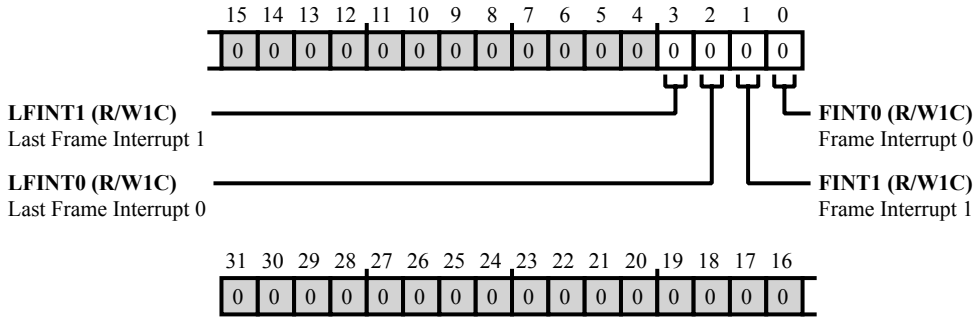


Figure 30-56: ADCC_FISTAT Register Diagram

Table 30-51: ADCC_FISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1C)	LFINT1	Last Frame Interrupt 1. The ADCC_FISTAT.LFINT1 bit indicates core/DMA last frame interrupt status for a Timer 1 frame. If the register NUMFRAM1 in ADCC is non zero(N), this bit gets set when all data corresponding to events of (N +1) frames at TIMER1 are successfully DMAed by the ADCC(and responses received) in DMA mode. In core mode, this bits gets set when all data of the (N+1) frames have been received by ADCC from ADC. This bit needs to be cleared before TIMER1 can accept the next trigger.	
2 (R/W1C)	LFINT0	Last Frame Interrupt 0. The ADCC_FISTAT.LFINT0 bit indicates core/DMA last frame interrupt status for a Timer 0 frame. If the register NUMFRAM0 in ADCC is non zero(N), this bit gets set when all data corresponding to events of (N +1) frames at TIMER0 are successfully DMAed by the ADCC(and responses received) in DMA mode. In core mode, this bits gets set when all data of the (N+1) frames have been received by ADCC from ADC. This bit needs to be cleared before TIMER0 can accept the next trigger.	
1 (R/W1C)	FINT1	Frame Interrupt 1. The ADCC_FISTAT.FINT1 bit indicates core/DMA interrupt status for a Timer 1 frame. In non-DMA mode, this bit is set when the data corresponding to all events (which were not missed) in a frame are received by the ADCC (from the ADC). In DMA mode, this bit is set when all data corresponding to events of a frame are successfully DMA transferred by the ADCC (and responses received).	
		0	No Status
		1	Frame Complete

Table 30-51: ADCC_FISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	FINT0	Frame Interrupt 0. The `ADCC_FISTAT.FINT0` bit indicates core/DMA interrupt status for a Timer 0 frame. In non-DMA mode, this bit is set when the data corresponding to all events (which were not missed) in a frame are received by the ADCC (from the ADC). In DMA mode, this bit is set when all data corresponding to events of a frame are successfully DMA transferred by the ADCC (and responses received).	
		0	No Status
		1	Frame Complete

Frame Increment 0 Register

The ADCC_FRINC0 register contains the address increment applied between the base-address of consecutive Timer 0 frames of DMA data (unless a fresh write of the ADCC_BPTR0 register occurred in between). The value is a signed, two's complement byte address increment.

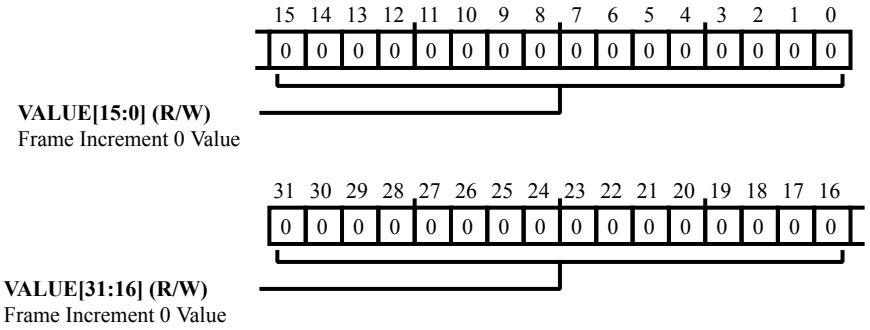


Figure 30-57: ADCC_FRINC0 Register Diagram

Table 30-52: ADCC_FRINC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Increment 0 Value. The ADCC_FRINC0.VALUE bits hold memory offset increment applied to the base-address of consecutive Timer 0 frames of DMA data. Note that bit 0 must be written as 0 to ensure correct address alignment.

Frame Increment 1 Register

The ADCC_FRINC1 register contains the address increment applied between the base-address of consecutive Timer 1 frames of DMA data (unless a fresh write of the ADCC_BPTR0 register occurred in between). The value is a signed, two's complement byte address increment.

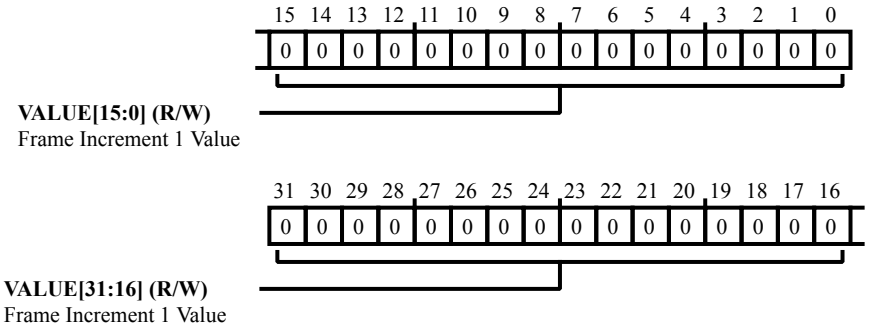


Figure 30-58: ADCC_FRINC1 Register Diagram

Table 30-53: ADCC_FRINC1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Increment 1 Value. The ADCC_FRINC1.VALUE bits hold memory offset increment applied to the base-address of consecutive Timer 1 frames of DMA data. Note that bit 0 must be written as 0 to ensure correct address alignment.

Timer0 Frame Limit Count Register

The ADCC_NUMFRAM0 register allows programming number of TIMER0 frames before frame interrupt 0 (FINT0).

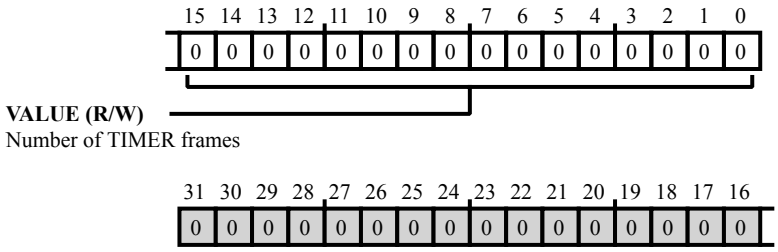


Figure 30-59: ADCC_NUMFRAM0 Register Diagram

Table 30-54: ADCC_NUMFRAM0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Number of TIMER frames. The ADCC_NUMFRAM0.VALUE bit field sets the number of TIMER0 frames after which FINT is generated.

Timer1 Frame Limit Count Register

The ADCC_NUMFRAM1 register allows programming number of TIMER1 frames before frame interrupt 0 (FINT0).

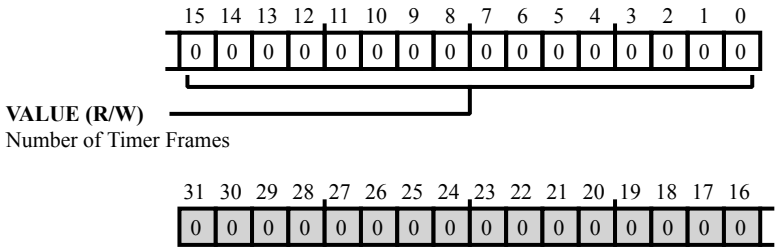


Figure 30-60: ADCC_NUMFRAM1 Register Diagram

Table 30-55: ADCC_NUMFRAM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Number of Timer Frames. The ADCC_NUMFRAM1.VALUE bit field sets the number of TIMER1 frames after which FINT is generated.

Timer 0 Status Register

The ADCC_T0STAT register indicates Timer 0 related status for the ADCC operation.

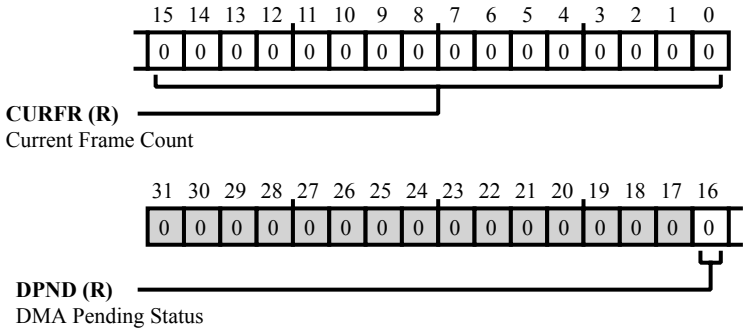


Figure 30-61: ADCC_T0STAT Register Diagram

Table 30-56: ADCC_T0STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	DPND	DMA Pending Status. The ADCC_T0STAT.DPND bit indicates pending status for data received from the ADC. When there are pending transfers, even if the ADCC is disabled while they are pending, the DMA finishes these pending data transfers.	
		0	No Status
		1	DMA Pending
15:0 (R/NW)	CURFR	Current Frame Count. The ADCC_T0STAT.CURFR bits hold the current frame number in the chronological order after ADCC enable occurred (linear buffer mode) or hold the frame number after the last wrap around (circular buffer mode). This field wraps over after 0xFFFF frames.	

Timer 1 Status Register

The ADCC_T1STAT register indicates Timer 1 related status for the ADCC operation.

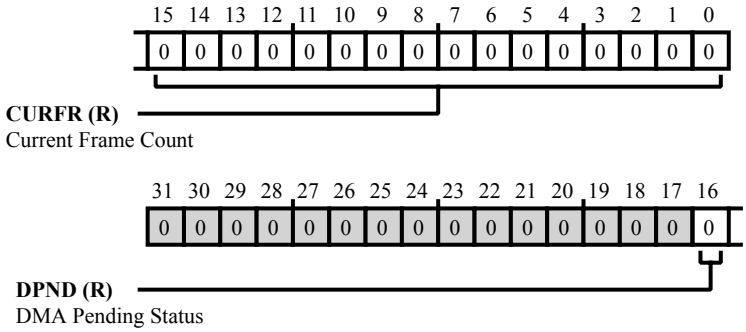


Figure 30-62: ADCC_T1STAT Register Diagram

Table 30-57: ADCC_T1STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	DPND	DMA Pending Status. The ADCC_T1STAT.DPND bit indicates pending status for data received from the ADC. When there are pending transfers, even if the ADCC is disabled while they are pending, the DMA finishes these pending data transfers.	
		0	No Status
		1	DMA Pending
15:0 (R/NW)	CURFR	Current Frame Count. The ADCC_T1STAT.CURFR bits hold the current frame number in the chronological order after ADCC enable occurred (linear buffer mode) or hold the frame number after the last wrap around (circular buffer mode). This field wraps over after 0xFFFF frames.	

Timing Control A (ADC0) Register

The ADCC_TCA0 register controls timing related to the unit A interface clock (ADCC_ACLK) and chip select signals.

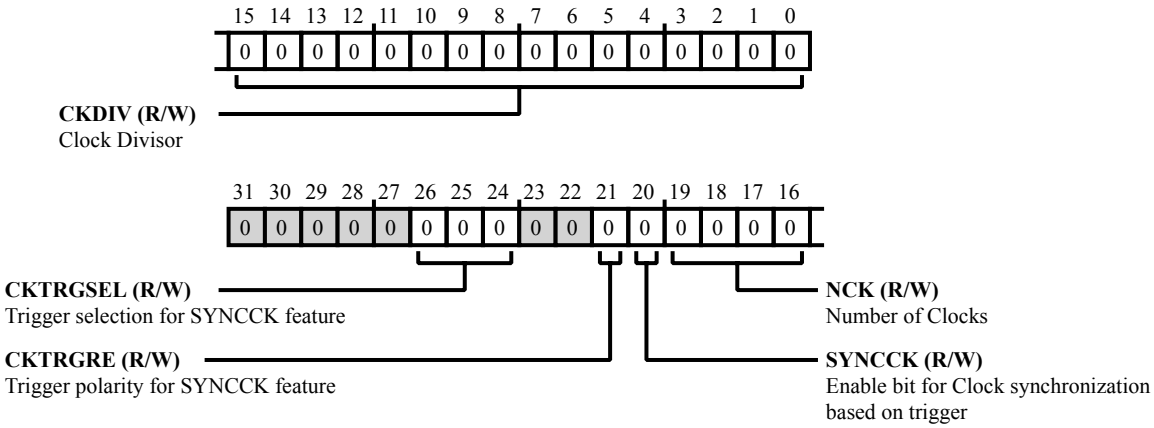


Figure 30-63: ADCC_TCA0 Register Diagram

Table 30-58: ADCC_TCA0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26:24 (R/W)	CKTRGSEL	Trigger selection for SYNCCK feature. The ADCC_TCA0.CKTRGSEL bit selects the trigger input to be used for synchronizing ADC clock. The valid values range from 0-5.
21 (R/W)	CKTRGRE	Trigger polarity for SYNCCK feature. The ADCC_TCA0.CKTRGRE bit provides a choice for the trigger polarity to be used for the trigger input with which the ADC clock is to be synchronized. A value of 1 indicates risedge based trigger, and a value of 0 indicates falledge based trigger identification.
20 (R/W)	SYNCCK	Enable bit for Clock synchronization based on trigger. The ADCC_TCA0.SYNCCK bit enables the clock synchronization w.r.t a trigger input to the block. A value of "1" enables this feature.
19:16 (R/W)	NCK	Number of Clocks. The ADCC_TCA0.NCK bits select the number of ADCC_ACLK ADC clock cycles for each ADCC_ACS chip select pulse. A value of 0 implies 16 clock cycles. The ADCC_TCA0.NCK programming should ensure that total number of data/control bits is less than or equal to 16 bits. If either data or control are on dual bit lines (ADCC_CTL.DSIZE =1), the maximum number of clock permitted is 8, and NCK=0 is not permitted.

Table 30-58: ADCC_TCA0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	CKDIV	Clock Divisor. The `ADCC_TCA0.CKDIV` bits select the clock divisor ratio (SCLK:ACK) for clocks to be sent to ADC calculated as: ADCC_ACLK frequency = (SCLK frequency) / (CKDIV+1) Yielding, `ADCC_TCA0.CKDIV` =0 represents a ratio of 1:1, =1 represents a ratio of 1:2, `ADCC_TCA0.CKDIV` =2 represents a ratio of 1:3, and so on.

Timing Control A (ADC1) Register

The ADCC_TCA1 register controls timing related to the Unit B interface clock (ADCC_ACLK) and and chip select signals.

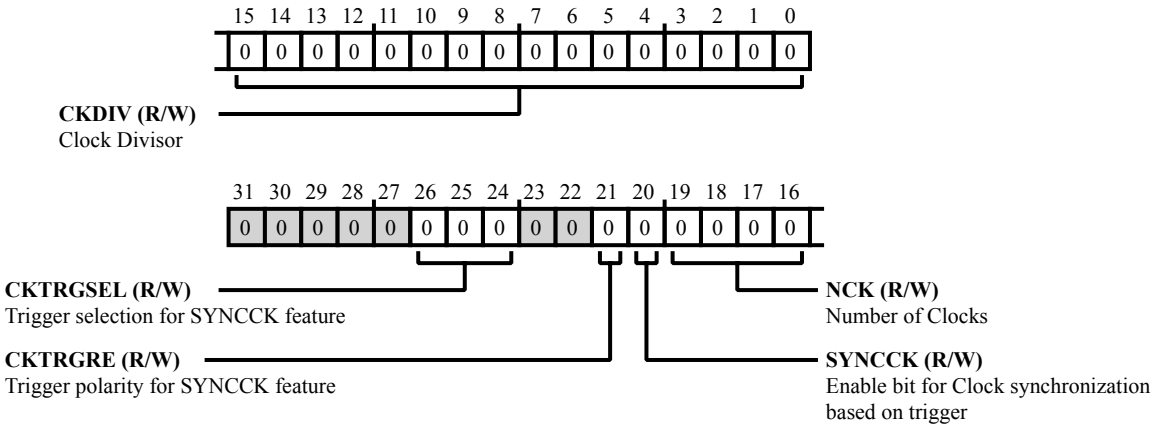


Figure 30-64: ADCC_TCA1 Register Diagram

Table 30-59: ADCC_TCA1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26:24 (R/W)	CKTRGSEL	Trigger selection for SYNCCK feature. The ADCC_TCA1.CKTRGSEL bit selects the trigger input to be used for synchronizing ADC clock. The valid values range from 0-5.
21 (R/W)	CKTRGRE	Trigger polarity for SYNCCK feature. The ADCC_TCA1.CKTRGRE bit provides a choice for the trigger polarity to be used for the trigger input with which the ADC clock is to be synchronized. A value of 1 indicates risedge based trigger, and a value of 0 indicates falledge based trigger identification.
20 (R/W)	SYNCCK	Enable bit for Clock synchronization based on trigger. The ADCC_TCA1.SYNCCK bit enables the clock synchronization w.r.t a trigger input to the block. A value of "1" enables this feature.
19:16 (R/W)	NCK	Number of Clocks. The ADCC_TCA1.NCK bits select the number of ADCC_ACLK ADC clock cycles for each ADCC_ACS chip select pulse. A value of 0 implies 16 clock cycles. The ADCC_TCA1.NCK programming should ensure that total number of data/control bits is less than or equal to 16 bits. If either data or control are on dual bit lines (ADCC_CTL.DSIZE =1), the maximum number of clock permitted is 8, and NCK=0 is not permitted.

Table 30-59: ADCC_TCA1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	CKDIV	Clock Divisor. The `ADCC_TCA1.CKDIV` bits select the clock divisor ratio (SCLK:ACK) for clocks to be sent to ADC calculated as: ADCC_ACLK frequency = (SCLK frequency) / (CKDIV+1) Yielding, `ADCC_TCA1.CKDIV` =0 represents a ratio of 1:1, =1 represents a ratio of 1:2, `ADCC_TCA1.CKDIV` =2 represents a ratio of 1:3, and so on.

Timing Control B (ADC0) Register

The ADCC_TCB0 register controls parameter values for timing relationships between the ADCC_ACS and ADCC_ACLK pin signals in the unit A interface. For a timing diagram illustrating these timing relationships, see the ADCC functional description.

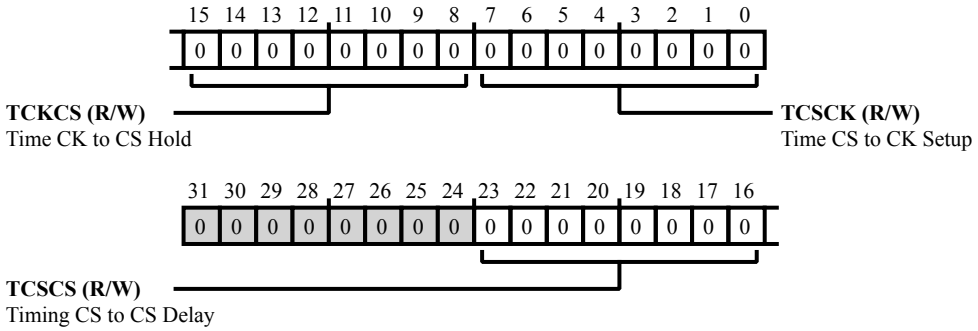


Figure 30-65: ADCC_TCB0 Register Diagram

Table 30-60: ADCC_TCB0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	TCSCS	Timing CS to CS Delay. The ADCC_TCB0.TCSCS value selects minimum delay time (in ADCC_ACLK cycles) required between on ADCC_ACS de-asserted edge and the next ADCC_ACS asserted edge. A value of 0 implies 256 cycles delay.
15:8 (R/W)	TCKCS	Time CK to CS Hold. The ADCC_TCB0.TCKCS value selects the minimum hold time (in ADCC_ACLK cycles) after (ADCC_TCB0.TCSCK + ADCC_TCA0.NCK) clock cycles have elapsed during ADCC_ACS asserted for which the chip select should be held asserted. A value of 0 implies 0 clock cycle delay.
7:0 (R/W)	TCSCK	Time CS to CK Setup. The ADCC_TCB0.TCSCK value selects the minimum setup time (in ADCC_ACLK cycles) required from the ADCC_ACS asserted to the first edge of the ADCC_ACLK clock. A value of 0 implies 256 clock cycles.

Timing Control B (ADC1) Register

The ADCC_TCB1 register controls parameter values for timing relationships between the ADCC_BCS and ADCC_BCLK pin signals in the unit B interface. For a timing diagram illustrating these timing relationships, see the ADCC functional description.

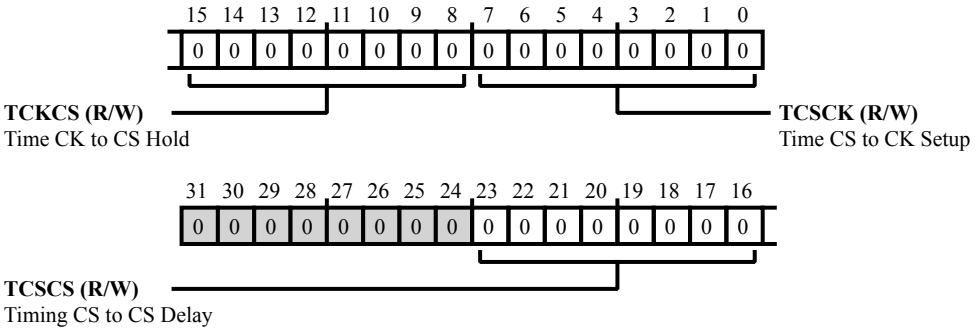


Figure 30-66: ADCC_TCB1 Register Diagram

Table 30-61: ADCC_TCB1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	TCSCS	Timing CS to CS Delay. The ADCC_TCB1.TCSCS value selects minimum delay time (in ADCC_BCLK cycles) required between on ADCC_BCS de-asserted edge and the next ADCC_BCS asserted edge. A value of 0 implies 256 cycles delay.
15:8 (R/W)	TCKCS	Time CK to CS Hold. The ADCC_TCB1.TCKCS value selects the minimum hold time (in ADCC_BCLK cycles) after (ADCC_TCB1.TCSCK + ADCC_TCA1.NCK) clock cycles have elapsed during ADCC_BCS asserted for which the chip select should be held asserted. A value of 0 implies 0 clock cycle delay.
7:0 (R/W)	TCSCK	Time CS to CK Setup. The ADCC_TCB1.TCSCK value selects the minimum setup time (in ADCC_BCLK cycles) required from the ADCC_BCS asserted to the first edge of the ADCC_BCLK clock. A value of 0 implies 256 clock cycles.

Timer 0 Current Count Register

The ADCC_TMR0 register holds the current count of the Timer 0 while a frame is in progress. This register may be used to determine the current timer count, which indicates approximately how much of the frame has been completed. The timer is initiated by the trigger and counts up starting from 0. The timer increments every SCLK cycle until the frame completion. In DMA mode, the frame completion happens when event all data have been placed in memory and responses are received. In non-DMA mode, the frame completion happens when all events' data have been received from the ADC. If a frame is not in progress, the ADCC_TMR0.CNT field is reset.

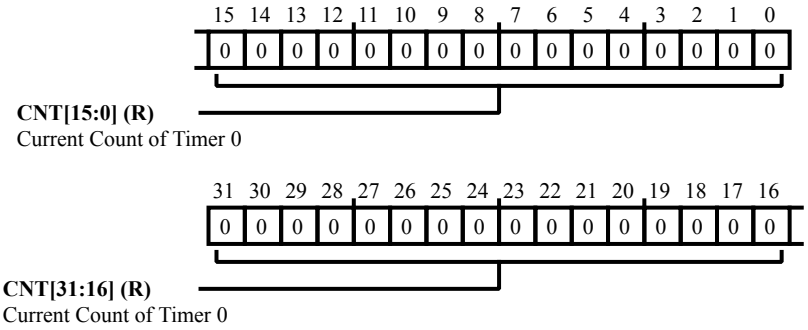


Figure 30-67: ADCC_TMR0 Register Diagram

Table 30-62: ADCC_TMR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Current Count of Timer 0. The ADCC_TMR0.CNT bits hold the current count of Timer 0.

Timer 1 Current Count Register

The ADCC_TMR1 register holds the current count of the Timer 1 while a frame is in progress. This register may be used to determine the current timer count, which indicates approximately how much of the frame has been completed. The timer is initiated by the trigger and counts up starting from 0. The timer increments every SCLK cycle until the frame completion. In DMA mode, the frame completion happens when event all data have been placed in memory and responses are received. In non-DMA mode, the frame completion happens when all events' data have been received from the ADC. If a frame is not in progress, the ADCC_TMR1.CNT field is reset.

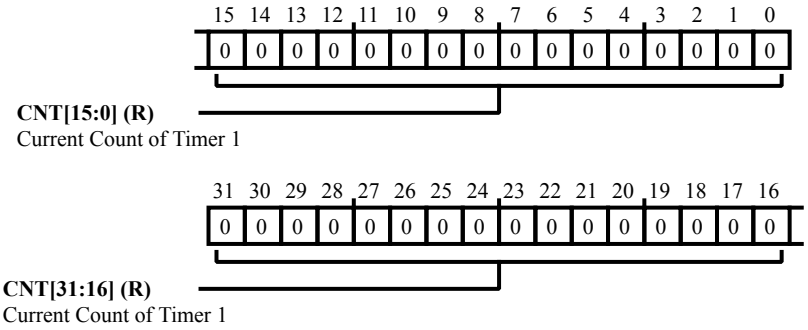


Figure 30-68: ADCC_TMR1 Register Diagram

Table 30-63: ADCC_TMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	CNT	Current Count of Timer 1. The ADCC_TMR1.CNT bits hold the current count of Timer 1.

Trigger Count TIMER0 Register

The ADCC_TRGCNT0 register holds the trigger number being serviced at TIMER0.

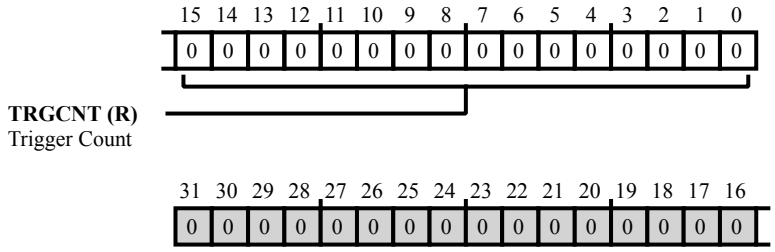


Figure 30-69: ADCC_TRGCNT0 Register Diagram

Table 30-64: ADCC_TRGCNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	TRGCNT	Trigger Count. The ADCC_TRGCNT0.TRGCNT bit field indicates the trigger number being serviced by TIMER0.

Trigger Count TIMER1 Register

The ADCC_TRGCNT1 register holds the trigger number being serviced at TIMER1.

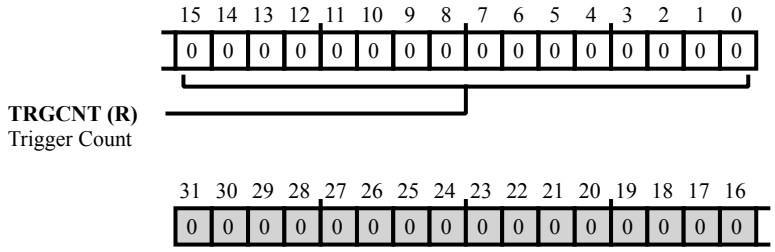


Figure 30-70: ADCC_TRGCNT1 Register Diagram

Table 30-65: ADCC_TRGCNT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/NW)	TRGCNT	Trigger Count. The ADCC_TRGCNT1.TRGCNT bit field indicates the trigger number being serviced by TIMER1.

31 Digital-to-Analog Converter Controller (DACC)

The DAC controller (DACC) automates DAC data conversion and simplifies DAC accesses. The DACC provides an interface that synchronizes the controls between the processor and a digital-to-analog converter (DAC).

NOTE: The DACC and ADCC chapters describe the control and data interface to the AFE. For information about the analog portion (I/O pins and electrical specifications) of the AFE, see the product data sheet.

Using the DACC permits flexible scheduling of data transfer and provides precise control execution of timing and analog output on the DACs. The DACC saves both processor MIPS and provides precise controllability for DAC conversion or output time. The DACC is designed to interface with the on-chip internal DACs which require minimal core intervention.

DACC Features

The DACC provides many architecture-based features (basic to the design) and mode-selectable features (usage is optional or configurable).

Architecture-based features of the DACC include:

- A single DAC Interface to control multiple DACs:
  - DAC0 12-bit DAC
  - DAC1 - 8-bit FOCP DAC, Over Voltage
  - DAC2 - 8-bit FOCP DAC, Under Voltage
- Automated DAC conversion with programmable timing
- Support of both core mode and DMA mode for updating the DAC buffer. The DMA interface width can be programmable as a 16-bit or 32-bit interface.
- Separate four-deep FIFO for each DAC interface to queue the data for conversion, reducing the data request rate to the core or DMA
- Serial clock, frame sync select, and data signal to control the DAC operations

- Internally-generated DAC clock from processor system clock

  This clock can be gated (for example, it is active only when controlling the DAC) to provide excellent noise immunity during the conversion process. The clock polarity (for example, the first edge after assertion of the frame sync signal) is configurable.

- Two built-in DMA units

  There is one DMA unit for each DAC. It used for transferring data for conversion. The DMA units support an optional mechanism for circular buffering.

- Error detection capabilities, including support for detection of memory access error, DAC data underflow, and address alignment error conditions

  The DACC does not perform a data transmission in case of underflow. An underflow interrupt request can be signaled to the processor core.

Mode-selectable features of the DACC include:

- Gated clock signal with configurable polarity.

  The DAC clock given to a DAC can be a continuous clock or can be gated when the DAC data is not driven. The clock edge polarity (that is, the driving edge of sync and data) can be configured to the rising edge or the falling edge.

- Frame sync signal with configurable width and polarity.

  The DAC frame sync signal has a programmable width in terms of the DAC clocks. The polarity of this signal is configurable as active high or active low signal. The DAC uses this signal to start the conversion, and to output the converted data.

- Timing for the DAC update frequency is configurable
- Data length for the DAC interface is programmable for up to 16-bit data lengths
- Data transmission to the DAC in LSB-first or MSB-bit first format
- Circular buffering for DMA data

  When using DMA mode to update the DAC FIFO, the buffer submitted to DMA can be configured for circular buffering to transmit the data continuously without core intervention. Optionally, an interrupt request can be enabled to signal the core at the end of each circular buffer.

- Status indication (status bits) and optional interrupt request generation to the core on DAC data underflow

DACC Functional Description

The following sections describe DACC functionality:

- DACC Signal Descriptions
- DACC Architectural Concepts

NOTE: Analog front end (AFE) start-up requires the loading of some initialization settings. This initialization occurs automatically when the ADCC is enabled. Therefore, enable the ADCC before enabling the DACC after power-on reset. If continued operation of the ADCC is unnecessary, it can be disabled while the DACC continues to operate.

CM41X_M4 DACC Register List

The DAC controller (DACC) automates the DAC data conversion process and simplifies DAC management. The DACC provides an interface that synchronizes the controls between the processor and a digital-to-analog converter (DAC). A set of registers govern DACC operations. For more information on DACC functionality, see the DACC register descriptions.

Table 31-1: CM41X_M4 DACC Register List

Name	Description
DACC_BCST_CTL	Broadcast (Write) Control Register
DACC_BPTR0	Base Pointer 0 Register
DACC_CNT0	Count 0 Register
DACC_CNTCUR0	Current Count 0 Register
DACC_CTL0	Control 0 Register
DACC_DAT0	Data FIFO 0 Register
DACC_ERRMSK	Error Mask Register
DACC_ERRMSK_CLR	Error Mask Clear Register
DACC_ERRMSK_SET	Error Mask Set Register
DACC_ERRSTAT	Error Status Register
DACC_IMSK	Interrupt Mask Register
DACC_IMSK_CLR	Interrupt Mask Clear Register
DACC_IMSK_SET	Interrupt Mask Set Register
DACC_ISTAT	Interrupt Status Register
DACC_MOD0	Modify 0 Register
DACC_STAT	Status Register
DACC_TC0	Timing Control 0 Register

DACC Signal Descriptions

The DACC controls the operation of internal DACs, based on its timing register settings. Because the DACs are internal, none of the DACC signals are available as external package pins.

NOTE: There is one instance of the DAC interface on the processor. Each offers separate clock, frame sync, and data signals for controlling three DACs independently.

A number of signals connect the DACC to the DACs for DAC clock, frame sync, and data. Using these signals, the DACC regulates the DAC data transfer and data conversion. These signals appear in the ***DACC-to-DAC0/1 Signal Descriptions*** table.

Table 31-2: DACC-to-DAC0/1 Signal Descriptions

Name	I/O	Description
DACC_ACLK	O	DACC DAC (A) clock
DACC_AFS	O	DACC DAC (A) frame sync
DACC_AD0	O	DACC DAC (A) data 0 for writing conversion data to the DAC

The DACC also has a number of signals to send the status and error information to the processor. These signals appear in the ***DACC-to-Core Signal Descriptions*** table.

Table 31-3: DACC-to-Core Signal Descriptions

Signal Name	I/O	Signal Description
DACC_DAC0	O	DAC0 Event when a DAC frame is completed
DACC_DAC1	O	DAC1 Event when a DAC frame is completed
DACC_ERR	O	DAC Error Interrupt

The following are more detailed descriptions of each DACC signal type.

DAC Clock (DACC_ACLK, DACC_BCLK)

This clock signal is given to DAC, based on which of the other two DAC signals, frame sync and data are driven. The clock edge polarity (that is, the driving edge of CS or SYNC and data) can be configured to rising edge or falling edge using DACC clock polarity bit setting.

The DAC clock signal is internally generated from processor SCLK and is configurable as follows:

```
DAC_CLK frequency = SCLK ÷ (DACC_TC.DCKDIV + 1)
```

The DAC clock can be given continuous free running or gated (that is, given out only when data is driven). This clock configuration uses the DACC gated clock enable bit setting.

DAC Frame Sync (DACC_AFS, DACC_BFS)

The DAC frame sync signal is active when valid data is driven out to the DAC. The DACC uses the data length field (for example, DACC_CTL0.DLEN) to decide the active period of the DAC frame sync.

The frequency of this signal determines the DAC update rate, which can be programmed using the DACC frame sync idle field (for example, DACC_TC0.FSIDLE). The DACC uses the frame sync idle field to decide the idle period between two DAC sync active pulses. This field is programmed in terms of the number of DAC clocks. It is a 16-bit field and a value of zero implies 1 DAC clock cycle.

Therefore, the periodicity of DACC_FS = (DACC_TC.DFSIDLE + 1) + DACC_CTL.DLEN.

If the DACC data length is zero, periodicity of DAC_FS = (DACC_TC.DFSIDLE + 1) + 16.

The polarity of the DAC sync signal can be configured to active high or active low, using the DACC sync polarity bit setting (for example,DACC_CTL0.SYNCPOL).

DAC Data (DACC_AD0, DACC_BD0)

The data to the DAC is serially driven out on the DACC data pin (for example, DACC_AD0).

The DACC data length field (for example, DACC_CTL0.DLEN) determines the DAC interface length. It is a four-bit field, that allows a DAC interface length of from 1 to 16 (a value of 4'h0 implies a data length of 16 bits).

The DAC data can be transmitted in LSB first format or MSB first format by configuring the DACC LSB first bit (for example, DACC_CTL0.LSBF).

On the processor, the DACC controls the 12-bit DACs, which support MSB first format. Configure the DACC data length to 0xC, and the DACC LSB first bit to 0.

There can be situations where an incomplete or larger DAC frame is sent to the DAC when the data length field is set incorrectly. In these cases, the DAC on the processor ignores frames smaller than 11-bits and discards extra bits if the frame sizes are more than 12 bits.

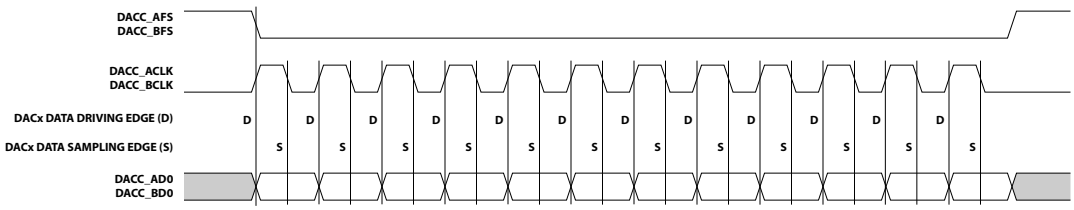


Figure 31-1: DACC Signals

DACC Architectural Concepts

The DACC Block Diagram shows the top-level architecture of DAC interface. The interface consists of:

- A DMA and core interface. This interface permits the core to read and write DACC registers for configuration, control, and word-by-word data transfer. The independent DMA engine of the DACC provides data transfer without core overhead. For more information about this interface, see Core and DMA Interfaces and Data Transfer Modes.
- DAC pending data FIFOs. Each DAC has a four-deep FIFO for pending data transfer. These FIFOs help prevent data underrun. For more information, see Pending Data FIFO.
- DAC clock generation. Each DAC has an independent clock signal that the DACC generates from the system clock (SCLK) according to the configuration in DACC registers. Optionally, this clock can be gated (only active while data is driven). For more information, see Clock Modes.

- Frame sync generation. Each DAC has an independent frame sync signal that controls the data transfer rate. Data and the gated clock are only driven while the frame sync is asserted. For more information, see Frame Sync Modes.
- Data output (shift registers). The last stage of each DAC FIFO is a serial shift register, which the DACC uses to transfer data from the FIFO to the DAC.
- Status and error interrupts. The DACC has an error interrupt request output, and the DACs each have a status interrupt output. These outputs go to system NVIC/SEC units for handling.

DACC Block Diagram

The *DACC Block Diagram* figure shows the functional blocks within the DACC.

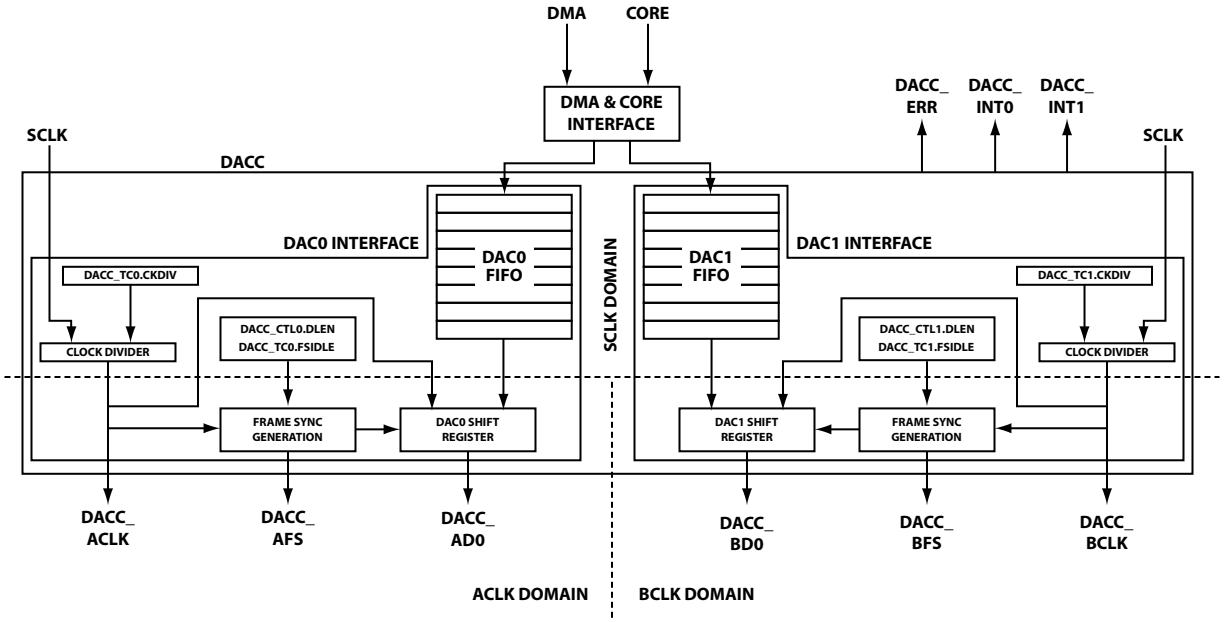


Figure 31-2: DACC Block Diagram

For more information about the DACC signals, see DACC Signal Descriptions.

For more information about the blocks in the diagram, see DACC Architectural Concepts.

Core and DMA Interfaces

The DACC has a 32-bit core interface through which the core programs the DACC control registers and reads the DACC status registers. The DACC also uses this interface to update the DAC FIFOs in core mode. It updates the DAC FIFO (with data for digital-to-analog conversion) by writing data to the data register of the DAC (for example, `DACC_DAT0`).

To minimize the core overhead, the DACC provides a 32-bit DMA interface for updating the DAC FIFOs. The DMA data interface width can be programmed to 16-bit or 32-bit width (for example, using the `DACC_CTL0.DMAW` bit).

The DMA interface supports an optional circular buffering mode for updating the DAC FIFOs. When circular buffering is enabled (for example, using the DACC_CTL0.CBUFEN bit), the DMA continuously transmits the data without core intervention. An interrupt can be enabled (for example, using the DACC_CTL0.CINTEN bit) to signal the core at the end of each circular buffer.

Pending Data FIFO

The DACC provides a separate, 4-deep FIFO for each DAC interface. It updates the FIFOs in core or DMA modes, according to the mode settings in the DACC control register (for example, DACC_CTL0).

The pending data FIFO helps reduce the data request rate to the core or DMA and reduces the possibility of underflows.

The DAC shift register (at the last FIFO stage) reads the data from the DACC FIFO and serially shifts out the data. It shifts the data based on active edges of the DAC clock (for example, DACC_ACLK) when the DAC sync pulse (for example, DACC_AFS) is active.

An *underflow* at the DAC interface occurs when the DAC frame sync is about to go active, but the FIFO of the DAC has no new data for transmission. When this situation occurs, the DACC does not generate the frame sync pulse. Optionally, the DACC can signal this condition to the core with an error interrupt request.

DACC Operating Modes

The operating modes of the DACC include its configurable options for data length, FIFO update rate, DAC clock divider and polarity, frame sync idle time and polarity, and broadcast controls. For more information about these options, see the following sections:

- Data Transfer Modes
- Data Length and Update Options
- Clock Modes
- Frame Sync Modes
- Broadcast Control Option

Data Transfer Modes

The DACC transfers data to the DACs through the corresponding FIFO. To update these FIFOs with data for transmission to the corresponding DAC, the DACC supports:

- Core-driven single word transfers
- DMA-driven multiple words transfers

DMA transfers can be set up to transfer a configurable number of words from internal or external memory of the processor to the DACC FIFOs automatically, without core intervention. Core-driven transfers can use DACC interrupt requests to signal the processor core to perform single word transfers to the DACC FIFOs.

Core-Driven Data Write Mode

The DACC provides a DAC data register (for example, DACC_DAT0), for accessing the top entry of the 4-deep FIFO in core mode. The DMA enable bit (for example, DACC_CTL0.DMAEN) selects the data transfer mode of DACC operation as either DMA mode or core mode. Typically, the core checks the FIFO status bits (for example, DACC_STAT.FSTAT0) or relies on related interrupt status before writing into the DAC data register. This register must be written only when there is space available in the DAC FIFO. If the core attempts a write into the data register when the DAC FIFO is full, the DACC ignores the core write transaction.

The DACC provides a way to ease status-checking for core write operations. It uses the core-write complete status bit (for example DACC_ISTAT.CINT0) to indicate whether the DAC FIFO has space available to accommodate new data writes by the core. Optionally, an interrupt request for the data request condition can be unmasked (enabled) in the DACC_IMSK register. Then, core writes to the DAC FIFO can be managed with a processor based interrupt service routine.

NOTE: The DAC interface considers only the lower 16 bits of DAC data FIFO register as DAC data. It ignores the upper 16 bits. Reads of this register return the top entry of the FIFO.

DMA-Driven Data Write Mode

The DACC provides a 32-bit DMA interface to minimize the core overhead for updating the DAC data FIFOs. The DMA enable bit (for example, DACC_CTL0.DMAEN) selects the data transfer mode of DACC operation as either DMA mode or core mode. The DMA data interface width bit (for example, DACC_CTL0.DMAW) selects whether the interface is 16 bits or 32 bits wide.

NOTE: When the DAC is in DMA mode, core writes to the DAC data register do not result in the data being written into the DACC FIFO, and these writes are ignored.

Program the DMA work-units with the following registers:

- DAC DMA base pointer register (for example, DACC_BPTR0). This register contains the memory address of the base pointer for starting a DMA read transfer.
- DAC DMA modify register (for example, DACC_MOD0). This register contains the address-increment applied between each DMA read from memory, starting from the base pointer.
- DAC DMA count register (for example, DACC_CNT0). This register holds the DMA count in a DMA work-unit.

The DMA unit supports *linear buffering* or *circular buffering* operating modes. The circular buffer enable bit (for example, DACC_CTL0.CBUFEN) selects between these modes of operation.

Linear Buffering

In *linear buffering*, the data transfer starts from the memory location pointed to by the base pointer register. With each data fetch (16 bits or 32 bits), the address is incremented by the number of bytes specified in the modifier register to calculate the next fetch address. The DMA fetches data the number of times programmed

in the count register and transmits the data to the DAC. After all the data is transmitted, an optional interrupt status bit is set.

Circular Buffering

In *circular buffering*, the data transfer starts from the memory location pointed to by the base pointer register. With each data fetch (16 bits or 32 bits), the address is incremented by the number of bytes specified in the modifier register to calculate the next fetch address. The DMA unit provides a looping back mechanism to the base pointer address. After the number of data fetches pointed by the count register, the next fetch is performed from the base pointer location, instead of incrementing the fetch address by the modifier.

When interrupts are disabled in circular buffer mode (for example, using the `DACC_CTL0.CINTEN` bit), the data fetches of the next circular buffer are initiated. The data fetch occurs even if the previous circular buffer has data-reads that are pending on the return from memory. If interrupts are enabled, the next circular buffer data request is made only after the reads of the previous circular buffer return. The interrupt request is signaled after the reads of a circular buffer return from memory.

Data Length and Update Options

The data length of the DAC interface can be configured, allowing the DACC to interface with serial DACs with 1-bit to 16-bit data length. The DACC can send data to the DAC in either LSB-first format or MSB-first format, based on the DACC LSB-first bit setting.

The DACC uses the DAC data length bit field (for example, `DACC_CTL0.DLEN`) to choose up to a 16-bit data length for the DAC interface. This value affects the number of DAC clock cycles for each data word and affects the rate for the DAC FIFO update.

The period of the DAC FIFO update is configured using the DAC frame sync signal (for example, `DACC_AFS`) for frame sync minimum idle time (for example, `DACC_TC0.FSIDLE`). The DAC clock divisor ratio (for example, `DACC_TC0.CKDIV`) also affects the period of the DAC FIFO update.

The following calculation describes the combination of these factors for the DAC FIFO update period (example DAC0 sync period) in DAC clock cycles:

DAC0 sync period = (`DACC_TC0.FSIDLE` + 1) + (`DACC_CTL0.DLEN`)

Applying this formula to calculate the required frame sync idle assumes the following:

- DAC data length is 12 bits
- System clock (SCLK) frequency is 100 MHz
- DAC0 clock frequency is 50 MHz
- DAC update frequency is 50 KSPs (kilo-samples-per-second)

First, calculate the DAC clock divisor (for DAC0) as:

`DACC_TC0.CKDIV` = (SCLK frequency ÷ DAC0 clock frequency) – 1

= (100 MHz / 50 MHz) – 1 = 1

Then, calculate the needed DAC frame sync idle time (for DAC0) as:

DACC_TC0.FSIDLE = (DAC clock frequency ÷ DAC update frequency) - DACC_CTL0.DLEN – 1

= (50 MHz / 50 KSPs) – 12 – 1 = (1000 – 12 – 1) = 987 = 0x3DB

Clock Modes

The DACC provides a clock signal to communicate with the interfaced DAC. The DAC clock also is available in gated format to ensure excellent noise immunity during the conversion process. (For example, the clock is active only during the time when data is driven to the DAC)

Clock Frequency Programming

The DAC clock is internally generated from system clock of processor. The DACC_TC0.CKDIV bit field specifies the divider to generate DAC0 clock signal from SCLK.

```
DACC_ACLK = (SCLK) ÷ (DACC_TC0.CKDIV + 1)
```

Alternatively, the clock divisor value for the required DAC0 clock frequency is calculated as:

```
DACC_TC0.CKDIV = (SCLK ÷ DACC_ACLK) – 1
```

Clock Polarity and Gated Clock Programming

The active DAC clock edge (polarity) can be selected as rising edge or falling edge. The DACC sync and data signals (for example,DACC_AFS and DACC_AD0) are driven on the active edges of DAC clock. Configure the clock polarity bit (for example, DACC_CTL0.CKPOL) based on the clock edge the DAC samples the DAC sync and data signals. If the DAC clock is active low (for example, DACC_CTL0.CKPOL =0), the DAC sync and data signals are driven from the falling edge of the DAC clock (for example, DACC_ACLK).

Consider an example where the DAC sync signal is active low (for example, DACC_CTL0.SYNCPOL =0) and the DAC clock signal driving edge polarity is selected as rising edge (for example, DACC_CTL0.CKPOL =1). In this example, the value of DAC data length is 12 (for example, DACC_CTL0.DLEN =12), and the value of DAC sync idle is 4 (for example, DACC_TC0.FSIDLE =4). Refer to the ***DAC Clock Programming (DACC_CTLx.GCKEN =0)*** figure.

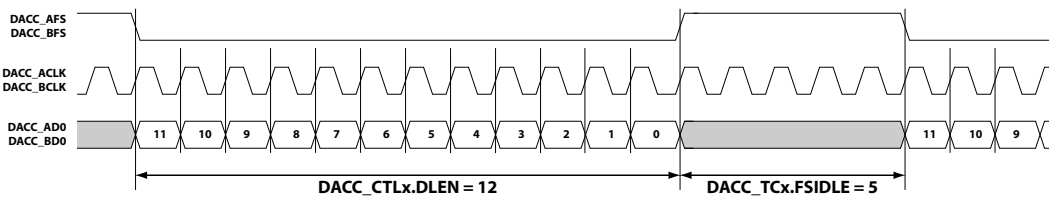


Figure 31-3: DAC Clock Programming (DACC_CTLx.GCKEN =0)

The DACC also provides the DAC clock in gated format, which is active only while the frame sync signal is asserted and valid DAC data is driven. This operation is enabled with the gated clock enable bit (for example,

DACC_CTL0.GCKEN). The ***DAC Clock Programming (DACC_CTLx.GCKEN =1)*** figure shows the DAC serial timing for gated clock mode.

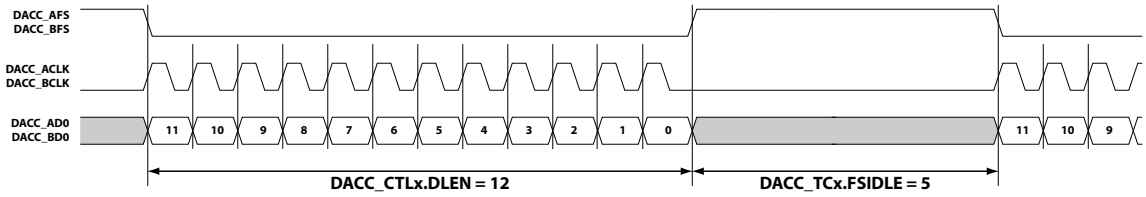


Figure 31-4: DAC Clock Programming (DACC_CTLx.GCKEN =1)

Frame Sync Modes

The DACC provides a frame select signal (for example, DACC_AFS) to select the DAC for communication. The signal (optionally with its edges) indicates the start of data transmission to the DAC (and data conversion). The frame select is asserted while data is driven to the DAC.

The frame sync signal provided to the DAC can be configured as an active-high signal or an active-low signal, based on the protocol the interfaced ADC supports. This sensitivity is configurable using the chip select polarity bit (for example, DACC_CTL0.SYNCPOL).

Between data write sequences to the DAC, when a valid data word is not driven to the DAC, the DAC sync lines must be driven to an inactive level (the lines are driven to a high or a low level). This action meets the DAC requirement for the frame period. The value is selected with the frame sync idle bit field (for example, DACC_TC0.FSIDLE).

For more information about frame sync idle programming, see Data Length and Update Options.

Broadcast Control Option

The DACC controls multiple DACs. To support a synchronized enable time for the DACs (if necessary), the DACC includes a broadcast control register (DACC_BCST_CTL), which provides broadcast write access to the control registers of the DAC. A memory-mapped register write to the DACC_BCST_CTL register writes the value to the DACs' control registers. Reading the DACC_BCST_CTL register returns 0x0000 0000.

DACC Event Control

The DACC can signal the core about its state and various error conditions that occur during its operation, by providing status and error bits through different registers. These conditions include:

- Interrupt status related to data FIFO operations in core mode and DMA mode.
- Error status related to DACC operations.
- Pending data status (which do not generate interrupt requests) related to data FIFO operations in core mode and DMA mode.

Interrupt Status

The DACC can generate interrupt requests to signal operation status for individual DACs or error status for the DACC to the system event controller (SEC).

The DACC provides a data interrupt channel for each DAC FIFO (for example, DACC_DAC0). In core mode, this interrupt indicates the request to fill the DACC FIFO. In DMA mode, this interrupt indicates the completion of a DMA work unit. The DACC uses the conditions flagged in the interrupt status register (DACC_ISTAT) to generate these interrupts.

The DACC error interrupt request signal (DACC_ERR) indicates error conditions related to DAC controller. The DACC uses the conditions flagged in the error status register (DACC_ERRSTAT) to generate these interrupt requests.

DAC DMA Mode Interrupt

In DMA mode, this interrupt request indicates the status of DMA work-unit. When in linear DMA mode, the DACC can generate this interrupt when a DMA work unit programmed for a DAC interface completes, and all data has transmitted to DAC. When in circular buffer DMA mode (with the circular buffer interrupt enabled), the DACC can generate this interrupt request when all read data in a buffer return from memory. The status bit related to this interrupt is sticky. A W1C operation clears the bit.

DAC Core Mode Interrupt

This interrupt request indicates the status of DAC FIFO update in core mode, when the DAC data FIFO has space to accommodate new core data writes. The status bit related to this interrupt is a read-only status bit and is cleared when the FIFO gets full.

Error Status

The DACC signals error conditions through the error status register (DACC_ERRSTAT) for data underflow errors and memory access errors. The DACC can use these conditions to generate an error interrupt request (DACC_ERR) when they are unmasked (enabled) in the error mask register (DACC_ERRMSK).

Clear all of these sticky error status bits with a W1C (write-1-to-clear) operation. Even if the DACC reports an error, normal operation of the DACC continues. Due to address alignment errors, the lower (violating) bits of the base pointer address or modifier registers are ignored. The data at the aligned address is written into the DAC FIFO. In the underflow error case, transmission of the DAC data is delayed until the DACC FIFO receives any data from the selected core or DMA interface.

Data Underflow Error

When an underflow occurs in a DAC FIFO, the related underflow error bit (for example, DACC_ERRSTAT.DUVF0) indicates the condition. For more information about underflow, see Pending Data FIFO.

Memory Access Error

When an address alignment error condition occurs while DACC DMA is enabled, the related address alignment error bit (for example, DACC_ERRMSK_SET.AER0) indicates the condition. An address alignment error occurs when:

- The LSB bit of the base pointer address register or modifier register is non-zero in 16-bit DMA mode.
- The lower 2 bits of the base pointer address register or modifier register are non-zero in 32-bit DMA mode.

When a read response error returns from memory for transfers to a DAC, the related memory access error bit (for example, DACC_ERRSTAT.MER0) indicates the condition. A read response error occurs when a DAC DMA attempts to access the reserved memory space of processor.

Pending Status

In addition to interrupt status and error status, the DACC provides bits to provide information about DAC FIFO pending data status.

DMA Data Pending Status

The DMA pending status bit (for example,DACC_STAT.DPND0) indicates that DMA read address requests have been made and the related read data is pending DAC interface reception. Before re-enabling the DMA of a DAC interface, check the corresponding pending status bit (for no data still pending).

DAC Data FIFO Status

The FIFO status bit (for example, DACC_STAT.FSTAT0) indicates the number of 16-bit data in the DAC data FIFO that await transmission to the DAC. The status increments when DMA mode transfers or core mode writes fill the DAC FIFO. The status decrements when a data transmission starts on the DAC interface.

DACC Programming Concepts

There are general programming concepts for using the DACC when it is present on any processor.

NOTE: There can also be processor-specific guidelines for programming the DACC.

The DAC does not work if the AFE is not "OK". See the ADCC chapter for more information

Take care when disabling or enabling the DACC in DMA modes. When disabling the DAC controller (for example, DACC_CTL0.EN =0), ensure that pending DMA transfers finish before re-enabling the DACC or re-enabling DMA (for example, DACC_CTL0.DMAEN =1). This status can be observed from the related DMA pending bit (for example, DACC_STAT.DPND0) of the DAC. When disabling the DACC or DMA in the control register, retain the programming of all other DMA-related controls (for example, DACC_CTL0.CBUFEN). Change all DMA-related programming in DACC registers only after the corresponding DMA pending status bit is cleared (DMA completed).

The memory mapped register status and counter-values are retained on disabling the DACC (helpful for debug). These registers (such as counters and other status) are cleared on a 0-to-1 transition of the enable bit. Error status is not cleared with this transition of 0-to-1. Clear the errors using a W1C operation.

If the DAC controller is disabled while a transaction on the DAC interface is in-progress, the transaction can be dropped midway. Disable the DACC after a DMA work unit completes in DMA mode, or disable the DACC when all data updates are complete in non-DMA mode.

After a DMA work unit finishes (interrupt given), initiate a new work unit using the following procedure:

- Disable both the DACC enable and DMA enable bits (for example, `DACC_CTL0.EN` =0 and `DACC_CTL0.DMAEN` =0).
- Configure the DMA count and DMA modifier (for example, `DACC_CNT0` and `DACC_MOD0`), then enable DMA operation (for example, `DACC_CTL0.DMAEN` =1).
- After the FIFO gets data from the DMA (for example, `DACC_STAT.DPND0` =1), enable the DACC (for example, `DACC_CTL0.EN` =1).

DACC Programming Model

The programming model for the DACC includes operation flow for core mode transfers and DMA mode transfers. Use the following steps in these flows.

Core Mode Operation Flow

Use this procedure to configure the DACC for core mode data transfers.

1. Set up the DACC, writing to its memory-mapped registers in core mode (for example, `DACC_CTL0.DMAEN` =0 and `DACC_CTL0.EN` =0).
2. Enable DACC operation, writing the `DACC_CTL0.EN` bit (=1) in core mode.

   The DACC clock starts and sync counters begin. The DACC sends sync pulses only if new data is present. With each sync pulse, the DACC updates the DAC with new data.
3. Write DAC data to the DACC FIFO, using register writes in core mode.
4. When DAC updates complete, disable DACC operation (if desired), writing to the `DACC_CTL0.EN` bit (=0) in core mode.

DMA Mode Operation Flow

Use this procedure to configure the DACC for DMA mode data transfers.

1. Set up the DACC, writing to its memory-mapped registers in DMA mode (`DACC_CTL0.DMAEN` =1 and `DACC_CTL0.EN` =0)

   DAC DMA starts reading data from memory.

2. Poll the FIFO status in DAC_STAT for FIFO >0 in core mode.
3. Enable DACC operation, writing the bit (=1) and DACC_CTL0.DMAEN (=1) in DMA mode.

   The DACC clock starts and sync counters begin. The DACC sends sync pulses only if new data is present. With each sync pulse, the DACC updates the DAC with new data during the selected DMA word count.
4. When DAC updates complete, disable DACC operation (if desired), writing to the DACC_CTL0.EN and DACC_CTL0.DMAEN bits (both =0) in core mode.

DAC Selection

Selecting the DAC is via setting up a programming sequence through AFE. For more information about this, see AFE Register Access Programming Model.

```
/*
DAC_ID: 0 - 12 bit DAC
        1 - Over Voltage DAC
        2 - Under Voltage DAC

#define BITM_USER_CONFIG_REG_USERCFG_DAC_SEL 0x0000C000
Enables Selection of DAC. 00 ->12 Bit DAC, 01->DAC_A, 10 ->DAC_B, 11 -> Not Used
*/

#define CTLWD_DACSEL    0xC0      //Bit[7] - RESERVED = 1
                                  //Bit[6] - WNR = 1
                                  //Bit[5] - RESERVED = 0
                                  //Bit[4-1] - REGADDRESS = 0000
                                  //Bit[0] - RESERVED = 0

void Select_DAC(unsigned char DAC_ID)
{

/*read user register and OR the value if it is required to retain value
   other bits of register*/
*pREG_ADCC0_ADCRW0=
        (((CTLWD_DACSEL  << BITP_ADCC_ADCRW0_CTLW) & BITM_ADCC_ADCRW0_CTLW ) |
        (( (DAC_ID << BITP_USER_CONFIG_REG_USERCFG_DAC_SEL) <<
         BITP_ADCC_ADCRW0_DAT) & BITM_ADCC_ADCRW0_DAT ));

//wait for register write to be complete
while ((*pREG_ADCC0_ADCRW0 & BITM_ADCC_ADCRW0_PND));

}
```

CM41X_M4 DACC Register Descriptions

DAC Controller (DACC) contains the following registers.

Table 31-4: CM41X_M4 DACC Register List

Name	Description
DACC_BCST_CTL	Broadcast (Write) Control Register
DACC_BPTR0	Base Pointer 0 Register
DACC_CNT0	Count 0 Register
DACC_CNTCUR0	Current Count 0 Register
DACC_CTL0	Control 0 Register
DACC_DAT0	Data FIFO 0 Register
DACC_ERRMSK	Error Mask Register
DACC_ERRMSK_CLR	Error Mask Clear Register
DACC_ERRMSK_SET	Error Mask Set Register
DACC_ERRSTAT	Error Status Register
DACC_IMSK	Interrupt Mask Register
DACC_IMSK_CLR	Interrupt Mask Clear Register
DACC_IMSK_SET	Interrupt Mask Set Register
DACC_ISTAT	Interrupt Status Register
DACC_MOD0	Modify 0 Register
DACC_STAT	Status Register
DACC_TC0	Timing Control 0 Register

Broadcast (Write) Control Register

The DACC_BCST_CTL register provides a broadcast write access to the DACC control register(s). A memory mapped register write to DACC_BCST_CTL writes the data to both control register. A memory mapped register read of the DACC_BCST_CTL register returns 0x0000.

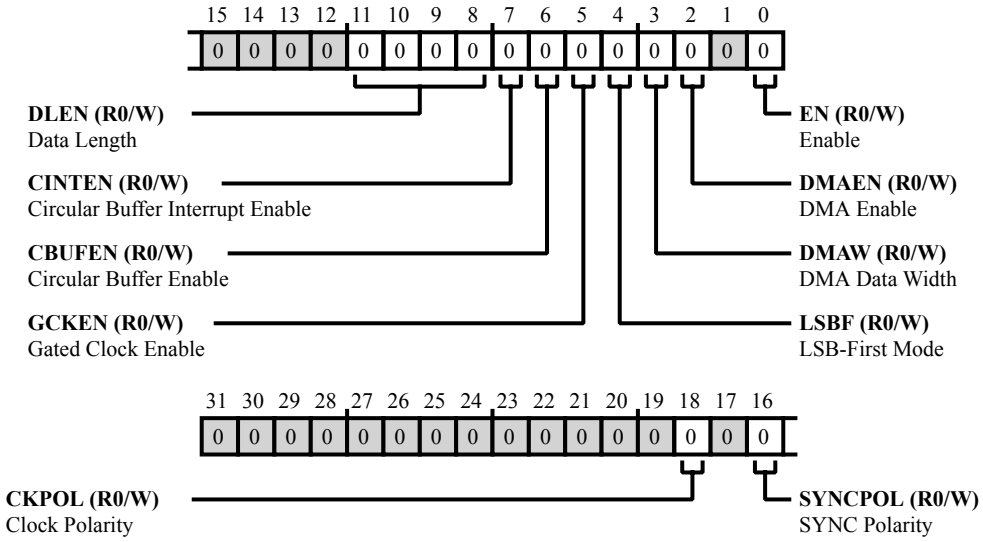


Figure 31-5: DACC_BCST_CTL Register Diagram

Table 31-5: DACC_BCST_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R0/W)	CKPOL	Clock Polarity. The DACC_BCST_CTL.CKPOL broadcast bit selects the polarity of the DAC0 and DAC1 interface clock signals (DACC_ACLK and DACC_BCLK pins) on which to drive the DAC0 output (DACC_AD0 pin) and/or DAC1 output (DACC_BD0 pin).
16 (R0/W)	SYNCPOL	SYNC Polarity. The DACC_BCST_CTL.SYNCPOL broadcast bit selects the polarity for the DAC0 and DAC1 interfaces' sync signals (DACC_AFS and DACC_BFS pins).
11:8 (R0/W)	DLEN	Data Length. The DACC_BCST_CTL.DLEN broadcast bits choose the data length (in bits) for the DAC0 and DAC1 interfaces. A value =0 for this field implies a data length of 16 bits. For data lengths less that 16 bits, use LSB-aligned data and zero fill unused bits.
7 (R0/W)	CINTEN	Circular Buffer Interrupt Enable. The DACC_BCST_CTL.CINTEN broadcast bit enables generation of the DACC_DAC0 and/or DACC_DAC1 interrupt requests at the end of each related circular buffer. This bit is only valid if the DACC_BCST_CTL.DMAEN bit =1 and the DACC_BCST_CTL.CBUFEN bit =1.

Table 31-5: DACC_BCST_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R0/W)	CBUFEN	Circular Buffer Enable. The DACC_BCST_CTL.CBUFEN broadcast bit enables circular buffer DMA mode for the DAC0 and DAC1 interfaces. This bit is only valid if the DACC_BCST_CTL.DMAEN bit =1.
5 (R0/W)	GCKEN	Gated Clock Enable. The DACC_BCST_CTL.GCKEN broadcast bit enables gated clock mode for the DAC0 and DAC1 interface clocks (DACC_ACLK and DACC_BCLK pins). When enabled, the related clock toggles only when valid data is driven on the DACC_AD0 pin and/or the DACC_BD0 pin. When disabled, the clocks are free running.
4 (R0/W)	LSBF	LSB-First Mode. The DACC_BCST_CTL.LSBF broadcast bit selects between LSB-first mode or MSB-first mode transfers for the DAC0 and DAC1 interfaces.
3 (R0/W)	DMAW	DMA Data Width. The DACC_BCST_CTL.DMAW broadcast bit selects the DMA data width for the DAC0 and DAC1 interfaces. This bit is only valid if the DACC_BCST_CTL.DMAEN bit =1.
2 (R0/W)	DMAEN	DMA Enable. The DACC_BCST_CTL.DMAEN broadcast bit enables DMA transfers for the DAC0 and DAC1 interfaces.
0 (R0/W)	EN	Enable. The DACC_BCST_CTL.EN broadcast bit enables operations for the DAC0 and DAC1 interfaces.

Base Pointer 0 Register

The DACC_BPTR0 register provides the base pointer (address) used by DMA read operations for the DAC0 interface. The value of base address (pointer) in the DACC_BPTR0 register at the start of the DMA work unit (start of frame) corresponds to one of the following transfers:

- The first transfer after DACC is enabled in DMA mode
- The first transfer after a wrap around occurs in circular buffering mode

The data for the first transfer is read from memory, starting at the address indicated with the value of the base address. Further DMA data is read from memory at addresses incremented by DACC_MOD0 bytes.

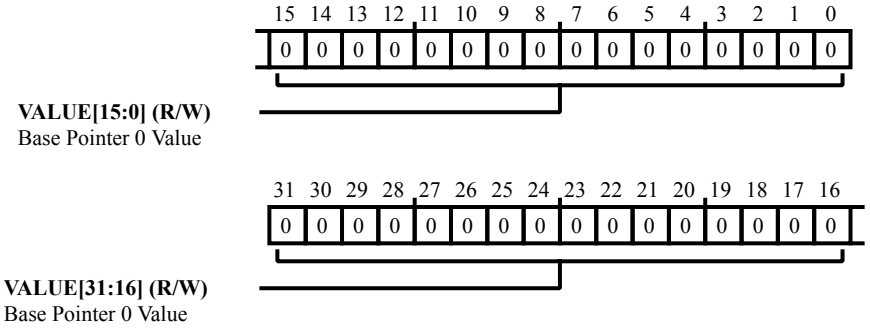


Figure 31-6: DACC_BPTR0 Register Diagram

Table 31-6: DACC_BPTR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Base Pointer 0 Value. The DACC_BPTR0.VALUE bits hold the base pointer (address) for the first DMA transfer.

Count 0 Register

In linear DMA mode, the DACC_CNT0 register holds the transfer count of a DMA work-unit for DAC0. The DMA fetches the indicated number of read data and generates an interrupt request after transmitting all data to the DAC. After the data corresponding to the count is transmitted (in linear DMA mode), the DACC does not send further syncs to the DAC.

In circular buffer mode, this register holds the count of DMA after which wrap around to the base pointer address ((DACC_BPTR0)) occurs. The DMA fetches the indicated number of read data and (optionally) generates an interrupt request after receiving all data corresponding to one set of DACC_CNT0 counts (one circular buffer) from memory.

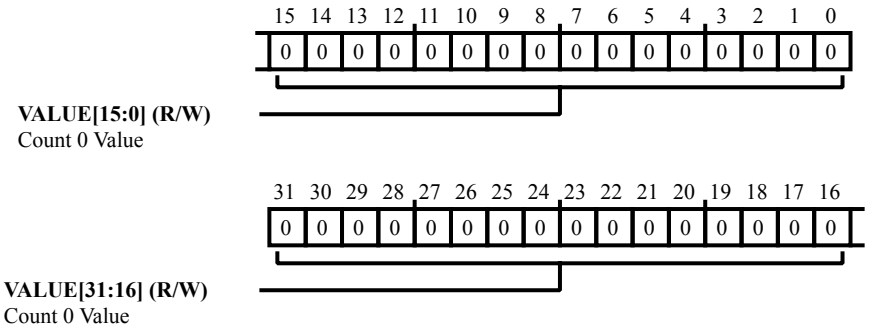


Figure 31-7: DACC_CNT0 Register Diagram

Table 31-7: DACC_CNT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Count 0 Value. The DACC_CNT0.VALUE bits select the transfer count of a DMA work-unit for DAC0.

Current Count 0 Register

The DACC_CNTCUR0 register holds the current count of the DMA work-unit of the DAC0 interface. This register is loaded from the DACC_CNT0 register when either of the following occur:

- The DACC is enabled in DMA mode
- A wrap around occurs in circular buffering mode

The count decrements with each DMA read from memory.

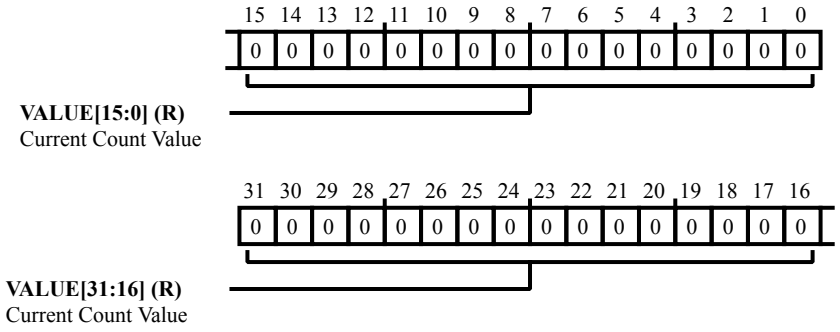


Figure 31-8: DACC_CNTCUR0 Register Diagram

Table 31-8: DACC_CNTCUR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Current Count Value. The DACC_CNTCUR0.VALUE bits hold the current transfer count of a DMA work-unit for DAC0.

Control 0 Register

The DACC_CTL0 register controls the DAC0 interface.

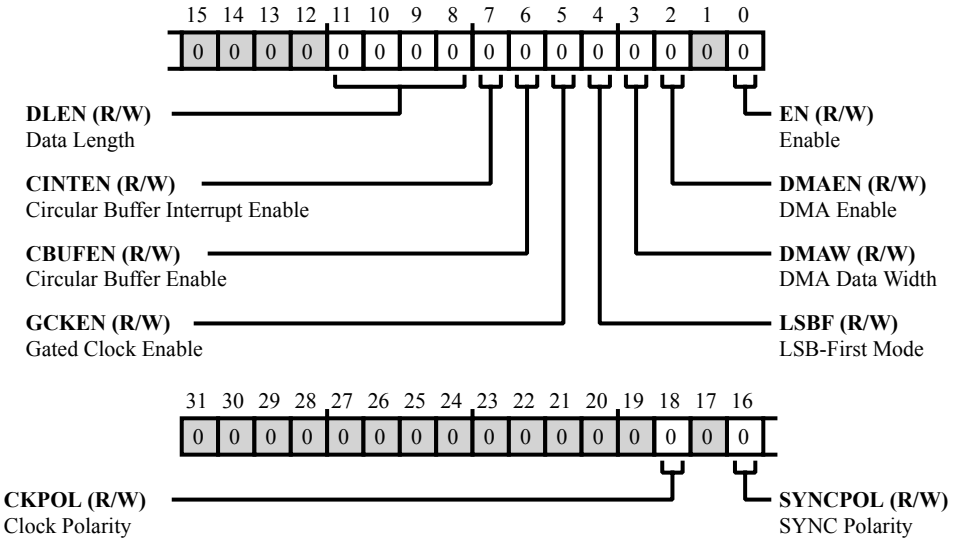


Figure 31-9: DACC_CTL0 Register Diagram

Table 31-9: DACC_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	CKPOL	Clock Polarity. The DACC_CTL0.CKPOL bit selects the polarity of the DAC0 interface clock signal (DACC_ACLK pin) on which to drive the DAC0 output (DACC_AD0 pin).	
		0	Drive on Clock Falling Edge
		1	Drive on Clock Rising Edge
16 (R/W)	SYNCPOL	SYNC Polarity. The DACC_CTL0.SYNCPOL bit selects the polarity for the DAC0 interface sync signal (DACC_AFS pin).	
		0	Active Low
		1	Active High
11:8 (R/W)	DLEN	Data Length. The DACC_CTL0.DLEN bits choose the data length (in bits) for the DAC0 interface. A value =0 for this field implies a data length of 16 bits. For data lengths less that 16 bits, use LSB-aligned data and zero fill unused bits.	

Table 31-9: DACC_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CINTEN	Circular Buffer Interrupt Enable. The DACC_CTL0.CINTEN bit enables generation of the DACC_DAC0 interrupt request at the end of each circular buffer. This bit is only valid if the DACC_CTL0.DMAEN bit =1 and the DACC_CTL0.CBUFEN bit =1.	
		0	Disable
		1	Enable
6 (R/W)	CBUFEN	Circular Buffer Enable. The DACC_CTL0.CBUFEN bit enables circular buffer DMA mode for the DAC0 interface. This bit is only valid if the DACC_CTL0.DMAEN bit =1.	
		0	Disable
		1	Enable
5 (R/W)	GCKEN	Gated Clock Enable. The DACC_CTL0.GCKEN bit enables gated clock mode for the DAC0 interface clock (DACC_ACLK pin). When enabled, the clock toggles only when valid data is driven on the DACC_AD0 pin. When disabled, the clock is free running.	
		0	Disable Gated Clock Mode
		1	Enable Gated Clock Mode
4 (R/W)	LSBF	LSB-First Mode. The DACC_CTL0.LSBF bit selects between LSB-first mode or MSB-first mode transfers for the DAC0 interface.	
		0	MSB-First Mode
		1	LSB-First Mode
3 (R/W)	DMAW	DMA Data Width. The DACC_CTL0.DMAW bit selects the DMA data width for the DAC0 interface. This bit is only valid if the DACC_CTL0.DMAEN bit =1.	
		0	16-Bit DMA Data
		1	32-Bit DMA Data
2 (R/W)	DMAEN	DMA Enable. The DACC_CTL0.DMAEN bit enables DMA transfers for the DAC0 interface. Please note that this bit needs to be cleared before updating the other DMA registers (BPTR,MOD,CNT) and then re-enabled before those changes take effect.	
		0	Disable
		1	Enable

Table 31-9: DACC_CTL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Enable. The `DACC_CTL0.EN` bit enables operations for the DAC0 interface.	
		0	Disable
		1	Enable

Data FIFO 0 Register

The DACC_DAT0 register provides a memory mapped register location for the DAC0 data FIFO. This location is used in non-DMA mode, permitting the core writes to transmit data to the DAC. Only the lower 16 bits are considered as DAC data; the upper 16 bits are ignored.

The core should only write this register when no DAC DMA is in progress (DACC_ISTAT.DINT0 =1). If the core attempts a write into this register during an active DAC DMA (DACC_ISTAT.DINT0 =0), the DACC ignores the write transaction. Reads of DACC_DAT0 return the value of the top entry of the DAC0 FIFO.

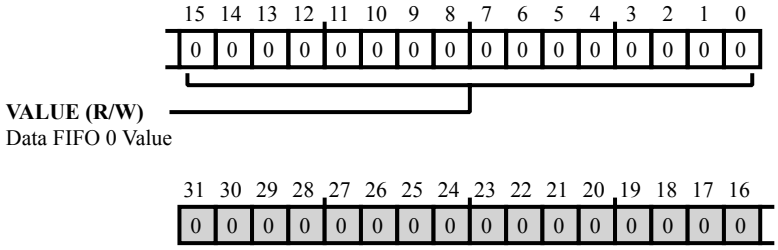


Figure 31-10: DACC_DAT0 Register Diagram

Table 31-10: DACC_DAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Data FIFO 0 Value. The DACC_DAT0.VALUE bits provides a memory mapped location for the DAC0 data FIFO. Core writes to these bits (when no DMA is active) are transmitted to the DAC. Reads of these bits return the value of the top entry of the DAC0 FIFO.

Error Mask Register

The DACC_ERRMSK register masks (disables) or unmasks (enables) reporting of DAC related errors.

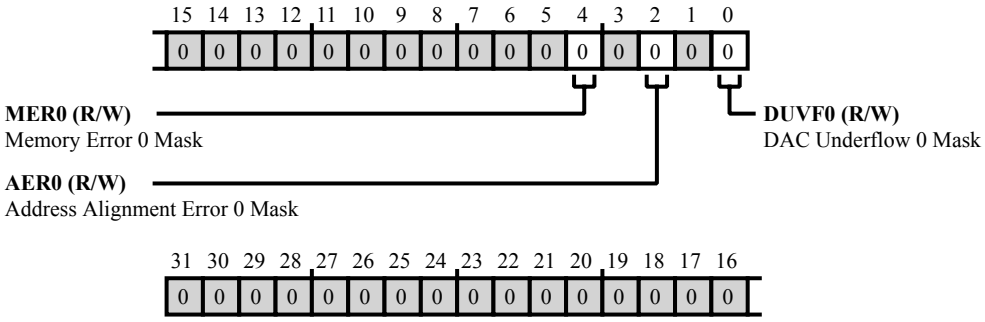


Figure 31-11: DACC_ERRMSK Register Diagram

Table 31-11: DACC_ERRMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	MER0	Memory Error 0 Mask. The DACC_ERRMSK.MER0 bit masks (disables) generating an error interrupt request on a DAC0 memory error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
2 (R/W)	AER0	Address Alignment Error 0 Mask. The DACC_ERRMSK.AER0 bit masks (disables) generating an error interrupt request on a DAC0 address alignment error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)
0 (R/W)	DUVF0	DAC Underflow 0 Mask. The DACC_ERRMSK.DUVF0 bit masks (disables) generating an error interrupt request on a DAC0 underflow error.	
		0	Unmask (Enable Reporting)
		1	Mask (Disable Reporting)

Error Mask Clear Register

The DACC_ERRMSK_CLR register can be used to selectively clear bits in the DACC_ERRMSK register without affecting other bits in the register. Writing a 1 to any bit position in DACC_ERRMSK_CLR clears the corresponding bit in DACC_ERRMSK. Reading the DACC_ERRMSK_CLR register returns the data present in the DACC_ERRMSK register.

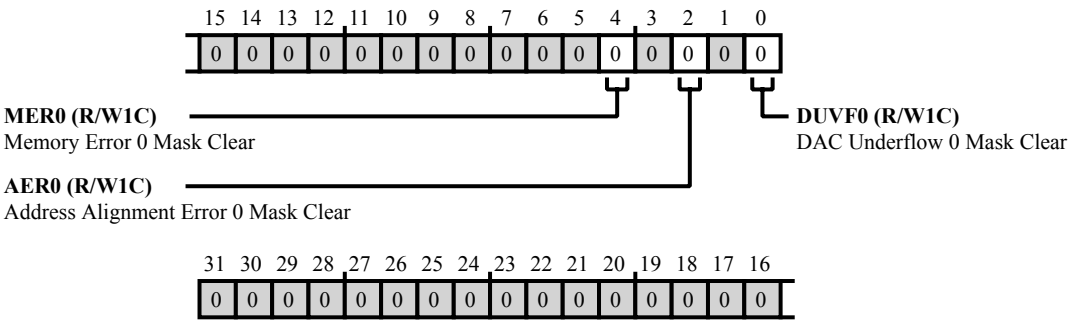


Figure 31-12: DACC_ERRMSK_CLR Register Diagram

Table 31-12: DACC_ERRMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W1C)	MER0	Memory Error 0 Mask Clear. Write 1 to DACC_ERRMSK_CLR.MER0 to clear the corresponding bit in DACC_ERRMSK.
2 (R/W1C)	AER0	Address Alignment Error 0 Mask Clear. Write 1 to DACC_ERRMSK_CLR.AER0 to clear the corresponding bit in DACC_ERRMSK.
0 (R/W1C)	DUVF0	DAC Underflow 0 Mask Clear. Write 1 to DACC_ERRMSK_CLR.DUVF0 to clear the corresponding bit in DACC_ERRMSK.

Error Mask Set Register

The DACC_ERRMSK_SET register can be used to selectively set bits in the DACC_ERRMSK register without affecting other bits in the register. Writing a 1 to any bit position in DACC_ERRMSK_SET sets the corresponding bit in DACC_ERRMSK. Reading the DACC_ERRMSK_SET register returns the data present in the DACC_ERRMSK register.

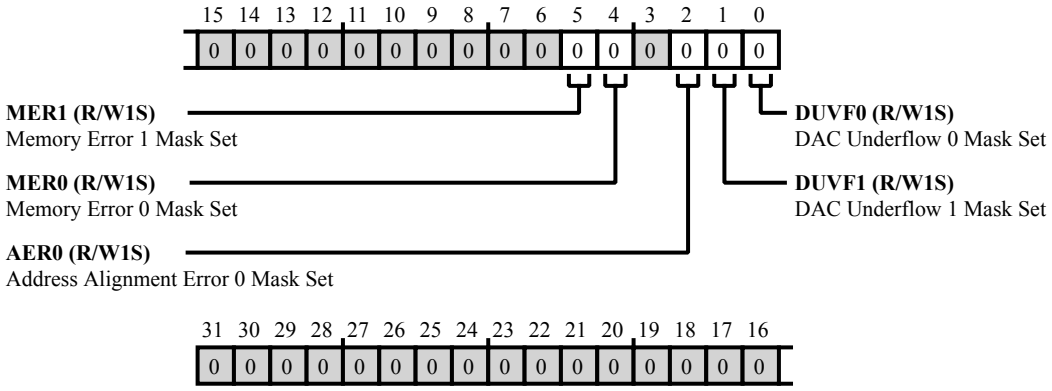


Figure 31-13: DACC_ERRMSK_SET Register Diagram

Table 31-13: DACC_ERRMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W1S)	MER1	Memory Error 1 Mask Set. Write 1 to DACC_ERRMSK_SET.MER1 to set the corresponding bit in DACC_ERRMSK.
4 (R/W1S)	MER0	Memory Error 0 Mask Set. Write 1 to DACC_ERRMSK_SET.MER0 to set the corresponding bit in DACC_ERRMSK.
2 (R/W1S)	AER0	Address Alignment Error 0 Mask Set. Write 1 to DACC_ERRMSK_SET.AER0 to set the corresponding bit in DACC_ERRMSK.
1 (R/W1S)	DUVF1	DAC Underflow 1 Mask Set. Write 1 to DACC_ERRMSK_SET.DUVF1 to set the corresponding bit in DACC_ERRMSK.
0 (R/W1S)	DUVF0	DAC Underflow 0 Mask Set. Write 1 to DACC_ERRMSK_SET.DUVF0 to set the corresponding bit in DACC_ERRMSK.

Error Status Register

The DACC_ERRSTAT register indicates error status for DACC operations. When any bit in this register is set, the DACC generates the DACC_ERR interrupt.

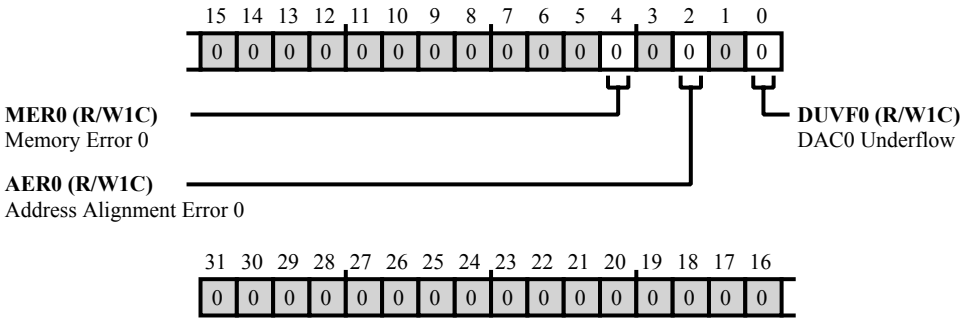


Figure 31-14: DACC_ERRSTAT Register Diagram

Table 31-14: DACC_ERRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W1C)	MER0	Memory Error 0. The DACC_ERRSTAT.MER0 bit indicates whether a memory error has occurred (erroneous read response received) during a DMA transfer for the DAC0 interface.	
		0	No Status
		1	Memory Error Occurred
2 (R/W1C)	AER0	Address Alignment Error 0. The DACC_ERRSTAT.AER0 bit indicates whether an address alignment error has occurred during a DMA transfer for the DAC0 interface. Recommended practice is to clear this bit (W1C) when enabling DMA with the DACC_CTL0.DMAEN bit.	
		0	No Status
		1	Alignment Error Occurred
0 (R/W1C)	DUVF0	DAC0 Underflow. The DACC_ERRSTAT.DUVF0 bit indicates whether a data underflow has occurred in the FIFO for the DAC0 interface (for example, no data was present in the FIFO when the DACC_AFS pin is about to be asserted).	
		0	No Status
		1	Underflow Occurred

Interrupt Mask Register

The DACC_IMSK register masks (disables) generation of DACC_DAC0 or DACC_DAC1 interrupt requests based on status in the DACC_ISTAT register.

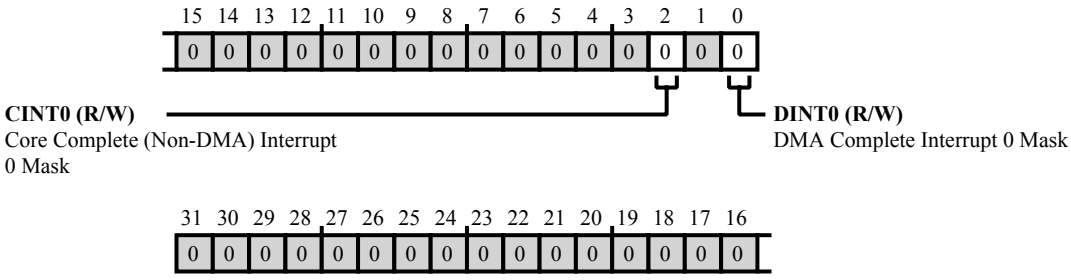


Figure 31-15: DACC_IMSK Register Diagram

Table 31-15: DACC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	CINT0	Core Complete (Non-DMA) Interrupt 0 Mask. The DACC_IMSK.CINT0 bits mask (disable) the DACC_ISTAT.CINT0 DMA complete interrupt requests for data transfer related to DAC0.	
		0	Unmask (Enable) DAC Interrupt
		1	Mask (Disable) DAC Interrupt
0 (R/W)	DINT0	DMA Complete Interrupt 0 Mask. The DACC_IMSK.DINT0 bits mask (disable) the DACC_ISTAT.DINT0 DMA complete interrupt requests for data transfer related to DAC0.	
		0	Unmask (Enable) DAC Interrupt
		1	Mask (Disable) DAC Interrupt

Interrupt Mask Clear Register

The DACC_IMSK_CLR register can be used to selectively clear bits in the DACC_IMSK register without affecting other bits in the register. Writing a 1 to any bit position in DACC_IMSK_CLR clears the corresponding bit in DACC_IMSK. Reading the DACC_IMSK_CLR register returns the data present in the DACC_IMSK register.

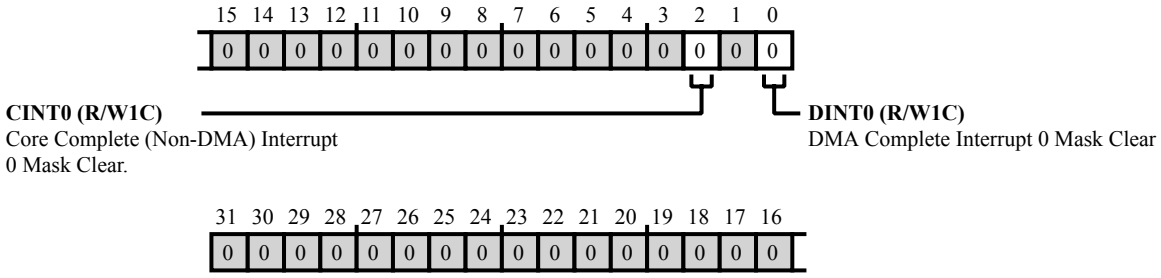


Figure 31-16: DACC_IMSK_CLR Register Diagram

Table 31-16: DACC_IMSK_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	CINT0	Core Complete (Non-DMA) Interrupt 0 Mask Clear.. The DACC_IMSK_CLR.CINT0 bits permit clearing the DACC_IMSK.CINT0 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_CLR.CINT0 bit to clear the DACC_IMSK.CINT0 bit.
0 (R/W1C)	DINT0	DMA Complete Interrupt 0 Mask Clear. The DACC_IMSK_CLR.DINT0 bits permit clearing the DACC_IMSK.DINT0 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_CLR.DINT0 bit to clear the DACC_IMSK.DINT0 bit.

Interrupt Mask Set Register

The DACC_IMSK_SET register can be used to selectively set bits in the DACC_IMSK register without affecting other bits in the register. Writing a 1 to any bit position in DACC_IMSK_SET sets the corresponding bit in DACC_IMSK. Reading the DACC_IMSK_SET register returns the data present in the DACC_IMSK register.

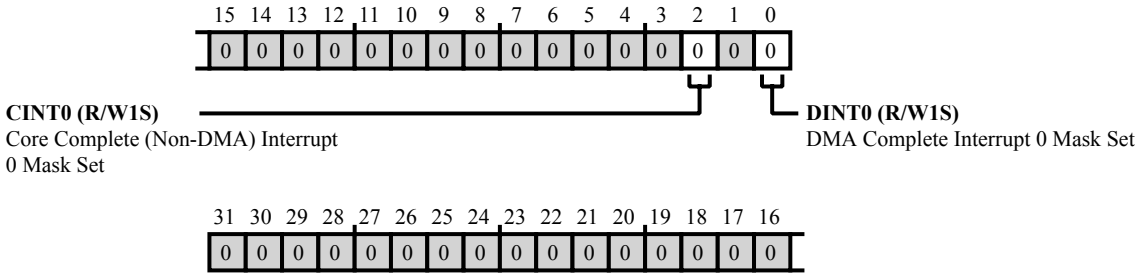


Figure 31-17: DACC_IMSK_SET Register Diagram

Table 31-17: DACC_IMSK_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1S)	CINT0	Core Complete (Non-DMA) Interrupt 0 Mask Set. The DACC_IMSK_SET.CINT0 bits permit setting the DACC_IMSK.CINT0 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_SET.CINT0 bit to set the DACC_IMSK.CINT0 bit.
0 (R/W1S)	DINT0	DMA Complete Interrupt 0 Mask Set. The DACC_IMSK_SET.DINT0 bits permit setting the DACC_IMSK.DINT0 bit without affecting other bits in the register. Write 1 to the DACC_IMSK_SET.DINT0 bit to set the DACC_IMSK.DINT0 bit.

Interrupt Status Register

The DACC_ISTAT register indicates DAC0 and DAC1 interrupt status. The DACC generates interrupt requests corresponding to DAC 0 on the DACC_DAC0 output and generates the interrupt requests corresponding to DAC1 on the DACC_DAC1 output.

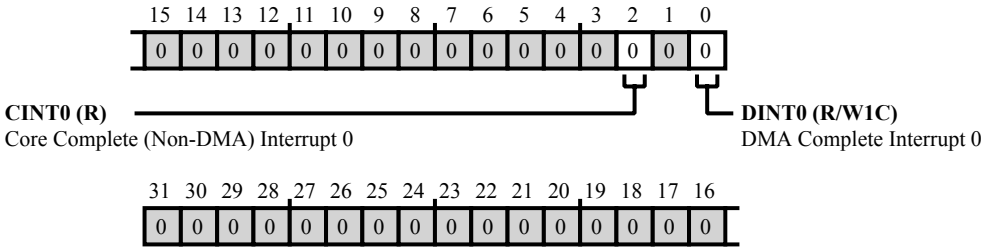


Figure 31-18: DACC_ISTAT Register Diagram

Table 31-18: DACC_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	CINT0	Core Complete (Non-DMA) Interrupt 0. The DACC_ISTAT.CINT0 bit indicates completion of a core write to the DAC0 interface, implying the DAC data FIFO again has space to accommodate new data writes by core. When cleared, this bit indicates the DAC FIFO is full.	
		0	No Status
		1	Core Complete (Non-DMA)
0 (R/W1C)	DINT0	DMA Complete Interrupt 0. The DACC_ISTAT.DINT0 bit indicates completion of the DMA work unit programmed for the DAC0 interface. When set in linear DMA mode, the DACC has transmitted all data to the DAC. When set in circular buffer mode, all read data in a buffer has returned from memory.	
		0	No Status
		1	DMA Complete

Modify 0 Register

The DACC_MOD0 register contains the address increment applied between each DMA read from memory, starting at the base-address (DACC_BPTR0). The value is a signed, two's complement byte address increment.

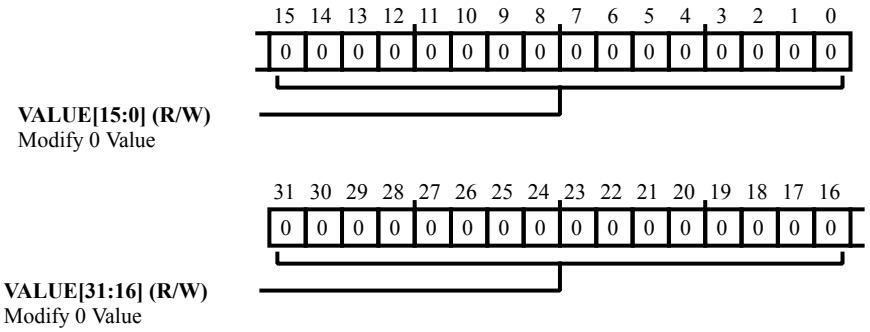


Figure 31-19: DACC_MOD0 Register Diagram

Table 31-19: DACC_MOD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Modify 0 Value. The DACC_MOD0.VALUE bits hold memory offset increment applied between each DMA read from memory, starting at the base address.

Status Register

The DACC_STAT register indicates status for DACC operations.

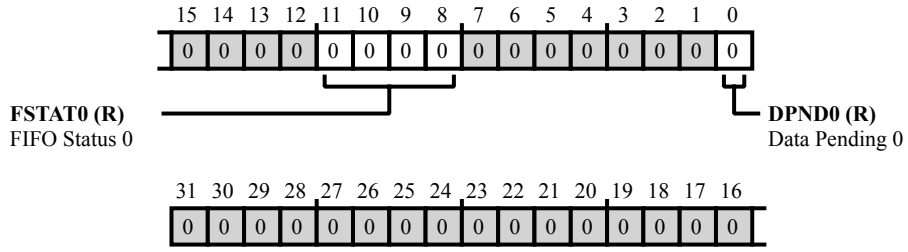


Figure 31-20: DACC_STAT Register Diagram

Table 31-20: DACC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11:8 (R/NW)	FSTAT0	FIFO Status 0. The DACC_STAT.FSTAT0 bits indicates the number of 16-bit data in DAC0 FIFO remaining to be transmitted to DAC. The status increments when DMA or core fills the DAC FIFO. The status decrements when a data transmission starts on the DAC interface. These bits are cleared when the DACC_CTL0.DMAEN bit has a 0-1 transition.	
0 (R/NW)	DPND0	Data Pending 0. The DACC_STAT.DPND0 bit indicates whether the DAC0 interface has made a DMA read access request and is pending (waiting) to receive the data. If DMA for the DAC0 interface is disabled (DACC_CTL0.DMAEN =0), wait until DACC_STAT.DPND0 =0 before enabling DMA for the DAC. This bit is cleared when DACC_CTL0.DMAEN has a 0-1 transition.	
		0	No Status
		1	Pending Read Data

Timing Control 0 Register

The DACC_TC0 register controls timing related to the DAC0 interface clock and sync signals.

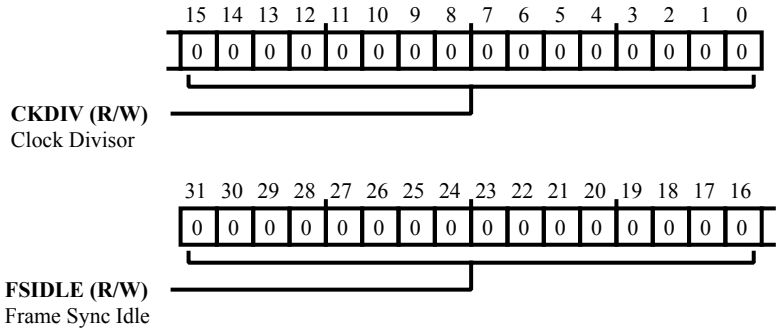


Figure 31-21: DACC_TC0 Register Diagram

Table 31-21: DACC_TC0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	FSIDLE	Frame Sync Idle. The DACC_TC0.FSIDLE value selects minimum idle time (in DACC_ACLK cycles) required between on DACC_AFS asserted edge and the next DACC_AFS asserted edge. A value of 0 implies 1 cycles idle. For DACC_CTL0.DLEN not =0, the DACC_AFS asserted period can be calculated from the DACC_TC0.FSIDLE and DACC_CTL0.DLEN values as: DACC_AFS period = (DACC_TC0.FSIDLE+1) + DACC_CTL0.DLEN For DACC_CTL0.DLEN =0, the DACC_AFS asserted period can be calculated as: DACC_AFS period = (DACC_TC0.FSIDLE+1) + 16
15:0 (R/W)	CKDIV	Clock Divisor. The DACC_TC0.CKDIV bits select the clock divisor ratio (SCLK:ACK) for clocks to be sent to ADC calculated as: ACK frequency = (SCLK frequency) / (CKDIV+1) Yielding, DACC_TC0.CKDIV =0 represents a ratio of 1:1, =1 represents a ratio of 1:2, DACC_TC0.CKDIV =2 represents a ratio of 1:3, and so on.

32 Fast Over Current Protection (FOCP)

The processor supports protection from over-current on its analog channels, via dedicated comparators in the analog front end subsystem (AFE). For complete information, see Analog-to-Digital Converter Controller (ADCC).

FOCP Features

The FOCP module has the following features

- Over limit and under limit programming available via 8-bit DACs.
- Programming of limits via DAC Controller (DACC).
- The comparators take in data from ADC analog inputs directly.
- The comparators can detect over current while the ADCs are sampling data.
- Interrupt facility as well as pin output status available upon detection.

FOCP Functional Description

The following sections provide information on the functional operation of the FOCP.

Architectural Concepts

The FOCP block is required to overcome the sampling rate requirement for certain inputs. Three comparators are available. The input of each comparator is connected internally to inputs A0, B0, and C0. The comparators have a common upper threshold (LIMIT_U) and lower threshold (LIMIT_L) which is set by internal 8-bit DACs. COMP_OUT_A/B/C outputs are accessible to users. If one or more comparators are signaling LIMT (availability of COMP_OUT_A/B/C), the AFE asserts an interrupt to the processor.

The AFE status register provides status bits indicating what event caused the assertion of the COMP_OUT_IRQ interrupt request. Note that accessing this register incurs a significant time latency. Creating an interrupt handler routine that performs an immediate response to the interrupt request before identifying the precise cause of the interrupt request may mitigate this latency.

The COMP_OUT_IRQ signal is provided with the following event connections:

Interrupts:

- The AFE_LIMIT interrupt to the M4 core
- The AFE_LIMIT interrupt to the M0 core

Trigger Masters:

- The AFE_LIMIT trigger master in the M4 TRU1 unit

Reading the status of the AFE_LIMIT is accomplished through an AFE sequence. An example is provided in Programming FOCP Comparators to Detect Over Current. The program reads the `AFE Registers (AFE_Regs.h)` register and verifies which of the following sources have caused the Limit interrupt. Reading of the status in software is optional, but the corresponding bits for each case in the status register are sticky if they are asserted, and reflects recent status. These bits are cleared during read operation.

- AFE_OK_FOCP — Check Stuck at Fault for FOCP_CLK
- FOCP_LATCH_0 — FOCP COMP_OUT_A
- FOCP_LATCH_1 — FOCP COMP_OUT_B
- FOCP_LATCH_2 — FOCP COMP_OUT_C
- AFE_LIMIT interrupt — Once this interrupt is triggered, it cannot be cleared by software. However, it can be cleared if the voltage is within the compartor limits. This is true for both COMP_OUT check as well as in a clock fault condition. Programs may mask the interrupt once necessary operations are performed or wait until the conditions are no longer faulty.

The AFE contains monitoring hardware to monitor the AFE_CLK signal which is necessary for proper FOCP comparator operation. By programming the AFE control registers, the monitor output can be arranged to generate the COMP_OUT_IRQ interrupt in the event the monitor detects a problem with the FOCP_CLK signal.

NOTE: It is not necessary to read the status of the AFE_LIMIT. In such cases, there is no need to access the ADC0 for AFE sequence. This is also true for a case where the comparators thresholds are programmed only once. If AFE_LIMIT status must be read or DAC thresholds must be programmed, ADC0 must be used.

If ADC0 has to be utilized, as in the above cases, ADC0 must not be sampling at that time.

FOCP Clock

The FOCP Analog Comparators require a clock called FOCP_CLK for proper operation. This clock is specified to be nominally 10 MHz and within the range 9 to 12.5 MHz, with an approximately 50% duty cycle. This clock is generated via the CGU in the processor. The FOCP_CLK divisor is enabled using the `PADS_FOCP_DIV.EN` control bit. The phase of the FOCP_CLK can be controlled by the use of this enable bit.

Figure 32-1: FOCP Clock

The FOCP_CLK clock is generated by the FOCP_CLK Divider mechanism, which uses the CGU DCLK output and a divisor specified by the PADS_FOCP_DIV.DAT register bit field, according to the following formula (note that this always produces a 50% duty cycle FOCP_CLK):

$f_{FOCP\_CLK} = f_{DCLK} / (2 \times (FOCP\_DIV:DAT))$ (for FOCP_DIV ≠0)

$f_{FOCP\_CLK} = f_{DCLK} / (32)$ (for FOCP_DIV = 0)

The frequency of DCLK, f_{DCLK}, is controlled by the CGU_DIV.DSEL, CGU_CTL.MSEL and CGU_CTL.DF bit fields in the CGU, such that:

$f_{DCLK} = (f_{PLLCLK}) / DSEL = [(f_{SYS\_CLKIN} / (DF+1)) \times MSEL] / DSEL$

To generate an appropriate 10 MHz FOCP_CLK given an f_{PLLCLK} for the system, the programmer selects a CGU_DIV.DSEL value in the range [1,32] and an FOCP_divisor in the range [1,16] so that:

$2 \times (FOCP\_Divisor) \times DSEL = (f_{PLLCLK} / 10\ MHz)$

where if the FOCP_Divisor is in the range [1–15], then set the register PADS_FOCP_DIV.DAT = FOCP_Divisor; else if FOCP_Divisor is 16, then set the register PADS_FOCP_DIV.DAT =0.

NOTE:
- The FOCP_CLK can be observed externally using the CLKOUTMUX if desired, for example for debug.
- When the CGU is in ACTIVE (bypass) mode, DCLK is equal to SYS_CLKIN. Care must be taken not to violate the AFE AC specifications during this time. If the FOCP_DIV divider is not disabled, then FOCP_CLK has a frequency of $f_{FOCP\_CLK} = f_{SYS\_CLKIN} \times (2 \times (FOCP\_DIV:DAT+1))$
- When FOCP is disabled, the output is low.
- The FOCP divide value is sampled in the divider logic upon the rising edge of the enable signal. This means to change the divider, you must disable and then re enable with the new divider.
- The FOCP clock fault check is enabled by setting the FOCP_CLK_CHECK_EN bit in the AFE Registers (AFE_Regs.h) register. For the FOCP clock fault check to be working, ADCC1 conversion must be on-going in the background. The internal circuit requires ADC1 clocks to be active and ADC1 to perform conversion whenever FOCP clock fault checking is done.

Diagnostic Mode

In diagnostic mode, the program can route the 12-bit DAC output to the comparators. This diagnostic feature is enabled by setting the DIAG_FOCP_EN bit in the AFE Registers (AFE_Regs.h) register.

FOCP Block Diagram

The FOCP is a real-time Fast Over Current Protection Unit designed in the processor.

It has the following advantages:

- No processor involvement to automatically trip PWM blocks or comparator assertion signal to external world.

- Saves additional BOM cost to implement over current protection.
- No additional signal conditioning to interface to processor.
- Direct access to comparator outputs.
- 2 x 8-bit programmable thresholds via internal DACs.

The DACC unit controls the AFE DAC outputs, and is used to set the levels for the Fast Over-Current Protection comparators. Three analog channels are available as inputs to the comparators. The program can generate an AFE_LIMIT signal as an interrupt to the processor to process the detection. Additionally, dedicated comparator output pins (COMP_OUT_A / COMP_OUT_B / COMP_OUT_C) are provided for each comparator so that it can be monitored externally. The program selects the correct DAC to write to so the comparators can detect the correct limits. This is enabled by running a routine.

- Over voltage DAC – DAC1
- Under voltage DAC – DAC2

NOTE: Setting the Over voltage DAC (DAC1) to 0xFF and Under voltage DAC (DAC2) to 0x0 turns off the FOCP comparators.

The ADC input channels for FOCP comparator detection are fixed and not configurable.

Programming the DACs are done via DACC unit, but selecting the DACs (1 or 2) has to be done by issuing an AFE sequence that utilizes the ADC0.

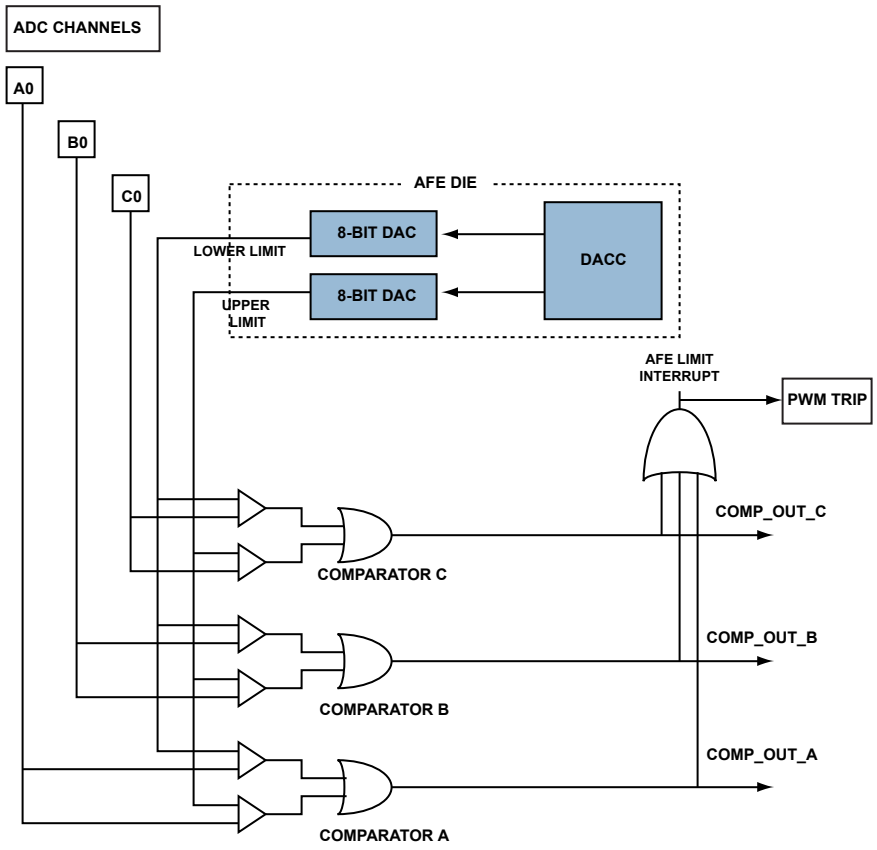


Figure 32-2: FOCP Block Diagram

FOCP Programming Concepts

Programming Examples

The following sections provide programming examples for the FOCP module.

Programming FOCP Comparators to Detect Over Current

To program FOCP comparators for the detection of over current:

1. Initialize CGU and all clocks. This includes DCLK clock which is source for FOCP clock.
2. Initialize the AFE. This does the device configuration.
3. Configure the Overvoltage DAC.
4. Configure the Undervoltage DAC.
5. Delay of 10 usec.
6. Configure the FOCP clock to 10 MHz via `PADS_FOCP_DIV.DAT`.
7. Enable the FOCP clock via `PADS_FOCP_DIV.EN`

8. Enable the Interrupt for AFE_LIMIT, if required.
9. Inside the handler, read the AFE status register to check which COMP has triggered the fault. The LIMIT status is ON as long as LIMIT has crossed, and therefore no control from digital side.

Re-initializing FOCP Comparators

To program the FOCP_CLK divisor without glitches in the clock:

1. Disable the Interrupt for AFE_LIMIT
2. Disable the FOCP clock.
3. Configure the Overvoltage DAC.
4. Configure the Undervoltage DAC.
5. Delay of 10 usec.
6. Enable the FOCP clock via `PADS_FOCP_DIV.EN`.
7. Enable the Interrupt for AFE_LIMIT, if required.

Verifying Comparator Limit

To determine which of the comparators have triggered an FOCP event, the program reads an AFE register as shown in programming examples. This register is ready via ADCC0 through the ADC0 interface. This register read must be initiated and completed only when ADC0 is idle. Also, ADCC0 must not have received a new trigger and must have completed all event activities related to previous triggers.

To verify which comparator has reached the limit, the program must perform a specific AFE sequence.

ADCC0/ADC0 is used as the interface to read and write the AFE registers, including those related to the FOCP. Refer to AFE Register Access Programming Model for more information on accessing these registers.

Accessing the AFE registers must be timed in such a way that it does not interfere with the sampling operation of ADC0 (normal data acquisition via ADC0). Accesses must be initiated before the start of one Event frame and must complete before another Event frame that uses ADC0.

```
void Enable_FOCP_ClockCheck()
{

//set focp clock to 10Mhz
*pREG_PADS1_FOCP_DIV= (( 5 << BITP_PADS_FOCP_DIV_DAT ) &
BITM_PADS_FOCP_DIV_DAT );
//enabling focp clock
*pREG_PADS1_FOCP_DIV | =BITM_PADS_FOCP_DIV_EN;

#define CTLWD_USERCFG         0x80 //Bit[7] - RESERVED = 1
                                   //Bit[6] - WNR = 0
                                   //Bit[5] - RESERVED = 0
                                   //Bit[4-1] - REGADDRESS = 0000
```

```
                                    //Bit[0] - RESERVED = 0

#define CTLWD_FOCPCLK         0xc0 //Bit[7] - RESERVED = 1
                                    //Bit[6] - WNR = 1
                                    //Bit[5] - RESERVED = 0
                                    //Bit[4-1] - REGADDRESS = 0000
                                    //Bit[0] - RESERVED = 0


//read register
*pREG_ADCC0_ADCRW0= ((CTLWD_USERCFG  << BITP_ADCC_ADCRW0_CTLW) &
BITM_ADCC_ADCRW0_CTLW );

// wait till ADCC operation is pending
while((*pREG_ADCC0_ADCRW0 & BITM_ADCC_ADCRW0_PND));

//enable fault check
user_cfg1= ( *pREG_ADCC0_ADCRW0 >>1);
*pREG_ADCC0_ADCRW0 = (((CTLWD_FOCPCLK  << BITP_ADCC_ADCRW0_CTLW) &
BITM_ADCC_ADCRW0_CTLW ) |
(((user_cfg1|( 1 << BITP_AFE_USERCFG_FOCP_CLK_CHECK_EN )) <<
BITP_ADCC_ADCRW0_DAT) & BITM_ADCC_ADCRW0_DAT));

// wait till ADCC operation is pending
while((*pREG_ADCC0_ADCRW0 & BITM_ADCC_ADCRW0_PND));
}
```

Enabling FOCP Fault Clock Detection

```
void AFE_LIMIT_Handler()
{
AFE_LIMIT_Flag = 1;
AFE_LIMIT_Cnt++;

#define CTLWD_STATUSREG     0x82 //Bit[7] - RESERVED = 1
//Bit[6] - WNR = 0
//Bit[5] - RESERVED = 0
//Bit[4-1] - REGADDRESS = 0001
//Bit[0] - RESERVED = 0

//Read status reg
*pREG_ADCC0_CTL = (uint32_t) (BITM_ADCC_CTL_EN | BITM_ADCC_CTL_MODE |
BITM_ADCC_CTL_DSIZE);
*pREG_ADCC0_ADCRW0= ((CTLWD_STATUSREG <<BITP_ADCC_ADCRW0_CTLW) &
BITM_ADCC_ADCRW0_CTLW);

// wait till ADCC operation is pending
while((*pREG_ADCC0_ADCRW0 & BITM_ADCC_ADCRW0_PND));
//if 19th bit of status register is set, means focp clock fault status is set
Focp_output= (( *pREG_ADCC0_ADCRW0 >>1) & 0x80000 );
```

```
//clear interrupt
*pREG_SYSBLK0_SISTAT10 =
((1<<BITP_SYSBLK_SISTAT10_SYS_AFE_LIMIT)&BITM_SYSBLK_SISTAT10_SYS_AFE_LIMIT);
__DSB();

}
```

FOCP Register Descriptions

The registers for the FOCP module are located in the System Block (SYSBLK) and PADS chapter.

33 Harmonic Analysis Engine (HAE)

The harmonic analysis engine (HAE) analyzes harmonic frequencies present on the voltage and current input samples. The HAE receives input samples from two source channels whose frequencies are 45–65 Hz. The HAE then processes the input samples and produces output results. The output results consist of power quality measurements of the fundamental and up to 12 more harmonics.

HAE Features

The HAE features include:

- Processing of two 24-bit signed input channels, consisting of one voltage and one current. The input full scale is limited to ~ +/-6,000,000
- Parity protection in data RAM
- Processing of fundamental frequencies between 45-65 Hz
- Processing of input samples at a nominal 8 kHz rate
- Processing of the fundamental plus 12 harmonic frequencies
- Active, reactive, apparent, I_{RMS}, V_{RMS}, and power factor on the fundamental frequency
- Active, reactive, apparent, I_{RMS}, V_{RMS}, power factor, I_{HD+n}, and V_{HD+n} on the 12 harmonic frequencies
- The accuracy of the measurements, relative to a full scale of +/-6,000,000:
    - Fundamental active or reactive powers 0.1% down to 1/1000 of full scale
    - Fundamental apparent power 0.2% down to 1/1000 of full scale
    - Fundamental I_{RMS}/V_{RMS} 0.1% down to 1/1000 of full scale
    - Fundamental power factor 0.3% based on active and apparent accuracy
    - Harmonic active or reactive powers 1% down to 1/1000 of full scale
    - Harmonic apparent power 2% down to 1/1000 of full scale
    - Harmonic I_{RMS}/V_{RMS} 1% down to 1/1000 of full scale

- Harmonic power factor 3% based on active and apparent accuracy
- Harmonic I_{HD+n} 2% down to 1/1000 FS based on the fundamental and harmonic I_{RMS}
- Harmonic V_{HD+n} 2% down to 1/1000 FS based on the fundamental and harmonic V_{RMS}

- Twelve 6-bit fields for selecting harmonics to analyze, limited by bandwidth of the input signal. The fundamental component always is provided.

HAE Functional Description

The following sections provide a functional description of the HAE.

Harmonic engine

The hardware block of the harmonic engine works with other HAE blocks to co-process full and partial results

Harmonic Analyzer

The harmonic analyzer block works with the harmonic engine to co-process full and partial results.

Data Transfer Module

The data transfer module transfers three channels of data to and from the HAE module.

CM41X_M4 HAE Register List

The Harmonic Analysis Engine (HAE) analyzes harmonics present on voltage and current input samples. The HAE receives input samples from two source channels, processes the samples, and produces output results. The output results consist of power quality measurements of the fundamental and up to twelve additional harmonic components. A set of registers governs HAE operations. For more information on HAE functionality, see the HAE register descriptions.

Table 33-1: CM41X_M4 HAE Register List

Name	Description
HAE_CFG0	Configuration 0 Register
HAE_CFG1	Configuration 1 Register
HAE_CFG2	Configuration 2 Register
HAE_CFG3	Configuration 3 Register
HAE_CFG4	Configuration 4 Register
HAE_DIDT_COEF	DIDT Coefficient Register
HAE_DIDT_GAIN	DIDT Gain Register

Table 33-1: CM41X_M4 HAE Register List (Continued)

Name	Description
HAE_H[nn]_INDX	Harmonic n Index Register
HAE_ISAMPLE	I (Current) Sample Register
HAE_IWAVEFORM	I (Current) Waveform Register
HAE_RUN	Run Register
HAE_STAT	Status Register
HAE_VLEVEL	Voltage Level Register
HAE_VSAMPLE	V (Voltage) Sample Register
HAE_VWAVEFORM	V (Voltage) Waveform Register

CM41X_M0 HAE Interrupt List

Table 33-2: CM41X_M0 HAE Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
6	HAE0_PERR	HAE0 Parity Error	Level	
27	HAE0_STAT	HAE0 Status	Level	

CM41X_M4 HAE Interrupt List

Table 33-3: CM41X_M4 HAE Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
24	HAE0_PERR	HAE0 Parity Error	Level	
144	HAE0_STAT	HAE0 Status	Level	

33.2.4 SPORT Trigger List

CM41X_M4 HAE DMA Channel List

Table 33-4: CM41X_M4 HAE DMA Channel List

DMA ID	DMA Channel Name	Description
DMA8	HAE0_RXDMA_CH0	HAE0 RX0: DMA->HAE DATAI
DMA9	HAE0_TXDMA	HAE0 TX: HAE->DMA
DMA10	HAE0_RXDMA_CH1	HAE0 RX1: DMA->HAE DATAV

HAE Block Diagram

The ***HAE Block Diagram*** shows the functional blocks within the HAE.

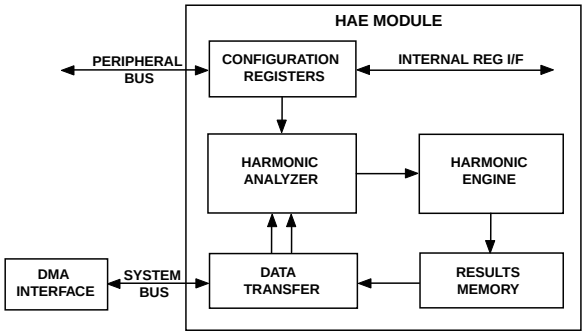


Figure 33-1: HAE Block Diagram

HAE Architectural Concepts

Using the HAE features and event control to their greatest potential requires an understanding of these architectural concepts.

- Harmonic Engine
- Harmonic Analyzer
- Data Transfer Module
- Results Memory

Harmonic Engine

The ***HAE Engine Block Diagram*** presents a synthesized diagram of the harmonic engine, its settings, and its output registers.

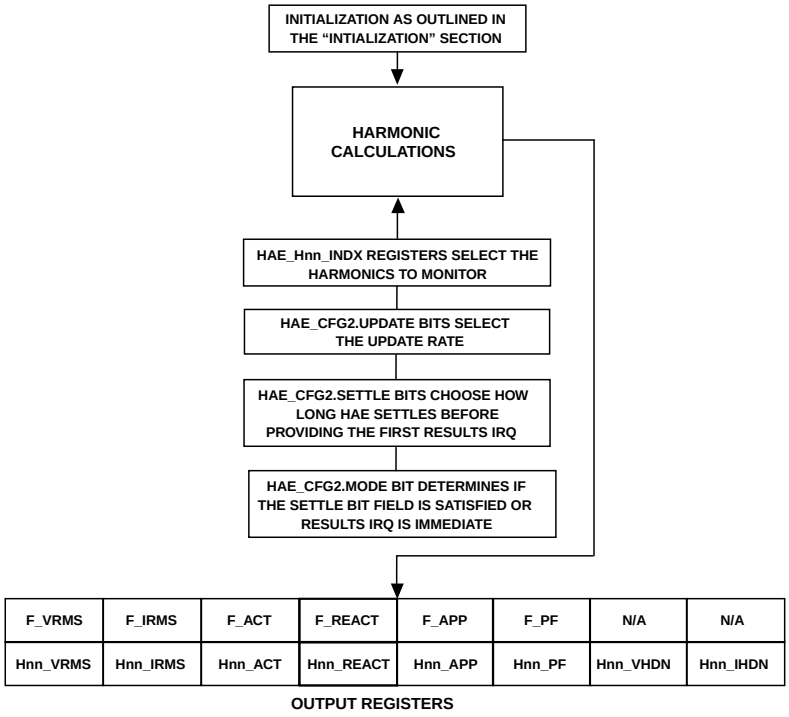


Figure 33-2: HAE Engine Block Diagram

The harmonic engine hardware block works with other HAE blocks to co-process full and partial results (see Harmonic Calculations). At the start of a new sampling period (described in Initialization), the harmonic engine cycles through predefined locations in data RAM, which contain the analyzer processing results. See Harmonic Analyzer for more information.

As the harmonic engine produces results in their final formats (described in HAE Result Ranges and Formats), the results are stored in the results memory (see Results Memory).

The HAE engine computes harmonic information for line frequencies 45–66 Hz. Neutral current can also be analyzed simultaneously with the sum of phase currents. See Theory of Operation for more information.

Harmonic Analyzer

The harmonic analyzer block works with the Harmonic Engine to co-process full and partial results.

To perform harmonic analysis, the HAE contains a pair of voltage and current inputs and a set of 12 indexes to indicate which harmonic components to extract. See Theory of Operation and Harmonic Calculations for more information.

High-Pass Filters (HPFs)

The voltage and current have the option of being high-pass filtered to remove DC offsets. The ***Frequency Response for High-Pass Filters*** figure shows the frequency response of the high-pass filters (the -3 dB point is around 1.1 Hz). The HPFs are enabled by default.

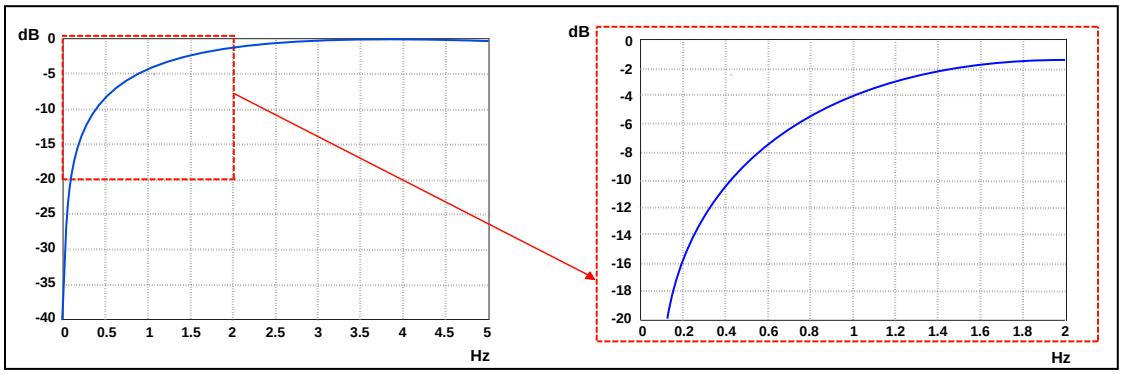


Figure 33-3: Frequency Response for High-Pass Filters

Digital Integrators

For cases when the current is sensed with a di/dt sensor, such as a Rogowski coil, the HAE offers an internal digital integrator for compensation. Ideally, it causes a perfect 90 degrees of phase shift at all the frequencies; at 50 Hz it is close to that value (see the ***Digital Integrator*** figure). The integrator is necessary to restore the signal to its original form before using the signal in HAE calculations. The digital integrator is disabled by default.

With digital integrator enabled for current channel, the amplitude of current will change with the frequency according to a slope of 20 dB per decade. For frequency of 50 Hz, the amplitude of current remains same after passing through the integrator. However, for other frequencies in the range 45-65 Hz, the amplitude of current changes with a slope of 20 dB per decade after passing through integrator.

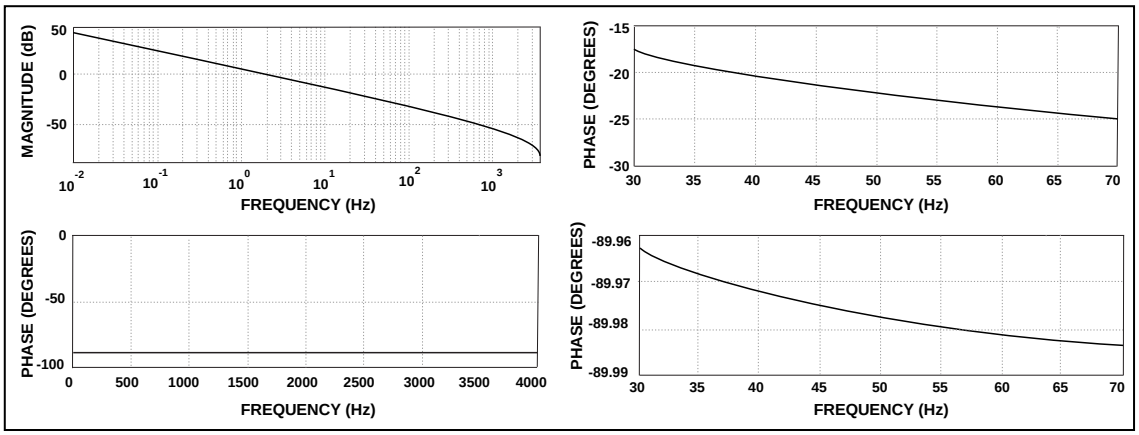


Figure 33-4: Digital Integrator

Phase-Locked Loop and Clock Control (PLL)

The fundamental frequency of the system is extracted from the voltage signal using the phase-locked loop and clock control (PLL) techniques. PLL techniques are optimized for signals with frequencies used in standard power grids around the world (50 Hz or 60 Hz). The techniques consider possible deviations of up to 5 Hz, so the final guaranteed operating range is from 45 Hz to 65 Hz.

The initial detection time of the frequency depends on its value and can take up to several seconds. This activity only happens at the start-up of the HAE block. Once the value of the fundamental frequency is detected, the PLL tracks it continuously.

Settling Times

The block that extracts all of the harmonic RMS and power values is replicated 12 times in parallel for all of the harmonic indexes plus once for the fundamental. Once an index for a particular harmonic changes, there is a settling time of about 700 mSec (~700 / 0.125 = 5600 8K samples). This settling time achieves less than 0.1% error needed for all the internal RMS and powers computations (see the ***Fundamental and Harmonic Values - Settling Times*** plots).

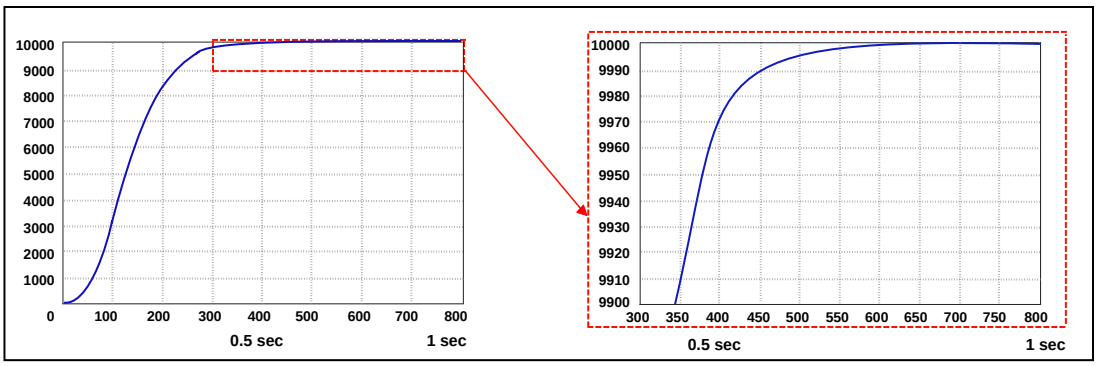


Figure 33-5: Fundamental and Harmonic Values - Settling Times

Data Transfer Module

The data transfer module transfers three channels of data to and from the HAE module.

RX I (Receive Current Sample) Channel

The RX I channel transfers HAE current (I) input samples from system memory through direct memory access (DMA). The RX I request occurs at a nominal 8 kHz rate, depending on the `HAE_CFG1.STARTDIV` bit field. The RX I samples typically come from a SINC filter through a memory buffer. For accurate harmonic results, derive the input samples at the same rate as the HAE sample loop. Therefore, program the SINC and HAE modules to be timing-matched. The SCLK / `HAE_CFG1.STARTDIV` determines the HAE sample loop, which must be programmed to be nominally 8 kHz in frequency. There is one `DWORD` transfer each HAE sample loop.

There is also an RX I memory-mapped location (`HAE_ISAMPLE` register) in peripheral space (MMR), which enables the MCU to provide the I channel samples, timed with the `HAE_STAT.RXIRQ` bit, if desired.

RX V (Receive Voltage Sample) Channel

The RX V channel transfers HAE voltage (V) input samples from system memory through DMA. The RX V request occurs at a nominal 8 kHz rate, depending on the `HAE_CFG1.STARTDIV` bit field. The RX V samples typically come from a SINC filter through a memory buffer. For accurate harmonic results, derive the input samples at the same rate as the HAE sample loop. Therefore, program the SINC and HAE modules to be timing-matched.

The SCLK / HAE_CFG1.STARTDIV, determines the HAE sample loop which must be programmed to be nominally 8 kHz in frequency. There is one DWORD transfer each HAE sample loop.

There is also an RX V memory-mapped location (HAE_VSAMPLE register) in peripheral space (MMR), which enables the MCU to provide the V channel samples, timed with the HAE_STAT.RXIRQ bit, if desired.

Example:

- The HAE_STAT.RXIRQ bit toggles prior to RX transfers:

  The RX interrupt is used internally in hardware to request I and V samples. The MCU can also use the interrupt when it supplies the input samples, rather than the DMA interface.

- RX transfers indicate that the RX channel is ready:

  The HAE RX channels request one RX sample at an 8 kHz rate, depending on the HAE_CFG1.STARTDIV bit field. The holding register stores the RX data, replacing the previous data. The holding register contents are moved up to an output register and driven to the Harmonic Analyzer module for processing. This operation amounts to a two-deep FIFO, allowing a full sample time of latency on the input sample arrival, nominally 125 uSec (8 kHz).

  If the MCU supplies the input samples, rather than the DMA RX interfaces, there are two memory-mapped locations (HAE_ISAMPLE and HAE_VSAMPLE), where the next samples are written. As previously mentioned, the HAE can use the HAE_STAT.RXIRQ bit to time the sample delivery.

TX (Transmit Results) Channel

The TX channel transfers HAE results from results memory to system memory through DMA. The results for the fundamental and 12 harmonics are stored in 13 8-location fields in the results memory. Therefore, the maximum number of DWORDs to transfer is 13 x 8 = 104. Use the HAE_CFG3.CHANEN bits to select the channels to transfer.

The HAE can request a transfer of results for each 8 kHz sample period, meaning that up to 104 DWORDs can be transferred each 125 uSec. Use the HAE_CFG2.UPDATE bit field to set the request rate to longer intervals. The results also are memory-mapped to peripheral address space, which enables the MCU to read the results, timed with the HAE_STAT.TXIRQ bit, if desired.

Example:

- The HAE_CFG3.CHANEN bit field is set to 0x009 to transfer the fundamental and harmonic contained in the HAE_H3_INDX register (programmed to 5^{th}).

- The HAE_STAT.TXIRQ bit toggles prior to TX transfers:

  The HAE uses the TX interrupt internally in hardware to start the transfer. The MCU can also use the interrupt when the MCU reads the HAE results, rather than the DMA interface transferring it.

- TX transfers indicate that the HAE results data is ready:

  The HAE TX channel can transfer one DWORD every other clock. The internal hardware parses through the HAE_CFG3.CHANEN bit field, eventually traversing all 13 bits. If consecutive channels are enabled, two more

IDLE clocks are inserted between the adjacent channels for a total of three IDLE clocks. When channels are skipped, one more IDLE clock is inserted for each unselected channel. Therefore, there are 3 (inter-channel IDLE) + 2 (skipped channels) = 5 IDLE clocks between the fundamental and HAE_H3_INDX transfers.

Results Memory

The results memory is organized by the fundamental and 12 harmonic indexes. Each of the 13 potentially analyzed frequencies has eight locations dedicated to store various power quality measurements. Each frequency is offset from the next by eight locations.

The ***HAE Results Memory*** figure shows the results memory contents.

	INDEX	0	1	2	3	4	5	6	7
GROUP	OFFSET								
FUNDAMENTAL	0x00	F_IRMS	F_VRMS	F_ACT	F_REACT	F_APP	F_PF	N/A	N/A
H1_INDEX	0x08	H1_IRMS	H1_VRMS	H1_ACT	H1_REACT	H1_APP	H1_PF	H1_IHDN	H1_VHDN
H2_INDEX	0x10	H2_IRMS	H2_VRMS	H2_ACT	H2_REACT	H2_APP	H2_PF	H2_IHDN	H2_VHDN
H3_INDEX	0x18	H3_IRMS	H3_VRMS	H3_ACT	H3_REACT	H3_APP	H3_PF	H3_IHDN	H3_VHDN
H4_INDEX	0x20	H4_IRMS	H4_VRMS	H4_ACT	H4_REACT	H4_APP	H4_PF	H4_IHDN	H4_VHDN
H5_INDEX	0x28	H5_IRMS	H5_VRMS	H5_ACT	H5_REACT	H5_APP	H5_PF	H5_IHDN	H5_VHDN
H6_INDEX	0x30	H6_IRMS	H6_VRMS	H6_ACT	H6_REACT	H6_APP	H6_PF	H6_IHDN	H6_VHDN
H7_INDEX	0x38	H7_IRMS	H7_VRMS	H7_ACT	H7_REACT	H7_APP	H7_PF	H7_IHDN	H7_VHDN
H8_INDEX	0x40	H8_IRMS	H8_VRMS	H8_ACT	H8_REACT	H8_APP	H8_PF	H8_IHDN	H8_VHDN
H9_INDEX	0x48	H9_IRMS	H9_VRMS	H9_ACT	H9_REACT	H9_APP	H9_PF	H9_IHDN	H9_VHDN
H10_INDEX	0x50	H10_IRMS	H10_VRMS	H10_ACT	H10_REACT	H10_APP	H10_PF	H10_IHDN	H10_VHDN
H11_INDEX	0x58	H11_IRMS	H11_VRMS	H11_ACT	H11_REACT	H11_APP	H11_PF	H11_IHDN	H11_VHDN
H12_INDEX	0x60	H12_IRMS	H12_VRMS	H12_ACT	H12_REACT	H12_APP	H12_PF	H12_IHDN	H12_VHDN

Figure 33-6: HAE Results Memory

RAM Parity Protection

The HAE implements parity protection on its data memories. An interrupt request can be generated (if enabled) upon detection of a parity error. The software driver or application needs fully initialize the HAE SRAM before starting the accelerator. This initializes the parity state bits and prevents spurious interrupt requests.

HAE Results Upper Byte ID

The HAE results are stored in a 24-bit wide memory as shown in Results Memory.

When the system bus moves the results into memory, the HAE hardware appends a unique CHANNEL:INDEX identifier to the upper byte of each results location. The intent of the ID byte is to help in data parsing of HAE results.

The 32-bit appended result is as follows:

CHANNEL[3:0] ||	INDEX[3:0] ||	RESULTS[23:0]

The CHANNEL in the upper byte corresponds to nn in Hnn_INDX of that particular results location. The INDX in the upper byte corresponds to the INDX within the Hnn_INDX grouping.

For example, location H7_PF has 0x75 in the upper byte of the system bus transfer of that HAE results memory location.

HAE Result Ranges and Formats

The HAE results, stored in the results memory, have formats appropriate for the given measurement. Measurements accuracy is relative to the full scale value of the input samples, and the assumed full scale is +/~6,000,000. If the full scale is above this value, overflow can occur and the results are undefined. A lower full scale limits the dynamic range accuracy of the measurements. For example, a full scale, which is 50% lower than the assumed 6,000,000 full scale, has the dynamic range reduced by 50%. Potentially, this result can be better or worse, as determined by the ADC noise floor supplying the input samples.

The HAE results are as linear as the input samples, within the accuracy ranges specified earlier. The results assume a full-scale input sample of ~=/-6,000,000 and sufficiently low ADC noise floor. Evaluate the system using your specific ADC and full scale specifications to predict the HAE range of accuracy and expected results.

The RMS values of voltage and current calculated from HAE are exactly not same as the theoretical RMS values. The ratio between HAE calculated RMS value and the theoretical value is around 1.11. A gain factor should be added in the software so as to match the theoretical and HAE calculated RMS values. Calibration can be done by calculating the ratio between HAE calculated RMS value and the theoretical one for a particular amplitude of V and I signal and line frequency.

The following list gives the HAE result formats.

- I_{RMS} and V_{RMS}: both fundamental and harmonic I_{RMS} and V_{RMS} are unsigned magnitudes with full scale of ~4,200,000
- Active and reactive power: both active and reactive powers are signed numbers with full scale of ~+/-4,200,000
- Apparent power: apparent power is an unsigned magnitude with full scale of ~4,200,000
- Power factor: each LSB of the fundamental and harmonic power factors equates to a weight of 2^{-23}. Hence, the maximum register value of 0x7FFFFF equates to a power factor value of 1. The minimum register value of 0x800000 corresponds to a power factor of -1. If the power factor is outside of the -1 to +1 range because of offset and gain calibrations, the result is set at -1 or +1. The result depends on the sign of the fundamental reactive power.

- I_{HD+n}/V_{HD+n}: the harmonic distortion plus noise ratios are computed using the RMS of the fundamental and the RMS of the harmonic under analysis. In other words, the ratio only covers the particular harmonic under analysis versus the fundamental. The ratios are stored as 24-bit values in 3.21 signed format. The ratios are limited to +3.9999, and all greater results are clamped to it. The `HD+n` ratios cannot be negative.

HAE Operating Modes

The HAE uses the DMA interface to transfer data to and from system memory. The HAE configuration registers enable the module and calibrate its frequency (clock) divide and other parameters, as described in HAE Programming Model. The HAE triggers and status signals indicate system events and errors.

HAE Data Transfer Modes

The HAE uses the RX DMA interface to transfer data from system memory. Samples for the current (I) and voltage (V) channels of the AFE or SINC filter compose the data: two `WORDS` are transferred each 8 kHz sample period into the HAE (I and V).

The HAE uses the TX DMA interface to transfer data to system memory. The fundamental and selected harmonic results compose the data: up to 13 (fundamental + 12 harmonics) x 8 = 104 `DWORD`s are transferred each 8K sample period.

See Data Transfer Module for more information.

HAE Event Control

The HAE module uses DMA to transfer samples to and data from system memory. The HAE also can use TX and RX events to time the arrival of input samples and extract the results by the MCU:

- The MCU uses the RX event (`HAE_STAT.RXIRQ`) as an IRQ to time when to write the I and V samples into the HAE.
- The MCU uses the TX event (`HAE_STAT.TXIRQ`) as an IRQ to time when to read the HAE results.

HAE Interrupt Signals

The interrupt signals to the HAE module include:

- The RX interrupt to time the delivery of waveform samples as inputs to the HAE.
- The TX interrupt to time when the HAE results memory is ready with new values for the current sample.

The HAE generates the RX interrupts at a rate specified by the `HAE_CFG1.STARTDIV` bit field and the TX interrupts at a rate specified by the `HAE_CFG2.UPDATE` bit field. Refer to Data Transfer Module for more information.

HAE Status and Error Signals

The trigger and status signals to the HAE module include:

- HAE_STAT.RDY (HAE ready status). When the bit is set, the HAE is fully accessible.
- HAE_STAT.RXIRQ (RX IRQ status). The bit mirrors the RX interrupt.
- HAE_STAT.TXIRQ (TX IRQ status). The bit mirrors the TX interrupt.

The status bit requires a 1 to clear and reenable the corresponding interrupt for the next sample period. Refer to Data Transfer Module for more information.

HAE Programming Model

The following sections provide basic procedures for configuring various HAE operations.

Current and voltage data can be transferred from system memory to HAE. HAE results can be transferred to system memory using core and DMA accesses. DMA transfers can be set up to transfer a configurable number of I and V samples and HAE results between HAE and system memory automatically. Core-driven transfers use HAE interrupts to signal the processor core to provide I and V data to the sample registers and read the HAE results from the system memory.

The following sections provide recommended programming guidelines to be followed for core and DMA mode.

Configuring the HAE for DMA Transfers

- Enable HAE operations by configuring the HAE_CFG2.EN bit.
- Poll for HAE ready status from the HAE_STAT.RDY bit configuration.
- Initialize the HAE configuration registers to configure update and settle rates, HPF, integrator for di/dt sensor and line frequency.
- Choose the harmonic channels to be monitored by configuring the HAE_H[nn]_INDX registers and enable channels by configuring the HAE_CFG3.CHANEN bits.
- Configure the RX DMA channel 0 and the RX DMA channel 1 to receive I and V samples. Configure the TX DMA channel 0 to transmit harmonic calculation results.
- Set the HAE clock divider using the HAE_CFG1.STARTDIV field to 8 kHz.
- Set the HAE_RUN register.

Configuring the HAE for Core Transfers

- Enable HAE operations by configuring the HAE_CFG2.EN bit.
- Poll for HAE ready status using the HAE_STAT.RDY bit.
- Initialize the HAE configuration registers to configure update and settle rates, HPF, integrator for di/dt sensor and line frequency.
- Enable receive and transmit IRQ using the HAE_CFG0 register.
- Configure the HAE_VLEVEL register depending on input voltage magnitude requirements.

- Configure the HAE_H[nn]_INDX registers to choose the harmonic channels to monitor. Enable the channels using the HAE_CFG3.CHANEN bits.
- Set the HAE clock divider using the HAE_CFG1.STARTDIV field to 8 kHz.
- Set the HAE_RUN register.
- At each receive IRQ (HAE_STAT.RXIRQ = 1), write I and V sample values to the HAE_ISAMPLE and the HAE_VSAMPLE registers respectively.
- At each transmit IRQ (HAE_STAT.TXIRQ = 1), read the HAE results from the HAE result RAM memory region based on the index.

HAE Programming Concepts

Using the HAE features to their greatest potential requires an understanding of some HAE-related concepts.

- Theory of Operation
- Initialization
- Harmonic Calculations
- Configuring Harmonic Calculations Update Rate

Theory of Operation

The HAE theory of operation can be described using the following scenario:

Consider a nonsinusoidal AC system supplied by a voltage, v(t), that consumes the current, i(t):

$$v(t) = \sum_{k=1}^{\infty} V_k \sqrt{2} \sin(k\omega t + \varphi_k)$$

$$i(t) = \sum_{k=1}^{\infty} I_k \sqrt{2} \sin(k\omega t + \gamma_k)$$

where:

- V_k, I_k are the RMS voltage and current, respectively, of each harmonic.
- Φ_k, γ_k are the phase delays of each harmonic.
- ω is the angular velocity at the fundamental (line) frequency f.

The HAE harmonic calculations are specified for line frequencies 45–65 Hz.

The maximum number of harmonics which the HAE can accept for a particular line frequency depends upon the bandwidth of the input waveforms supplied.

When the HAE analyzes I and V samples, the following metering quantities are computed:

- Fundamental current RMS: I_1

- Fundamental voltage RMS: V_1
- RMS of up to 12 harmonics of the current channel: I_n, n = 2, 3,…, 12
- RMS of up to 12 harmonics of the voltage channel: V_n, n = 2, 3,…, 12
- Fundamental active power: $P_1 = V_1 I_1 cos(\varphi_1 - \gamma_1)$
- Fundamental reactive power: $Q_1 = V_1 I_1 sin(\varphi_1 - \gamma_1)$
- Fundamental apparent power: $S_1 = V_1 I_1$
- Power factor of the fundamental:

  $$pf_1 = \text{sgn}(Q_1) \cdot \frac{P_1}{S_1}$$

  Active power of up to 12 harmonics:

  $P_n = V_n I_n cos(\varphi_n - \gamma_n)$, n = 2, 3,…, 12
- Reactive power of up to 12 harmonics:

  $Q_n = V_n I_n sin(\varphi_n - \gamma_n)$, n = 2, 3,…, 12
- Apparent power of up to 12 harmonics:

  $S_n = V_n I_n$, n = 2, 3, …, 12
- Power factor of up to 12 harmonics:

  $$pf_n = \text{sgn}(Q_n) \cdot \frac{P_n}{S_n}, \text{ n = 2, 3,..., 12}$$

- Total harmonic distortion of the current channel:

  $$(THD)_I = \frac{\sqrt{I^2 - I_1^2}}{I_1}$$

- Total harmonic distortion of the voltage channel:

  $$(THD)_V = \frac{\sqrt{V^2 - V_1^2}}{V_1}$$

- Harmonic distortion of up to 12 harmonics on the current channel:

  $$HD_{I_n} = \frac{I_n}{I_1}, \text{ n = 2, 3,..., 12}$$

- Harmonic distortion of up to 12 harmonics on the voltage channel:

$$HD_{V_n} = \frac{V_n}{V_1}, \text{ n} = 2, 3, \ldots, 12$$

Initialization

The HAE typically is initialized with parameters and settings for a given usage scenario. It is interrupt-driven when new results are available.

The HAE programming depends on whether the line frequency is 50 Hz or 60 Hz. If the external sensor is a di/dt type, there are also some differences. The ***HAE Initialization*** flowchart shows the initialization sequence.

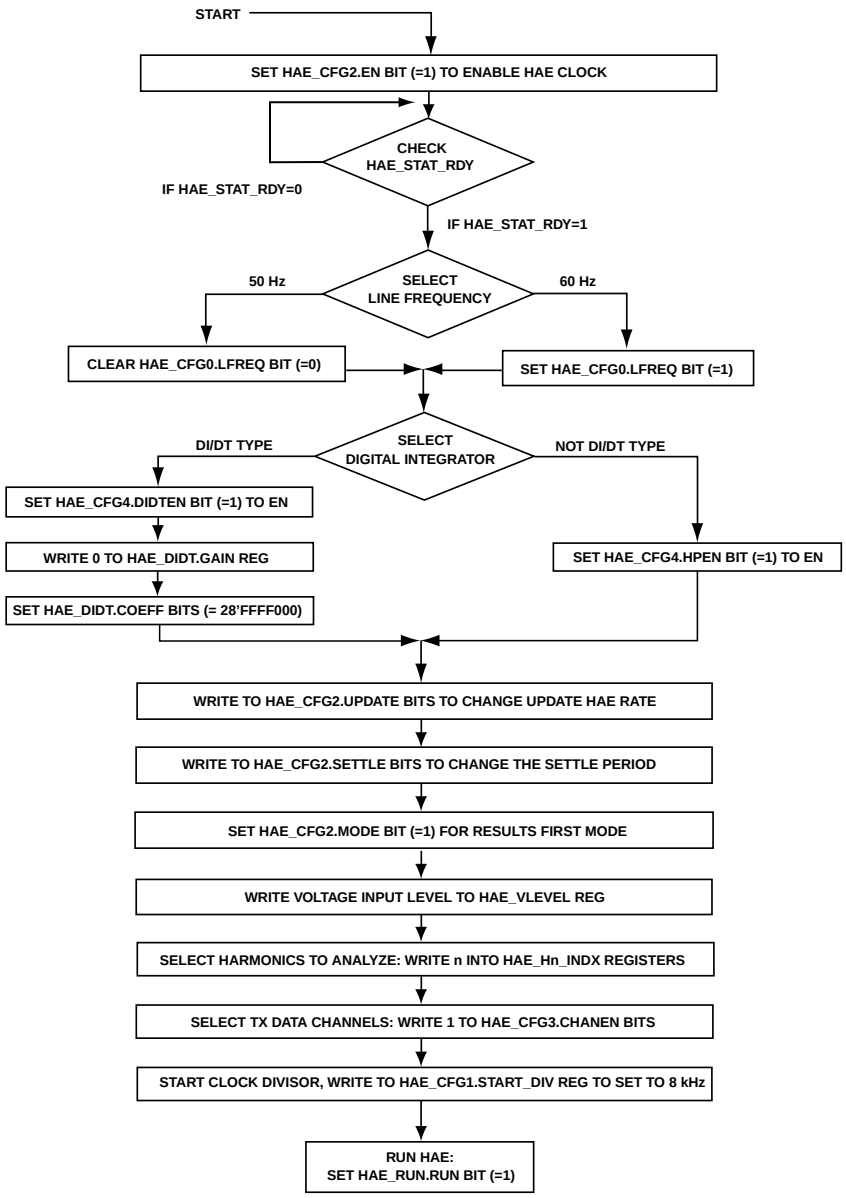


Figure 33-7: HAE Initialization

Harmonic Calculations

When the harmonic engine runs, it computes information about the fundamental and up to 12 harmonics. The HAE simultaneously monitors the indexes of the additional 12 harmonics, provided by the 8-bit registers HAE_H[nn]_INDX. Write the index of the harmonic into the register for the harmonic to be monitored. If the second harmonic is monitored, write 2. If harmonic 51 is desired, write 51. The HAE always monitors the fundamental component, independent of the values written into HAE_H[nn]_INDX. Therefore, if one of these registers is equal to 1, the HAE monitors the fundamental components multiple times. The maximum index allowed in the HAE_H[nn]_INDX registers is 63.

As a reference, the ***Harmonic Engine Outputs and Registers where Values are Stored*** table presents the harmonic engine outputs and registers that store the outputs.

Table 33-5: Harmonic Engine Outputs and Registers where Values are Stored

Quantity	Definition	HAE Registers
RMS of the Fundamental Component	V_1, I_1	F_VRMS, F_IRMS
RMS of a Harmonic Component	V_n, I_n, n = 2, 3,…, 12	Hnn_VRMS, Hnn_XIRMS
Active Power of the Fundamental Component	$P_1 = V_1 I_1 \cos(\varphi_1 - \gamma_1)$	F_ACT
Active Power of a Harmonic Component	$P_n = V_n I_n \cos(\varphi_n - \gamma_n)$, n = 2, 3,…, 12	Fnn_ACT
Reactive Power of the Fundamental Component	$Q_1 = V_1 I_1 \sin(\varphi_1 - \gamma_1)$	F_REACT
Reactive Power of a Harmonic Component	$Q_n = V_n I_n \sin(\varphi_1 - \gamma_1)$, n = 2, 3,…, 12	Hnn_REACT
Apparent Power of the Fundamental Component	$S_1 = V_1 I_1$	F_APP
Apparent Power of a Harmonic Component	$S_n = V_n I_n$, n = 2, 3, …, 12	Hnn_APP
Power Factor of the Fundamental Component	$pf_1 = \mathrm{sgn}(Q_1) \cdot \frac{P_1}{S_1}$	F_PF
Power Factor of a Harmonic Component	$pf_n = \mathrm{sgn}(Q_n) \cdot \frac{P_n}{S_n}$, n = 2, 3,…, 12	Hnn_PF
Harmonic Distortion of a Harmonic Component	$HD_{V_n} = \frac{V_n}{V_1}$, $HD_{I_n} = \frac{I_n}{I_1}$, n = 2, 3,…, 12	Hnn_VHDN, Hnn_IHDN

Configuring Harmonic Calculations Update Rate

The harmonic engine functions at an 8 kHz rate. From the moment the HAE_CFG2 register is initialized, and the harmonic indexes are set in the HAE_H[nn]_INDX index registers, the HAE calculations take typically 750 mSec to settle within the specification parameters.

The HAE module uses the HAE_CFG2.UPDATE bits to manage the update rate of the output registers for the harmonic engine. It manages the update rate independent of the calculations rate of 8 kHz for the engine. The default value of 000 means that the registers are updated every 125 uSec (8 kHz rate). Other update periods are: 250

uSec (001), 1 mSec (010), 16 mSec (011), 128 mSec (100), 512 mSec (101), 1.024 mSec (110). If the HAE_CFG2.UPDATE bits are 111, the harmonic calculations are disabled.

The HAE module provides two ways to manage the harmonic computations. It enables the first approach when bit HAE_CFG2.MODE is cleared to its default value of 0. The state sets status bit HAE_STAT.TXIRQ to 1 after a certain period and then every time the harmonic calculations are updated at HAE_CFG2.UPDATE frequency. This functionality allows an external microcontroller to access the harmonic calculations only after they have settled. The HAE uses the state of bits HAE_CFG2.SETTLE to determine the time period. The possible values of settling time of the harmonic calculations are 512 mSec (00), 768 mSec(01), 1024 mSec (10), and 1280 mSec (11).

The HAE module enables the second approach when bit HAE_CFG2.MODE is set to 1. The state sets the status bit HAE_STAT.TXIRQ to 1 every time the harmonic calculations are updated at the HAE_CFG2.UPDATE frequency, without waiting for the harmonic calculations to settle. This functionality allows an external microcontroller to access the harmonic calculations immediately after starting. A write to the HAE_STAT register clears the status bit. The corresponding bit (HAE_STAT.TXIRQ) is set to 1.

CM41X_M4 HAE Register Descriptions

Harmonic Analysis Engine (HAE) contains the following registers.

Table 33-6: CM41X_M4 HAE Register List

Name	Description
HAE_CFG0	Configuration 0 Register
HAE_CFG1	Configuration 1 Register
HAE_CFG2	Configuration 2 Register
HAE_CFG3	Configuration 3 Register
HAE_CFG4	Configuration 4 Register
HAE_DIDT_COEF	DIDT Coefficient Register
HAE_DIDT_GAIN	DIDT Gain Register
HAE_H[nn]_INDX	Harmonic n Index Register
HAE_ISAMPLE	I (Current) Sample Register
HAE_IWAVEFORM	I (Current) Waveform Register
HAE_RUN	Run Register
HAE_STAT	Status Register
HAE_VLEVEL	Voltage Level Register
HAE_VSAMPLE	V (Voltage) Sample Register
HAE_VWAVEFORM	V (Voltage) Waveform Register

Configuration 0 Register

The HAE_CFG0 register configures high-level interrupts and specifies the line frequency for HAE operations.

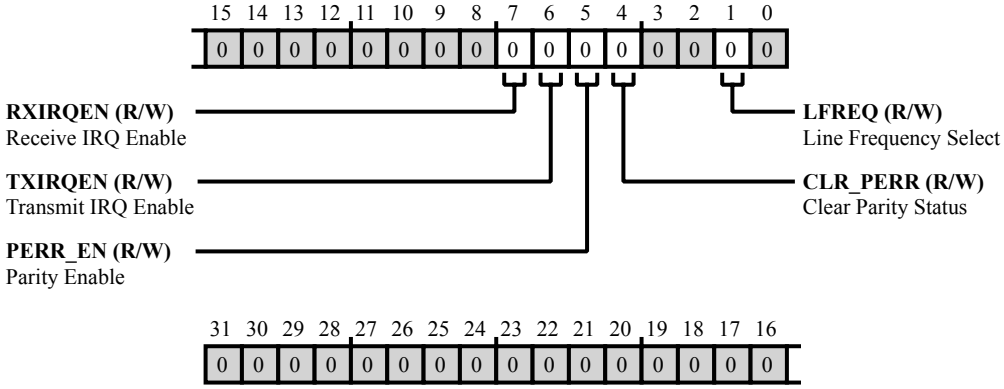


Figure 33-8: HAE_CFG0 Register Diagram

Table 33-7: HAE_CFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	RXIRQEN	Receive IRQ Enable. The HAE_CFG0.RXIRQEN bit enables an interrupt, which the HAE triggers on each request for a new input sample on both the I and V channels.	
		0	Disable
		1	Enable
6 (R/W)	TXIRQEN	Transmit IRQ Enable. The HAE_CFG0.TXIRQEN bit enables an interrupt, which the HAE triggers on each result as the result is calculated and ready to transmit.	
		0	Disable
		1	Enable
5 (R/W)	PERR_EN	Parity Enable. The HAE_CFG0.PERR_EN bit enables parity checking	
4 (R/NW)	CLR_PERR	Clear Parity Status. The HAE_CFG0.CLR_PERR bit clears parity error status	
1 (R/W)	LFREQ	Line Frequency Select. The HAE_CFG0.LFREQ bit specifies the line frequency for the HAE. Set the bit to match the line frequency being analyzed.	
		0	50 Hz
		1	60 Hz

Configuration 1 Register

The HAE_CFG1 register configures the HAE frequency (clock) divider.

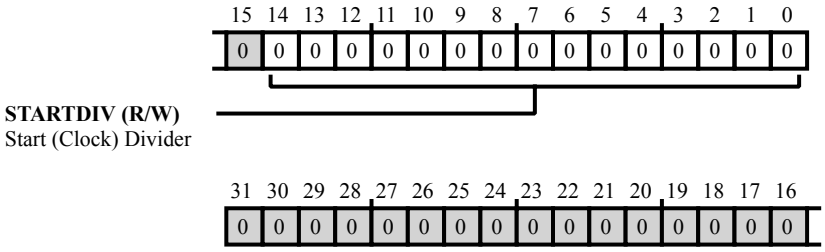


Figure 33-9: HAE_CFG1 Register Diagram

Table 33-8: HAE_CFG1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	STARTDIV	Start (Clock) Divider. The HAE_CFG1.STARTDIV bits provide the sample clock divider. Write the value to divide the main clock down to 8 kHz. The HAE sample loop is determined by SCLK / (HAE_CFG1.STARTDIV).

Configuration 2 Register

The HAE_CFG2 register enables and configures HAE operations as related to output results.

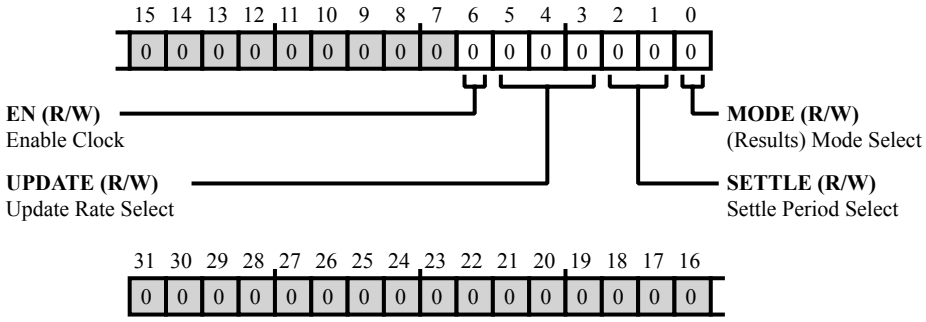


Figure 33-10: HAE_CFG2 Register Diagram

Table 33-9: HAE_CFG2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	EN	Enable Clock. The HAE_CFG2.EN bit enables the HAE the main clock, enabling HAE operation.	
		0	Disable
		1	Enable
5:3 (R/W)	UPDATE	Update Rate Select. The HAE_CFG2.UPDATE bits determine the rate (in microseconds or milliseconds) at which the HAE updates the output results. The HAE_CFG2.UPDATE bits can also disable harmonic calculations.	
		0	125 uSec
		1	250 uSec
		2	1 mSec
		3	16 mSec
		4	128 mSec
		5	512 mSec
		6	1024 mSec
		7	Disable

Table 33-9: HAE_CFG2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2:1 (R/W)	SETTLE	Settle Period Select. The `HAE_CFG2.SETTLE` bits determine the time (in milliseconds) that the HAE waits before producing results. The time is based on an 8K sample count.	
		0	512 mSec
		1	768 mSec
		2	1024 mSec
		3	1280 mSec
0 (R/W)	MODE	(Results) Mode Select. The `HAE_CFG2.MODE` bit determines whether the HAE produces results immediately or waits per the `HAE_CFG2.SETTLE` bit field.	
		0	Results after Settle
		1	Results First

Configuration 3 Register

The HAE_CFG3 register configures HAE data transfer operations by selecting the fundamental and potential of twelve additional harmonic channels. The selected harmonics have their data (results) transferred to memory using DMA. First, the fundamental; followed by the selected channel n, in the order from the lowest to the highest numbered channel. Each selected channel has its eight result words transferred.

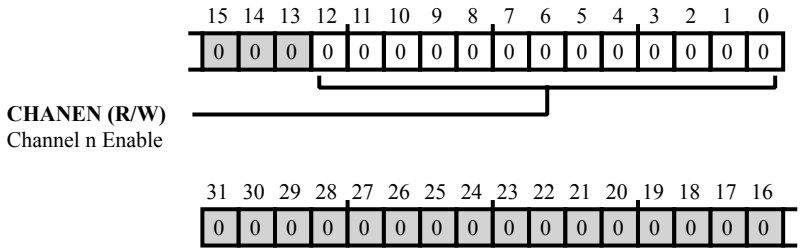


Figure 33-11: HAE_CFG3 Register Diagram

Table 33-10: HAE_CFG3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12:0 (R/W)	CHANEN	Channel n Enable. Each HAE_CFG3.CHANEN bit enables the fundamental and potential of twelve additional harmonic data channels. The following enumerations apply to each bit. Bit 0 denotes the fundamental channel. Bits 1-12 denote the harmonic channels 1-12, accordingly.	
		0	Disable channel n
		8191	Enable channel n

Configuration 4 Register

The HAE_CFG4 register configures the internal digital integrator and high-pass filters (HPS) for HAE operations. The digital integrator is disabled and the filters are enabled by default.

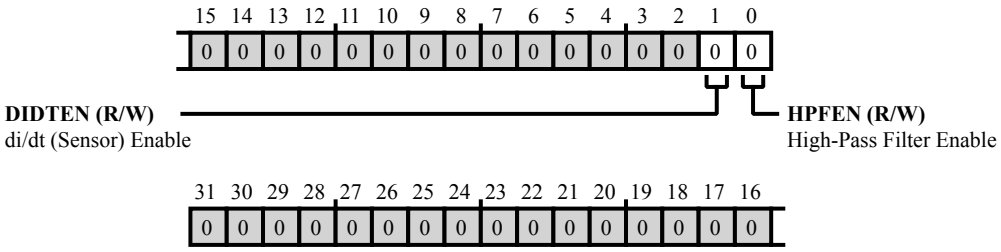


Figure 33-12: HAE_CFG4 Register Diagram

Table 33-11: HAE_CFG4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DIDTEN	di/dt (Sensor) Enable. The HAE_CFG4.DIDTEN bit enables the internal digital integrator for a di/dt sensor. Set the bit (=1) if the sensor is a di/dt type sensor.	
		0	Disable
		1	Enable
0 (R/W)	HPFEN	High-Pass Filter Enable. The HAE_CFG4.HPFEN bit enables the high-pass filters. Set the bit (=1) unless measuring DC levels.	
		0	Disable
		1	Enable

DIDT Coefficient Register

The HAE_DIDT_COEF register sets the di/dt sensor's coefficient.

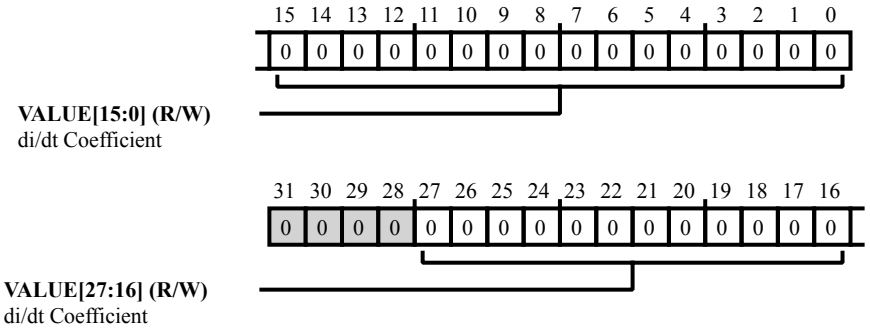


Figure 33-13: HAE_DIDT_COEF Register Diagram

Table 33-12: HAE_DIDT_COEF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	di/dt Coefficient. The HAE_DIDT_COEF.VALUE bits provide the coefficient for a di/dt type sensor. If the HAE_CFG4.DIDTEN bit is set (=1), set this bit field to 28'FFFF000.

DIDT Gain Register

The HAE_DIDT_GAIN register provides the di/dt sensor's gain.

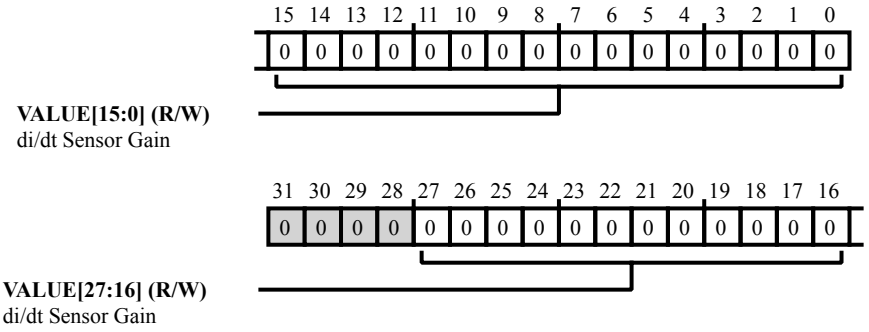


Figure 33-14: HAE_DIDT_GAIN Register Diagram

Table 33-13: HAE_DIDT_GAIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	di/dt Sensor Gain. The HAE_DIDT_GAIN.VALUE bits provide the gain for a di/dt type sensor. If the HAE_CFG4.DIDTEN bit is set (=1), set this bit field to 0.

Harmonic n Index Register

The HAE_H[nn]_INDX registers select harmonics for HAE operations. The fundamental always is provided. The harmonic results appear in the results RAM memory region based on the index.

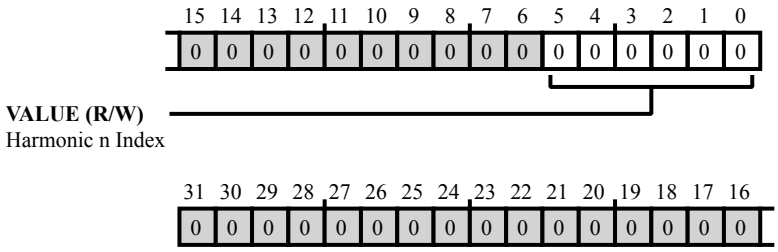


Figure 33-15: HAE_H[nn]_INDX Register Diagram

Table 33-14: HAE_H[nn]_INDX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:0 (R/W)	VALUE	Harmonic n Index. The HAE_H[nn]_INDX.VALUE bits select the harmonic channel n to analyze. Write the index of the harmonic into the HAE_H[nn]_INDX register for that harmonic to be selected.

I (Current) Sample Register

The HAE_ISAMPLE register provides current (I) input samples for HAE calculations.

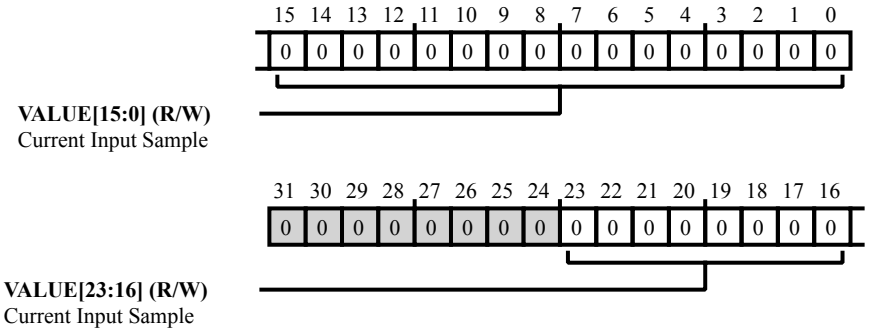


Figure 33-16: HAE_ISAMPLE Register Diagram

Table 33-15: HAE_ISAMPLE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Current Input Sample. The HAE_ISAMPLE.VALUE bits hold the current sample if provided by the MCU. The sample can be timed with the HAE_STAT.RXIRQ bit.

I (Current) Waveform Register

The HAE_IWAVEFORM register holds processed current waveforms produced by the HAE. After some amplitude and phase delay, the waveform follows a sample input.

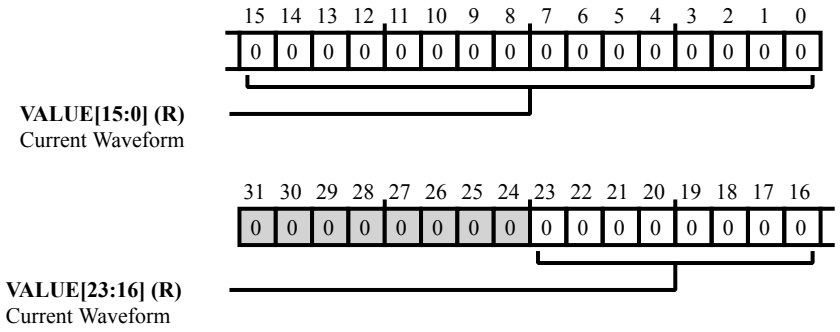


Figure 33-17: HAE_IWAVEFORM Register Diagram

Table 33-16: HAE_IWAVEFORM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/NW)	VALUE	Current Waveform. The HAE_IWAVEFORM.VALUE bits hold the processed current input sample.

Run Register

The HAE_RUN register starts/idles HAE harmonic calculations.

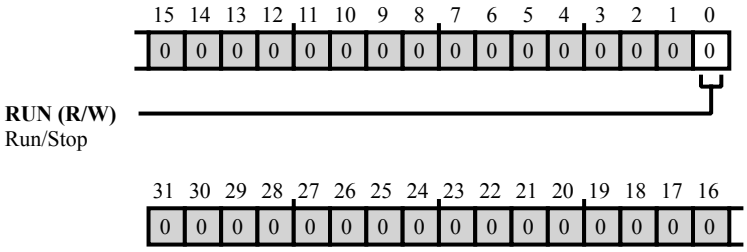


Figure 33-18: HAE_RUN Register Diagram

Table 33-17: HAE_RUN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	RUN	Run/Stop. The HAE_RUN.RUN bit starts the HAE harmonic calculations.	
		0	Stop
		1	Run

Status Register

The HAE_STAT register indicates status for the HAE module and its interrupts.

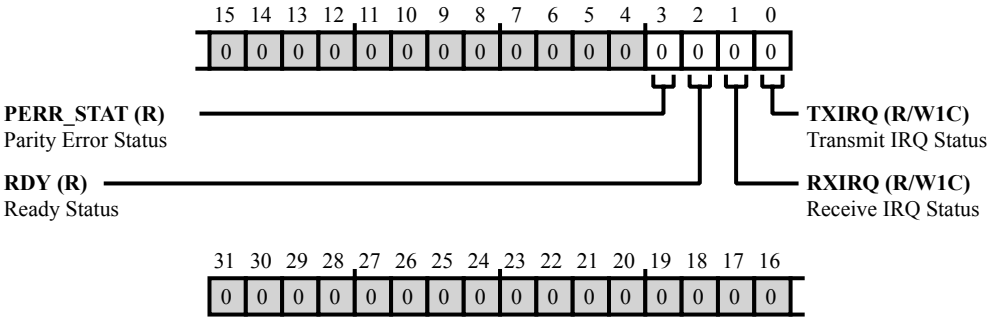


Figure 33-19: HAE_STAT Register Diagram

Table 33-18: HAE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	PERR_STAT	Parity Error Status.	
2 (R/NW)	RDY	Ready Status. The HAE_STAT.RDY bit indicates status for the HAE. When the bit is set (=1), the HAE is fully accessible, following the setting of the HAE_CFG2.EN bit.	
		0	No Status
		1	Ready
1 (R/W1C)	RXIRQ	Receive IRQ Status. The HAE_STAT.RXIRQ bit indicates status for an HAE RX interrupt. The bit mirrors the HAE_CFG0.RXIRQEN bit status.	
		0	No Status
		1	Interrupt Detected
0 (R/W1C)	TXIRQ	Transmit IRQ Status. The HAE_STAT.TXIRQ bit indicates status for an HAE TX interrupt. The bit mirrors the HAE_CFG0.TXIRQEN bit status.	
		0	No Status
		1	Interrupt Detected

Voltage Level Register

The HAE_VLEVEL register is used to scale the fixed fundamental voltage level internally. This assists the HAE in locking onto the fundamental voltage channel.

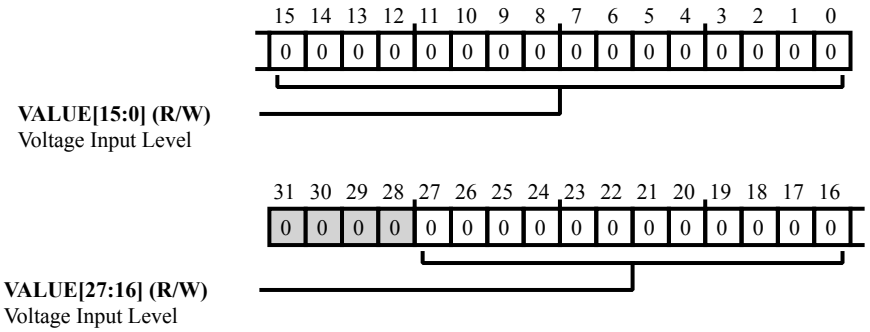


Figure 33-20: HAE_VLEVEL Register Diagram

Table 33-19: HAE_VLEVEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:0 (R/W)	VALUE	Voltage Input Level. The HAE_VLEVEL.VALUE bits hold a value to scale the fixed fundamental voltage level. Use this formula: VLEVEL = FS/V_{IN} * 5033168, where V_{IN} is the fundamental voltage input level.

V (Voltage) Sample Register

The HAE_VSAMPLE register provides voltage (V) input samples for HAE calculations.

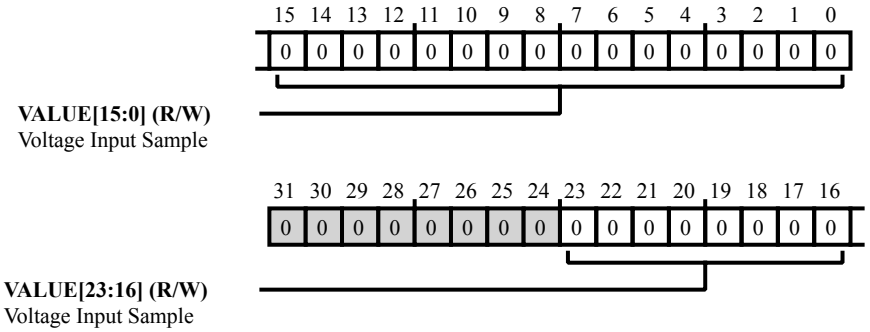


Figure 33-21: HAE_VSAMPLE Register Diagram

Table 33-20: HAE_VSAMPLE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Voltage Input Sample. The HAE_VSAMPLE.VALUE bits hold the voltage sample if provided by the MCU. The sample can be timed with the HAE_STAT.RXIRQ bit.

V (Voltage) Waveform Register

The HAE_VWAVEFORM register contains processed voltage waveforms produced by the HAE. After some amplitude and phase delay, the waveform follows a sample input.

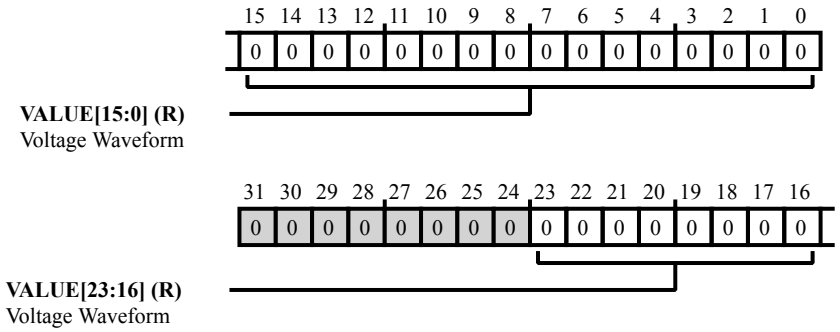


Figure 33-22: HAE_VWAVEFORM Register Diagram

Table 33-21: HAE_VWAVEFORM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/NW)	VALUE	Voltage Waveform. The HAE_VWAVEFORM.VALUE bits hold the processed voltage input sample.

34 Sinus Cardinalis (SINC) Filter

The sinus cardinalis (SINC) filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. A SINC filter converts the bit stream from a sigma-delta front-end modulator into a digital word representing the signal level presented to the modulator.

The filter consists of a set of integration and decimation stages implemented directly in logic for efficient execution. The SINC filter supports capture of current or voltage feedback signals from an isolating analog-to-digital converter (ADC). Each modulator bit stream connects to two SINC filters: a primary filter for controlling feedback; a secondary filter for overcurrent detection. The SINC module includes four filter channels and two modulator clock generators.

SINC Filter Features

The SINC features include:

- Four-bit stream filter channels for current or voltage feedback signal processing
- Each channel includes two SINC filter pairs:
  - A primary filter for feedback signal processing
  - A secondary filter for overload detection
- Up to two modulator clock sources with phase control options
- Configuration of SINC filter channels according to a modulator clock selection
- Programmable order and decimation rates
- Primary filters:
  - Programmable bias and gain with output saturation
  - Dedicated direct memory access (DMA) channels with data interleaving and programmable data ready output triggers
- Secondary filters:
  - Detecting a fault when signals exceed amplitude and duration values

  - Registers preserving the eight most recent samples before a fault event
- Multiple interrupt trigger sources for overload fault and data overflow events

SINC Functional Description

The SINC filter has the following functionality:

Digital filter

The filter removes the modulator sample clock and recovers a digital value of the sampled signal.

DC gain and data resolution

The DC gain of the digital filter is a function of the order and decimation rate.

Frequency response

The frequency response of the filter depends on the order, decimation rate, and modulator clock frequency.

Output scaling

The output scaling and postprocessing functions embedded in the SINC filter blocks differ, depending on the function.

CM41X_M4 SINC Register List

The SINC filter module processes four independent sigma-delta bit streams by applying a pair of SINC filters to each stream. A SINC filter converts the bit stream from a sigma-delta front-end modulator into a digital word representing the signal level presented to the modulator. Each modulator bit stream connects to two SINC filters: a primary filter for controlling feedback, and a secondary filter for overcurrent detection. A set of registers governs SINC operations. For more information on SINC functionality, see the SINC register descriptions.

Table 34-1: CM41X_M4 SINC Register List

Name	Description
SINC_BIAS0	Bias for Group 0 Register
SINC_BIAS1	Bias for Group 1 Register
SINC_CLK	Clock Control Register
SINC_CTL	Control Register
SINC_HIS_STAT	History Status Register
SINC_LEVEL0	Level Control for Group 0 Register
SINC_LEVEL1	Level Control for Group 1 Register

Table 34-1: CM41X_M4 SINC Register List (Continued)

Name	Description
SINC_LIMIT0	(Amplitude) Limits for Secondary Filter 0 Register
SINC_LIMIT1	(Amplitude) Limits for Secondary Filter 1 Register
SINC_LIMIT2	(Amplitude) Limits for Secondary Filter 2 Register
SINC_LIMIT3	(Amplitude) Limits for Secondary Filter 3 Register
SINC_P0SEC_HIST[n]	Pair 0 Secondary (Filter) History n Register
SINC_P1SEC_HIST[n]	Pair 1 Secondary (Filter) History n Register
SINC_P2SEC_HIST[n]	Pair 2 Secondary (Filter) History n Register
SINC_P3SEC_HIST[n]	Pair 3 Secondary (Filter) History n Register
SINC_PHEAD0	Primary (Filters) Head for Group 0 Register
SINC_PHEAD1	Primary (Filters) Head for Group 1 Register
SINC_PPTR0	Primary (Filters) Pointer for Group 0 Register
SINC_PPTR1	Primary (Filters) Pointer for Group 1 Register
SINC_PTAIL0	Primary (Filters) Tail for Group 0 Register
SINC_PTAIL1	Primary (Filters) Tail for Group 1 Register
SINC_RATE0	Rate Control for Group 0 Register
SINC_RATE1	Rate Control for Group 1 Register
SINC_STAT	Status Register

CM41X_M4 SINC Interrupt List

Table 34-2: CM41X_M4 SINC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
153	SINC0_STAT	SINC0 Status	Level	

CM41X_M4 SINC Trigger List

Table 34-3: CM41X_M4 SINC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
33	SINC0_P0_OVLD	SINC0 Pair 0 Overload Indicator	Edge
34	SINC0_P1_OVLD	SINC0 Pair 1 Overload Indicator	Edge
35	SINC0_P2_OVLD	SINC0 Pair 2 Overload Indicator	Edge
36	SINC0_P3_OVLD	SINC0 Pair 3 Overload Indicator	Edge

Table 34-3: CM41X_M4 SINC Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
37	SINC0_DATA0	SINC0 Data Move 0	Edge
38	SINC0_DATA1	SINC0 Data Move 1	Edge

Table 34-4: CM41X_M4 SINC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
34	SINC0_SYNC0	SINC0 Synchronization Input 0	Pulse
35	SINC0_SYNC1	SINC0 Synchronization Input 1	Pulse

SINC Definitions

To make the best use of the SINC, it is useful to understand the following terms.

Decimation

Decimation is the process of discarding samples from a data stream.

Decimation Rate

The decimation rate is the ratio of the filter input data rate to the filter output data rate.

Filter Order

The SINC filter order is the number of integration and decimation stages in the filter.

Modulator Order

The modulator order is the number of comparator and integrator stages in a sigma-delta modulator.

Sigma-Delta Modulator

The sigma-delta modulator is an oversampling analog-to-digital conversion circuit that generates a digital bit stream whose pulse density is proportional to the analog voltage presented to the input.

SINC Block Diagram

The *SINC Block Diagram* figure shows the functional blocks within the SINC module.

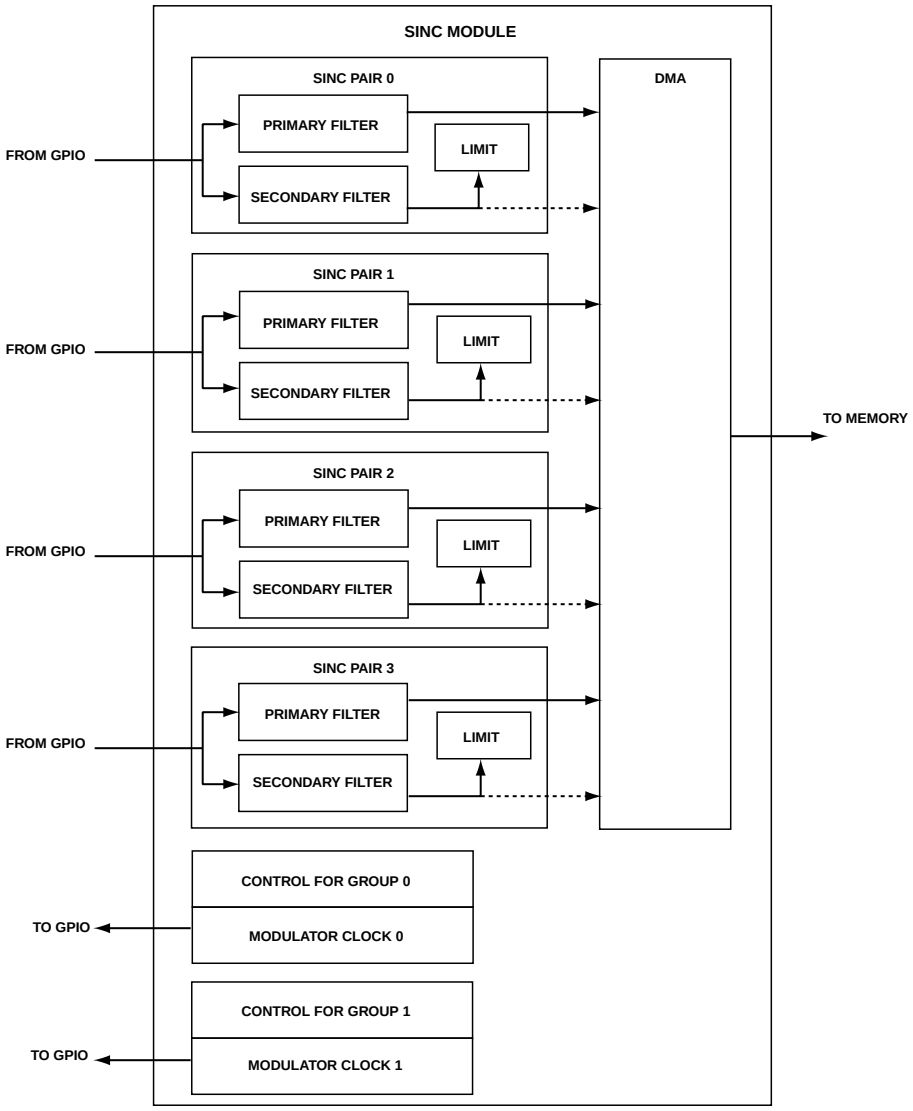


Figure 34-1: SINC Block Diagram

NOTE: For the ADSP-CM41x, only one of the two MCLK outputs is available to connect using the pin mux, so only Group 0 can be used directly. If a system needs to use Group 1, (to have a different decimation rate than Group 0 for example), then the Group 1 MCLK1 must be configured to exactly match MCLK0, with the same settings and the same trigger source.

The block diagram shows four SINC filter pairs (SINC0–3), two modulator clock sources, and two banks of control registers (units). The module accepts four sigma-delta bit streams from the GPIO input pins and directs the modulator clock source of GROUP 0 to the GPIO output pin. A pulse-width modulation (PWM) signal synchronizes the modulator clocks to optimize system performance. Each SINC filter pair includes the primary filter, secondary filter, DMA interface, and overload limit detection functions.

The primary SINC filter transmits its data to memory using DMA. The secondary SINC filter generates overload signals, which can be routed through the trigger routing unit (TRU) to trip a PWM modulator and generate an interrupt request.

The SINC filter pairs can be assigned to either set of control units, where multiple channels of current or voltage-feedback share common filter parameters. The primary filters generate high-resolution signals for closing the feedback control loop. The secondary filters are for rapid-overload fault detection, require lower resolution, but a faster response. The primary and secondary filters have programmable order and decimation rates. The primary filters also have the programmable output gain stage, while the secondary filters have the programmable overload limit thresholds.

To use the primary and secondary filters, set up the filter parameters once, prior to using the filters. The feedback control algorithm reads the data from the primary filter directly from memory. A PWM interrupt request signal can generate the algorithm timing signal, or the SINC module generates a data trigger. The data history of the secondary filter is saved in buffer registers once an overload fault signal is detected. The data history supports fault diagnostics.

SINC Architectural Concepts

The architecture of the SINC includes the following:

- Digital Filter
- DC Gain and Data Resolution
- Frequency Response
- Output Scaling

Digital Filter

The SINC filter has a transfer function that lends itself to an implementation in digital logic, using a series of summation and decimation functions. The filter removes the modulator sample clock and recovers a digital value of the sampled signal. The filter design matches a bipolar SD modulator. The design produces a 50% pulse density for a 0V input, over 50% for positive inputs and less than 50% for negative inputs.

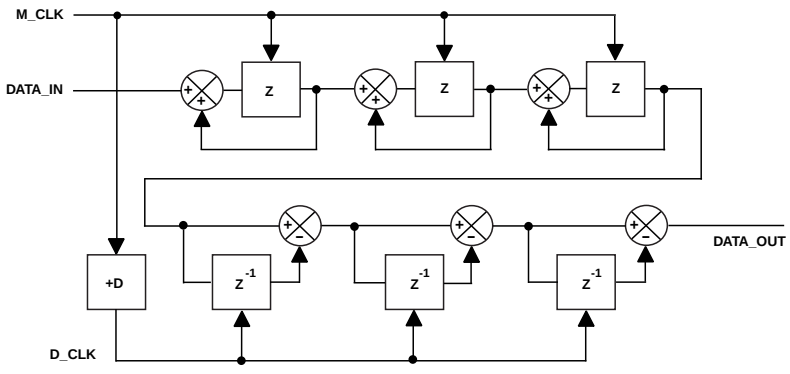


Figure 34-2: SINC Digital Filter

The digital filter is a set of accumulators driven by the modulator clock (M_CLK), followed by a set of differentiators driven by the decimation clock (D_CLK). The input accumulators convert the input bit stream into a multibyte word, while the output differentiators derive the average 1's density of the bit stream. The number of accumulator and differentiator stages can be three or four, depending on the order of the filter. The DC gain and bandwidth of the filter are functions of the filter order (O) and the decimation rate (D), which is the ratio of the modulator to the decimation clock.

The calculation of the transfer function of the SINC filter includes the product of the transfer functions for the accumulators and differentiators, and in the z domain. The following equation gives the calculation:

$$H(z) = \left[\frac{1}{D} \text{ x } \frac{1 - z^{-D}}{1 - z^{-1}}\right]^{O}$$

DC Gain and Data Resolution

The DC gain of the digital filter is a function of the order and decimation rate. At 100% ones density input, each accumulator stage counts D pulses, and the gain of the filter is given as follows:

$$G_{dc} = D^{o}$$

The higher the decimation rate, the higher the resolution of the output data. The number of usable data bits is a function of the SNR; the ***Filter Order versus Decimation*** table shows ENOB versus the decimation rate.

Table 34-5: ENOB versus Decimation

Decimation		4	5	6	7	8	16	32	64	128	256
O = 3	SNR (dB)	6.42	11.47	16.41	20.57	23.55	35.02	48.59	62.26	76.46	89.59
	ENOB	0.8	1.6	2.4	3.1	3.6	5.5	7.8	10.0	12.4	14.6
O = 4	SNR (dB)	9.08	14.77	19.78	23.41	25.9	38.05	51.29	64.67	79.15	
	ENOB	1.2	2.1	3.1	3.6	4.0	6.0	8.2	10.4	12.8	

Notes: ENOB versus order and decimation rate.

Test conditions are for a 1.22 kHz tone and a 10 MHz modulator.

Frequency Response

The frequency response of the filter depends on the order, decimation rate, and modulator clock frequency, f_M. The equation is obtained by substituting $e^{j \text{ x } Ts}$ for z in the transfer function, where Ts is the period of the modulator clock:

$$H\left(e^{j\frac{\omega}{f_M}}\right) = \left[\frac{1}{D} \times \frac{\sin\left(D\frac{\omega}{2f_M}\right)}{\sin\left(\frac{\omega}{2f_M}\right)} \times e^{-j(D-1)\frac{\omega}{2f_M}}\right]^{O}$$

The filter has a linear phase response with a constant group delay given by:

$$\tau_d = \left(\frac{D-1}{2}\right)\frac{O}{f_M}$$

The ***Frequency Response*** plots show zeros at multiples the decimation frequency, where the `sin` term in the numerator goes to zero. This response makes it possible to remove some PWM ripple components from the motor current waveform by matching the decimation frequency to the PWM switching frequency. There are some limitations at lower PWM frequencies based on available decimation rates. High decimation rates limit the bandwidth of the control loop because of the phase delay, which is 3π radians at the decimation frequency.

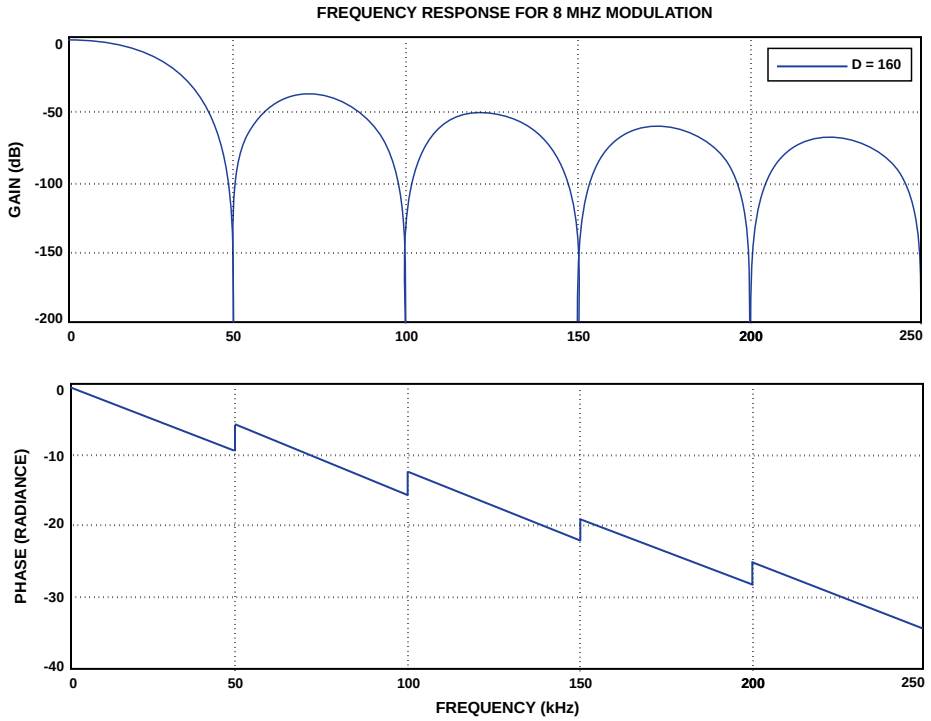


Figure 34-5: Frequency Response

Output Scaling

The output scaling and postprocessing functions embedded in the SINC filter blocks differ, depending on the function. The primary filter used for feedback signal processing includes the output bias and scaling blocks to present a 16-bit signed integer to the control code. The scaling is required at decimation rates higher than 32 to keep the lower 16 bits of the output word.

The secondary filter supports overload detection functions. The secondary filter can detect signals crossing maximum and minimum thresholds. It has a glitch filter that only accepts faults with a minimum number of counts (`c`) within a certain count window (`w`). The secondary filter has no output scaling, so the minimum and maximum values in the overload registers must be calculated from the DC gain of the secondary filter. The response time to a step input is approximately 2 x O decimation clock cycles.

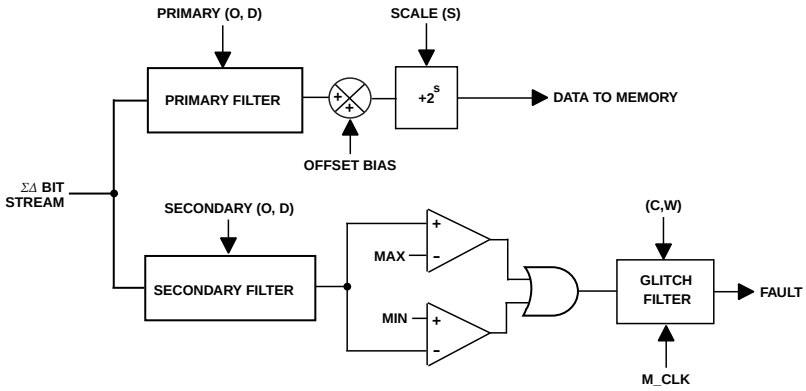


Figure 34-4: Output Scaling

SINC Operating Modes

The SINC filter module has only one operating mode. The module generates the clock source for a sigma-delta modulator analog front end and filters the output data stream for the modulator. The primary SINC filter transfers its data to memory through DMA. The secondary SINC filter output generates an overload trigger signal that the SINC filter module can use as a PWM trip signal. The SINC control registers enable the module and set up the modulator clock sources, filter parameters, DMA transfers, and interrupts masks, as described in SINC Programming Model.

For self-testing purposes, the SINC data inputs can be driven by bit streams transferred from the SPORT0A port through DMA from a slave memory space (typically M4 main memory). The following programming selections are required to enable SINC self-test mode:

- `SYSBLK_SINC_TEST.EN`[3:0] – set each bit to 1 for each SINC channel to be selected for test. If 1, the SINC channel input will be driven by the SPORT, not the associated GPIO.
- Enable filters to route SPORT data to SINC unit with the write: `*pREG_TESYS1_SINC_TEST = 0xf`
- The SINC unit must be configured as follows.
  - Timer Group 0; MCLK0
- The SPORT (A) must be configured as follows.
  - External Clock
  - 16 bit data recommended
  - No multichannel mode
  - Enable SPORT0_A in transmit mode, external clock, internal FS, and I2S Mode
    `*pREG_SPORT0_CTL_A = 0x021548f1`
- Any data transmitted on SPORT0 Data A channel is sent to SINC block.

SINC Data Transfer Modes

The only mode of data transfer between the primary SINC filter and memory is through DMA (See Primary DMA Configuration and Data Interrupts for more information. Reading the history registers for the secondary filter is the only way to transfer data between the secondary SINC filter and memory. See Overload Detection for more information.

SINC Signal Modes

The SINC filter has an interrupt request signal and a number of triggers and status signals to indicate system events and errors.

- Primary data transfer trigger:

  The SINC filter can generate a trigger after a user-specified number of primary output sets are transferred to memory. There is one trigger source for each filter group. See Primary DMA Configuration and Data Interrupts for more information.

- Secondary data overload trigger:

  The SINC filter can generate a trigger when one of the secondary filters detects an overload condition. There is one trigger source for each secondary filter. See Overload Detection for more information.

- SINC status bits:

  The SINC status bits indicate secondary filter overload errors, primary filter saturation errors, primary filter transfer count exceeded, and primary filter data buffer errors.

  - Secondary filter overload errors:

    A number of status bits indicate the type of error and the filter channel when a secondary filter detects an overload condition. The status bits `SINC_STAT.GLIM0` and `SINC_STAT.GLIM1` indicate the control group of the secondary filter that detected the overload. The status bits `SINC_STAT.MAX0` through `SINC_STAT.MAX3` indicate when the error a maximum limit on one of the secondary filter channels is passed, causing the error. The status bits `SINC_STAT.MIN0` through `SINC_STAT.MIN3` indicate when a minimum limit on one of the secondary filter channels is passed, causing the error.

  - Primary filter data saturation errors:

    A number of status bits indicate the group and filter channel when the SINC filter detects data saturation . The status bits `SINC_STAT.GSAT0` and `SINC_STAT.GSAT1` indicate the filter control group when the SINC filter detects data saturation. The status bits `SINC_STAT.PSAT0` through `SINC_STAT.PSAT3` indicate a primary filter channel that detects data saturation.

  - Primary filter transfer count exceeded:

    The status bits `SINC_STAT.PCNT0` and `SINC_STAT.PCNT1` are set every time a specified number of primary filter data sets for that filter group are transferred to memory. The primary filter data set for a group is the data for all the channels in the group. The specified number of data sets is the value in the

SINC_LEVEL0.PCNT-SINC_LEVEL1.PCNT bits. Write 1 to clear the bits before the next data transfer to generate a trigger.

- Primary filter data buffer errors:

  A number of status bits indicate data buffer errors. The status bits SINC_STAT.FOVF0 and SINC_STAT.FOVF1 indicate the filter control group when there is a SINC data buffer overflow. This error occurs when a third sample is presented to the buffer before the first sample transfers to memory. The status bits SINC_STAT.PFAB0 and SINC_STAT.PFAB1 indicate the filter group when an error occurs while writing the data to memory.

- SINC status interrupt request:

  There is a single SINC filter interrupt request output that can indicate secondary filter overload errors, primary filter data saturation, or primary filter data buffer overrun. There is one interrupt mask bit for each of these conditions per filter group. See Interrupt Masking for more information.

SINC Event Control

The SINC provides status and error bits through different registers to signal the core about its state and various error conditions that occur during its operation. These conditions include:

- Interrupt status related to data overload, data saturation, data FIFO fault conditions
- Error status related to SINC operations
- History status (which do not generate interrupts) related to data FIFO operations

SINC Interrupt Signals

The interrupt request and trigger signals to the SINC filter module include:

- One interrupt request signal, SINC_STAT, triggered by fault events, such as detected overload limits and data transfer errors. Manage interrupt request generation with the masking bits in the SINC_CTL register:
  - Bits SINC_CTL.ELIM0-SINC_CTL.ELIM1 can enable (unmask) interrupt request generation on overload faults when the SINC_STAT.GLIM0-SINC_STAT.GLIM1 bit is set, respectively.
  - Bits SINC_CTL.ESAT0-SINC_CTL.ESAT1 can mask interrupt request generation on data saturation faults when the SINC_STAT.GSAT0-SINC_STAT.GSAT1 bit is set, respectively.
  - Bits SINC_CTL.EFOVF0-SINC_CTL.EFOVF1 can mask interrupt request generation on data buffer overruns when the SINC_STAT.FOVF0-SINC_STAT.FOVF1 bit is set, respectively.

  The fault bits in the SINC_STAT register must be cleared to clear the interrupt request.
- Two data count triggers, one trigger per each control group. The SINC filter module regularly uses the data count triggers to generate a software interrupt request or trigger an event. First, set the SINC_CTL.EPCNT0 or SINC_CTL.EPCNT1 masking bit to enable the data count trigger. Then, the TRU must assign the data count master (SINC0_DATA0-1) to an interrupt request input.

- Four overload triggers, one trigger per each channel. The SINC filter module can use overload triggers to trip the appropriate PWM block in the case of a fault. The overload trigger is always enabled, and the TRU must assign the masters (SINC0_P0_OVLD through SINC0_P4_OVLD) to the appropriate PWM trip input slave (PWMn_TRIP_TRIGn).

SINC Status and Error Signals

The status and error signals related to SINC operations are as follows:

- SINC_STAT signals:
  - The amplitude and duration limit error signals for secondary SINC filters: SINC_STAT.MAX0 through SINC_STAT.MAX3, SINC_STAT.MIN0 through SINC_STAT.MIN3, and SINC_STAT.GLIM0-SINC_STAT.GLIM1.
  - The output saturation error signals for primary SINC filters: SINC_STAT.MAX0 through SINC_STAT.MAX3, SINC_STAT.MIN0 through SINC_STAT.MIN3, and SINC_STAT.GLIM0-SINC_STAT.GLIM1.
  - The output FIFO overflow error signals for primary SINC filters: SINC_STAT.FOVF0 and SINC_STAT.FOVF1.
  - The output count error signals for primary SINC filters: SINC_STAT.PCNT0 and SINC_STAT.PCNT1.
  - The SCB fabric-related error signals for primary SINC filters: SINC_STAT.PFAB0-SINC_STAT.PFAB1.
- SINC_CLK signals:
  - The phase shift signals for SINC modulator clocks: SINC_CLK.MREQ0-SINC_CLK.MREQ1.
- SINC_HIS_STAT signals:
  - The history saved signals for secondary SINC filters: SINC_HIS_STAT.P0HISPTR through SINC_HIS_STAT.P3HISPTR, which indicate that the data history of the filter is saved in buffer registers due to a detected overload error signal.

SINC Programming Model

The pin multiplexer enables the device input and output pins and connects the signals to the SINC module. Decide the filter grouping in advance. The filter parameters are defined according to the control register group.

Follow these steps to configure the filters:

1. Define the primary and secondary filter parameters by setting the appropriate bits in the control register for each filter channel group.

2. Set the upper and lower overload limits to maximum for each channel to avoid overload trips due to the filter startup transient.
3. Define the modulator clock frequency and startup mode.
4. Enable the SINC channels and assign them to the selected group of control registers.

Set the running overload limits after the filter settles, which is (order × decimation) modulator clock cycles after startup. When the filters are running, the module transfers its data to data RAM on the dedicated DMA channels. Once configured, the control registers do not need accessing, but the status and some data buffer registers typically are read after fault events.

In general, adjusting filter parameters during operation leads to unpredictable results. However, programs can write to the trigger and interrupt mask registers, as well as to the secondary threshold level registers during operation.

The DC gain of the converter subsystem depends on the gain of the input modulator (G_M), filter order (O), and decimation rate (D). The primary filter has an output binary scalar (S) to fit data into a 16-bit range:

$$G_M = 0.625 \times (D^O \div 2^S)$$

SINC Programming Concepts

Using the features and event control for the SINC to their greatest potential requires an understanding of some SINC-related concepts:

- Channel Configuration
- Trigger Masking
- Interrupt Masking
- Modulator Clock
- Filter Configuration
- Primary Filter Parameters
- Primary DMA Configuration and Data Interrupts
- Secondary Filter Parameters
- Overload Detection

Channel Configuration

The control bits, SINC_CTL.EN0 through SINC_CTL.EN3, configure SINC module channels. These control bits enable or disable the selected SINC filter channel and assign the channel to one of the two control register groups. The selected control register group also determines the filter clock source.

Trigger Masking

The SINC module has two data count triggers, one trigger per group. The module can use the data count triggers to generate a software interrupt regularly or trigger an event. First, set the SINC_CTL.EPCNT0 and SINC_CTL.EPCNT1 masking bits to enable the data count trigger. Then, the TRU must assign the data count master (SINC_DATn) to an interrupt input.

There are also four overload triggers, one trigger per each channel. The SINC module can use overload triggers to trip the appropriate PWM block when there is a fault. The overload trigger is always enabled, and the TRU must assign the masters (SINC0_Pn_OVLD) to the appropriate PWM trip input slave (PWMn_TRIP_TRIGn).

Interrupt Masking

The SINC filter can generate a SINC_STAT interrupt request signal when triggered by fault events, such as detected overload limits or data transfer errors.

Enable (unmask) interrupt request generation with the SINC_CTL register bits:

- Bits SINC_CTL.ELIM0-SINC_CTL.ELIM1 can enable interrupt request generation on overload faults when the SINC_STAT.GLIM0-SINC_STAT.GLIM1 bit is set, respectively.
- Bits SINC_CTL.ESAT0-SINC_CTL.ESAT1 can enable interrupt request generation on data saturation faults when the SINC_STAT.GSAT0-SINC_STAT.GSAT1 bit is set, respectively.
- Bits SINC_CTL.EFOVF0-SINC_CTL.EFOVF1 can enable interrupt request generation on data buffer overruns when the SINC_STAT.FOVF0-SINC_STAT.FOVF1 bit is set, respectively.

The fault bits in the SINC_STAT register must be cleared to clear the interrupt request.

Modulator Clock

The SINC filter has two modulator clock sources. Out of the two modulator clock sources, only the modulator clock for GROUP 0 is available on the GPIO port. Each clock source can be set with an output frequency in the range of 1-20 MHz. The SINC module uses bits in the SINC_CLK register to control the modulator clock output, frequency, and phase. The modulator clocks are assigned to the SINC filter channels according to their control group assignments. The SINC module uses the SINC_CLK.MCEN0-SINC_CLK.MCEN1 bit fields to enable the modulator clocks and control the startup behavior of the clock. Start the clock immediately or enable the clock on the first rising edge of an external trigger connected to the SINC0_SYNCn input of the module. This action synchronizes the modulator clock with a PWM waveform source by routing a PWMn_SYNC master to a SINC0_SYNC0 or SINC0_SYNC1 slave using the TRU.

The target frequency is in the range and derived from SYSCLK using an integer divisor in the SINC_CLK.MDIV0 or SINC_CLK.MDIV1 bits. Write to the SINC_CLK.MREQ0 or SINC_CLK.MREQ1 bit to adjust the phase of the clock. This adjustment lengthens the next clock period by the number of SCLK periods stored in the respective SINC_CLK.MADJ0 or SINC_CLK.MADJ1 bit field. The SINC_CLK.MREQ0 or SINC_CLK.MREQ1 bit is cleared automatically once the adjustment is complete.

Filter Configuration

Configure the primary and secondary filter parameters, according to the group number, by setting the appropriate bits in the SINC_RATE0 – SINC_RATE1, SINC_LEVEL0 – SINC_LEVEL1, and SINC_BIAS0 – SINC_BIAS1 control registers. Configure the DMA transfers by setting the appropriate bits in the SINC_PHEAD0 – SINC_PHEAD1 and SINC_PTAIL0 – SINC_PTAIL1 registers. Set the maximum and minimum levels for overload detection in the four limit registers, SINC_LIMIT0 – SINC_LIMIT3. Set the overload filtering parameters in the SINC_LEVEL0 – SINC_LEVEL1 registers.

Primary Filter Parameters

Set the primary filter to the 3rd or 4th order by the SINC_LEVEL0.PORD or SINC_LEVEL1.PORD bit assigned to the channel. Set the decimation rate for the primary filter using the SINC_RATE0.PDEC or SINC_RATE1.PDEC bits assigned to the channel. Valid decimation rates are in the range 4–256. Set the phase of the primary filter output relative to the number of modulator clocks after enabling the filter using the SINC_RATE0.PADJ or SINC_RATE1.PADJ bits assigned to the channel. Valid PADJ values are in the range 0 to PDEC - 1.

The raw filter output is a 32-bit wide integer, has an offset added, and is scaled to a 16-bit number before transfer to memory. Store the 32-bit two's compliment offset value in the SINC_BIAS0 or SINC_BIAS1 register of the channel. Set the binary scale factor by a mantissa in the range 4–32 stored in the SINC_LEVEL0.PSCALE or SINC_LEVEL1.PSCALE bits. The output is a valid 16-bit signed number. If the number is outside of the valid range, the output is saturated to 0x8000 or 0x7FFF, while the SINC_STAT.PSAT0 or SINC_STAT.GSAT1 fault bit (according to the channel group) is set.

Primary DMA Configuration and Data Interrupts

Transfer the primary SINC filter outputs to a circular buffer in data memory using DMA. There are separate DMA streams for each filter channel group. The output from the primary filter is interleaved with outputs from other primary filters in the same group. The interleaving order is from the lowest to the highest numbered filter.

The SINC module stores the circular buffer head address in the SINC_PHEAD0 or SINC_PHEAD1 register of the channel. It stores the tail address in the SINC_PTAIL0 or SINC_PTAIL1 register of the channel. The data address wraps around to the head address after the tail address is reached. The head and tail addresses must be 16-bit aligned and can be set to the same address. The SINC_PPTR0 or SINC_PPTR1 register of the channel is a read-only register that contains the address of the most recent primary SINC filter data. If there is an overflow condition in the SINC filter output data FIFO, due to a delay DMA transfer, the SINC_STAT.FOVF0, or SINC_STAT.FOVF1 fault bit (according to the channel group) is set.

A SINC data trigger can be generated after a user-specified number of primary filter outputs (data transfers) completes. Specify the data count value by the SINC_LEVEL0.PCNT or SINC_LEVEL1.PCNT bits assigned to the channel, and the trigger is generated every PCNT + 1 data transfers.

The *SINC Data Buffer Organization* figure shows the SINC data buffer organization. In the figure, SINC_OUT_X_M[n] is the data for the n th most recent sample in the M th channel in the filter group X, and n = 0 is the most recent data.

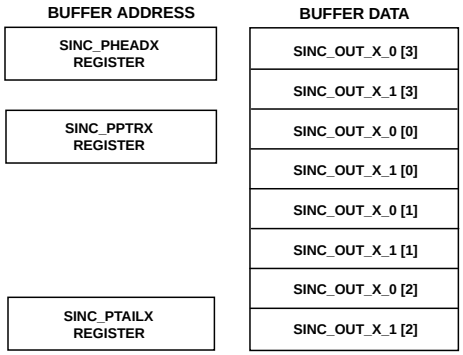


Figure 34-5: SINC Data Buffer Organization

Secondary Filter Parameters

Set the secondary filter to the 3rd or 4th order by the SINC_LEVEL0.SORD or SINC_LEVEL1.SORD bit assigned to the channel. Set the decimation rate for the secondary filter using the SINC_RATE0.SDEC or SINC_RATE1.SDEC bits assigned to the channel. The secondary filter outputs are limited to 16-bit values. Limit the decimation rate according to the filter order:

- Valid decimation rates are in the range 4–40 for the 3rd order filters
- Valid decimation rates are in the range of 4–16 for the 4th order filters

Set the phase of the primary filter output relative to the number of modulator clocks after using the SINC_RATE0.PADJ or SINC_RATE1.SADJ bits to enable the filter. Valid PADJ values are in the range 0 to SDEC - 1.

Overload Detection

The function of the secondary SINC filter is to detect AC current overload conditions and set up the upper and lower limit detection thresholds. There are event count filters on the overload detector outputs to reject short-term transients, if desired. Define the overload thresholds in four 32-bit registers SINC_LIMIT0-SINC_LIMIT3, according to the channel number. Each register contains the 16-bit LMAX and LMIN overload threshold values. The SINC filter module detects an overload condition when the secondary filter output exceeds the threshold for a minimum number of counts (LCNT) within the detection window (LWIN). Set the LCNT and LWIN count values in the SINC_LEVEL0 or SINC_LEVEL1 register assigned to the channel. When the SINC filter module detects an overload condition, the appropriate SINC0_Px_OVLD trigger is generated, and the SINC_STAT.GLIM0 or SINC_STAT.GLIM1 fault bit is set.

The SINC filter module saves the eight most recent data samples for the secondary filter in a local circular buffer to support diagnostics after a fault is triggered. Since 16-bit data is saved, only four buffer registers are required per channel. For example, SINC_P1SEC_HIST0-3 store the eight most recent 16-bit secondary filter outputs from

channel 1. The SINC_HIS_STAT register contains four pointers (SINC_HIS_STAT.P0HISPTR through SINC_HIS_STAT.P3HISPTR) to the buffer location of the most recent secondary current samples, per channel.

CM41X_M4 SINC Register Descriptions

SINC (SINC) contains the following registers.

Table 34-6: CM41X_M4 SINC Register List

Name	Description
SINC_BIAS0	Bias for Group 0 Register
SINC_BIAS1	Bias for Group 1 Register
SINC_CLK	Clock Control Register
SINC_CTL	Control Register
SINC_HIS_STAT	History Status Register
SINC_LEVEL0	Level Control for Group 0 Register
SINC_LEVEL1	Level Control for Group 1 Register
SINC_LIMIT0	(Amplitude) Limits for Secondary Filter 0 Register
SINC_LIMIT1	(Amplitude) Limits for Secondary Filter 1 Register
SINC_LIMIT2	(Amplitude) Limits for Secondary Filter 2 Register
SINC_LIMIT3	(Amplitude) Limits for Secondary Filter 3 Register
SINC_P0SEC_HIST[n]	Pair 0 Secondary (Filter) History n Register
SINC_P1SEC_HIST[n]	Pair 1 Secondary (Filter) History n Register
SINC_P2SEC_HIST[n]	Pair 2 Secondary (Filter) History n Register
SINC_P3SEC_HIST[n]	Pair 3 Secondary (Filter) History n Register
SINC_PHEAD0	Primary (Filters) Head for Group 0 Register
SINC_PHEAD1	Primary (Filters) Head for Group 1 Register
SINC_PPTR0	Primary (Filters) Pointer for Group 0 Register
SINC_PPTR1	Primary (Filters) Pointer for Group 1 Register
SINC_PTAIL0	Primary (Filters) Tail for Group 0 Register
SINC_PTAIL1	Primary (Filters) Tail for Group 1 Register
SINC_RATE0	Rate Control for Group 0 Register
SINC_RATE1	Rate Control for Group 1 Register
SINC_STAT	Status Register

Bias for Group 0 Register

The SINC_BIAS0 register controls an output bias added to primary SINC filters of group 0.

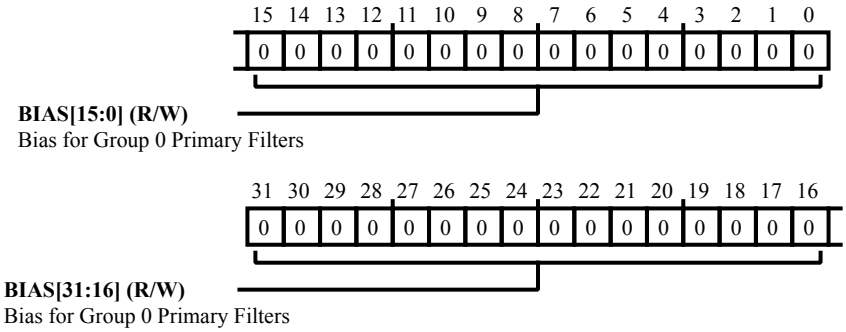


Figure 34-6: SINC_BIAS0 Register Diagram

Table 34-7: SINC_BIAS0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	BIAS	Bias for Group 0 Primary Filters. The SINC_BIAS0.BIAS bits specify a bias for the primary SINC filters output. The bias is added to the output prior to saturation and DMA memory transfer. The valid value is represented in two's complement format; thus, must be programmed to be equal to -(d ^ o) / 2, where d = SINC_RATE0.PDEC and o = SINC_LEVEL0.PORD.

Bias for Group 1 Register

The SINC_BIAS1 register controls an output bias added to primary SINC filters of group 1.

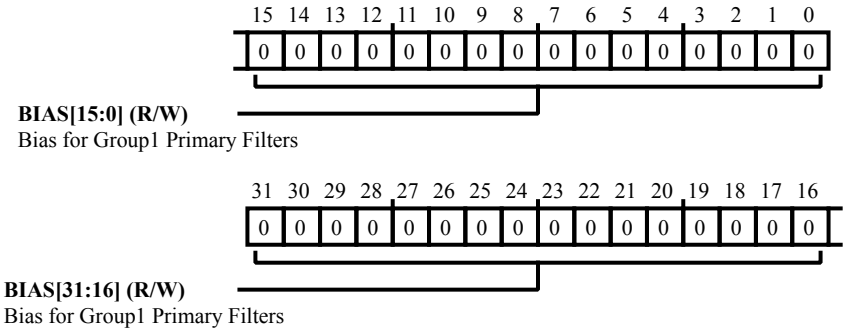


Figure 34-7: SINC_BIAS1 Register Diagram

Table 34-8: SINC_BIAS1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	BIAS	Bias for Group1 Primary Filters. The SINC_BIAS1.BIAS bits specify a bias for the primary SINC filters output. The bias is added to the output prior to saturation and DMA memory transfer. The valid value is represented in two's complement format; thus, must be programmed to be equal to -(d ^ o) / 2, where where d = SINC_RATE1.PDEC and o = SINC_LEVEL1.PORD.

Clock Control Register

The SINC_CLK register generates and enables two SINC modulator clocks. The register also controls each clock's output, frequency, phase, and start-up behavior.

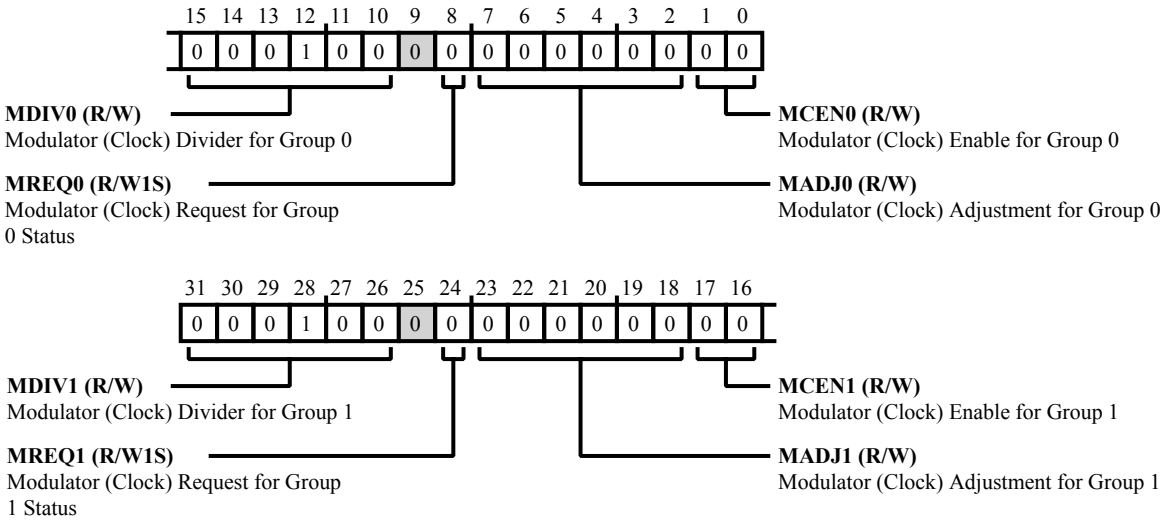


Figure 34-8: SINC_CLK Register Diagram

Table 34-9: SINC_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:26 (R/W)	MDIV1	Modulator (Clock) Divider for Group 1. The SINC_CLK.MDIV1 bits provide the SCLK divider to generate the modulator clock for group 1. The valid value is between 1 and 63.	
24 (R/W1S)	MREQ1	Modulator (Clock) Request for Group 1 Status. The SINC_CLK.MREQ1 bit indicates status for a phase shift request of the modulator clock for group 1. If the bits state is changed from clear (=0) to set (=1), the following modulator clock 1 period is lengthened by the number of SCLK periods specified by the SINC_CLK.MADJ1 bits. Any writes to this bit while the bit is set are ignored. The bit is cleared by hardware (and only by hardware) once a requested modulator clock adjustment is complete.	
		0	Inactive
		1	Active
23:18 (R/W)	MADJ1	Modulator (Clock) Adjustment for Group 1. The SINC_CLK.MADJ1 bits provide the adjustment value for the modulator clock of group 1. The valid value is between 1 and 63 when SINC_CLK.MREQ1 is set (=1). A write to this bit field effects only an active modulator clock adjustment. See the SINC_CLK.MREQ1 bit filed description.	

Table 34-9: SINC_CLK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17:16 (R/W)	MCEN1	Modulator (Clock) Enable for Group 1. The SINC_CLK.MCEN1 bits enable/disable the modulator clock for group 1 and control the clocks start-up behavior. Commence the clock immediately upon making it enabled, or enable and commence upon the next rising edge of PWMSYNC (PWM synchronizing output clock).	
		0	Disable
		1	Reserved
		2	Enable and Commence
		3	Enable and Commence on Next Rising Edge
15:10 (R/W)	MDIV0	Modulator (Clock) Divider for Group 0. The SINC_CLK.MDIV0 bits provide the SCLK divider to generate the modulator clock for group 0. The valid value is between 1 and 63.	
8 (R/W1S)	MREQ0	Modulator (Clock) Request for Group 0 Status. The SINC_CLK.MREQ0 bit indicates status for a phase shift request of the modulator clock for group 0. If the bits state is changed from clear (=0) to set (=1), the following modulator clock 0 period is lengthened by the number of SCLK periods specified by the SINC_CLK.MADJ0 bits. Any writes to this bit while the bit is set are ignored. The bit is cleared by hardware (and only by hardware) once a requested modulator clock adjustment is complete.	
		0	Inactive
		1	Active
7:2 (R/W)	MADJ0	Modulator (Clock) Adjustment for Group 0. The SINC_CLK.MADJ0 bits provide the adjustment value for the modulator clock of group 0. The valid value is between 1 and 63 when SINC_CLK.MREQ1 is set (=1). A write to this bit field effects only an active modulator clock adjustment. See the SINC_CLK.MREQ1 bit filed description.	
1:0 (R/W)	MCEN0	Modulator (Clock) Enable for Group 0. The SINC_CLK.MCEN0 bits enable/disable the modulator clock for group 0 and control the clocks start-up behavior. Commence the clock immediately upon making it enabled, or enable and commence upon the next rising edge of PWMSYNC (PWM synchronizing output clock).	
		0	Disable
		1	Reserved
		2	Enable and Commence
		3	Enable and Commence on Next Rising Edge

Control Register

The SINC_CTL register masks (disables) and unmasks (enables) SINC high-level interrupt request output signals triggered by fault events. The register also enables and assigns SINC filter pairs to one of two control groups.

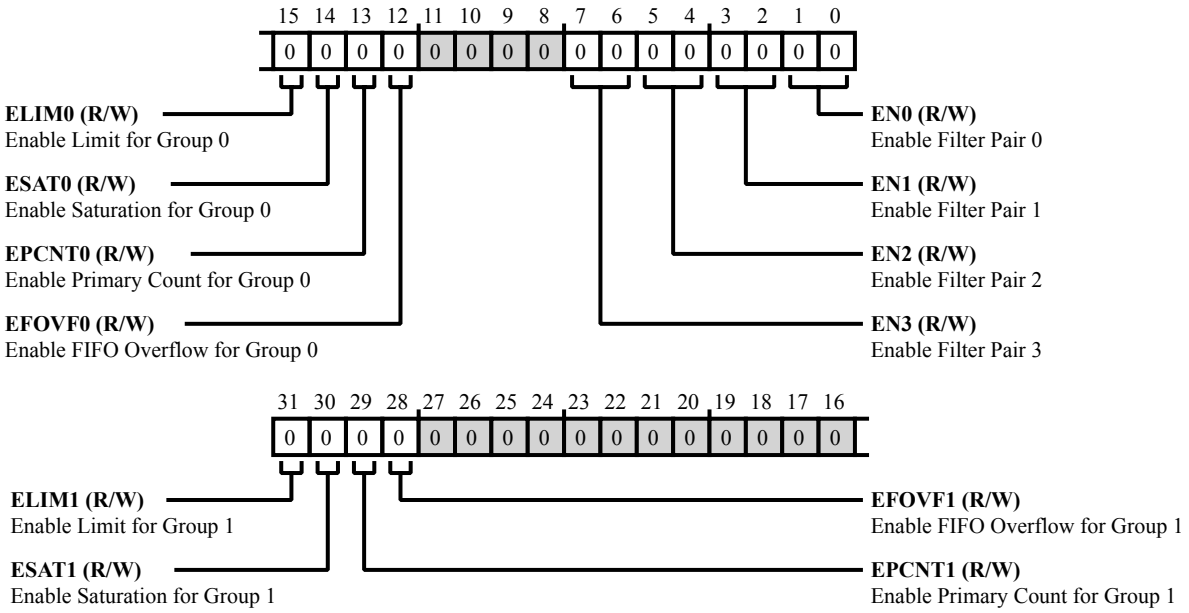


Figure 34-9: SINC_CTL Register Diagram

Table 34-10: SINC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	ELIM1	Enable Limit for Group 1. The SINC_CTL.ELIM1 bit enables (unmasks) the SINC_STAT interrupt request on overload conditions if this bit and status bit SINC_STAT.GLIM1 are set (=1).	
		0	Disable
		1	Enable
30 (R/W)	ESAT1	Enable Saturation for Group 1. The SINC_CTL.ESAT1 bit enables (unmasks) the SINC_STAT interrupt request on output saturation conditions if this bit and bit SINC_STAT.GSAT1 are set (=1).	
		0	Disable
		1	Enable

Table 34-10: SINC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	EPCNT1	Enable Primary Count for Group 1. The SINC_CTL.EPCNT1 bit enables a trigger event on each SINC_DATA1 interrupt request if this bit and status bit SINC_STAT.PCNT1 are set (=1).	
		0	Disable
		1	Enable
28 (R/W)	EFOVF1	Enable FIFO Overflow for Group 1. The SINC_CTL.EFOVF1 bit enables (unmasks) the SINC_STAT interrupt request on data FIFO overflow conditions if this bit and status bit SINC_STAT.FOVF1 are set (=1). The SINC_STAT.FOVF1 bit is set (=1) when the group 1 output data FIFO overflows due to delayed SCB fabric ready response.	
		0	Disable
		1	Enable
15 (R/W)	ELIM0	Enable Limit for Group 0. The SINC_CTL.ELIM0 bit enables (unmasks) the SINC_STAT interrupt request on overload conditions if this bit and status bit SINC_STAT.GLIM0 are set (=1).	
		0	Disable
		1	Enable
14 (R/W)	ESAT0	Enable Saturation for Group 0. The SINC_CTL.ESAT0 bit enables (unmasks) the SINC_STAT interrupt request on output saturation conditions if this bit and status bit SINC_STAT.GSAT0 are set (=1).	
		0	Disable
		1	Enable
13 (R/W)	EPCNT0	Enable Primary Count for Group 0. The SINC_CTL.EPCNT0 bit enables a trigger event on each SINC_DATA0 interrupt request if this bit and status bit SINC_STAT.PCNT0 are set (=1).	
		0	Disable
		1	Enable

Table 34-10: SINC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	EFOVF0	Enable FIFO Overflow for Group 0. The SINC_CTL.EFOVF0 bit enables (unmasks) the SINC_STAT interrupt request on data FIFO overflow conditions if this bit and status bit SINC_STAT.FOVF0 are set (=1). The SINC_STAT.FOVF0 bit is set (=1) when the group 0 output data FIFO overflows due to delayed SCB fabric ready response.	
		0	Disable
		1	Enable
7:6 (R/W)	EN3	Enable Filter Pair 3. The SINC_CTL.EN3 bits enable/disable and assign SINC filter pair 3 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved
5:4 (R/W)	EN2	Enable Filter Pair 2. The SINC_CTL.EN2 bits enable/disable and assign SINC filter pair 2 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved
3:2 (R/W)	EN1	Enable Filter Pair 1. The SINC_CTL.EN1 bits enable/disable and assign SINC filter pair 1 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved

Table 34-10: SINC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	EN0	Enable Filter Pair 0. The `SINC_CTL.EN0` bits enable/disable and assign SINC filter pair 0 to the control group.	
		0	Disable
		1	Reserved
		2	Enable and Assign to Group 0
		3	Reserved

History Status Register

The SINC_HIS_STAT provides status for data histories of secondary SINC filters, in the corresponding history buffer registers. The SINC history buffer registers save the most recent filter samples once an overload fault signal is detected.

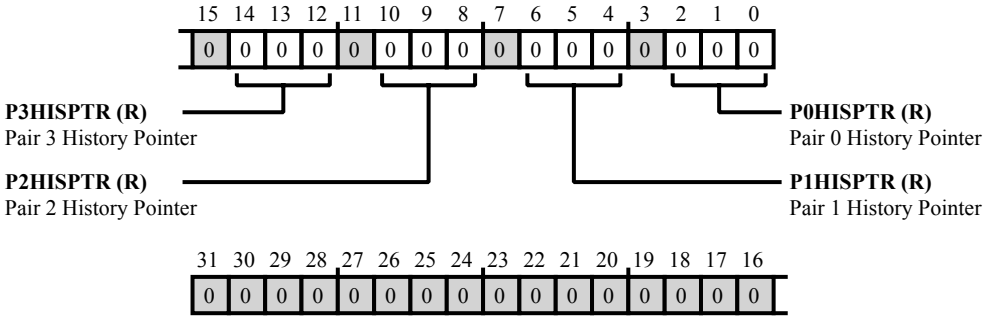


Figure 34-10: SINC_HIS_STAT Register Diagram

Table 34-11: SINC_HIS_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14:12 (R/NW)	P3HISPTR	Pair 3 History Pointer. The SINC_HIS_STAT.P3HISPTR bits indicate the position for the most recent data sample of secondary SINC filter 3 in the corresponding SINC_P3SEC_HIST[n] register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
10:8 (R/NW)	P2HISPTR	Pair 2 History Pointer. The SINC_HIS_STAT.P2HISPTR bits indicate the position for the most recent data sample of secondary SINC filter 2 in the corresponding SINC_P2SEC_HIST[n] register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS

Table 34-11: SINC_HIS_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
6:4 (R/NW)	P1HISPTR	Pair 1 History Pointer. The SINC_HIS_STAT.P1HISPTR bits indicate the position for the most recent data sample of secondary SINC filter 1 in the corresponding SINC_P1SEC_HIST[n] register block.	
		0	History Register 3, MS
		1	History Register, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS
2:0 (R/NW)	P0HISPTR	Pair 0 History Pointer. The SINC_HIS_STAT.P0HISPTR bits indicate the position for the most recent data sample of secondary SINC filter 0 in the corresponding SINC_P0SEC_HIST[n] register block.	
		0	History Register 3, MS
		1	History Register 0, LS
		2	History Register 0, MS
		3	History Register 1, LS
		4	History Register 1, MS
		5	History Register 2, LS
		6	History Register 2, MS
		7	History Register 3, LS

Level Control for Group 0 Register

The SINC_LEVEL0 register controls output scaling and count, excursion limit and window, as well as orders for primary and secondary SINC filters assigned to group 0.

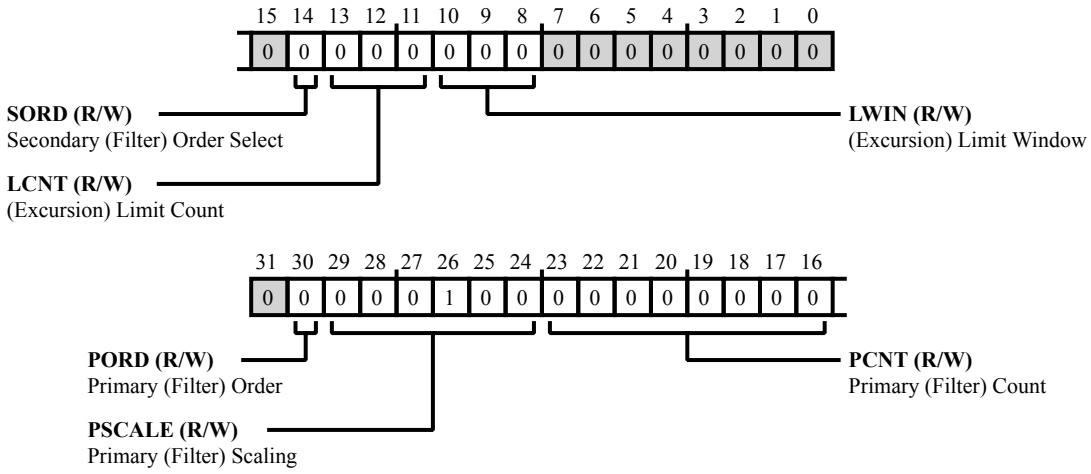


Figure 34-11: SINC_LEVEL0 Register Diagram

Table 34-12: SINC_LEVEL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	PORD	Primary (Filter) Order. The SINC_LEVEL0.PORD bit determines the order for group 1 primary filters.	
		0	Third Order
		1	Fourth Order

Table 34-12: SINC_LEVEL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29:24 (R/W)	PSCALE	Primary (Filter) Scaling. The SINC_LEVEL0.PSCALE bits specify the scaling applied to the output of group 0 primary filters, prior to DMA transfer to memory. The valid value is between 4 to 32. The SINC integrator, decimator, and bias adjustment produce an integer value up to 32 bits wide. The range of a full-scale signal of a bit stream filtered by a primary SINC filter is approximately (BIAS +- ((0.625 * SINC_RATE0.PDEC) ^ order)). The value requires about (ln2(SINC_RATE0.PDEC) * order) bits of precision (where 'order' is 3 or 4, as specified by the SINC_LEVEL0.PORD bit. This bit field specifies the bit position of the intermediate value, which is transferred on the MSB of 16-bit DMA sample. Thus, the intermediate value is right-shifted by (SINC_LEVEL0.PSCALE - 16) bits if SINC_LEVEL0.PSCALE >= 16, or left-shifted by (16 - SINC_LEVEL0.PSCALE) bits if SINC_LEVEL0.PSCALE < 16. If SINC_LEVEL0.PSCALE >= 16, thus selecting a right shift, the shifted value is rounded up (as if 0.5 * LSB is added) before truncation. Rounding is not necessary for a left shift. If the scaled and rounded value exceeds the range of a signed 16-bit number, the sample is saturated (to 0x8000 or 0x7FFF), and the corresponding saturation status bit (SINC_STAT.PSAT3, SINC_STAT.PSAT2, SINC_STAT.PSAT1, or SINC_STAT.PSAT0 is set.	
23:16 (R/W)	PCNT	Primary (Filter) Count. The SINC_LEVEL0.PCNT bits specify the modulo number of outputs for group 0 primary filters. The number must be one less than a desired modulo. Each time the number of outputs specified by this bit filed is transferred, the SINC_STAT.PCNT0 status bit is set (=1). When the SINC_STAT.PCNT0 bit is set (unless masked), it causes a TRU trigger. For example: 8'h00 written to the SINC_LEVEL0.PCNT bit field sets bit SINC_STAT.PCNT0 to 1 after every primary SINC filter output is transferred. 8'hFF written to the SINC_LEVEL0.PCNT bit field sets bit SINC_STAT.PCNT0 to 1 after every 256 primary SINC filter outputs transferred.	
14 (R/W)	SORD	Secondary (Filter) Order Select. The SINC_LEVEL0.SORD bit determines the order for group 0 secondary filters.	
		0	Third Order
		1	Fourth Order

Table 34-12: SINC_LEVEL0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13:11 (R/W)	LCNT	(Excursion) Limit Count. The SINC_LEVEL0.LCNT bits specify the number (count) of output excursions beyond the amplitude specified for group 0 secondary filters. The number of excursions greater than specified by registers SINC_LIMIT3, SINC_LIMIT2, SINC_LIMIT2, and SINC_LIMIT0 is perceived as an overload and sets (=1) a corresponding MAX or MIN bit in the SINC_STAT register. The valid count is between 1 to 8. If the count is greater than SINC_LEVEL0.LWIN, the bit fields behavior is as it is equal to SINC_LEVEL0.LWIN. See SINC_LEVEL0.LWIN for details. The valid count must be one less than a desired count: =000 require one excursion above the amplitude limit =111 require eight excursions above the amplitude limit.
10:8 (R/W)	LWIN	(Excursion) Limit Window. The SINC_LEVEL0.LWIN bits specify the window size for excursion checking for group 0 secondary filters. The window size is the number of the most recent outputs to be included in a measurement specified by the SINC_LEVEL0.LCNT bits. The valid value must be one less than a desired count (1 to 8), meaning the valid value is 0 to 7.

Level Control for Group 1 Register

The SINC_LEVEL1 register controls output scaling and count, excursion limit and window, as well as orders for primary and secondary SINC filters assigned to group 1.

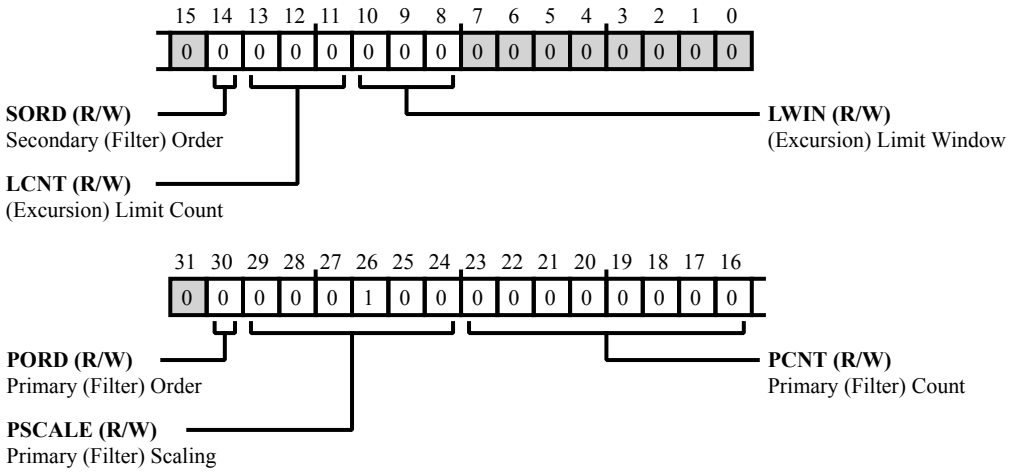


Figure 34-12: SINC_LEVEL1 Register Diagram

Table 34-13: SINC_LEVEL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	PORD	Primary (Filter) Order. The SINC_LEVEL1.PORD bits determines the order for group 1 primary filters.	
		0	Third Order
		1	Fourth Order

Table 34-13: SINC_LEVEL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/W)	PSCALE	Primary (Filter) Scaling. The SINC_LEVEL1.PSCALE bits specify the scaling applied to the output of group 1 primary filters, prior to DMA transfer to memory. The valid value is between 4 to 32. The SINC integrator, decimator, and bias adjustment produce an integer value up to 32 bits wide. The range of a full-scale signal of a bit stream filtered by a primary SINC filter is approximately (BIAS +- ((0.625 * SINC_RATE1.PDEC) ^ order)). The value requires about (ln2(SINC_RATE1.PDEC) * order) bits of precision (where 'order' is 3 or 4, as specified by the SINC_LEVEL1.PORD bit. This bit field specifies the bit position of the intermediate value, which is transferred on the MSB of 16-bit DMA sample. Thus, the intermediate value is right-shifted by (SINC_LEVEL1.PSCALE - 16) bits if SINC_LEVEL1.PSCALE >= 16, or left-shifted by (16 - SINC_LEVEL1.PSCALE) bits if SINC_LEVEL1.PSCALE < 16. If SINC_LEVEL1.PSCALE >= 16, thus selecting a right shift, the shifted value is rounded up (as if 0.5 * LSB is added) before truncation. Rounding is not necessary for a left shift. If the scaled and rounded value exceeds the range of a signed 16-bit number, the sample is saturated (to 0x8000 or 0x7FFF), and the corresponding saturation status bit (SINC_STAT.PSAT3, SINC_STAT.PSAT2, SINC_STAT.PSAT1, or SINC_STAT.PSAT0 is set.
23:16 (R/W)	PCNT	Primary (Filter) Count. The SINC_LEVEL1.PCNT bits specify the modulo number of outputs for group 1 primary filters. The number must be one less than a desired modulo. Each time the number of outputs specified by this bit filed is transferred, the SINC_STAT.PCNT1 status bit is set (=1). When the SINC_STAT.PCNT1 bit is set (unless masked), it causes a TRU trigger. For example: 8'h00 written to the SINC_LEVEL1.PCNT bit field sets bit SINC_STAT.PCNT1 to 1 after every primary SINC filter output is transferred. 8'hFF written to the SINC_LEVEL1.PCNT bit field sets bit SINC_STAT.PCNT1 to 1 after every 256 primary SINC filter outputs transferred.
14 (R/W)	SORD	Secondary (Filter) Order. The SINC_LEVEL1.SORD bit determines the order for group 1 secondary filters. The SINC_LEVEL1.SORD bit determines the order for group 1 secondary filters. 0 – Third Order 1 – Fourth Order

Table 34-13: SINC_LEVEL1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
13:11 (R/W)	LCNT	(Excursion) Limit Count. The SINC_LEVEL1.LCNT bits specify the number (count) of output excursions beyond the amplitude specified for group 1 secondary filters. The number of excursions greater than specified by registers SINC_LIMIT3, SINC_LIMIT2, SINC_LIMIT2, and SINC_LIMIT0 is perceived as an overload and sets (=1) a corresponding MAX or MIN bit in the SINC_STAT register. The valid count is between 1 to 8. If the count is greater than SINC_LEVEL1.LWIN, the bit fields behavior is as it is equal to SINC_LEVEL1.LWIN. See SINC_LEVEL1.LWIN for details. The valid count must be one less than a desired count: =000 require one excursion above the amplitude limit =111 require eight excursions above the amplitude limit.
10:8 (R/W)	LWIN	(Excursion) Limit Window. The SINC_LEVEL1.LWIN bits specify the window size for excursion checking for group 1 secondary filters. The window size is the number of the most recent outputs to be included in a measurement specified by the SINC_LEVEL1.LCNT bits. The valid value must be one less than a desired count (1 to 8), meaning the valid value is 0 to 7.

(Amplitude) Limits for Secondary Filter 0 Register

The SINC_LIMIT0 register controls amplitude limits for a secondary filter of SINC pair 0.

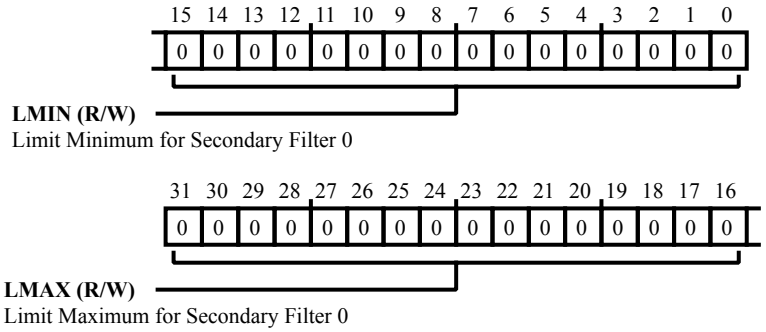


Figure 34-13: SINC_LIMIT0 Register Diagram

Table 34-14: SINC_LIMIT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 0. The SINC_LIMIT0.LMAX bits specify the output signal conditions for the secondary SINC filter 0. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 0. The SINC_LIMIT0.LMIN bits specify the output signal conditions for the secondary SINC filter 0. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

(Amplitude) Limits for Secondary Filter 1 Register

The SINC_LIMIT1 register controls amplitude limits for a secondary filter of SINC pair 1.

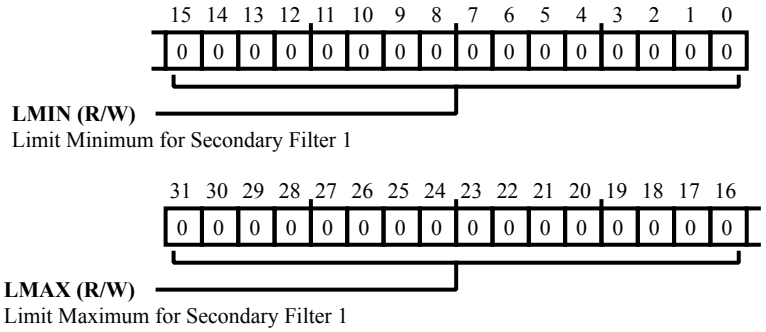


Figure 34-14: SINC_LIMIT1 Register Diagram

Table 34-15: SINC_LIMIT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 1. The SINC_LIMIT1.LMAX bits specify the output signal conditions for the secondary SINC filter 1. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 1. The SINC_LIMIT1.LMIN bits specify the output signal conditions for the secondary SINC filter 1. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

(Amplitude) Limits for Secondary Filter 2 Register

The SINC_LIMIT2 register controls amplitude limits for a secondary filter of SINC pair 2.

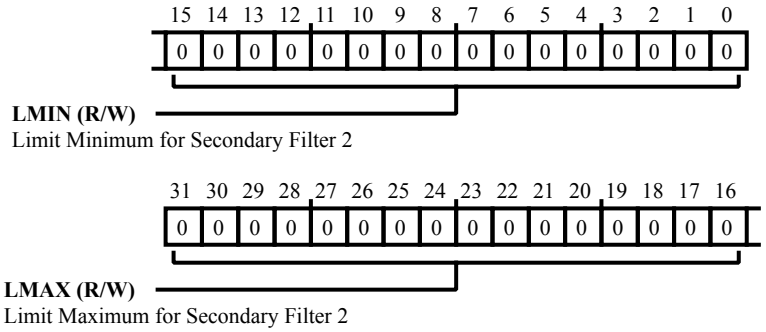


Figure 34-15: SINC_LIMIT2 Register Diagram

Table 34-16: SINC_LIMIT2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 2. The SINC_LIMIT2.LMAX bits specify the output signal conditions for the secondary SINC filter 2. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 2. The SINC_LIMIT2.LMIN bits specify the output signal conditions for the secondary SINC filter 2. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

(Amplitude) Limits for Secondary Filter 3 Register

The SINC_LIMIT3 register controls amplitude limits for a secondary filter of SINC pair 3.

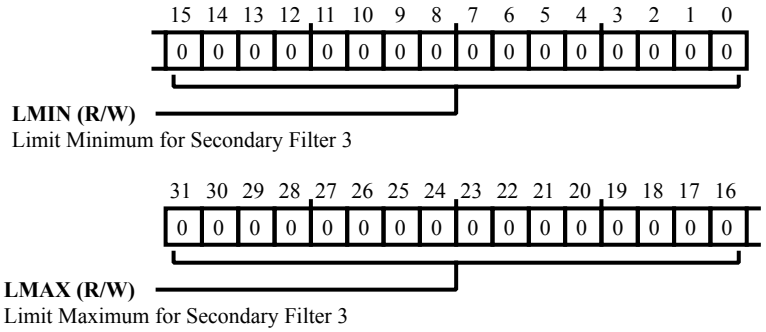


Figure 34-16: SINC_LIMIT3 Register Diagram

Table 34-17: SINC_LIMIT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	LMAX	Limit Maximum for Secondary Filter 3. The SINC_LIMIT3.LMAX bits specify the output signal conditions for the secondary SINC filter 3. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated maximum limit warning bit in register SINC_STAT.
15:0 (R/W)	LMIN	Limit Minimum for Secondary Filter 3. The SINC_LIMIT3.LMIN bits specify the output signal conditions for the secondary SINC filter 3. In conjunction with bits LCNT and LWIN in register SINC_LEVEL1 or SINC_LEVEL0, this bit field specifies conditions for an associated minimum limit warning bit in register SINC_STAT.

Pair 0 Secondary (Filter) History n Register

The SINC_P0SEC_HIST[n] read-only register provides the eight most recent samples produced by secondary SINC filter 0. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first SINC_P0SEC_HIST[n] register. The stored values, one compared to the limit, count, and window settings, set the SINC_STAT.MAX0 and SINC_STAT.MIN0 bits.

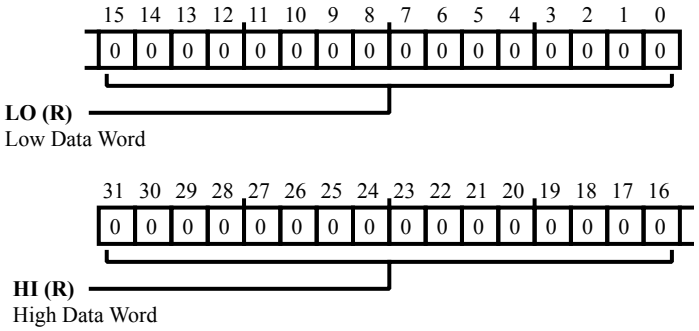


Figure 34-17: SINC_P0SEC_HIST[n] Register Diagram

Table 34-18: SINC_P0SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The SINC_P0SEC_HIST[n].HI bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The SINC_P0SEC_HIST[n].LO bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 1 Secondary (Filter) History n Register

The SINC_P1SEC_HIST[n] read-only register provides the eight most recent samples produced by secondary SINC filter 1. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first SINC_P1SEC_HIST[n] register. The stored values, compared to the limit, count, and window settings, set the SINC_STAT.MAX1 and SINC_STAT.MIN1 bits.

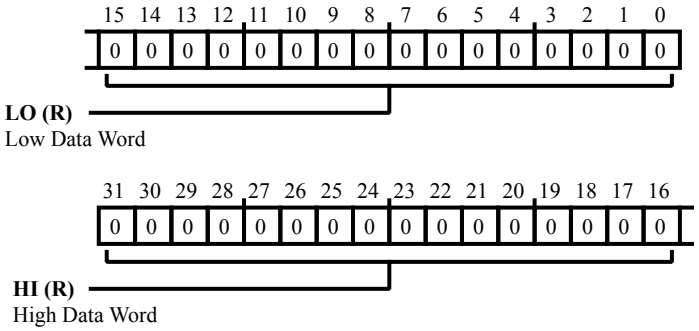


Figure 34-18: SINC_P1SEC_HIST[n] Register Diagram

Table 34-19: SINC_P1SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The SINC_P1SEC_HIST[n].HI bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The SINC_P1SEC_HIST[n].LO bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 2 Secondary (Filter) History n Register

The SINC_P2SEC_HIST[n] read-only register provides the eight most recent samples produced by secondary SINC filter 2. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first SINC_P2SEC_HIST[n] register. The stored values, compared to the limit, count, and window settings, set the SINC_STAT.MAX2 and SINC_STAT.MIN2 bits.

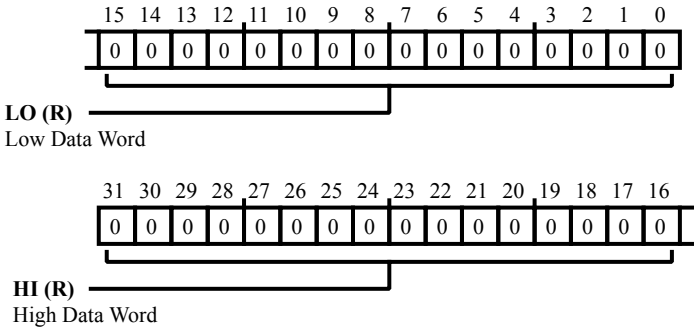


Figure 34-19: SINC_P2SEC_HIST[n] Register Diagram

Table 34-20: SINC_P2SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The SINC_P2SEC_HIST[n].HI bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The SINC_P2SEC_HIST[n].LO bits provide the 16-bit sample in the least significant half of the 32- bit register.

Pair 3 Secondary (Filter) History n Register

The SINC_P3SEC_HIST[n] read-only register provides the eight most recent samples produced by secondary SINC filter 3. The 16-bit samples are stored in the 32-bit register in circular manner, starting with the low-order field of the first SINC_P3SEC_HIST[n] register. The stored values, compared to the limit, count, and window settings, set the SINC_STAT.MAX3 and SINC_STAT.MIN3 bits.

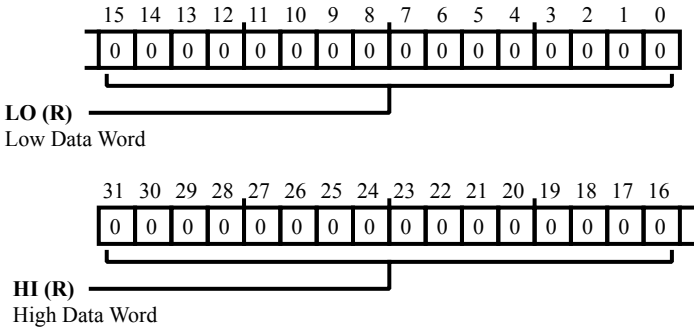


Figure 34-20: SINC_P3SEC_HIST[n] Register Diagram

Table 34-21: SINC_P3SEC_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	HI	High Data Word. The SINC_P3SEC_HIST[n].HI bits provide the 16-bit sample in the most significant half of the 32- bit register.
15:0 (R/NW)	LO	Low Data Word. The SINC_P3SEC_HIST[n].LO bits provide the 16-bit sample in the least significant half of the 32- bit register.

Primary (Filters) Head for Group 0 Register

The SINC_PHEAD0 register stores the head address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 0 assignments).

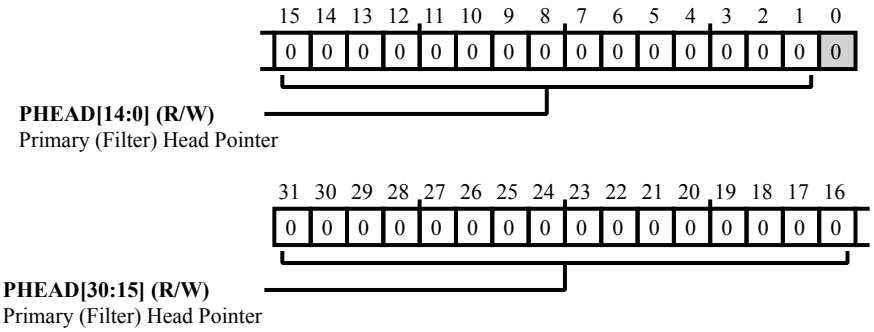


Figure 34-21: SINC_PHEAD0 Register Diagram

Table 34-22: SINC_PHEAD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:1 (R/W)	PHEAD	Primary (Filter) Head Pointer. The SINC_PHEAD0.PHEAD bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to SINC_PHEAD0.PHEAD after SINC_PTAIL0.PTAIL is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 0). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Head for Group 1 Register

The SINC_PHEAD1 register stores the head address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

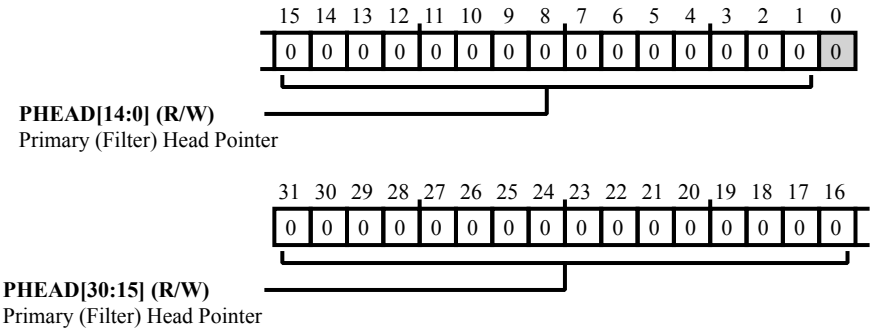


Figure 34-22: SINC_PHEAD1 Register Diagram

Table 34-23: SINC_PHEAD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:1 (R/W)	PHEAD	Primary (Filter) Head Pointer. The SINC_PHEAD1.PHEAD bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to SINC_PHEAD1.PHEAD after SINC_PTAIL1.PTAIL is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Pointer for Group 0 Register

The SINC_PPTR0 read-only register points to a circular buffer holding the most recent results of primary SINC filters, according to control group 0 assignments.

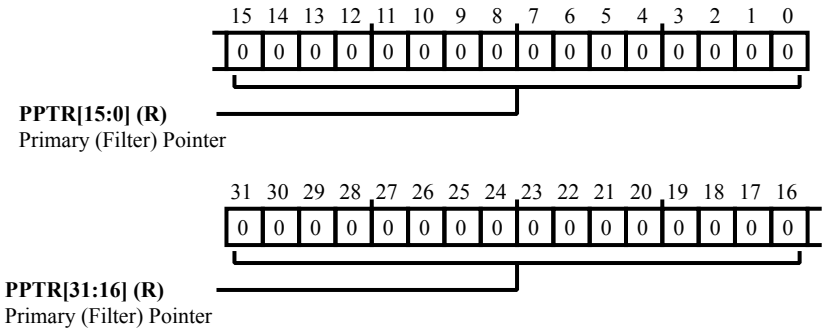


Figure 34-23: SINC_PPTR0 Register Diagram

Table 34-24: SINC_PPTR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	PPTR	Primary (Filter) Pointer. The SINC_PPTR0.PPTR bits hold the address for the last memory location of the most recent set of primary SINC filter results (group 0). The address is incremented once all of the primary SINC filter data (assigned to group 0 and associated to a particular time stamp) is successfully presented to the system fabric. Memory locations beyond the location reported by this register may be partially updated, so the entire circular buffer is not considered valid. Note that in real-time operation, due to fabric latency, write data may be in flight on the system fabric after the point when this bit field is updated. Thus, the write data may not be observed in memory until it has transited the fabric.

Primary (Filters) Pointer for Group 1 Register

The SINC_PPTR1 read-only register points to a circular buffer holding the most recent results of primary SINC filters, according to control group 1 assignments.

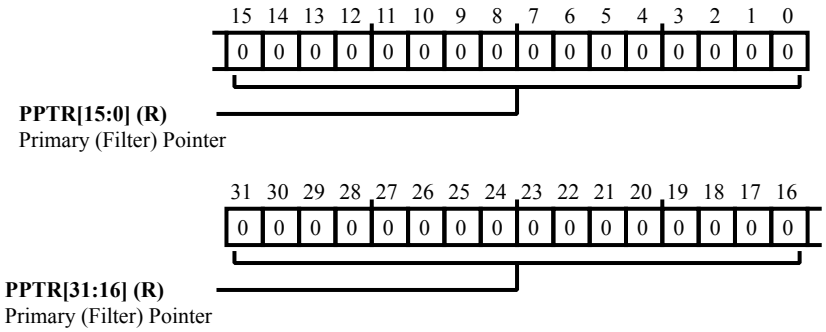


Figure 34-24: SINC_PPTR1 Register Diagram

Table 34-25: SINC_PPTR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	PPTR	Primary (Filter) Pointer. The SINC_PPTR1.PPTR bits hold the address for the last memory location of the most recent set of primary SINC filter results (group 1). The address is incremented once all of the primary SINC filter data (assigned to group 1 and associated to a particular time stamp) is successfully presented to the system fabric. Memory locations beyond the location reported by this register may be partially updated, so the entire circular buffer is not considered valid. Note that in real-time operation, due to fabric latency, write data may be in flight on the system fabric after the point when this bit field is updated. Thus, the write data may not be observed in memory until it has transited the fabric.

Primary (Filters) Tail for Group 0 Register

The SINC_PTAIL0 register stores the tail address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

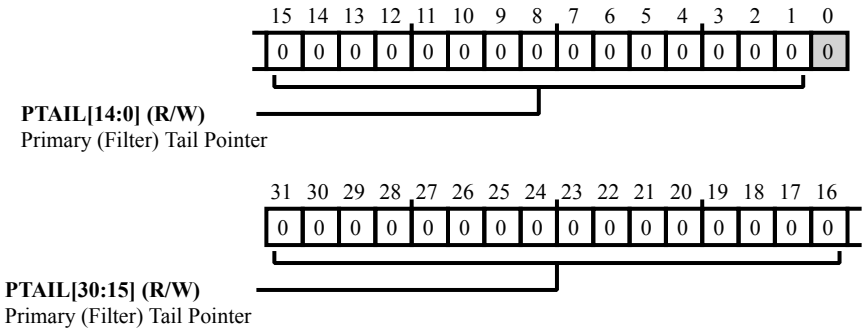


Figure 34-25: SINC_PTAIL0 Register Diagram

Table 34-26: SINC_PTAIL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:1 (R/W)	PTAIL	Primary (Filter) Tail Pointer. The SINC_PTAIL0.PTAIL bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to SINC_PHEAD0.PHEAD after SINC_PTAIL0.PTAIL is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Primary (Filters) Tail for Group 1 Register

The SINC_PTAIL1 register stores the tail address for a circular buffer in data memory to which to transfer the primary SINC filter outputs (according to control group 1 assignments).

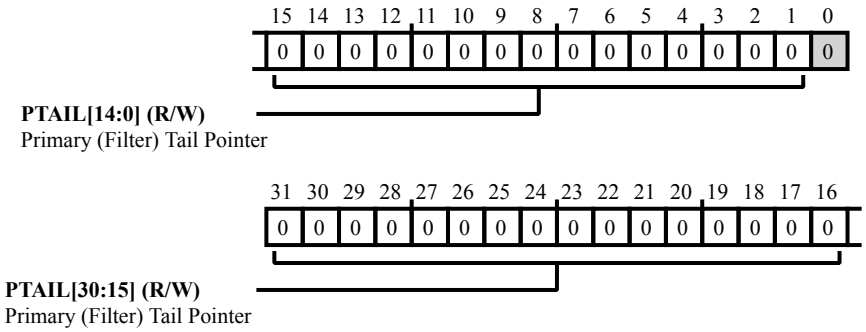


Figure 34-26: SINC_PTAIL1 Register Diagram

Table 34-27: SINC_PTAIL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:1 (R/W)	PTAIL	Primary (Filter) Tail Pointer. The SINC_PTAIL1.PTAIL bits hold the pointer (address) for DMA transfer to memory. Commencing at and wrapping back to SINC_PHEAD1.PHEAD after SINC_PTAIL1.PTAIL is reached, it forms a circular buffer, to which to transfer the primary SINC filter outputs (group 1). The valid address must be 16-bit aligned (address must be even).

Rate Control for Group 0 Register

The SINC_RATE0 register controls phase adjustments and decimation rates for primary and secondary SINC filters assigned to group 0.

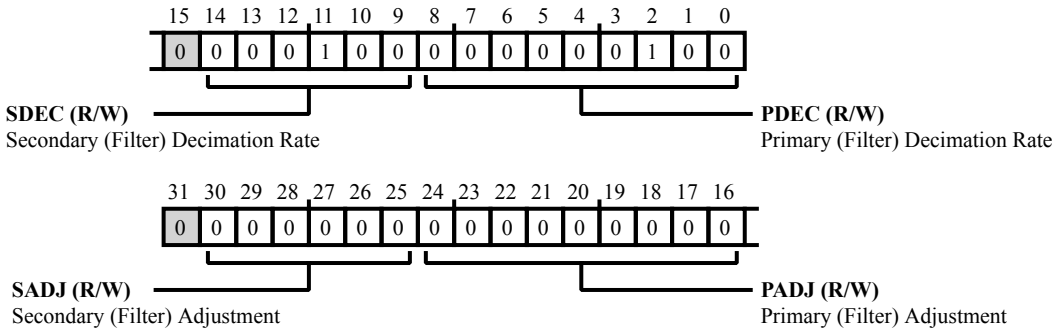


Figure 34-27: SINC_RATE0 Register Diagram

Table 34-28: SINC_RATE0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:25 (R/W)	SADJ	Secondary (Filter) Adjustment. The SINC_RATE0.SADJ bits provide the phase adjustment for the decimated output of group 0 secondary filters. The valid adjustment is between 0 and (SINC_RATE0.SDEC - 1), in modulator clock cycles, relative to the time the filter is enabled in the SINC_CTL register. The secondary SINC filter calculates an output in modulator clock cycle equivalent to ((SINC_RATE0.SDEC * n) - SINC_RATE0.SADJ), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.
24:16 (R/W)	PADJ	Primary (Filter) Adjustment. The SINC_RATE0.PADJ bits provide the phase adjustment for the decimated output of group 0 primary filters. The valid adjustment is between 0 and (SINC_RATE0.PDEC - 1), in modulator clock cycles, relative to the time the filter is enabled in the SINC_CTL register. The primary SINC filter calculates an output in modulator clock cycle equivalent to ((SINC_RATE0.PDEC * n) - SINC_RATE0.PADJ), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.

Table 34-28: SINC_RATE0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:9 (R/W)	SDEC	Secondary (Filter) Decimation Rate. The `SINC_RATE0.SDEC` bits provide the decimation rate for group 0 secondary filters. The valid range depends on the SINC order selected. If the third order (`SINC_LEVEL0.SORD` = 0), the valid range is 4 to 40. If the forth order (`SINC_LEVEL0.SORD` = 1), the valid rate is 4 to 16.
8:0 (R/W)	PDEC	Primary (Filter) Decimation Rate. The `SINC_RATE0.PDEC` bits provide the decimation rate for group 0 primary filters. The valid rate is 256 to 4.

Rate Control for Group 1 Register

The SINC_RATE1 register controls phase adjustments and decimation rates for primary and secondary SINC filters assigned to group 1.

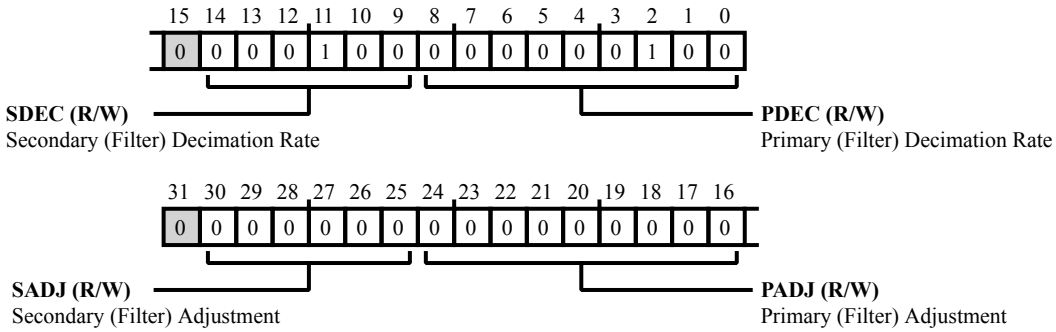


Figure 34-28: SINC_RATE1 Register Diagram

Table 34-29: SINC_RATE1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:25 (R/W)	SADJ	Secondary (Filter) Adjustment. The SINC_RATE1.SADJ bits provide the phase adjustment for the decimated output of group 1 secondary filters. The valid adjustment is between 0 and (SINC_RATE1.SDEC - 1), in modulator clock cycles, relative to the time the filter is enabled in the SINC_CTL register. The secondary SINC filter calculates an output in modulator clock cycle equivalent to ((SINC_RATE1.SDEC * n) - SINC_RATE1.SADJ), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.
24:16 (R/W)	PADJ	Primary (Filter) Adjustment. The SINC_RATE1.PADJ bits provide the phase adjustment for the decimated output of group 1 primary filters. The valid adjustment is between 0 and (SINC_RATE1.PDEC - 1), in modulator clock cycles, relative to the time the filter is enabled in the SINC_CTL register. The primary SINC filter calculates an output in modulator clock cycle equivalent to ((SINC_RATE1.PDEC * n) - SINC_RATE1.PADJ), where n is an integer > 1. This bit field can be changed while the filter is running and takes effect after the next decimation sample is generated. The effect of the change requires time to ripple through the filter: a number of output sample periods is equal to the filter order.

Table 34-29: SINC_RATE1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
14:9 (R/W)	SDEC	Secondary (Filter) Decimation Rate. The `SINC_RATE1.SDEC` bits provide the decimation rate for group 1 secondary filters. The valid range depends on the SINC order selected. If the third order (`SINC_LEVEL1.SORD` = 0), the valid range is 4 to 40. If the forth order (`SINC_LEVEL1.SORD` = 1), the valid rate is 4 to 16.
8:0 (R/W)	PDEC	Primary (Filter) Decimation Rate. The `SINC_RATE1.PDEC` bits provide the decimation rate for group 1 primary filters. The valid rate is 256 to 4.

Status Register

The SINC_STAT register indicates status for SINC output saturation, amplitude and duration limits, overload conditions, and data transfer errors.

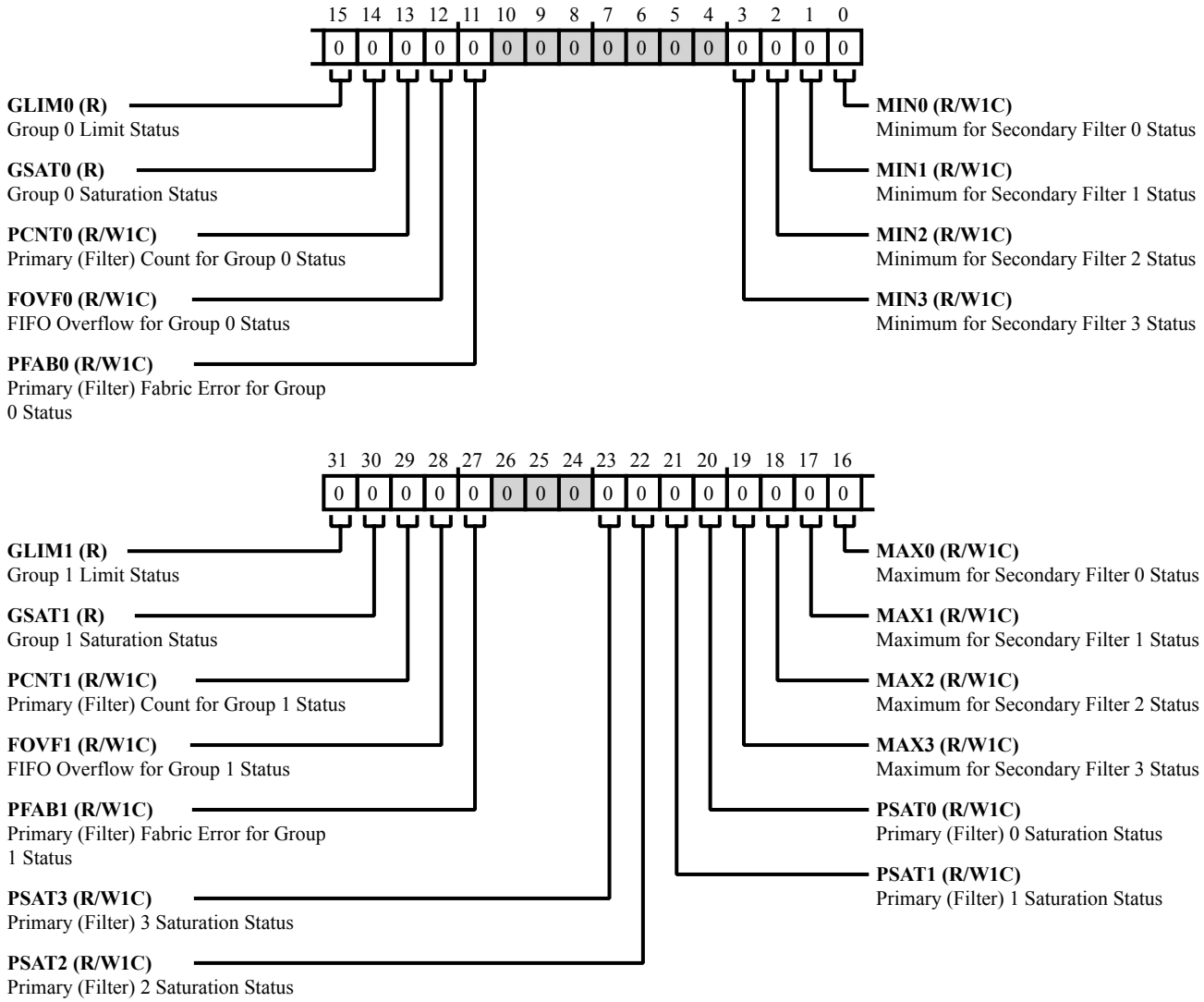


Figure 34-29: SINC_STAT Register Diagram

Table 34-30: SINC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/NW)	GLIM1	Group 1 Limit Status. The SINC_STAT.GLIM1 indicates status for an amplitude and duration limit of secondary SINC filters assigned to group 1. This bit is set (=1) if any limit specified by registers SINC_LIMIT3, SINC_LIMIT2, SINC_LIMIT1, or SINC_LIMIT0, within the duration count and window specified by bits SINC_LEVEL0.LCNT and SINC_LEVEL0.LWIN are exceeded. To identify the offending secondary SINC filter, examine the filters status bits SINC_STAT.MAX3, SINC_STAT.MAX2, SINC_STAT.MAX1, SINC_STAT.MAX0, SINC_STAT.MIN3, SINC_STAT.MIN2, SINC_STAT.MIN1 and SINC_STAT.MAX0 according to the group 1 assignments in the SINC_CTL register.	
		0	Not Exceeded
		1	Exceeded
30 (R/NW)	GSAT1	Group 1 Saturation Status. The SINC_STAT.GSAT1 indicates status for the output saturation bit of primary SINC filters assigned to group 1. The bit is set (=1) if any filter of group 1 has its saturation status bit set (=1). To identify the offending SINC primary filter, examine bits SINC_STAT.PSAT3, SINC_STAT.PSAT2, SINC_STAT.PSAT1, and SINC_STAT.PSAT0 according to the group 1 assignments specified by the SINC_CTL.EN3, SINC_CTL.EN2, SINC_CTL.EN1, and SINC_CTL.EN0 bits.	
		0	Not Set
		1	Set
29 (R/W1C)	PCNT1	Primary (Filter) Count for Group 1 Status. The SINC_STAT.PCNT1 indicates status for the output count of primary SINC filters assigned to group 1. The bit is set (=1) each time the modulo number of outputs (specified by the SINC_LEVEL1.PCNT bits) has been transferred for each primary SINC filter assigned to group 1. Each count in SINC_LEVEL1.PCNT corresponds to one complete set or vector of samples from all SINC filter pairs assigned to group 1. For example, if group 1 is assigned three SINC filters pairs 0, 1, and 3, and SINC_LEVEL1.PCNT is set to 5, then this status bit is set after the transfer of every 5th complete sample vector, comprising 3 x 5= 15 data samples. This bit asserts when the memory transfer on the system SCB fabric is complete, and a valid SCB write data response is received by the SINC filter unit. If this status bit and bit SINC_CTL.EPCNT1 are set (=1), the SINC_DATA1 trigger is asserted. Write 1 to clear.	
		0	Not Reached
		1	Reached

Table 34-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W1C)	FOVF1	FIFO Overflow for Group 1 Status. The SINC_STAT.FOVF1 indicates status for the data output FIFO bit of primary SINC filters assigned to group 1. This bit is set (= 1) if the output FIFO for any filter in group 1 overflows due to slow SCB fabric response. The FIFO for each primary SINC filter contains two data sample locations. An overflow occurs if a third data sample is generated before the first sample's data is transferred into the SCB fabric write data channel. After any overflow signaled by this bit occurs, all further SCB transmissions generated by group 1 are UNSPECIFIED until all SINC filters of the group are shut down and restarted. Clearing this status bit (=0) alone is not sufficient to re-sync the DMA stream. Write 1 to clear. If this status bit and bit SINC_CTL.EFOVF1 are set (=1), the SINC_STAT interrupt is asserted.	
		0	No Overflow
		1	Overflow
27 (R/W1C)	PFAB1	Primary (Filter) Fabric Error for Group 1 Status. The SINC_STAT.PFAB1 indicates error status for the output of any primary SINC filter assigned to group 1. The bit is set (=1) if the SCB fabric provides a write error response for a filter output transfer associated with group 1, or if an overrun occurs for a filter in group 1. An interrupt is requested whenever this bit =1 (not maskable).	
		0	Disabled
		1	Enabled
23 (R/W1C)	PSAT3	Primary (Filter) 3 Saturation Status. The SINC_STAT.PSAT3 bit indicates whether the primary SINC filter 3 requires saturation.	
		0	Not Saturated
		1	Saturated
22 (R/W1C)	PSAT2	Primary (Filter) 2 Saturation Status. The SINC_STAT.PSAT2 bit indicates whether the primary SINC filter 2 requires saturation.	
		0	Not Saturated
		1	Saturated

Table 34-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W1C)	PSAT1	Primary (Filter) 1 Saturation Status. The SINC_STAT.PSAT1 bit indicates whether the primary SINC filter 1 requires saturation.	
		0	Not Saturated
		1	Saturated
20 (R/W1C)	PSAT0	Primary (Filter) 0 Saturation Status. The SINC_STAT.PSAT0 bit indicates whether the primary SINC filter 0 requires saturation.	
		0	Not Saturated
		1	Saturated
19 (R/W1C)	MAX3	Maximum for Secondary Filter 3 Status. The SINC_STAT.MAX3 bit indicates whether the output of the secondary SINC filter 3 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT3.LMAX bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN3 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded
18 (R/W1C)	MAX2	Maximum for Secondary Filter 2 Status. The SINC_STAT.MAX2 bit indicates whether the output of the secondary SINC filter 2 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT2.LMAX bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN2 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded

Table 34-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W1C)	MAX1	Maximum for Secondary Filter 1 Status. The SINC_STAT.MAX1 bit indicates whether the output of the secondary SINC filter 0 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT1.LMAX bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN1 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded
16 (R/W1C)	MAX0	Maximum for Secondary Filter 0 Status. The SINC_STAT.MAX0 bit indicates whether the output of the secondary SINC filter 0 exceeded its maximum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT0.LMAX bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN0 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded

Table 34-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	GLIM0	Group 0 Limit Status. The SINC_STAT.GLIM0 indicates status for an amplitude and duration limit of secondary SINC filters assigned to group 0. This bit is set (=1) if any limit specified by registers SINC_LIMIT3, SINC_LIMIT2, SINC_LIMIT1, or SINC_LIMIT0, within the duration count and window specified by bits SINC_LEVEL1.LCNT and SINC_LEVEL1.LWIN are exceeded. To identify the offending secondary SINC filter, examine the filters status bits SINC_STAT.MAX3, SINC_STAT.MAX2, SINC_STAT.MAX1, SINC_STAT.MAX0, SINC_STAT.MIN3, SINC_STAT.MIN2, SINC_STAT.MIN1 and SINC_STAT.MAX0 according to the group 0 assignments in the SINC_CTL register.	
		0	Not Exceeded
		1	Exceeded
14 (R/NW)	GSAT0	Group 0 Saturation Status. The SINC_STAT.GSAT0 indicates status for the output saturation bit of primary SINC filters assigned to group 0. The bit is set (=1) if any filter of group 0 has its saturation status bit set (=1). To identify the offending SINC primary filter, examine bits SINC_STAT.PSAT3, SINC_STAT.PSAT2, SINC_STAT.PSAT1, and SINC_STAT.PSAT0 according to the group 0 assignments specified by the SINC_CTL.EN3, SINC_CTL.EN2, SINC_CTL.EN1, and SINC_CTL.EN0 bits.	
		0	Not Set
		1	Set
13 (R/W1C)	PCNT0	Primary (Filter) Count for Group 0 Status. The SINC_STAT.PCNT0 indicates status for the output count of primary SINC filters assigned to group 0. The bit is set (=1) each time the modulo number of outputs (specified by the SINC_LEVEL0.PCNT bits) has been transferred for each primary SINC filter assigned to group 0. Each count in SINC_LEVEL0.PCNT corresponds to one complete set or vector of samples from all SINC filter pairs assigned to group 1. For example, if group 0 is assigned three SINC filters pairs 0, 1, and 3, and SINC_LEVEL0.PCNT is set to 5, then this status bit is set after the transfer of every 5th complete sample vector, comprising 3 x 5= 15 data samples. This bit asserts when the memory transfer on the system SCB fabric is complete, and a valid SCB write data response is received by the SINC filter unit. If this status bit and bit SINC_CTL.EPCNT0 are set (=1), the SINC_DATA0 trigger is asserted. Write 1 to clear.	
		0	Not Reached
		1	Reached

Table 34-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	FOVF0	FIFO Overflow for Group 0 Status. The SINC_STAT.FOVF0 indicates status for the data output FIFO bit of primary SINC filters assigned to group 0. This bit is set (= 1) if the output FIFO for any filter in group 0 overflows due to slow SCB fabric response. The FIFO for each primary SINC filter contains two data sample locations. An overflow occurs if a third data sample is generated before the first sample's data is transferred into the SCB fabric write data channel. After any overflow signaled by this bit occurs, all further SCB transmissions generated by group 1 are UNSPECIFIED until all SINC filters of the group are shut down and restarted. Clearing this status bit (=0) alone is not sufficient to re-sync the DMA stream. Write 1 to clear. If this status bit and bit SINC_CTL.EFOVF0 are set (=1), the SINC_STAT interrupt is asserted.	
		0	No Overflow
		1	Overflow
11 (R/W1C)	PFAB0	Primary (Filter) Fabric Error for Group 0 Status. The SINC_STAT.PFAB0 indicates error status for the output of any primary SINC filter assigned to group 0. The bit is set (=1) if the SCB fabric provides a write error response for a filter output transfer associated with group 0, or if an overrun occurs for a filter in group 0. An interrupt is requested whenever this bit is =1 (not maskable).	
		0	Disabled
		1	Enabled
3 (R/W1C)	MIN3	Minimum for Secondary Filter 3 Status. The SINC_STAT.MIN3 bit indicates whether the output of the secondary SINC filter 3 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT3.LMIN bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN3 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded

Table 34-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	MIN2	Minimum for Secondary Filter 2 Status. The SINC_STAT.MIN2 bit indicates whether the output of the secondary SINC filter 2 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT2.LMIN bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN2 bits. For group 0, the duration limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the duration limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded
1 (R/W1C)	MIN1	Minimum for Secondary Filter 1 Status. The SINC_STAT.MIN1 bit indicates whether the output of the secondary SINC filter 1 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the SINC_LIMIT1.LMIN bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the SINC_CTL.EN1 bits. For group 0, the limit is SINC_LEVEL0.LCNT counts within a window of SINC_LEVEL0.LWIN samples. For group 1, the limit is SINC_LEVEL1.LCNT counts within a window of SINC_LEVEL1.LWIN samples.	
		0	Not Exceeded
		1	Exceeded

Table 34-30: SINC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W1C)	MIN0	Minimum for Secondary Filter 0 Status. The `SINC_STAT.MIN0` bit indicates whether the output of the secondary SINC filter 0 exceeded its minimum amplitude and duration level. This bit is set (=1) if the limit is exceeded. The amplitude limit is specified by the `SINC_LIMIT0.LMIN` bits. The duration limit is specified in terms of an excursion count and window for the filter group to which the filter is assigned by the `SINC_CTL.EN0` bits. For group 0, the limit is `SINC_LEVEL0.LCNT` counts within a window of `SINC_LEVEL0.LWIN` samples. For group 1, the limit is `SINC_LEVEL1.LCNT` counts within a window of `SINC_LEVEL1.LWIN` samples.	
		0	Not Exceeded
		1	Exceeded

35 MATH Accelerator Unit

The MATH unit accelerates highly accurate single-precision floating-point computations of common transcendentals such as trigonometric and exponential functions and their inverses. It provides faster reciprocals and square roots than provided by the Cortex-M4. It also provides accelerated functions to convert between rectangular and polar coordinates. Most operations by this tightly-coupled accelerator complete in a deterministic number of core clock cycles.

The MATH unit provides an easy-to-use function calculator for general programming operations. Its operands and results use the IEEE-754 single-precision floating-point numeric format. The functions follow the definitions in IEEE-754-2008 Section 9, Recommended Functions, including the set of valid arguments (domain) and exceptions. In general, results returned are accurate to within a standard relative error of 23.5 bits compared to the infinitely precise mathematical result.

A MATH operation generally takes the following form.

1. Write one or more values to the command registers. For example, write x to `MATH_SINF` for sin(x).
2. Compute the correctly rounded results. Make them available in one or more result registers (for example, `MATH_RES1`).
3. Signal detected exceptions (for example, DivByZero) in the `MATH_ISTAT` register and the `MATH_FSTAT` register. Generate interrupt requests, if enabled, through the `MATH_CTL` interrupt enable register.

NOTE: While the calculation is in-progress, the core can proceed with other tasks without delay.

The MATH unit accelerator requires no handshaking. When the result is needed, the processor core reads the result registers back into its own registers. The first read operation is stalled, if necessary, to allow the computation to complete.

For example, consider the following function call in C:

```
float x, y;
y = adi_sinf( x );
```

The compiler translates this call to assembly instructions:

```
VSTR S0, [R0, #0xNNNN] // store operand x to MATH:SINF register
VLDR S1, [R0, #0xNNNN] // load result y from MATH:RES1 register
```

It is straightforward to achieve parallelism in C code containing MATH operations, to promote acceleration of calculations in parallel with any related code. The C program performs the operand store and the result load as separate operations, with the desired parallel code in between. Since the MATH registers are declared as volatile in the processor's header files, optimizing compilers will respect the stated order of operations while ensuring maximum performance.

The MATH unit is efficient in its use of processor resources. It does not consume scarce Rx or Sx registers in the Cortex-M4 core, so that sequences of calculations can proceed without fill or spill to the stack. Further, it is possible to perform calculations directly to or from general-purpose registers (R0-R12), which has some performance advantages in the ARM M4 architecture.

MATH Features

The MATH unit has the following features:

- Accepts and produces IEEE 754 Single-Precision Floating-Point operands and results for all functions
- Provides unique MMR addresses for all function operands
- Provides all exceptions compliant to the IEEE 754 Single-Precision Floating-Point standard, including handling of infinities (INF), nans (QNAN and SNAN), signed zero, divided-by-zero, underflow, overflow, inexact, and invalid operations
- Provides an interrupt accumulate status register (`MATH_ISTAT`), a function status register (`MATH_FSTAT`), and a state status register (`MATH_SSTAT`)
- Processes denormalized numbers and returns denormalized results, when appropriate
- Performs preprocess range reduction, when necessary
- Provides save and restore

Supported MATH Functions

- Trigonometric functions and their inverses: sinf, cosf, tanf, asinf, acosf, atanf, with angles in radians
- Exponential functions expf (e^x) and exp2f (2^x)
- Logarithmic functions: lnf and log2f
- Square root
- Reciprocal
- Two-operand functions: atan2f(y/x) and hypotf(x,y) = $\sqrt{(x^2 + y^2)}$
- Rectangular and polar coordinate conversions: (x,y)=ptor(r,a), (r,a) = rtop(x,y), with angles in radians

Summary of MATH Modes and Status

For each of the IEEE-754 standard status flags (overflow, underflow, divide by zero, invalid operation, inexact result), the MATH unit provides the following status, and modes:

- (RW) Status of most recent operation
- (RW) Sticky status, accumulated from all operations since sticky status last cleared by writing a 0 to the status bit
- (RW) Interrupt enable (mode) for interrupt request after any operation which generates the associated status flag
- (W1C) Interrupt pending status

MATH Functional Description

The MATH function unit provides a DSP-based system with fast single-precision floating-point functions through a simple MMR interface in the core clock domain. The MATH function unit is an accelerator for highly accurate single-precision floating point computation of common transcendental functions. The tightly-coupled accelerator completes within a defined number of core clock cycles for each function, which is more competitive than the functions provided by the Cortex-M4.

CM41X_M4 MATH Interrupt List

Table 35-1: CM41X_M4 MATH Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
61	MATH0_MSTAT	MATH0 MATH unit Accumulate Interrupt Status	Level	

CM41X_M4 MATH Register List

A set of registers are used for MATH operations. For more information on MATH functionality, see the MATH register descriptions.

Table 35-2: CM41X_M4 MATH Register List

Name	Description
MATH_ACOSF	arccos(x) Function Register
MATH_ASINF	arcsin(x) Function Register
MATH_ATAN2F_X	arctan(y/x) Function x Operand Register
MATH_ATAN2F_Y	arctan(y/x) Function y Operand Register
MATH_ATANF	arctan(x) Function Register

Table 35-2: CM41X_M4 MATH Register List (Continued)

Name	Description
MATH_COSF	cos(x) Function Register
MATH_CTL	Math Unit Interrupt Enable Control
MATH_EXP2F	exp2(x) Function Register
MATH_EXPF	exp(x) Function Register
MATH_FSTAT	Math Unit Function Status Register
MATH_GX	Generic X Function Register (GX)
MATH_GY	Generic Y Function Register (GY)
MATH_HYPOTF_X	hypot(x,y) Function x Operand Register
MATH_HYPOTF_Y	hypot(x,y) Function y Operand Register
MATH_ISTAT	Math Unit Interrupt Register
MATH_LNF	ln(x) Function Register
MATH_LOG2F	log2(x) Function Register
MATH_PTORF_A	Polar to Rectangular Function a Operand Register
MATH_PTORF_R	Polar to Rectangular Function r Operand Register
MATH_RECIPF	Reciprocal(x) or 1/x Function Register
MATH_RES1	Math Unit Function Result 1
MATH_RES2	Math Unit Function Result 2
MATH_RTOPF_X	Rectangular to Polar Function x Operand Register
MATH_RTOPF_Y	Rectangular to Polar Function y Operand Register
MATH_SINF	sin(x) Function Register
MATH_SQRTF	Square Root or sqrt(x) Function
MATH_SSTAT	Math Unit Function State Status Register
MATH_TANF	tan(x) Function Register

MATH Block Diagram

The ***MATH Interface Block Diagram*** shows the functional blocks within the MATH module.

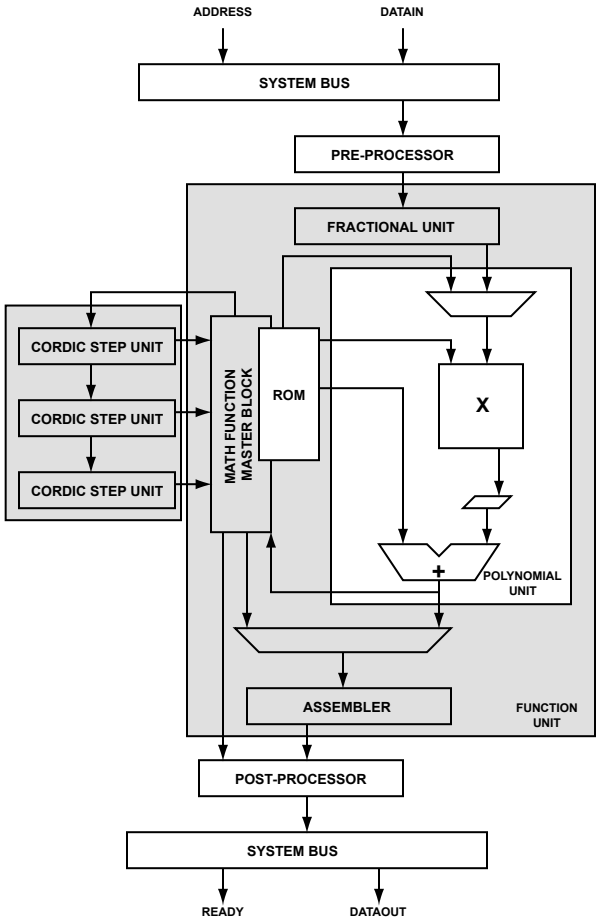


Figure 35-1: MATH Unit Block Diagram

MATH Definitions

To make the best use of the MATH unit, it is useful to understand the following terms.

Single-Precision Floating-Point Format

Single-precision floating-point format (shown in the figure) is a computer number format that occupies 4 bytes (32 bits). It represents a wide dynamic range of values by using a floating point (for example, C datatype 'float' numbers). In IEEE 754–2008, the 32-bit base 2 format is officially referred to as binary32.

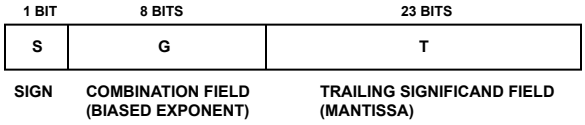


Figure 35-2: Single-Precision Binary Floating-Point Format

Normal Domain

For a given MATH unit function, this region of the input operands (for example, domain) is where the operation completes in the nominal number of cycles. It produces a result with Relative numerical Error (RE) which is no greater than the MATH unit Standard Relative Error (SRE).

The MATH unit functions, when used within constrained (most commonly used) regions, provides results faster than the ARM Cortex-M4F. This region is called the normal domain for that particular function.

Full Domain

An area of input operands. For a given MATH unit function, IEEE-754 2008 Section 9 specifies that this region of input operands (the domain) is well-defined.

Series Expansion

In mathematics, series expansion is a method for calculating a function that cannot be expressed by only elementary operators (addition, subtraction, multiplication, and division). The resulting series often can be limited to a finite number of terms, thus yielding an approximation of the function.

For example, the Taylor series of sinusoidal function.

$\sin(x) = x-x^3/3!+x^55/5!- ...= c1*x+c2*x^3+c3\times x^5... = (c1+c2*x^2+c3*x^4...)*x$

CORDIC

Coordinate Rotation Digital Computer (CORDIC) is also known as the digit-by-digit method and Volder's algorithm. It is a simple and efficient algorithm used to calculate hyperbolic and trigonometric functions. The MATH unit uses the CORDIC algorithm to compute trigonometric functions.

Range Reduction

The process of reducing the argument of x(input) into a smaller range. For example, while the Taylor series can be used to approximate and converge to a function with a polynomial. It only converges well to a finite series when x(input) is near to zero. Therefore, reducing the argument of x(input) into a smaller range is necessary.

MATH Architectural Concepts

The MATH unit communicates with the Cortex-M4F directly through the M4 subsystem's SYS bus crossbar. It services MMR addresses associated with the MATH function unit and translates the addresses into specific transcendental math functions. The results are read from common result registers shared by all the functions in single-precision floating-point format. The block stalls all read-accesses until the result is ready.

The MATH unit accepts and produces IEEE 754 Single-Precision Floating-Point operands and results for all functions. It provides unique MMR addresses (for example, `MATH_ATAN2F_X`) for all function operands. A write to an associated operand address initiates the processing of the function indicated by the address.

The MATH unit provides all exceptions compliant to the IEEE 754 Single-Precision Floating-Point standard, including handling of infinities (INF), nans (QNAN and SNAN), signed zero, divided-by-zero, underflow, overflow, inexact, and invalid operations.

The MATH unit provides an interrupt cumulative status register (MATH_ISTAT), a function status register (MATH_FSTAT), and a state status register (MATH_SSTAT). The status register is organized in the same way as the ARM M4 floating-point status register (invalid operation, overflow, underflow, divide-by- zero, and inexact). An interrupt-enable register can generate an interrupt request on any of the status indications.

Supported Operand Functions

The MATH unit produces results that conform to IEEE 754–2008 section 9 (Recommended Operations). This standard specifies the argument domain of the operation and the handling of NaNs, infinities, and invalid operands specific to that function as well as exception signaled.

Single-Operand Functions

The single-operand functions are be initiated by a write to a single write operand register. The ***Single-operand Functions with Single Result*** table shows the single-operand functions with a single result:

Table 35-3: Single-Operand Functions with Single Result

Function	Command Register	Expression	Full Domain of Argument	Exceptions
adi_recipf	RECIPF	$1/x$	$[-\infty, +\infty]$	$\|x\|$=0: divByZero
adi_sqrtf	SQRTF	$\sqrt{x}$	$[0, +\infty]$	x<0: invalidOp
adi_expf	EXPF	e^x	$[-\infty, +\infty]$	overflow; underflow
adi_exp2f	EXP2F	2^x	$[-\infty, +\infty]$	overflow; underflow
adi_logf	LOGF	$\ln(x)$	$[0, +\infty]$	x = 0: divByZero; x < 0: invalidOp
adi_log2f	LOG2F	$\log_2(x)$	$[0, +\infty]$	x = 0: divByZero; x < 0: invalidOp
adi_sinf	SINF	$\sin(x)$	$(-\infty, +\infty)$	$\|x\| = \infty$: invalidOp; underflow
adi_cosf	COSF	$\cos(x)$	$(-\infty, +\infty)$	$\|x\| = \infty$: invalidOp; underflow
adi_tanf	TANF	$\tan(x)$	$(-\infty, +\infty)$	$\|x\| = \infty$: invalidOp; underflow
adi_asinf	ASINF	$\sin^{-1}(x)$	$[-1, +1]$	$\|x\| > 1$: invalidOp; underflow
adi_acosf	ACOSF	$\cos^{-1}(x)$	$[-1, +1]$	$\|x\| > 1$: invalidOp; underflow

Table 35-3: Single-Operand Functions with Single Result (Continued)

Function	Command Register	Expression	Full Domain of Argument	Exceptions
adi_atanf	ATANF	$\tan^{-1}(x)$	$[-\infty, +\infty]$	underflow

Multi-Operand Functions with Single Result

The multiple operand functions are initiated by a write to two operand registers. The ***Multi-Operand Functions with Single Result*** table shows the multiple operand functions with single result.

NOTE: Writing to the `MATH_ATAN2F_X`/ `MATH_HYPOTF_X` register initiates the operation. First, write to the `MATH_ATAN2F_Y` /`MATH_HYPOTF_Y` register and then to the `MATH_ATAN2F_X`/ `MATH_HYPOTF_X` register.

Table 35-4: Multi-Operand Functions with Single Result

Function	Command Registers	Expression	Full Domain of Arguments	Exceptions
adi_atan2	ATAN2_X ATAN2_Y	$\tan^{-1}(y/x)$	x: $[-\infty, +\infty]$ × y: $[-\infty, +\infty]$	overflow, underflow
adi_hypot	HYPOT_X HYPOT_Y	$\sqrt{(x^2 + y^2)}$	x: $[-\infty, +\infty]$ × y: $[-\infty, +\infty]$	overflow

Multi-Operand Functions with Multiple Results

The ***Multi-Operand Functions with Multiple Result*** table shows the multiple operand functions with multiple results.

NOTE: Writing to `MATH_RTOPF_X`/ `MATH_PTORF_A` register initiates the operation. First, write to the `MATH_RTOPF_Y`/`MATH_PTORF_R` register and then to the `MATH_RTOPF_X` / `MATH_PTORF_A` register.

NOTE: Reading from the `MATH_RES1` register stalls the operation until the result is ready. First, read the `MATH_RES1` register and then the `MATH_RES2` register.

Table 35-5: Multi-Operand Functions with Multiple Results

Function	Command Registers	Result Expressions	Domain of Arguments	Exceptions
adi_rtop	RTOP_X RTOP_Y	RES1 = $\tan^{-1}(y/x)$ RES2 = $\sqrt{(x^2 + y^2)}$	x: $[-\infty, +\infty]$ × y: $[-\infty, +\infty]$	overflow; underflow

Table 35-5: Multi-Operand Functions with Multiple Results (Continued)

Function	Command Registers	Result Expressions	Domain of Arguments	Exceptions
adi_ptor	PTOR_A PTOR_R	RES1 $=r \cdot \cos(a)$ RES2 $= r \cdot \sin(a)$	r: $[-\infty, +\infty]$ × a: $(-\infty, +\infty)$ ×	$\|a\| = \infty$: invalidOp; overflow; underflow

Argument Normal and Full Domains

For any of its supported operations, and for any operands, the MATH unit produces results conforming to IEEE-754-2008 section 9 (Recommended Operations). This standard specifies the argument domain of the operation and the handling of NaNs, infinities, and invalid operands specific to that function (√x for x< 0), as well as the exceptions signaled.

For a given MATH unit function, the *normal domain* is the region of the input operands (the domain) in which the operation completes in the nominal number of cycles. The operation produces a result with relative numerical error (RE) which is no greater than the MATH unit Standard Relative Error (SRE). For example, the MATH unit implementation of sin(x) produces a result in nominal time accurate to within the SRE for the normal domain of x ∈(-8, +8).

For a given MATH unit function, the *full domain* is the region of input operands (the domain) for which IEEE-754 2008 Section 9 specifies that the function result be well-defined. For example, the standard specifies that the function sin(x) must accept a domain of x ∈ (-∞, +∞).

When an operand is outside the function's normal domain, the MATH block detects that range reduction is necessary, and in those cases it performs range reduction calculations automatically and without loss of accuracy. It is never necessary for the user's application to perform range checking tests or range reduction s on MATH unit operands.

The ***MATH Function Domain*** table shows the domains for the functions supported by the MATH unit. Details of the behavior of the function outside the normal domain are documented in the definition of the function.

Table 35-6: MATH Function Domain

Function	Full Domain	Normal Domain
adi_recipf (1/x)	$[-\infty, +\infty]$, $x \neq 0$	$[-\infty, +\infty]$, $x \neq 0$
adi_sqrtf	[0, +∞]	[0, +∞]
adi_expf (e^x)	$[-\infty, +\infty]$	$[-\infty, +\infty]$
adi_exp2f (2^x)	$[-\infty, +\infty]$	$[-\infty, +\infty]$
adi_lnf	[0, +∞]	[0, +∞]
adi_log2f	[0, +∞]	[0, +∞]
adi_sinf	$(-\infty, +\infty)$	(-8, +8)
adi_cosf	$(-\infty, +\infty)$	(-8, +8)

Table 35-6: MATH Function Domain (Continued)

Function	Full Domain	Normal Domain
adi_tanf	(−∞, +∞)	(-8, +8)
adi_asinf	[−1, +1]	[−1, +1]
adi_acosf	[−1, +1]	[−1, +1]
adi_atanf	[−∞, +∞]	[−1, +1]
adi_atan2f	x: [−∞, +∞] y: [−∞, +∞]	x: [−∞, +∞] y: [−∞, +∞]
adi_hypotf	x: [−∞, +∞] y: [−∞, +∞]	x: [−∞, +∞] y: [−∞, +∞]
adi_rtopf	x: [−∞, +∞] y: [−∞, +∞]	x: [−∞, +∞] y: [−∞, +∞]
adi_ptorf	r:[0, +∞) *a*:(−∞, +∞)	r:[0, +∞) *a*:(−8, +8)

Numerical Error

The numerical error of computations performed by the MATH unit is specified in terms Standard Relative Error (SRE). All results of computations are accurate across the entire full domain to the infinitely precise result within the SRE, as defined below. This holds even if the function argument is well outside the normal domain, for example cos(1234.5*pi). All IEEE-754-defined boundary conditions and exceptional states are handled as specified in the standard.

Table 35-7: MATH Numerical Error

Item	Definition	Max Value	Units
SRE	Standard Relative Error is given by: |R-P|/P For results of MATH unit operations on operands within the normal domain of the operation, where: R= actual result P=infintely precise result	8.429×10^{-8}	(dimensionless)
SRE_{ulp}	Standard Relative Error, relative to the unit in the least place (ULP) of the IEEE-754 single precision format	0.707	ULP
SRE_{bits}	Standard Relative Error, expressed in bits of precision: SRE bits = -log 2 (SRE) Note: For denormalized numbers, the error is expressed relative to the magnitude of the denormal numerical range, 2^{-126}.	-23.50	bits

Operand Performance (Core Clock Cycles)

The *MATH Function Domain* table defines the number of clock cycles for each operand supported by the MATH unit. The first clock cycle is defined at the write data cycle. The last cycle is defined at the ready signal of the read data or read operand status register.

NOTE: The *MATH Function Domain* table does not include additional latencies associated with load and store to MATH accelerator registers.

Table 35-8: MATH Function Domain

Function	Full Domain	Normal Domain
adi_recipf (1/*x*)	9 cycles	9
adi_sqrtf	9 cycles	9
adi_expf (e^x)	9 cycles	9
adi_exp2f (2^x)	8 cycles	8
adi_log2f	10 cycles	11
adi_lnf	11 cycles	10
adi_sinf	x = [−∞, +∞]: 14 cycles	x = (-8,+8): 9 cycles
adi_cosf	x = [−∞, +∞]: 14 cycles	x = (-8,+8): 9 cycles
adi_tanf	x = [−∞, +∞]: 20 cycles	x = (-8,+8): 13 cycles
adi_asinf	|x| ≤0.5: 11 cycles 0.5< |x| ≤0.75: 12 cycles 0.75< |x| ≤1: 15 cycles	
adi_acosf	|x| ≤0.5: 12 cycles 0.5< |x| ≤0.75: 13 cycles 0.75< |x| ≤1: 14 cycles (and negative x is15)	
adi_atanf	|x| ≤0.00325: 8 cycles |x| >0.00325: 20 cycles	
adi_atan2f	x: [−∞, +∞] y: [−∞, +∞]: 22 cycles	
adi_hypotf	|x| ≤0.00325: 10 cycles |x| >0.00325: 22 cycles	
adi_rtopf	x: [−∞, +∞] y: [−∞, +∞]: 33 cycles	
adi_ptorf	r:[0,+∞) *a*:(−∞, +∞): 20 cycles	r:[0,+∞) *a*:(−8, +8): 15 cycles

MATH Programming Model

At the most basic level, the MATH unit is operated by stores and loads of operands and results. All hardware synchronization is handled automatically. The unit provides a function library that makes conversion from a standard C library to the ADI math accelerator library nearly seamless.

For advanced programmers, inlined functions and optimized code rearrangement can dramatically improve throughput of function-intensive applications.

MATH Programming Concepts

Using the features, operating modes, and event control for the MATH unit to their greatest potential requires an understanding of some MATH-related concepts.

General Operations

The MATH unit functions are simple to control: store operands to operand registers, and read the results from the result registers. The bus hardware handles all synchronization with the accelerator. It inserts stalls, as necessary.

```
float angle = 0.78424;
pADI_MATH0->SINF = angle;    /* start a single-operand function */
float sine = pADI_MATH0->RES1;  /* read the result; hardware stalls until ready
*/

float x = 1.286e+4;
float y = -0.315e+3;
float radius;
pADI_MATH0->RTOPF_Y = y;    /* load the first operand (y first) */
pADI_MATH0->RTOPF_X = x; /* load the second operand and start the operation */
angle = pADI_MATH0->RES1; /* read the first result; hw stalls until ready */
radius = pADI_MATH0->RES2; /* read the second result */
```

Inlining Functions

Inlined functions provide an efficient and readable method of programming for the MATH unit. The method for causing a function to be inlined can vary from compiler to compiler. In the IAR tool set, this programming can be performed with `#pragma=forced` syntax.

```
#pragma inline=forced
float adi_sinf( float x ) {
pADI_MATH0->SINF = x;
return pADI_MATH0->RES1;
 }

float angle = 0.789123;
float sine = adi_sinf( angle ); /* will be converted to instructions inline
*/
```

Inlining Functions with Assembler

To achieve optimal performance, functions can be defined using inline assembler code. This code can guide the compiler to generate more efficient load-multiple or store-multiple instructions. In the following example, a two-element coordinate structure is used to pass arguments and results from an inlined function.

```
typedef struct { float x, y; } f_pair_t;

// inline assembler
#pragma inline=forced
f_pair_t adi_atan2f( f_pair_t xy ) {
  f_pair_t ra;
  float *pxy = pADI_MATH0->ATAN2_Y;
  float *pr12 = pADI_MATH0->RES1;
  asm("VSTM.32 %[px], {%[x], %[y]}\n"
  "VLDM.32 %[pr], {%[r], %[a]}"
      : [pr]"r" (pr12),
        [r]"=t" (ra.x),
        [a]"=t" (ra.y)
      : [px]"r" (pxy),
        [x]"t" (xy.x),
        [y]"t" (xy.y)
          );
    return ra;
   }

 /* do rectangular to polar coordinate conversion */
 f_pair_t rect = {1.2, 3.4};
 f_pair_t polar = adi_atan2f( rect ); /* generates VSTM and VLDM instructions */
```

Interleaving Code for Maximum Performance

The MATH unit does not require the ARM Cortex-M4F core to remain idle while it performs calculations. To optimize code for performance, it is useful to move unrelated code between the operand-store and the result-load operations. In this manner, the effective time of a function call is almost eliminated.

```
float *farray;
float angle;
int index, index2, offset, scale;

// Before Optimization
pADI_MATH0->SINF = angle;        /* start a single-operand function */
float sine = pADI_MATH0->RES1;   /* read the result; hardware stalls until ready
*/
index2 = index + (offset*scale);      /* calculate address to store result */
farray[index2] = sine;

// After Optimization
pADI_MATH0->SINF = angle;        /* start a single-operand function */
```

```
index2 = index + (offset*scale);      /* do useful work while waiting for result
*/
float sine = pADI_MATH0->RES1;   /* read the result; hardware stalls until ready
*/
farray[index2] = sine;
```

MATH Assembler Programming Model

Maximum performance of the MATH unit is attained when the optimal Cortex-M4 instruction sequences are used for common operations. In normal use, macros or inlined functions can be used from C to perform functions with the MATH block. This section documents the compiler-generated instruction sequences.

For advanced programmers, inlined functions and optimized code rearrangement can dramatically improve the throughput of function-intensive applications. It is assumed that exceptions are handled by interrupts or by cumulative (sticky) status, and are not interrogated after each specific operation.

Load and Store Efficiency

In MATH unit operations, the operands and results can be transferred either from general-purpose registers (Rx) or from floating-point registers within the FPU (Sx).

Single loads and stores of both types can take two cycles in the M4 architecture. In some cases, the general-purpose Rx registers load and store faster (just 1 cycle if they are pipelined with other load and store operations). The FPU Sx registers always require at least two cycles to load and store.

Multiple-register load and store operations are faster than a series of single load and store operations, for both Rx and Sx register types. The LDM or STM (for Rx) and VLDM or VSTM (for Sx) instructions each take 1 + N cycles for N registers.

Single Operand Functions

Single operand functions consist of a store of the operand to the desired function register, and a load of the result from the `MATH_RES1` register. It is advantageous to keep a pointer to the base of the MATH unit register block in a general-purpose M4 register. In the following examples, the R0 register is used for this pointer.

```
LDR.N R0, REG_MATH0_CTL; base address

 // Using General-Purpose registers Rx: compute R2 = sinf(R1)
 STR R1, [R0, #imm5]   //Start sinf(R1). imm5 is offset to SINF register
 // Stalls inserted automatically as needed (see tables).
 // Unrelated code may be placed here for parallel operation.
 LDR R2, [R0, #imm5] //finish sinf{R1}. imm5 is offset to RES1 register

 // Using FPU registers Sx: compute S2 = sinf(S1)
 VSTR S1, [R0, #imm8] //Start sinf(S1). imm8 is offset to SINF register
 // Stalls inserted automatically as needed (see tables).
 // Unrelated code may be placed here for parallel operation.
 VLDR S2, [R0, #imm8] //finish sinf{S1}. imm8 is offset to RES1 register
```

Two-operand and Two-result Functions

Two-operand functions are performed by a store of the two operands to the desired function registers. The Y register perfroms the first function and the X register performs the last function, followed by a load of the result from the MATH_RES1 register. Reading from the MATH_RES1 and MATH_RES2 result registers in any order concludes the two-result functions.

The operand registers for two-operand functions are arranged in the order (Y, then X) so it is possible to use an STM store-multiple instruction to initiate the function and an LDM load-multiple instruction to retrieve the results. (These instructions access memory addresses in increasing order.) The equivalent instructions for FPU registers are VSTM and VLDM.

Single-word loads and stores can be used to perform multi-operand and multi-result functions. These instructions have more available address modes than LDM or STM. So, while the load and store of the operands can take longer, the single LDR or STR instructions can permit a reduction of the overall cycle count.

```
//Using General-Purpose registers: {R3-R4} = rtopf({R1-R2} )
LDR.N R0, REG_MATH0_RTOPF_Y     // base address
STM R0, {R1-R2}    // 3 cycles
//Stalls inserted automatically as needed (see tables).
//Unrelated code may be placed here for parallel operation.
LDM R0, {R3-R4}    // 3 cycles

//Using FPU registers: {S3-S4} = rtopf( {S1-S2} )
VSTM R0, {S1-S2}   // 3 cycles
// Stalls inserted automatically as needed (see tables).
// Unrelated code may be placed here for parallel operation.
VLDM R0, {S3-S4}   // 3 cycles
```

Saving and Restoring Context

The context of the MATH unit consists of a consecutive 5-register block which includes MATH_GY, MATH_GX, MATH_RES1, MATH_RES2 , and MATH_FSTAT. This context can be most efficiently saved and restored using load-multiple and store-multiple instructions.

```
LDR.N R0, REG_MATH0_GY  //base address of the 5-register context group
LDM R0, {R1-R5}
PUSH {R1-R5}          // save context on the stack for later
// another set of MATH unit operations intervenes
POP {R1-R5}
STM R0, {R1-R5}      // restore prior context from stack
```

NOTE: MATH unit can still be in YPARTIAL state after Save context, which can cause sequence errors during another set of MATH unit operations . To clear the state, set the MATH_FSTAT.YPARTIAL bit to zero after PUSH {R1-R5}. Or a more efficient way is to set the MATH_FSTAT register to zero after PUSH {R1-R5}.

MATH Operation Flow

This section provides a description of the execution flow of MATH unit operations, including stalls and interrupt requests.

Initiation of MATH Unit Operations

A MATH unit operation is initiated by writing one or two operands to the command registers, where the selection of command register indicates the desired function.

Single-operand functions are initiated by a write to a single operand register, for example, MATH_SINF.

Multiple-operand functions are initiated by a write to two operand registers, the first to the associated Y register (for example, MATH_ATAN2F_Y) and second to the associated X register (for example, MATH_ATAN2F_X). The write to the first (Y) register does not start another operation and has no side effects on status, results, or interrupt requests. The write to the second (X) register initiates the desired operation.

If the operands are written in the wrong order (X without a preceding Y), or to registers that do not correspond to the same function, the SEQERR condition results and no function result is computed. To detect these errors, set the MATH_CTL.SEQEN (SEQERR interrupt enable) bit (=1).

Once an operation has been initiated by a write to an X register, the MATH unit enters the BUSY state (as indicated by the MATH_SSTAT.DIRTY bit), and begins the calculation of the result.

NOTE: Writing to the generic operand registers MATH_GX or MATH_GY does not initiate an operation or cause the MATH unit to become BUSY. These registers are provided to facilitate context save-and-restore.

MATH Unit Results, Status, and Stalls

To retrieve the results of a calculation, the processor reads the result registers. When the MATH unit is in the BUSY state, any read of any result register or the MATH_FSTAT register result register stalls the processor until the calculation of the result is complete. No polling by the processor is necessary.

Most function calculations complete within 20 or fewer core clocks, provided their arguments are in the normal range of the function. Calculation on arguments outside the normal range of the function can cause extra stalls while the argument range is automatically reduced to a normal range. The maximum duration of any MATH unit calculation is bounded at 28 cycles.

Reads of the MATH_FSTAT register stall when the MATH unit is in the BUSY state (if the MATH_SSTAT.DIRTY bit is set). Reads of the MATH_SSTAT register do not stall, and allow stall-free inspection of the state of the MATH unit at any time. If the MATH_SSTAT.DIRTY bit is clear, then the MATH unit is in the IDLE state, and the contents of the register are always consistent with the result registers. All the status bits in the register indicate the status results of the most recently completed operation.

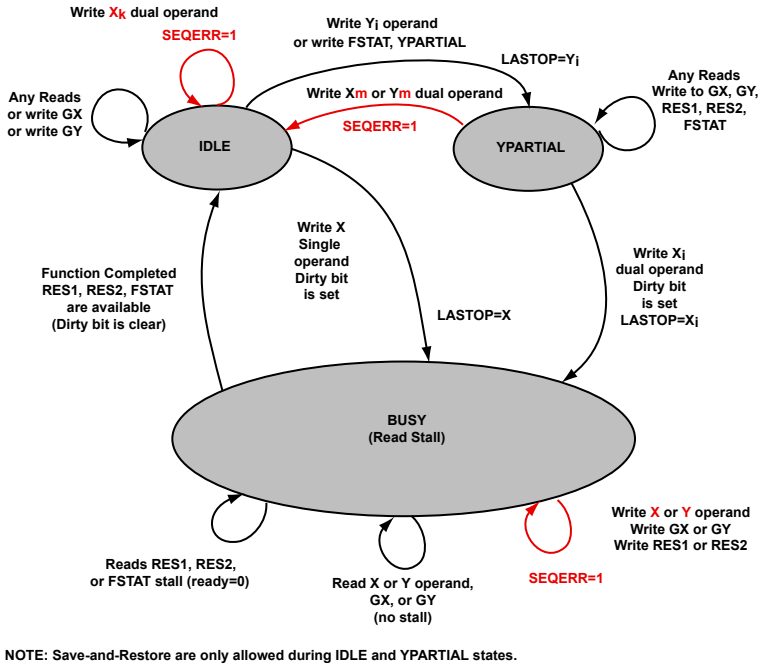


Figure 35-3: MATH Unit Register Operation Sequence

MATH Unit States and Sequence Error

These sections describe the various states of the MATH unit and how the sequence error SEQERR is resolved. The main states of the MATH unit are IDLE, YPARTIAL, and BUSY.

Typically, programs do not need to be concerned with these states, as long as the sequence of MATH register accesses is followed (for example, write Y, then write X, then read results or status). The details of these states become useful to understand if the normal sequence is not followed, or is interrupted, or if the state of the block is unknown.

IDLE State

The MATH unit is in an IDLE state when the `MATH_SSTAT.DIRTY` bit =0. In the IDLE state, if the `MATH_FSTAT.SEQERR` bit =0, the contents of the `MATH_FSTAT`, `MATH_RES1`, `MATH_RES2`, `MATH_GX`, and `MATH_GY` registers must all be mutually consistent. If the `MATH_FSTAT.SEQERR` bit =1, the contents of the `MATH_FSTAT`, `MATH_RES1`, `MATH_RES2`, `MATH_GX`, and `MATH_GY` registers are not always mutually consistent.

Normal status between operations is: (state = IDLE)

- `MATH_FSTAT.YPARTIAL` = 0
- `MATH_FSTAT.LASTOP` is any legal operation, whatever last finished. `MATH_RES1`, `MATH_RES2` = results of last operation

- If SEQERR = 0: MATH_GX, MATH_GY= operands of the last operation. (MATH_RES1, MATH_RES2, MATH_GX, and MATH_GY, MATH_FSTAT.LASTOP, and STATUS must be mutually consistent: the results of a valid operation)
- If SEQERR = 1: MATH_GX, MATH_GY = operands of last operation may not be consistent with the contents ofMATH_FSTAT, MATH_RES1, MATH_RES2 registers
- MATH_SSTAT.DIRTY =0 (not in the BUSY state)
- Status and sticky status can have any value

YPARTIAL State

The YPARTIAL state is entered when the first operand (Y) of a two-operand function is written. The YPARTIAL state is indicated by the MATH_FSTAT.SEQERR bit =1 and the MATH_FSTAT.YPARTIAL bit =1. In this state, the register sets contents may not be mutually consistent. This state is exited back to IDLE by writing MATH_FSTAT.YPARTIAL to zero. (This bit is a R/W bit and can be written to 0 or 1.) There is no SEQERR as a consequence.

The state can be restored from the IDLE state to YPARTIAL by writing MATH_FSTAT.YPARTIAL to 1 (as part of a save-and-restore sequence).

The YPARTIAL state is normally advanced to the BUSY state by writing a function X register to start the two-operand function.

The sequence error status is indicated when the MATH_FSTAT.SEQERR bit =1. The sequence error is set by hardware detection of a sequence error, or a write to MATH_FSTAT.SEQERR (as part of save-and-restore). When hardware sets the MATH_FSTAT.SEQERR bit, the contents of the 5-register set may not be mutually consistent. Writing zero to the MATH_FSTAT.SEQERR bit clears a sequence error. Setting the MATH_CTL.SEQEN bit enables the SEQERR cumulative interrupt. The register cumulative interrupt status can be read in using the W1C MATH_ISTAT.SEQERR bit.

BUSY State

The BUSY state can be detected directly by reading MATH_SSTAT.DIRTY bit value = 1. Or, this state can be detected indirectly by reading the MATH_FSTAT, MATH_RES1, or MATH_RES2 register and getting a stall. In practice, there is no need to poll (check for BUSY) to obtain the results of a function. Instead, the application reads the MATH_FSTAT, MATH_RES1, or MATH_RES2 registers as needed. The core stalls for a short time as required for the result to become available. The BUSY state automatically exits to IDLE when the function completes.

The BUSY state must not persist for longer than a short, bounded time limit defined in the specification (for example, 20 cycles for single operands and 28 cycles for dual operands). Any write to any input or output register while in the BUSY state sets the MATH_FSTAT.SEQERR bit. Any read of the output registers (MATH_FSTAT, MATH_RES1, or MATH_RES2) while in the BUSY state stalls until the function is not in the BUSY state.

Unknown State

If the prior state of the math unit is unknown, any use of the math unit must be preceded with a read of the MATH_FSTAT register. This forces the completion of any BUSY operations. If the prior state of the MATH unit is valuable, for example after an interrupt, then a save of the 5-register state must be performed (MATH_FSTAT, MATH_RES1, MATH_RES2MATH_GX, and MATH_GY).

Write the MATH_FSTAT.YPARTIAL bit with a value of 0 to clear the MATH_FSTAT.SEQERR bit.

Save-and-Restore

The MATH_FSTAT.SEQERR bit exhibits the following behavior during a save-and-restore operation:

- If the MATH_FSTAT.SEQERR bit is 0, and a save-and-restore brackets an error-free unrelated use of the MATH unit, then SEQERR ends up 0. (read 0, write 0).
- If the MATH_FSTAT.SEQERR bit is 1, and a save-and-restore happens (regardless of whether the intervening activity is error-free or not), then the restore of the saved MATH_FSTAT register preserves the SEQERR state or cleared. A sequence error is a detected departure from a valid operation sequence. This departure can cause an interrupt request if the MATH_CTL.SEQEN interrupt enable is set.
- The MATH_FSTAT.SEQERR indicates that the contents of MATH_FSTAT, MATH_RES1, MATH_RES2, MATH_GX, and MATH_GY may not be mutually consistent.
- No new function starts upon a restore to MATH_FSTAT.

MATH Unit Context

The context of the MATH unit refers to the set of registers that fully describe the state of the unit. In the programming model, successful context switching depends on properly saving and restoring the full context of the MATH unit.

Components of the MATH Unit Context

The MATH unit contains a control register, MATH_CTL and the status registers MATH_ISTAT, MATH_FSTAT, and MATH_SSTAT.

The status registers provide the following functions.

- The MATH_FSTAT.LASTOP bit field provides the function code for the function in-progress or the IDLE function when the unit is paused. The MATH unit sets this bit field whenever an X or Y operand register (but not MATH_GX and MATH_GY) is written. The bit field is cleared when the operation completes (BUSY returns to IDLE state).
- Status (exceptions and sequence error) of the most recently completed function
- The related sticky status and interrupt accumulated status
- The MATH_FSTAT.YPARTIAL bit indicates that a Y operand has been written without the associated X operand.

- The MATH_SSTAT.DIRTY bit indicates that some register has been modified since the last time and the MATH unit is now in the BUSY state. The MATH_SSTAT.DIRTY bit is cleared when the unit is no longer busy with the current function.

The MATH unit contains two result registers MATH_RES1 and MATH_RES2 and two operand registers MATH_GX and MATH_GY.

The five registers, (MATH_GX, MATH_GY, MATH_RES1, MATH_RES2, and MATH_FSTAT) in the MATH unit context are placed at adjacent addresses to facilitate saving and restoring of context using ARM Cortex-M4 load-multiple (LDM) and store-multiple (STM) instructions. The sequence is ordered so that blocking reads of a MATH_RES1 or MATH_RES2 register precede blocking reads of the MATH_FSTAT register when an LDM instruction is used, so that the complete context is self-consistent.

The sequence ensures that the MATH_FSTAT register is loaded last, so that any restarted operation finds all the operand values and required control settings.

NOTE: The MATH_SSTAT register is a non-blocking read so that the MATH_SSTAT.DIRTY bit is available for stall-free examination of the MATH block's state.

Reading Operand Register Context

Most of the operand registers are pseudo-registers. They are aliases of two internal storage elements, described as MATH_GX and MATH_GY. Writing to any Y operand register writes to the MATH_GY storage element. Writing to any X operand register writes to the MATH_GX storage element and initiates the associated computation.

Reads of the MATH_GX register or any of the X operand registers alias to the same value, that is the last value written to any X operand register. Similarly, reads of the MATH_GY register or any of the Y operand registers alias to the same value, that is the last value written to any Y operand register.

The generic operand registers MATH_GX and MATH_GY facilitate saving and restoring the MATH unit context without unintentionally initiating new computations. Writing to these registers does not initiate a new computation.

Interrupts in MATH Unit Operations

MATH unit operations are fully interruptible and restartable. An interrupt thread which itself needs to use the MATH unit can save the MATH context using an LDM instruction. A software system might further use an external semaphore (for example a global variable) to signal whether the MATH block is in use as part of a chain of calculations, so that interrupt handlers might omit save or restore operations when not needed (for a lazy save/restore.)

- The LDM stalls, only if needed, on the read of the MATH_RES1 or MATH_RES2 registers, and provides a self-consistent set of register values. This configuration works effectively if saving the MATH context can be positioned after other handler initial code, to allow the operation to complete.
- When restored with STM, the result registers are reloaded with the result values expected by the interrupted thread, with no additional latency.

The MATH_FSTAT.YPARTIAL bit indicates to an observer that the contents of the results, status, and X operand registers are not consistent with the Y operand register. This inconsistency is because a Y operand register was written without the accompanying X register (which would have initiated the computation). This discrepancy can be due to an operand write sequence that was interrupted.

An interrupt handler needs write MATH_FSTAT.YPARTIAL = 0 after save-context operation. After returning from a save-and-restore operation, the interrupted thread completes the operand-load sequence and the intended calculation completes successfully.

A context save operation proceeds as follows.

1. The processor (for example, in an interrupt handler) starts an LDM operation to read the block of five registers into five M4 general purpose registers (for example R0-R4), typically at a rate of one core clock per register. The first two registers (MATH_GX, MATH_GY) do not stall, and preserve the original operand(s) of any operation that may be in progress.
2. The next register read (MATH_RES1) is a blocking read, and if an operation is running (BUSY), the MATH block will stall the LDM until the operation is complete. (Note that the impact on interrupt latency is minimal, since the processor will have taken several cycles to transfer control and save registers before the handler's MATH context-save instructions begin executing, and in most cases any running operation will have already completed.)
3. The last two registers then complete without stall. At this point the processor registers now contain the entire self-consistent MATH block context, together with the completed function result. This is true even if a two-operand function was only partially initiated after writing the Y operand (YPARTIAL state), as the MATH_FSTAT register value indicates the YPARTIAL state and the register value preserves the first operand value.
4. The set of five saved context values can then be stored elsewhere with an STM instruction, for example pushed onto the stack.

NOTE: The MATH unit can still be in YPARTIAL state after Step 4, which can cause sequence errors during any subsequent MATH unit operations. To clear the state, set MATH_FSTAT.YPARTIAL = 0 after Step 4. Else, set the MATH_FSTAT register to zero after Step 4.

A context restore operation proceeds in reverse, as follows.

1. Before returning from the handler, the handler retrieves the five-word context from storage (for example, by an LDM or pop from the stack), and
2. The handler then initiates a STM of the five MMR registers starting with MATH_GX. These writes all proceed without stall, and do not initiate any new function calculations.
3. When the STM is complete, all operands, results, and function status are restored, including any YPARTAL or SEQERR conditions, so that any interrupted code may resume.

CM41X_M4 MATH Register Descriptions

Math Unit Function (MATH) contains the following registers.

Table 35-9: CM41X_M4 MATH Register List

Name	Description
MATH_ACOSF	arccos(x) Function Register
MATH_ASINF	arcsin(x) Function Register
MATH_ATAN2F_X	arctan(y/x) Function x Operand Register
MATH_ATAN2F_Y	arctan(y/x) Function y Operand Register
MATH_ATANF	arctan(x) Function Register
MATH_COSF	cos(x) Function Register
MATH_CTL	Math Unit Interrupt Enable Control
MATH_EXP2F	exp2(x) Function Register
MATH_EXPF	exp(x) Function Register
MATH_FSTAT	Math Unit Function Status Register
MATH_GX	Generic X Function Register (GX)
MATH_GY	Generic Y Function Register (GY)
MATH_HYPOTF_X	hypot(x,y) Function x Operand Register
MATH_HYPOTF_Y	hypot(x,y) Function y Operand Register
MATH_ISTAT	Math Unit Interrupt Register
MATH_LNF	ln(x) Function Register
MATH_LOG2F	log2(x) Function Register
MATH_PTORF_A	Polar to Rectangular Function a Operand Register
MATH_PTORF_R	Polar to Rectangular Function r Operand Register
MATH_RECIPF	Reciprocal(x) or 1/x Function Register
MATH_RES1	Math Unit Function Result 1
MATH_RES2	Math Unit Function Result 2
MATH_RTOPF_X	Rectangular to Polar Function x Operand Register
MATH_RTOPF_Y	Rectangular to Polar Function y Operand Register
MATH_SINF	sin(x) Function Register
MATH_SQRTF	Square Root or sqrt(x) Function
MATH_SSTAT	Math Unit Function State Status Register
MATH_TANF	tan(x) Function Register

arccos(x) Function Register

The MATH_ACOSF register, when written, initiates the arccos(x) function.

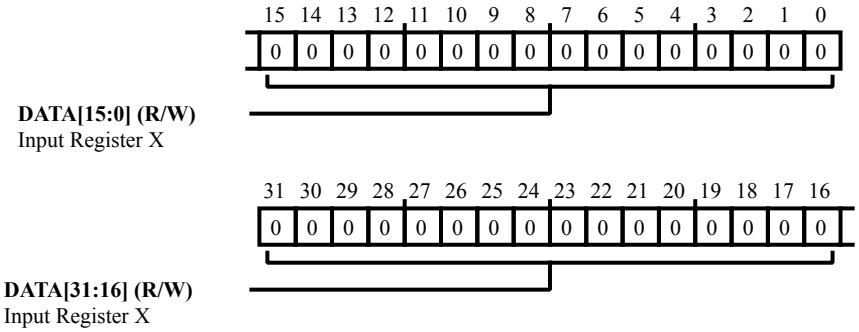


Figure 35-4: MATH_ACOSF Register Diagram

Table 35-10: MATH_ACOSF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

arcsin(x) Function Register

The MATH_ASINF register, when written, initiates the arcsin(x) function.

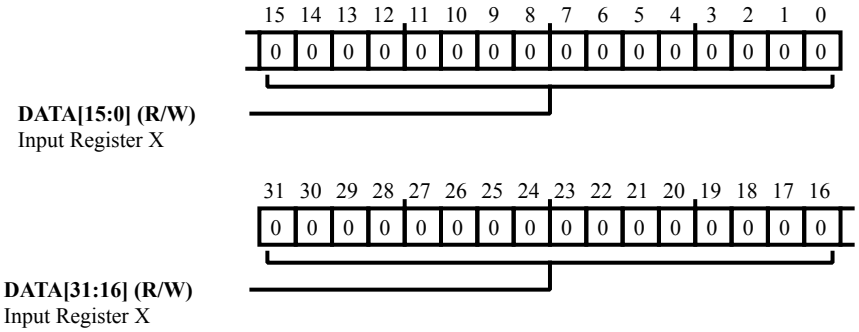


Figure 35-5: MATH_ASINF Register Diagram

Table 35-11: MATH_ASINF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

arctan(y/x) Function x Operand Register

The MATH_ATAN2F_X register is the command register for the second operand (x) of the arctan(y/x) function. A write to the register initiates the arctan(y/x) function.

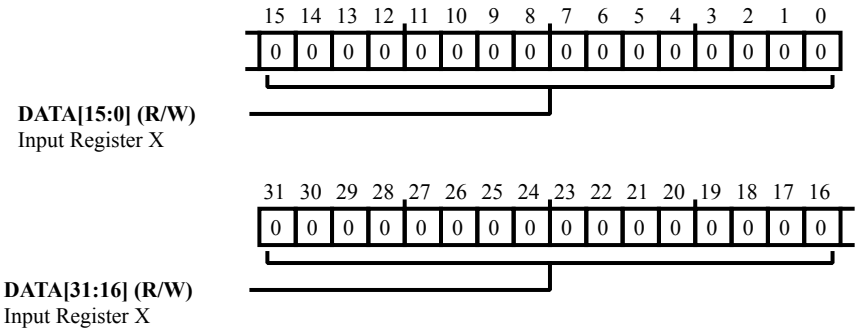


Figure 35-6: MATH_ATAN2F_X Register Diagram

Table 35-12: MATH_ATAN2F_X Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

arctan(y/x) Function y Operand Register

The MATH_ATAN2F_Y register is the command register for the first operand (y) of the arctan(y/x) function. It does not initiate a new operation.

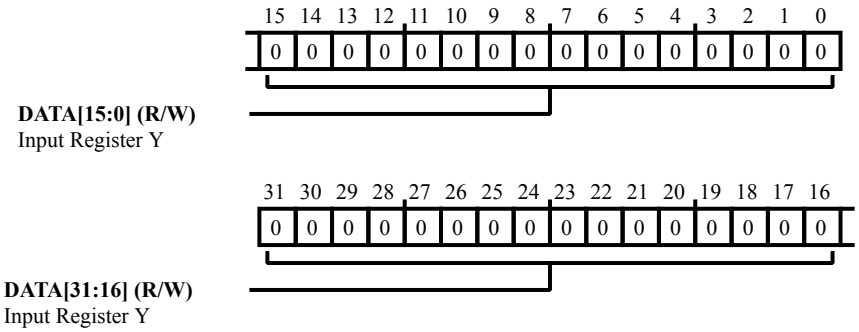


Figure 35-7: MATH_ATAN2F_Y Register Diagram

Table 35-13: MATH_ATAN2F_Y Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register Y.

arctan(x) Function Register

The MATH_ATANF register, when written, initiates the arctan(x) function.

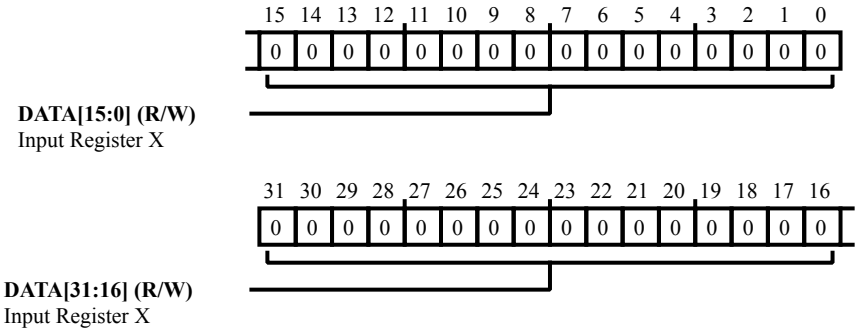


Figure 35-8: MATH_ATANF Register Diagram

Table 35-14: MATH_ATANF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

cos(x) Function Register

The MATH_COSF register, when written, initiates the cos(x) function.

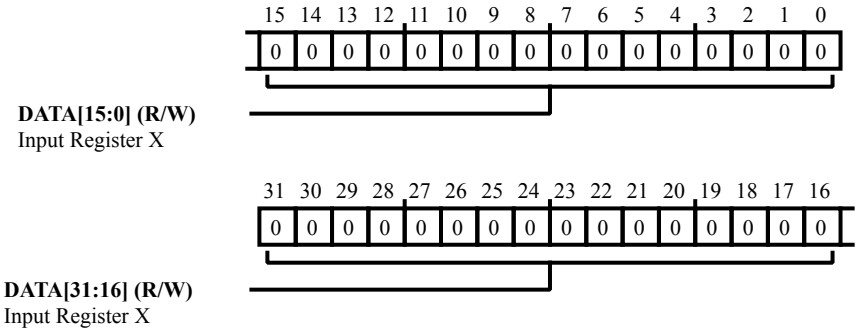


Figure 35-9: MATH_COSF Register Diagram

Table 35-15: MATH_COSF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X. The MATH_COSF.DATA bits hold the data for the cos (x) function.

Math Unit Interrupt Enable Control

The MATH_CTL register enables interrupts for invalid operations, overflow and underflow errors, divide by zero, and sequence errors.

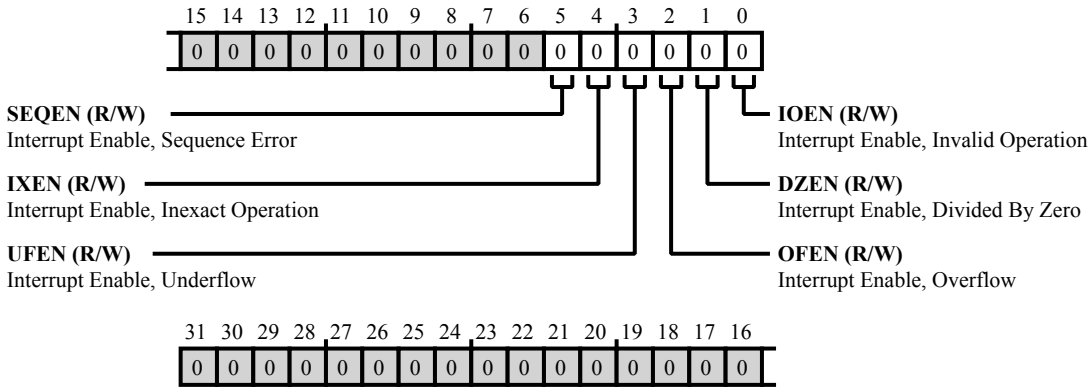


Figure 35-10: MATH_CTL Register Diagram

Table 35-16: MATH_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	SEQEN	Interrupt Enable, Sequence Error. Write 1 to enable the interrupt request.
4 (R/W)	IXEN	Interrupt Enable, Inexact Operation. Write 1 to enable the interrupt request.
3 (R/W)	UFEN	Interrupt Enable, Underflow. Write 1 to enable the interrupt request.
2 (R/W)	OFEN	Interrupt Enable, Overflow. Write 1 to enable the interrupt request.
1 (R/W)	DZEN	Interrupt Enable, Divided By Zero. Write 1 to enable the interrupt request.
0 (R/W)	IOEN	Interrupt Enable, Invalid Operation. Write 1 to enable the interrupt request.

exp2(x) Function Register

The MATH_EXP2F register, when written, initiates the exp2(x) or 2^x function.

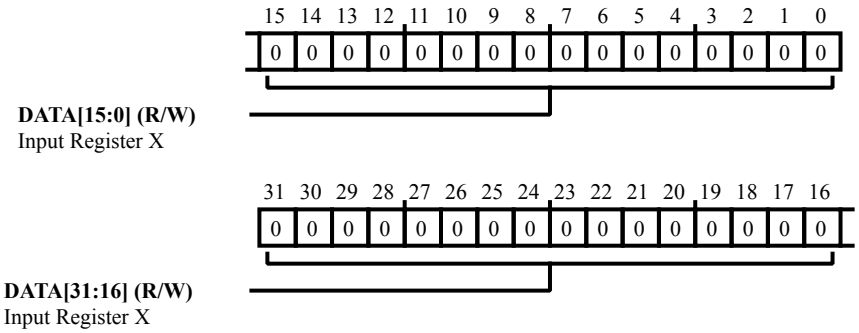


Figure 35-11: MATH_EXP2F Register Diagram

Table 35-17: MATH_EXP2F Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

exp(x) Function Register

The MATH_EXPF register, when written, initiates the exp(x) or e^x function.

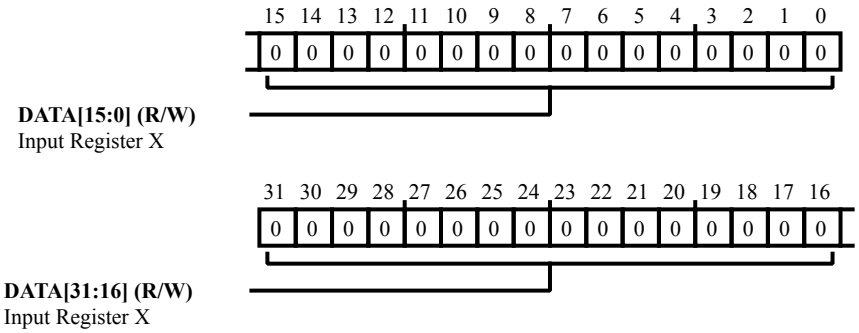


Figure 35-12: MATH_EXPF Register Diagram

Table 35-18: MATH_EXPF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

Math Unit Function Status Register

The MATH_FSTAT register indicates the current or last function status.

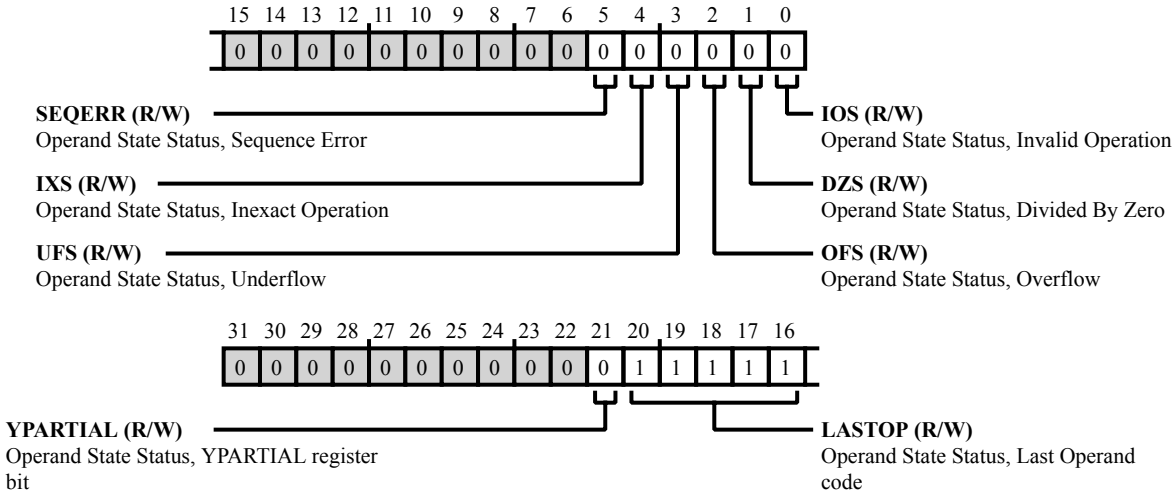


Figure 35-13: MATH_FSTAT Register Diagram

Table 35-19: MATH_FSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	YPARTIAL	Operand State Status, YPARTIAL register bit. The MATH_FSTAT register can be written by hardware as well as a software access. When executing a function x register, the MATH_FSTAT.YPARTIAL bit is set to indicate that the y operand register is modified without the associated x operand. (Refer to the MATH Unit States and Sequence Error section for more information.)	
20:16 (R/W)	LASTOP	Operand State Status, Last Operand code. The MATH_FSTAT register can be written by hardware as well as a software access. When executing a function X register, the MATH_FSTAT.LASTOP bit field is updated. (Refer to the MATH Unit States and Sequence Error section for more information.)	
		0	Last operation was EXP.
		1	Last operation was EXP2.
		2	Last operation was RECIP.
		3	Last operation was SQRT.
		4	Last operation was LN.
		5	Last operation was LOG2.
		6	Last operation was SIN.
		7	Last operation was COS.

Table 35-19: MATH_FSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		8	Last operation was TAN.
		9	Last operation was ASIN.
		10	Last operation was ACOS.
		11	Last operation was ATAN.
		16	Last operation was ATAN2.
		17	Last operation was HYPOT.
		18	Last operation was RTOP.
		19	Last operation was PTOR.
5 (R/W)	SEQERR	Operand State Status, Sequence Error. Read/write state register.	
4 (R/W)	IXS	Operand State Status, Inexact Operation. Read/write state register.	
3 (R/W)	UFS	Operand State Status, Underflow. Read/write state register.	
2 (R/W)	OFS	Operand State Status, Overflow. Read/write state register.	
1 (R/W)	DZS	Operand State Status, Divided By Zero. Read/write state register.	
0 (R/W)	IOS	Operand State Status, Invalid Operation. Read/write state register.	

Generic X Function Register (GX)

The MATH_GX register is used for the generic X function.

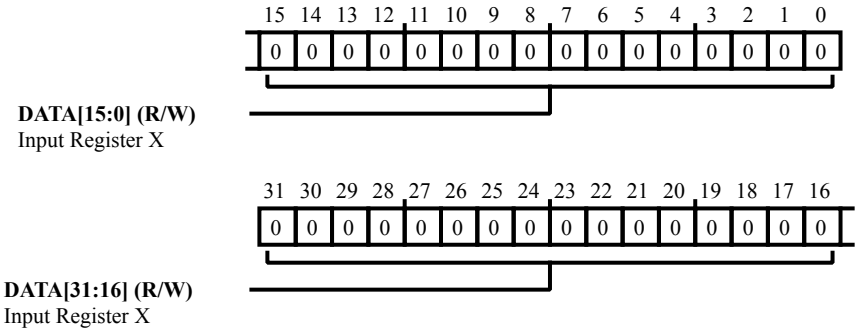


Figure 35-14: MATH_GX Register Diagram

Table 35-20: MATH_GX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

Generic Y Function Register (GY)

The MATH_GY register is used for the generic Y function.

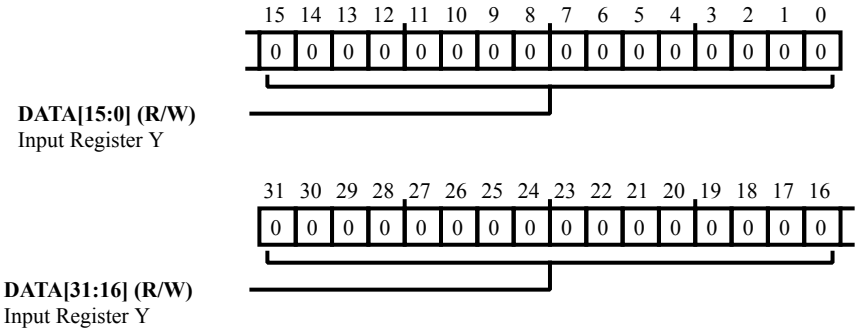


Figure 35-15: MATH_GY Register Diagram

Table 35-21: MATH_GY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register Y.

hypot(x,y) Function x Operand Register

The MATH_HYPOTF_X register is the command register for the second operand (x) of the arctan(y/x) function. A write to the register initiates the hypot(x,y) = sqrt(x^2 + y^2) function.

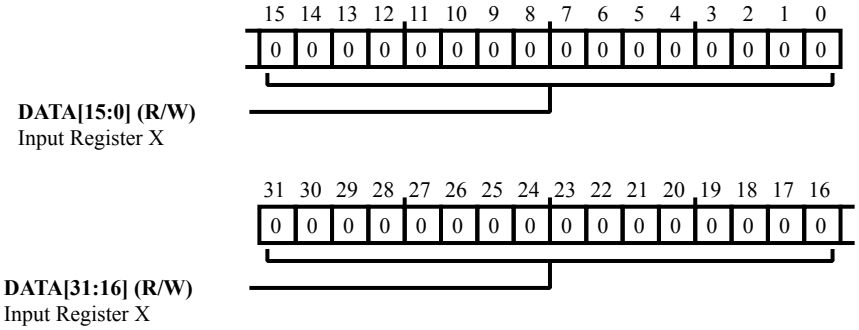


Figure 35-16: MATH_HYPOTF_X Register Diagram

Table 35-22: MATH_HYPOTF_X Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

hypot(x,y) Function y Operand Register

The MATH_HYPOTF_Y register is the command register for the first operand (y) of the hypot(x,y) = sqrt(x^2 + y^2) function. It does not initiate a new operation.

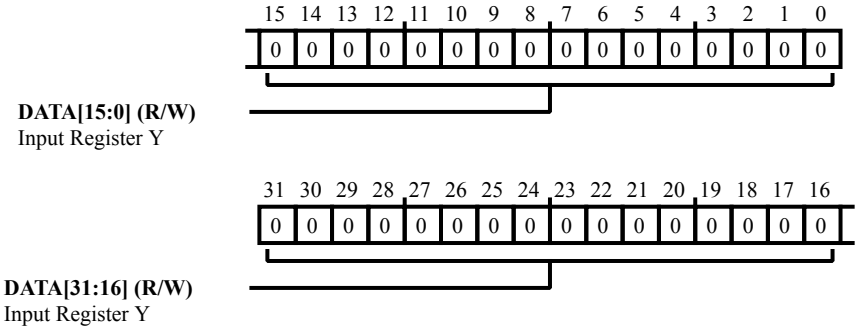


Figure 35-17: MATH_HYPOTF_Y Register Diagram

Table 35-23: MATH_HYPOTF_Y Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register Y.

Math Unit Interrupt Register

The MATH_ISTAT register indicates the cumulative interrupt status of the MATH unit.

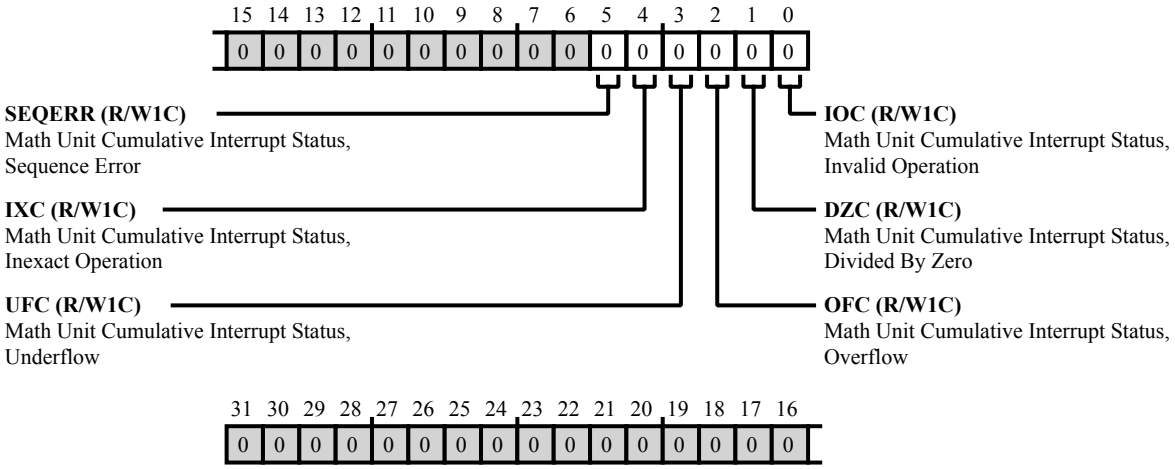


Figure 35-18: MATH_ISTAT Register Diagram

Table 35-24: MATH_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W1C)	SEQERR	Math Unit Cumulative Interrupt Status, Sequence Error. Write 1 to clear status.
4 (R/W1C)	IXC	Math Unit Cumulative Interrupt Status, Inexact Operation. Write 1 to clear status.
3 (R/W1C)	UFC	Math Unit Cumulative Interrupt Status, Underflow. Write 1 to clear status.
2 (R/W1C)	OFC	Math Unit Cumulative Interrupt Status, Overflow. Write 1 to clear status.
1 (R/W1C)	DZC	Math Unit Cumulative Interrupt Status, Divided By Zero. Write 1 to clear status.
0 (R/W1C)	IOC	Math Unit Cumulative Interrupt Status, Invalid Operation. Write 1 to clear status.

ln(x) Function Register

The MATH_LNF register, when written, initiates the ln(x) function.

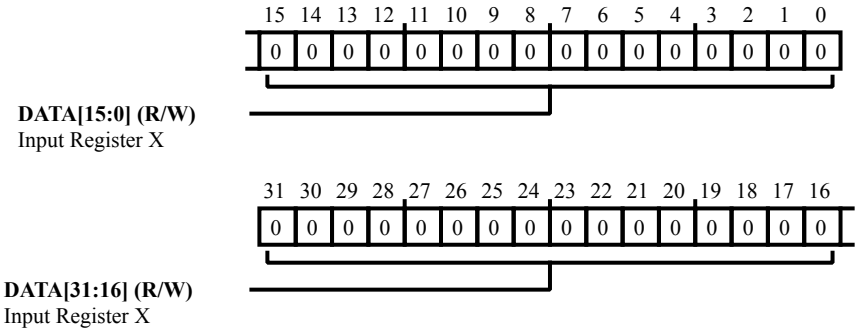


Figure 35-19: MATH_LNF Register Diagram

Table 35-25: MATH_LNF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

log2(x) Function Register

The MATH_LOG2F register, when written, initiates the log2(x) function.

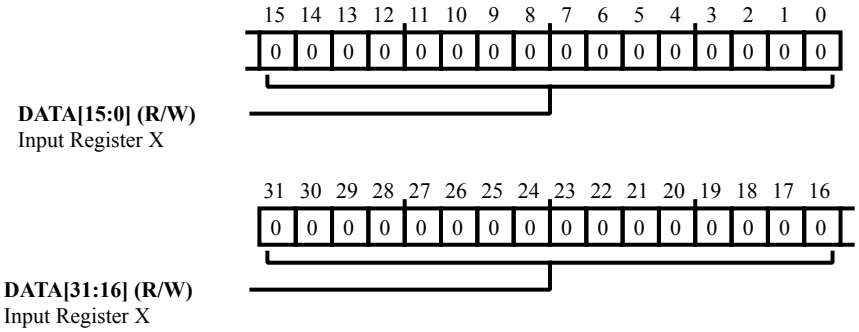


Figure 35-20: MATH_LOG2F Register Diagram

Table 35-26: MATH_LOG2F Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

Polar to Rectangular Function a Operand Register

The MATH_PTORF_A register is the command register for the first operand (a, or angle in radians) of the r * sin(a) and r * cos(a) function multifunction. (The function converts from polar to rectangular coordinates.) A write to this register does not initiate a new operation.

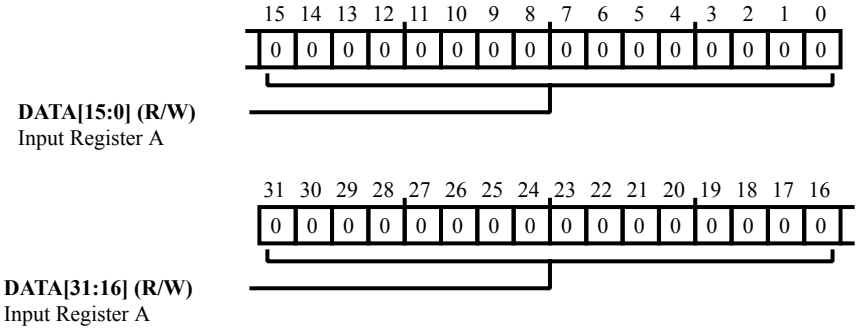


Figure 35-21: MATH_PTORF_A Register Diagram

Table 35-27: MATH_PTORF_A Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register A.

Polar to Rectangular Function r Operand Register

The MATH_PTORF_R register is the command register for the second operand (r, or radius) of the r * sin(a) and r * cos(a) multifunction. (The function converts from polar to rectangular coordinates.) A write to this register initiates the r * sin(a) and r * cos(a) multifunction.

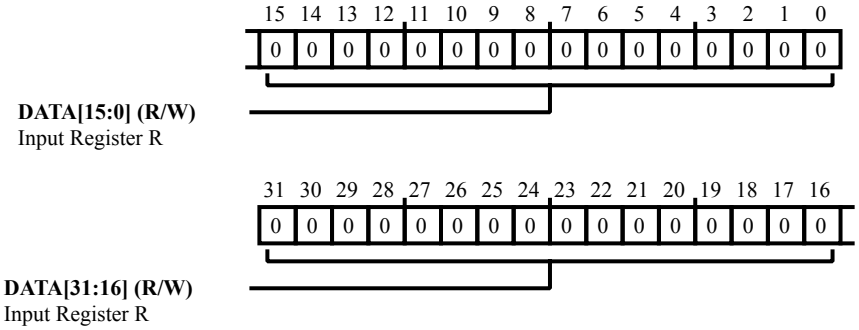


Figure 35-22: MATH_PTORF_R Register Diagram

Table 35-28: MATH_PTORF_R Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register R.

Reciprocal(x) or 1/x Function Register

The MATH_RECIPF register, when written, initiates the reciprocal(x) or 1/x function.

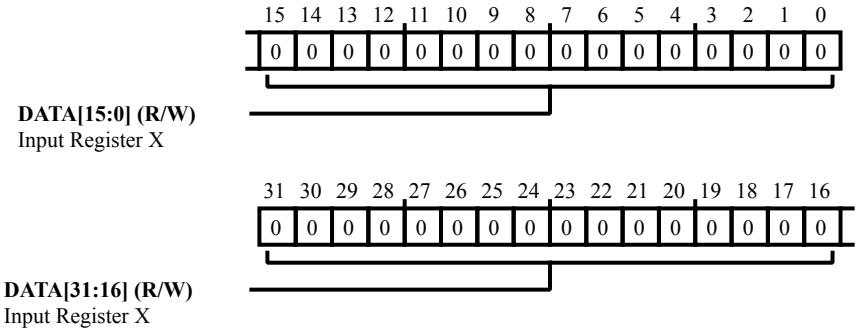


Figure 35-23: MATH_RECIPF Register Diagram

Table 35-29: MATH_RECIPF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

Math Unit Function Result 1

The MATH_RES1 register contains the result for a single result operation or the first result of a dual result function.

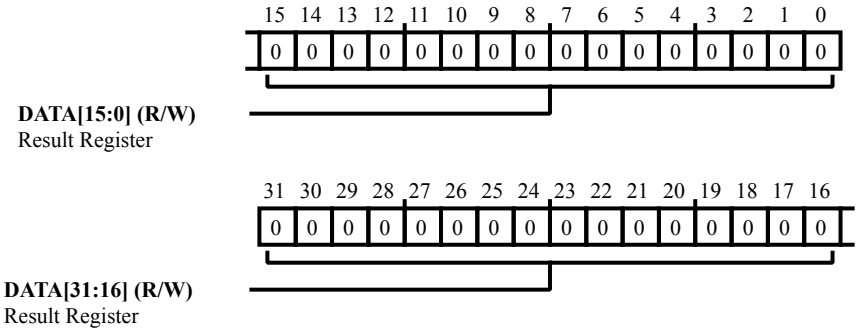


Figure 35-24: MATH_RES1 Register Diagram

Table 35-30: MATH_RES1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Result Register.

Math Unit Function Result 2

The MATH_RES2 register contains the second result of a dual result function.

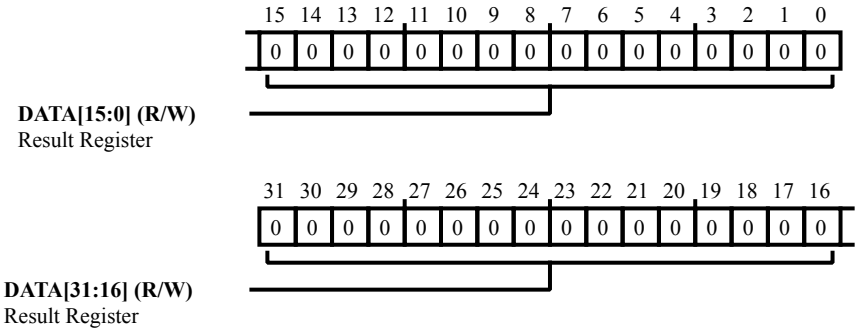


Figure 35-25: MATH_RES2 Register Diagram

Table 35-31: MATH_RES2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Result Register.

Rectangular to Polar Function x Operand Register

The MATH_RTOPF_X register is the command register for the second operand (x) of the sqrt($x^2 + y^2$) and arctan(y/x) multifunction. (The function converts from rectangular to polar coordinates.) A write to this register initiates the sqrt($x^2 + y^2$) and arctan(y/x) multifunction.

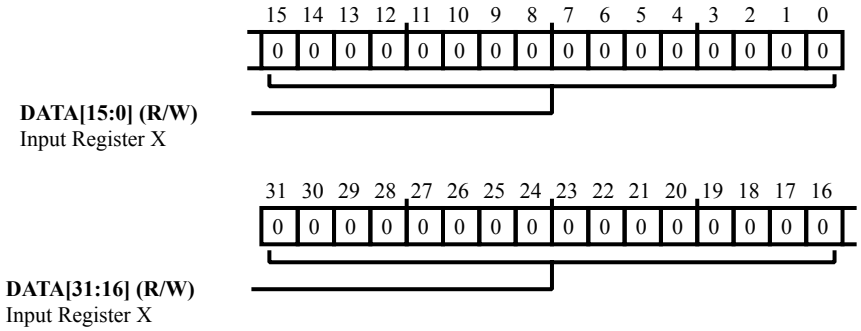


Figure 35-26: MATH_RTOPF_X Register Diagram

Table 35-32: MATH_RTOPF_X Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

Rectangular to Polar Function y Operand Register

The MATH_RTOPF_Y register is the command register for the first operand (y) of the sqrt($x^2 + y^2$) and arctan(y/x) multifunction. (The function converts from rectangular to polar coordinates.) A write to this register does not initiate a new operation.

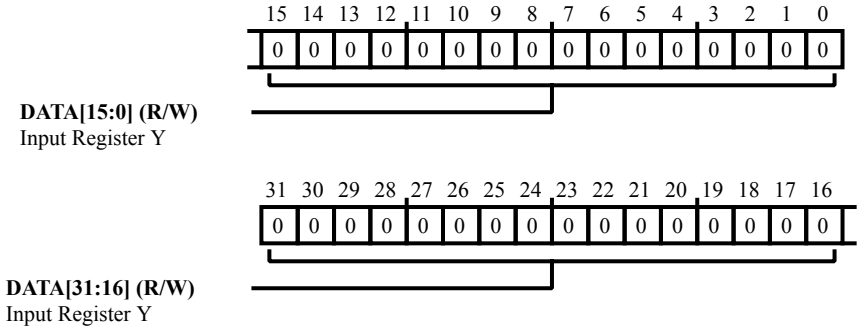


Figure 35-27: MATH_RTOPF_Y Register Diagram

Table 35-33: MATH_RTOPF_Y Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register Y.

sin(x) Function Register

The MATH_SINF register, when written, initiates the sin (x) function.

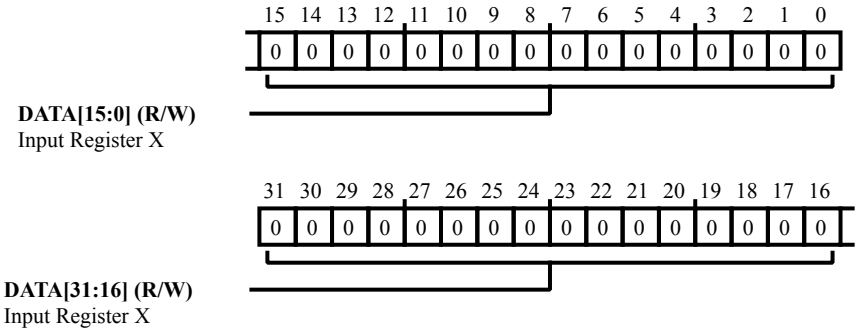


Figure 35-28: MATH_SINF Register Diagram

Table 35-34: MATH_SINF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

Square Root or sqrt(x) Function

The MATH_SQRTF register, when written, initiates the sqrt(x) function.

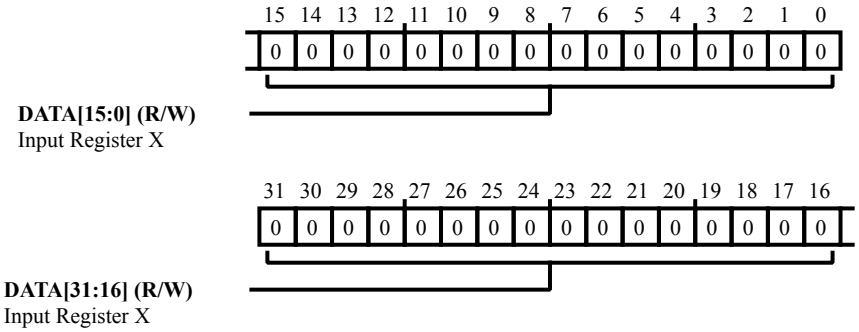


Figure 35-29: MATH_SQRTF Register Diagram

Table 35-35: MATH_SQRTF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

Math Unit Function State Status Register

The MATH_SSTAT register indicates the status of the MATH unit.

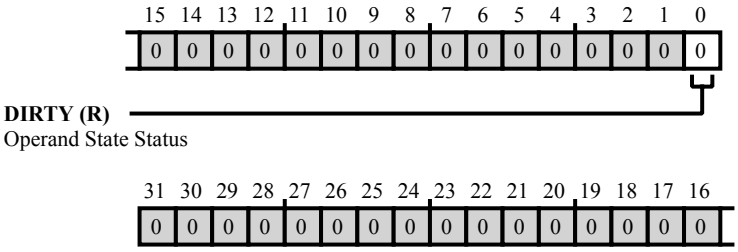


Figure 35-30: MATH_SSTAT Register Diagram

Table 35-36: MATH_SSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	DIRTY	Operand State Status. The MATH_SSTAT.DIRTY (sticky) bit indicates that the context of the function register has been modified since the last context was saved, facilitating a lazy stacking of state. The bit is also known as the busy bit.

tan(x) Function Register

The MATH_TANF register, when written, initiates the tan(x) function.

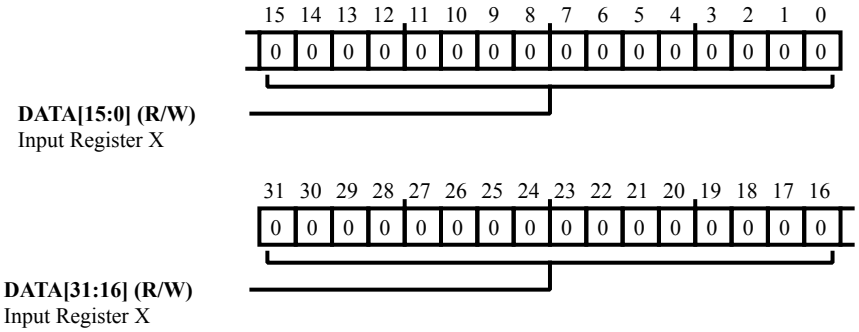


Figure 35-31: MATH_TANF Register Diagram

Table 35-37: MATH_TANF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Input Register X.

36 Floating-Point Saturation Unit (FSAT)

It is often desirable to implement a 32-bit floating-point saturate. This function compares a 32-bit floating point number to a maximum and a minimum value. The function returns the minimum value if the number is below minimum, the function returns the maximum value if the number is above the maximum. Otherwise the function returns the input number. There are a small set of special answers to have basic floating-point compatibility to basic IEEE standards.

The ARM-Cortex-M4F core takes a significant number of cycles more than some competing processors to implement this function. It has no direct instruction for a max/min operation. This unit is intended to act as a very narrow coprocessor that implements a limited accelerated saturate function in hardware.

Functional Description

The unit can operate on the bottom 1KB of configured data memory space. If enabled, accesses to this range goes through a simple co-process step to implement the minimum/maximum on written data. This is intended to sit off of the system bus.

The minimum/maximum values are at the two data locations just prior to the beginning of data space. This may allow the compiler to use small offsets and share pointer values between the minimum/maximum and the data values, maximizing code throughput. The entire 1KB region where the minimum/maximum resides aligns with the basic ARM minimum size range.

From a programmer's perspective, the FSAT essentially implements a faster way to perform floating-point saturation, as depicted in following code samples.

- **Code for floating-point saturation in C language**

```
Float fsat(Float min, Float max, Float val)
    {
    if(val < min) return (min);
    else if (val > max) return (max);
    return (val)
    }
```

- **Code for floating-point saturation using FSAT design**

```
float fsat( float min, float max, float var)
    {
    *pFSAT_MAX = max;
    *pFSAT_MIN = min;
    *pFSAT_VAR     = var;

    return ( *pFSAT_VAR );
    }
```

FSAT Performance

The FSAT unit is closely tied to the ARM Cortex-M4 core and the ARM Cortex-M4 SRAM memory. Additional processor cycles (other than what is expected on typical SRAM writes and reads) are not incurred using the FSAT unit.

Supported ARM Cortex-M4P Configuration

The following ARM Cortex-M4P SRAM configurations are supported with the FSAT unit.

Table 36-1: SRAM_CFG

SRAM_CFG	Register	Address	Description
SRAM_CFG=1	SAT_MIN	0x2000_7FF8	Min value for sat function
	SAT_MAX	0x2000_7FFC	Max value for sat function
	SAT_VAR	0x2000_8000 0x2000_83FC	If FSAT enabled: Min/Max function performed on aligned 32-bit writes. Else normal memory operation.
SRAM_CFG=2	SAT_MIN	0x2000_FFF8	Min value for sat function
	SAT_MAX	0x2000_FFFC	Max value for sat function
	SAT_VAR	0x2001_0000 0x2001_03FC	If FSAT enabled: Min/Max function performed on aligned 32-bit writes. Else normal memory operation.
SRAM_CFG=3	SAT_MIN	0x2001_7FF8	Min value for sat function
	SAT_MAX	0x2001_7FFC	Max value for sat function
	SAT_VAR	0x2001_8000 0x2001_83FC	If FSAT enabled: Min/Max function performed on aligned 32-bit writes. Else normal memory operation.
SRAM_CFG=4	SAT_MIN	0x2002_FFF8	Min value for sat function
	SAT_MAX	0x2002_FFFC	Max value for sat function
	SAT_VAR	0x2002_0000 0x2002_03FC	If FSAT enabled: Min/Max function performed on aligned 32-bit writes. Else normal memory operation.

In each SRAM configuration case mentioned above, up to 1 KB of variable memory is available as variable memory on which the saturation can be performed, while the MIN and MAX are just single 32-bit memory locations. Once FSAT is turned ON and an SRAM Config is fixed, the typical sequence would be:

- Write to MIN location
- Write to MAX location
- Write the variable value to one of 1K locations
- Read the Saturated value from the above variable location.

If MIN and MAX are initialized and do not vary, further operations can be as simple as:

- Write the variable value to one of 1K locations
- Read the Saturated value from the above variable location.

Atomicity of Operation

There are no atomic protections on write operations on MIN, MAX values or an on-going FSAT operation on a variable, since these are just treated as SRAM memory location accesses. If interrupted, the application must ensure that new writes are not performed until existing writes are verified as complete. Since there are no status bits to reflect operation status, the application may use ARM synchronization instructions. Optionally, interrupts maybe turned off during FSAT access.

Programming Model, Enable FSAT

To enable the FSAT unit, the `M4P_SRAM_CFG.SAT_COPROC` bit must be enabled.

- `#define BITM_M4P_SRAM_CFG_BIT_28 0x10000000`
- `*pREG_M4P_SRAM_CFG |= BITM_M4P_SRAM_CFG_BIT_28;`

Configure the desired ARM Cortex-M4 SRAM prior to enabling or using the FSAT block.

Event Control

FSAT is a simple block which operates directly on the ARM Cortex-M4P SRAM memory. There are no event control or other registers or configurations used with this unit.

Register Descriptions

Table 36-2: SRAM_CFG=1

Register	Address	Description
SAT_MIN	0x2000_7FF8	Minimum value for saturate function

Table 36-2: SRAM_CFG=1 (Continued)

Register	Address	Description
SAT_MAX	0x2000_7FFC	Maximum value for saturate function
SAT_VAR	0x2000_8000 0x2000_83FC	0 = normal memory operation 1 = Min/max function performed on aligned 32-bit writes in flight

Table 36-3: SRAM_CFG=2

Register	Address	Description
SAT_MIN	0x2000_FFF8	Minimum value for saturate function
SAT_MAX	0x2000_FFFC	Maximum value for saturate function
SAT_VAR	0x2001_0000 0x2001_03FC	0 = normal memory operation 1 = Min/max function performed on aligned 32-bit writes in flight

Table 36-4: SRAM_CFG=3

Register	Address	Description
SAT_MIN	0x2001_7FF8	Minimum value for saturate function
SAT_MAX	0x2001_7FFC	Maximum value for saturate function
SAT_VAR	0x2001_8000 0x2001_83FC	0 = normal memory operation 1 = Min/max function performed on aligned 32-bit writes in flight

37 Logic Block Array (LBA)

The Logic Block Array (LBA) contains a number of logic blocks which can be programmed to perform a variety of logical or arithmetic functions. The logical or arithmetic function can be defined in either Look-up Table (LUT) or product term mode. Each logic block generates one output as a function of up to 8 or 16 inputs depending upon the mode chosen. The exact function is defined by programming eight 32-bit function registers which are mapped into the processor register space.

LBA Features

The following list contains the features of the LBA module:

- Configurable per output in either LUT or PTA modes
- LUT (Look-up table) mode allows any 8-input combinational logic function
- PTA (Product Term Array) mode allows eight product terms with up to 16-inputs
- Scalable, typically up to eight independent outputs
- Combinatorial or registered output
- System inputs can be connected to system-specific signals (for example, timer outputs, TRU slaves)
- System outputs can be connected to system-specific signals (for example, TRU masters, core interrupts)

LBA Functional Description

The logic block array has eight logic block elements. The 16 logical inputs A[7:0] and B[7:0] are available to each logic block element when the LBA is configured for PTA mode. Eight logical inputs A[7:0] are available to each logic block element when LBA is configured for LUT mode. The logic inputs A[7:0] can be sourced from the general-purpose port pins, the on-chip system-level inputs (SYS_IN), the memory-mapped input register (`LBA0_DIN`), or the system feedback outputs (SYS_OUT). GPIO pins are not available as source-to-logic inputs B[7:0]. But, B[7:0] can be sourced from the system level inputs, the memory-mapped input register (`LBA0_DIN`), or the system feedback outputs. The source for each of the 16 possible inputs A[7:0] and B[7:0] is configured through the `LBA0_CFG.B7MXSEL` and `LBA0_CFG.A0MXSEL` bit fields. The ***A and B Input Sources*** table shows the possible sources for individual A and B inputs as determined by the these fields.

Table 37-1: A and B Input Sources

MXSEL[1:0]	A[n] source	B[n] source
00	Primary input GP_IN[n]	System output SYS_OUT[n]
01	System output SYS_OUT[n]	System output SYS_OUT[n]
10	Input register LBA0_DIN[n]	Input register LBA0_DIN[n]
11	System input SYS_IN[n]	System input SYS_IN[n]

The outputs Y[7:0] of the logic block array are assigned to GPIO pins by proper configuration of the system-level master pin multiplexer. One GPIO pin is available as an output for each logic block element. A total of eight GPIO pins are available for the logic block array. The system-level pin resources for LBA are limited. The system designer must choose one direction for each of the eight pins. If the system requires four LBA outputs available as GPIO pins, then only four GPIOs are available as inputs.

The logic inputs from GPIOs (GP_IN[n]) can be individually synchronized to the system clock or used asynchronously, under the control of the corresponding LBA0_SYNC_CTL.SYNCIN bit. This selection applies to all LBA blocks globally.

Clocking

The LBA module operates on internal clock which is derived from the system clock. LBA0_CLK_CTL register is used to control the clock divider. The clock divider periodically creates a synchronous update pulse whose period is controlled by the LBA0_CLK_CTL.DIVDAT field. This update pulse controls how often the logic block array registered outputs are updated. When enabled (LBA0_CLK_CTL.EN =1) the clock divide ratio is equal to LBA0_CLK_CTL.DIVDAT + 1. If disabled (LBA0_CLK_CTL.EN =0) the update signal is always high and the logic block array is updated every system clock cycle to implement a divide-by-1.

Data Input and Output

Specific data sources for the LBA module are selected using the LBA0_CFG register. The 2-bit multiplex select bit fields (LBA0_CFG.A0MXSEL – LBA0_CFG.B7MXSEL) control individual 4:1 input multiplexers for each A or B input.

The data input register (LBA0_DIN) provides another optional data input source. It is one of the data choices for each A or B input as defined by the LBA0_CFG register. This LBA0_DIN register provides direct access to a given A or B input via a memory-mapped register write. This feature can be used for:

- Providing a convenient way to select different logic functions.
- Accepting a software input into a logic function or state machine
- Accepting input from a DMA source originating in another peripheral

The LBA0_DOUT register provides real-time observability of the output states of the N logic block array registered outputs. When reading this register, the application must take into consideration that the outputs are dynamic and may change every system clock cycle.

The number of input and output pins for the LBA module in the system is limited. The LBA0_DIR register is used to control output enables and input enables at the system level. If the direction is IN, the input enable is activated for the target GPIO and the output enable is deasserted. If the direction is OUT, the output enable is activated for the target GPIO and the input enable is deasserted.

The LBA0_BLK[n]_FN0 – LBA0_BLK[n]_FN7 registers are used to implement the input-output logic for each logic block element. Eight block function registers are available for each logic block element. When using LUT mode, these eight registers are used as 8 x 32-bit RAM. The A[7:0] inputs are used as address bits for this 256 bit RAM. Depending on the address, a bit is selected as output from 256 bit RAM. In PTA mode, each LBA0_BLK[n]_FN0 – LBA0_BLK[n]_FN7 register is used to configure a product term for 16 inputs. Finally eight product terms are OR'ed to create the product-term-array function.

CM41X_M4 LBA Trigger List

Table 37-2: CM41X_M4 LBA Trigger List Masters

Trigger ID	Name	Description	Sensitivity
96	LBA0_SYS_OUT1	LBA0 Registered LBA outputs available to system 1	Edge
97	LBA0_SYS_OUT2	LBA0 Registered LBA outputs available to system 2	Edge

Table 37-3: CM41X_M4 LBA Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
96	LBA0_SYS_IN3	LBA0 LBA system inputs from Trigger Slaves, Timer Outputs etc.. 3	Pulse
97	LBA0_SYS_IN4	LBA0 LBA system inputs from Trigger Slaves, Timer Outputs etc.. 4	Pulse

CM41X_M4 LBA Interrupt List

Table 37-4: CM41X_M4 LBA Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
157	LBA0_SYS_OUT0	LBA0 Registered LBA outputs available to system	Edge	

CM41X_M4 LBA0 Register List

The LBA0 module controls system interface signal features for a number of module interfaces.

Table 37-5: CM41X_M4 LBA0 Register List

Name	Description
LBA0_BLK[n]_CTL	Logic Block Control Register
LBA0_BLK[n]_FN0	Logic Block Function 0 Register
LBA0_BLK[n]_FN1	Logic Block Function 1 Register
LBA0_BLK[n]_FN2	Logic Block Function 2 Register
LBA0_BLK[n]_FN3	Logic Block Function 3 Register
LBA0_BLK[n]_FN4	Logic Block Function 4 Register
LBA0_BLK[n]_FN5	Logic Block Function 5 Register
LBA0_BLK[n]_FN6	Logic Block Function 6 Register
LBA0_BLK[n]_FN7	Logic Block Function 7 Register
LBA0_CFG	Logic Block Array Configuration Register
LBA0_CLK_CTL	Logic Block Array Clock Divisor Register
LBA0_DIN	Logic Block Array Data Input Register
LBA0_DIR	Logic Block Array Pin Direction Register
LBA0_DOUT	Logic Block Array Data Output Register
LBA0_SYNC_CTL	Logic Block Array GPIO Input Synchronization Control Register

LBA Block Diagram

The following figures show the LBA block in LUT (Look Up Table) mode and PTA (Combinatorial) mode.

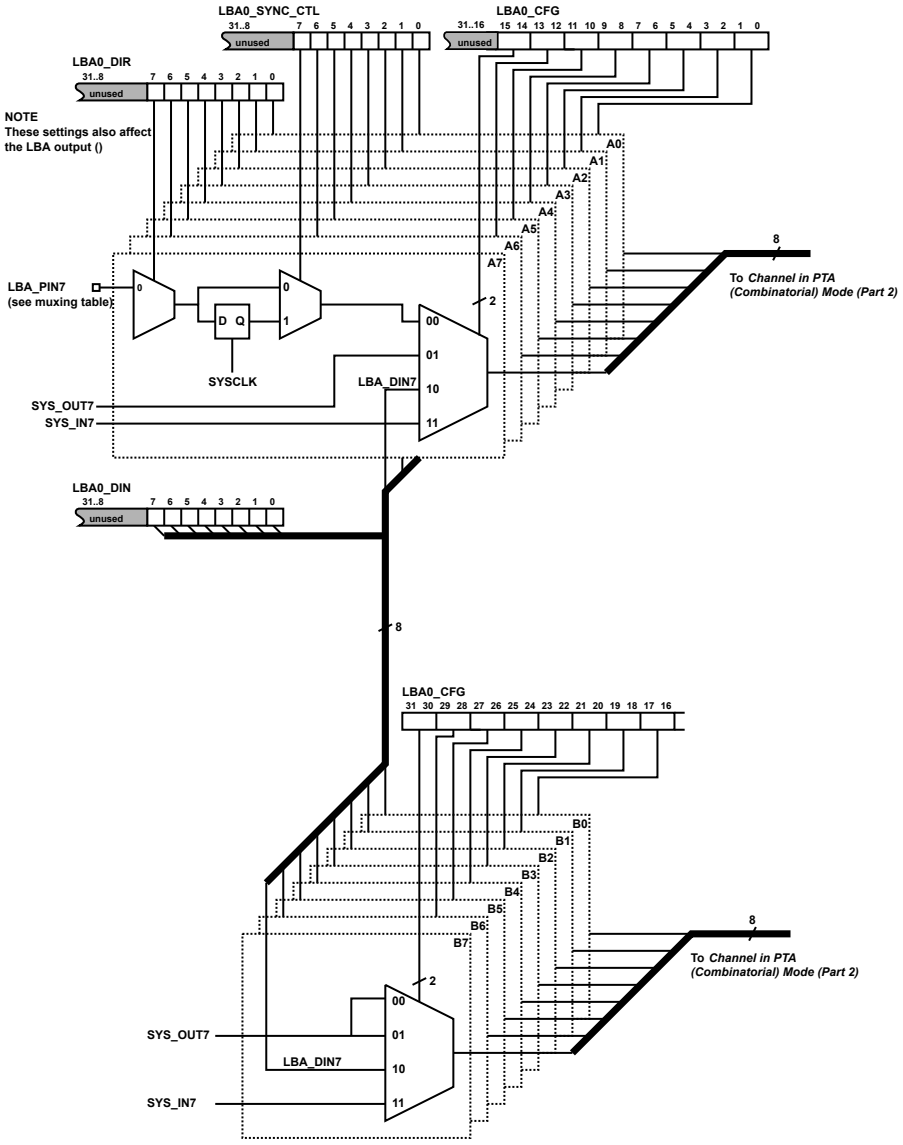


Figure 37-1: Channel in PTA (Combinatorial) Mode (Part 1)

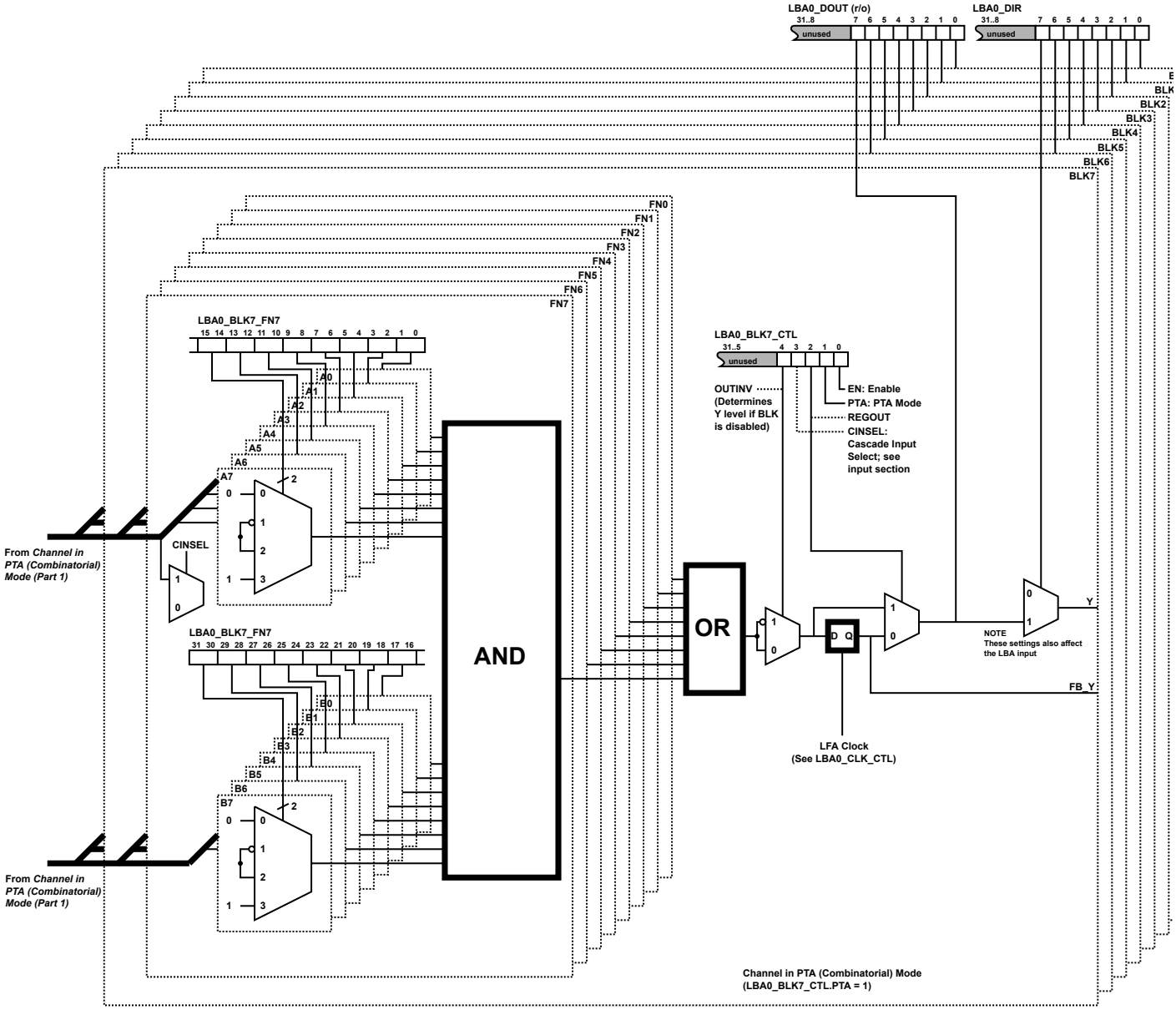


Figure 37-2: Channel in PTA (Combinatorial) Mode (Part 2)

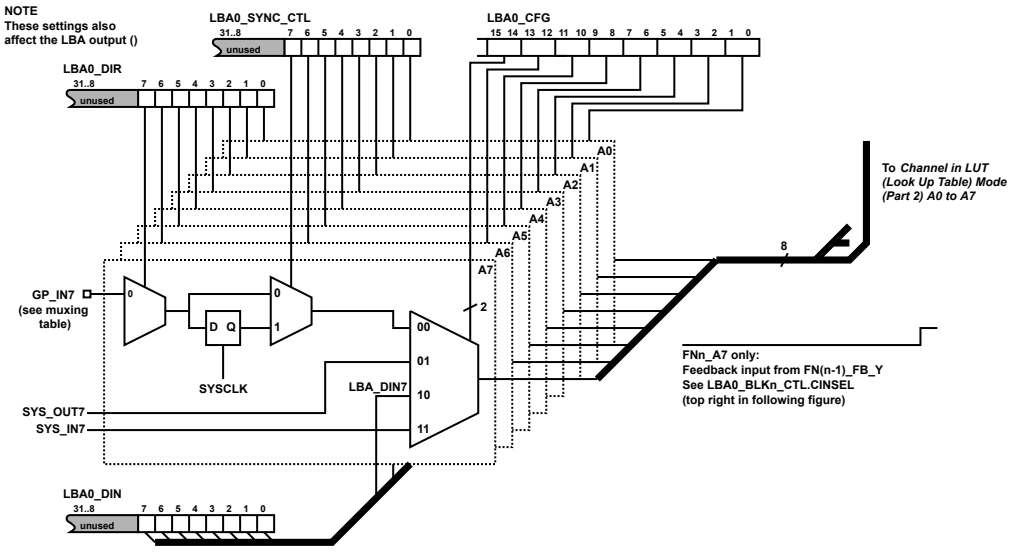


Figure 37-3: Channel in LUT (Look Up Table) Mode (Part 1)

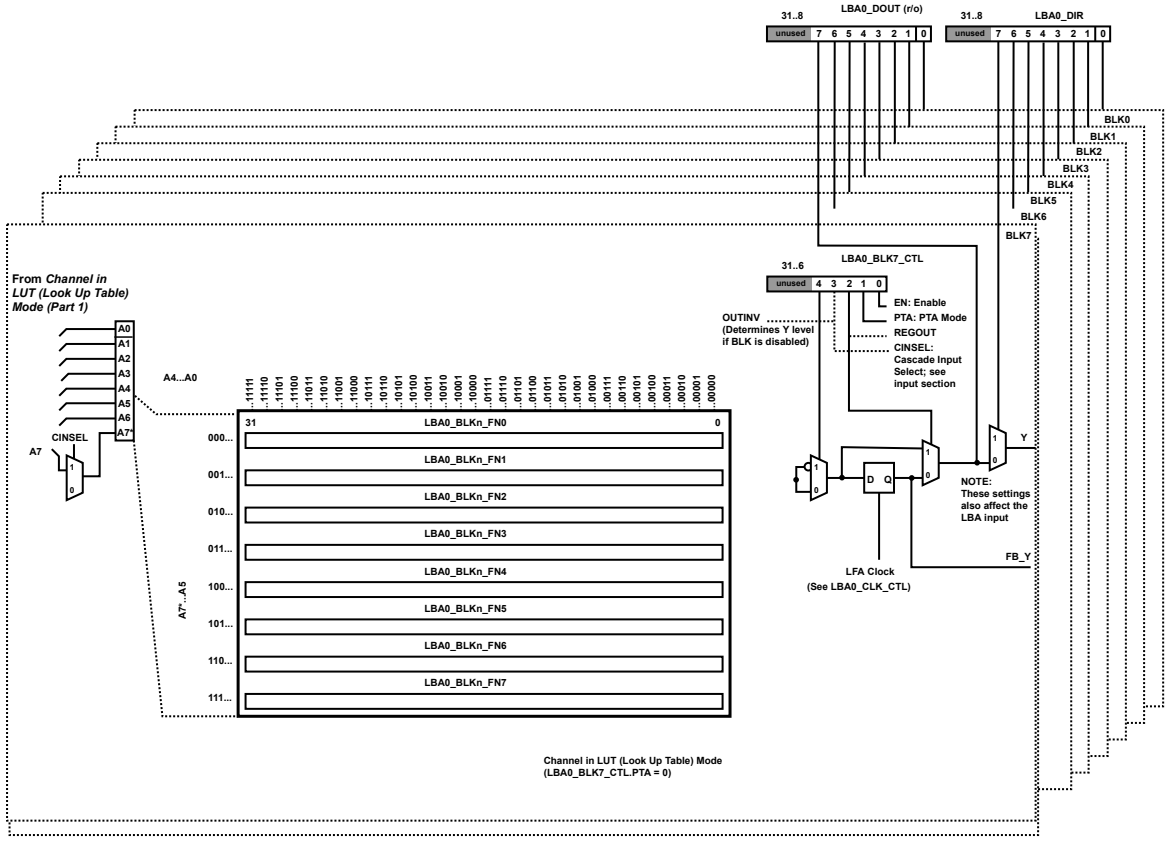


Figure 37-4: Channel in LUT (Look Up Table) Mode (Part 2)

LBA Architectural Concepts

The LBA supports the various configurations described in following sections.

Registered Output

The LBA supports registering of outputs. The registered outputs are synchronous to SYSCLK. Program the `LBA0_BLK[n]_CTL.REGOUT` bit to select registered outputs.

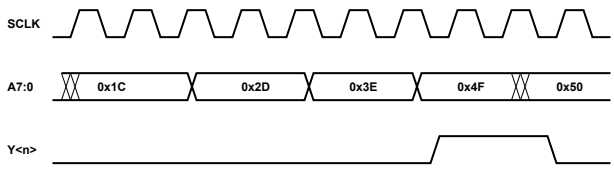


Figure 37-5: Registered Output Mode Timing

LBA Operating Modes

The LBA supports the following operating modes.

- Product Term Array Mode
- Look-up Table Mode
- Registered Output Mode
- Combinatorial Mode

Product Term Array Mode

In product term mode, each function register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X. The ***A and B Input Sources*** table describes the configurations.

Table 37-6: A and B Input Sources

2-bit Connectivity Field	Input Function	Description
00	0	Disable the entire product term
01	~A	Inverted
10	A	Non-inverted
11	X	Do-not-care

The 16 modified inputs are AND'ed to create one product term. Eight product terms are OR'ed to create the product term array function. The polarity of function is selectable. Registered output can be bypassed, if desired. Synchronized or asynchronous A[7:0] inputs can be selected using `LBA0_SYNC_CTL.SYNCIN` (synchronizer at LBA not shown). Program `LBA0_BLK[n]_CTL.PTA` to 1 to select the product term array mode. Refer to the programming concepts section for more details.

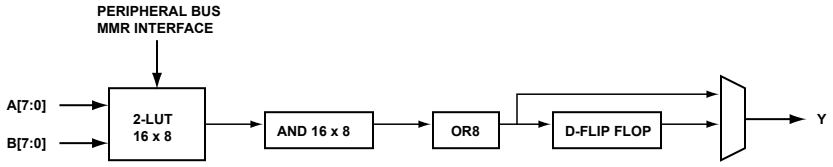


Figure 37-6: Product Term Array Mode Block

Look-up Table Mode

In look-up table mode, all 256 possible input combinations must be defined. Each bit of a function register (`LBA0_BLK[n]_FN0`) represents the output state for a specific combination of the eight inputs. The registers are organized similar to random access memory so that the A[7:0] inputs act like address bits to access a specific bit in the function registers.

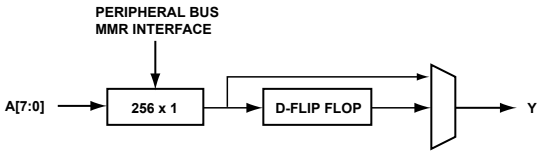


Figure 37-7: Look Up Table Mode Block

Registered Output Mode

In registered output mode, the output of the logic block is registered by the system clock rising edges. This feature allows clean output edges to be presented to the GPIO pads or to be used as feedback or cascade input terms for other logic blocks.

Combinatorial Mode

Each logic block can be individually programmed to be in either combinatorial mode or registered output mode. In combinatorial mode, there is an asynchronous logic path from the input to the output of the logic block. This feature provides flexibility for system designs, but also comes with some caveats. The output signal can transition multiple times depending primarily upon varying input timing, input delay paths and secondarily on internal logic delays.

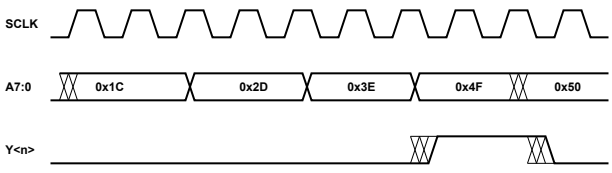


Figure 37-8: Combinatorial Mode Timing

LBA Event Control

No interrupts or status are generated by the logic block array directly.

The LBA0_SYS_IN[n] and LBA0_SYS_OUT[n] signals may be used by system designers to implement interrupts, trigger inputs and outputs as required. The LBA0_SYS_OUT2 and LBA0_SYS_OUT1 signals are m aster trigger signals for TRU1. A high pulse on these signals can trigger other trigger slaves as configured in TRU. The LBA0_SYS_OUT7 and LBA0_SYS_OUT6 signals can be used as input signals for counter. The LBA's system output can be selected as input to the counter can be by programming the SYSBLK_ROT_UPDN_CFG register. The LBA0_SYS_OUT0 signal can interrupt M4 core. The high level on he LBA0_SYS_OUT0 signal will result in the recognition of interrupt by interrupt controller of M4 core.

The trigger signals generated by various trigger masters of TRU1 can be used as source of inputs to LBA by using the the LBA0_SYS_OUT4 and the LBA0_SYS_OUT3 input signals. The TIMER_TMR3 output signal can also be used as input to LBA by using the the LBA0_SYS_OUT3 signal.

See the ***Internal SYS_IN and SYS_OUT Routing Scheme*** table.

Table 37-7: Internal SYS_IN and SYS_OUT Routing Scheme

Port Number	SYS_IN[n] Port Connection	SYS_OUT[n] Port Connection
0	0	M4/SEC1 Interrupt INTR_LBA_SYS_OUT0
1	0	TRU1 Trigger Master TRGM_LBA_SYS_OUT1
2	0	TRU1 Trigger Master TRGM_LBA_SYS_OUT2
3	TRU1 Trigger slave TRGS_LBA_SYS_IN3	Reserved
4	TRU1 Trigger slave TRGS_LBA_SYS_IN4	Reserved
5	TMR1_OUT[3] output waveform	Reserved
6	0	Counter CNT0's CNT_UD/PhaseA input
7	0	Counter CNT0's CNT_DG/PhaseB input

LBA Programming Concepts

Using the features, operating modes, and event control for the LBA to their greatest potential requires an understanding of some LBA related concepts.

Cascading Output Functions

The LBA supports cascading output functions of one block to the input of another block. Program the LBA0_BLK[n]_CTL.CINSEL bit to 1 to select cascading.

When the LBA0_BLK[n]_CTL.CINSEL bit =1, the FB_Y registered output from block N-1 is made available as input A[7] into block N. (For block 0, when the LBA0_BLK[n]_CTL.CINSEL bit =1, input A[7] is assigned a logic 0.)

NOTE: It is not possible to cascade the combinatorial, unregistered output of one block into the next block.

Output Inversion

The LBA supports selectable polarity of the programmed function equation. This inversion is applied after the LUT or AND/OR array, and before the output register and output port. It, therefore, affects the polarity of the output regardless of the setting of the LBA0_BLK[n]_CTL.REGOUT bit, and it affects the polarity of any cascaded output from the block to the next block.

If the block is not enabled (LBA0_BLK[n]_CTL.EN =0), the LBA0_BLK[n]_CTL.OUTINV bit selects the polarity of the (dsiabled) output port: if set, it selects a logic 1; otherwise, a logic 0.

Program the LBA0_BLK[n]_CTL.OUTINV to 1 to select inversion.

Registered Output

The LBA supports registering of outputs. Program the LBA0_BLK[n]_CTL.REGOUT bit to select registered outputs.

LBA Programming Examples

The logic block array allows flexible logic or arithmetic functions to be created through programming of the function registers. All function registers must be programmed before the logic block array is enabled to avoid spurious outputs during the programming. Disable each logic block before reprogramming begins. Once programmed, a logic block can be enabled and the desired programmed logic function is then available.

Look-up Table Mode Example

All function registers must be programmed before the logic block is enabled. In look-up table mode, the PTA mode LBA0_BLK[n]_CTL.PTA bit is set to 0 and the LBA0_BLK[n]_CTL.EN bit is set to 1. Programming examples are for an 8-input AND function and a 4-input OR function. The programming can be easier to understand when considering it as a 256 x 1 table of functions even though it is programmed as eight 32-bit registers.

Programming Example for an 8-input AND

In this example, Y= A7 × A6 × A5 × A4 × A3 × A2 × A1 × A0. For this function, there is only one entry in the 256 x 1 table, which is 1. Therefore, function registers LBA0_BLK[n]_FN0 through LBA0_BLK[n]_FN6 can be written to 0x0000_00000. Function register LBA0_BLK[n]_FN7 must be written to 0x8000_0000.

Register	A[7:5]	Byte3 A[4:3]=11	Byte2 A[4:3]=10	Byte1 A[4:3]=01	Byte0 A[4:3]=00
FN0	000	0000_0000	0000_0000	0000_0000	0000_0000
FN1	001	0000_0000	0000_0000	0000_0000	0000_0000
FN2	010	0000_0000	0000_0000	0000_0000	0000_0000
FN3	011	0000_0000	0000_0000	0000_0000	0000_0000
FN4	100	0000_0000	0000_0000	0000_0000	0000_0000
FN5	101	0000_0000	0000_0000	0000_0000	0000_0000

Register	A[7:5]	Byte3 A[4:3]=11	Byte2 A[4:3]=10	Byte1 A[4:3]=01	Byte0 A[4:3]=00
FN6	110	0000_0000	0000_0000	0000_0000	0000_0000
FN7	111	1000_0000	0000_0000	0000_0000	0000_0000

Look-up Table Programming for an 4-input OR function of A[3:0]

In this example, Y= A3+A2+A1+A0.

Register/Data	A[7:5]	Byte3 A[4:3]=11	Byte2 A[4:3]=10	Byte1 A[4:3]=01	Byte0 A[4:3]=00
FN0	000	1111_1111	1111_1110	1111_1111	1111_1110
FN1	001	1111_1111	1111_1110	1111_1111	1111_1110
FN2	010	1111_1111	1111_1110	1111_1111	1111_1110
FN3	011	1111_1111	1111_1110	1111_1111	1111_1110
FN4	100	1111_1111	1111_1110	1111_1111	1111_1110
FN5	101	1111_1111	1111_1110	1111_1111	1111_1110
FN6	110	1111_1111	1111_1110	1111_1111	1111_1110
FN7	111	1111_1111	1111_1110	1111_1111	1111_1110

Product Term Array Mode Examples

Example 1: 8-input AND Function

In this example:

- Y=A0 × A1 × A2 × A3 × A4 × A5 × A6 × A7
- All 2-bit input connectivity functions for A inputs must be programmed as `10` (non-inverted).
- All 2-bit input connectivity functions for B inputs must be programmed as `11` (do-not-care).
- Only one product term is required for this particular function.

Register	Product Term #	Byte3 B[7:4]	Byte2 B[3:0]	Byte1 A[7:4]	Byte0 A[3:0]
FN0	0	1111_1111	1111_1111	1010_1010	1010_1010
FN1	1	0000_0000	0000_0000	0000_0000	0000_0000
FN2	2	0000_0000	0000_0000	0000_0000	0000_0000
FN3	3	0000_0000	0000_0000	0000_0000	0000_0000
FN4	4	0000_0000	0000_0000	0000_0000	0000_0000

Register	Product Term #	Byte3 B[7:4]	Byte2 B[3:0]	Byte1 A[7:4]	Byte0 A[3:0]
FN5	5	0000_0000	0000_0000	0000_0000	0000_0000
FN6	6	0000_0000	0000_0000	0000_0000	0000_0000
FN7	7	0000_0000	0000_0000	0000_0000	0000_0000

Example 2: A 4-input OR Function

In this example:

- Y= A3 + A2 + A1 + A0;
- This function can be created with 4 individual product terms(*)
- Only one A[3:0] is programmed as `10` (non-inverted) pre product term
- All other inputs for the four product terms are programmed as `11` (do-not-care).
- The other four product terms are not required.

Register	Product Term #	Byte3 B[7:4]	Byte2 B[3:0]	Byte1 A[7:4]	Byte0 A[3:0]
FN0	0	1111_1111	1111_1111	1111_1111	1111_1110
FN1	1	1111_1111	1111_1111	1111_1111	1111_1011
FN2	2	1111_1111	1111_1111	1111_1111	1110_1111
FN3	3	1111_1111	1111_1111	1111_1111	1011_1111
FN4	4	0000_0000	0000_0000	0000_0000	0000_0000
FN5	5	0000_0000	0000_0000	0000_0000	0000_0000
FN6	6	0000_0000	0000_0000	0000_0000	0000_0000
FN7	7	0000_0000	0000_0000	0000_0000	0000_0000

CM41X_M4 LBA0 Register Descriptions

Logic Block Array (LBA0) contains the following registers.

Table 37-8: CM41X_M4 LBA0 Register List

Name	Description
`LBA0_BLK[n]_CTL`	Logic Block Control Register
`LBA0_BLK[n]_FN0`	Logic Block Function 0 Register
`LBA0_BLK[n]_FN1`	Logic Block Function 1 Register
`LBA0_BLK[n]_FN2`	Logic Block Function 2 Register

Table 37-8: CM41X_M4 LBA0 Register List (Continued)

Name	Description
LBA0_BLK[n]_FN3	Logic Block Function 3 Register
LBA0_BLK[n]_FN4	Logic Block Function 4 Register
LBA0_BLK[n]_FN5	Logic Block Function 5 Register
LBA0_BLK[n]_FN6	Logic Block Function 6 Register
LBA0_BLK[n]_FN7	Logic Block Function 7 Register
LBA0_CFG	Logic Block Array Configuration Register
LBA0_CLK_CTL	Logic Block Array Clock Divisor Register
LBA0_DIN	Logic Block Array Data Input Register
LBA0_DIR	Logic Block Array Pin Direction Register
LBA0_DOUT	Logic Block Array Data Output Register
LBA0_SYNC_CTL	Logic Block Array GPIO Input Synchronization Control Register

Logic Block Control Register

The LBA0_BLK[n]_CTL register contains bits that configure the various LBA0 modes of operation.

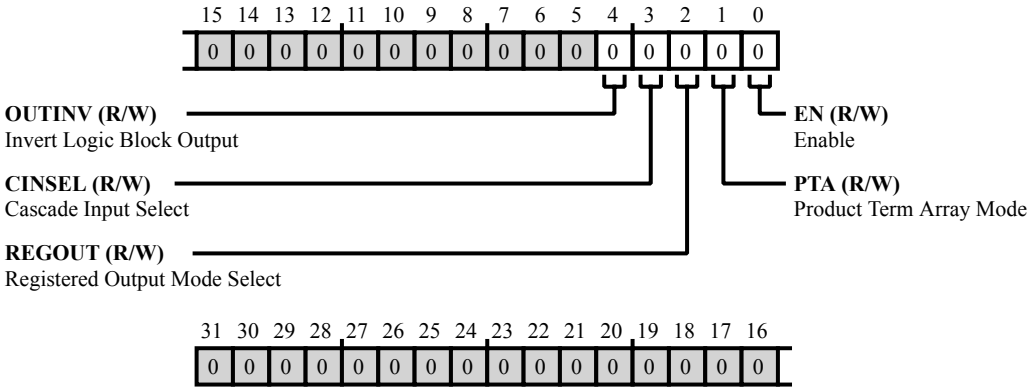


Figure 37-9: LBA0_BLK[n]_CTL Register Diagram

Table 37-9: LBA0_BLK[n]_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	OUTINV	Invert Logic Block Output. The LBA0_BLK[n]_CTL.OUTINV selects polarity of the programmed function equation. This inversion is applied after the LUT or AND/OR array, and before the output register and output port. It, therefore, affects the polarity of the output regardless of the setting of LBA0_BLK[n]_CTL.REGOUT bit, and it affects the polarity of any cascaded output from the block to the next block.
3 (R/W)	CINSEL	Cascade Input Select. The LBA0_BLK[n]_CTL.CINSEL bit selects cascading output functions of one block to the input of another block. When the LBA0_BLK[n]_CTL.CINSEL =1, the FB_Y registered output from block N-1 is made available as input A[7] into block N. (For block 0, when LBA0_BLK[n]_CTL.CINSEL =1, input A[7] is assigned a logic 0.) Note that it is not possible to cascade the combinatorial, unregistered output of one block into the next block.
2 (R/W)	REGOUT	Registered Output Mode Select. The LBA0_BLK[n]_CTL.REGOUT bit selects registered output mode. In this mode, the output of the logic block is registered by the system clock rising edges. This feature allows clean output edges to be presented to the GPIO pads or to be used as feedback and cascade input terms for other logic blocks.
1 (R/W)	PTA	Product Term Array Mode. The LBA0_BLK[n]_CTL.PTA bit configures product term array mode. The default operating mode is look-up table mode.
0 (R/W)	EN	Enable.

Logic Block Function 0 Register

In look-up table mode, the LBA0_BLK[n]_FN0 register represents the output state for a specific combination of the 8 inputs. In this mode, the register is organized similar to a random access memory so that the A7:0 inputs act like address bits to access a specific bit in this register.

In product term mode, the LBA0_BLK[n]_FN0 register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X.

For more information, see the "Programming Examples" section of this chapter.

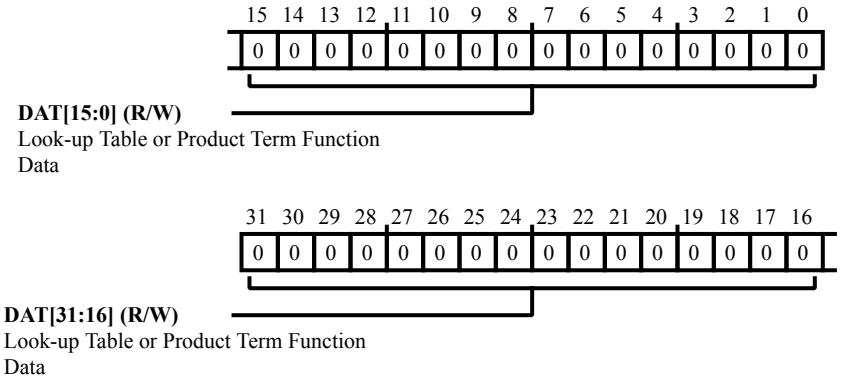


Figure 37-10: LBA0_BLK[n]_FN0 Register Diagram

Table 37-10: LBA0_BLK[n]_FN0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DAT	Look-up Table or Product Term Function Data.

Logic Block Function 1 Register

In look-up table mode, the LBA0_BLK[n]_FN1 register represents the output state for a specific combination of the 8 inputs. In this mode, the register is organized similar to a random access memory so that the A7:0 inputs act like address bits to access a specific bit in this register.

In product term mode, the LBA0_BLK[n]_FN1 register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X.

For more information, see the "Programming Examples" section of this chapter.

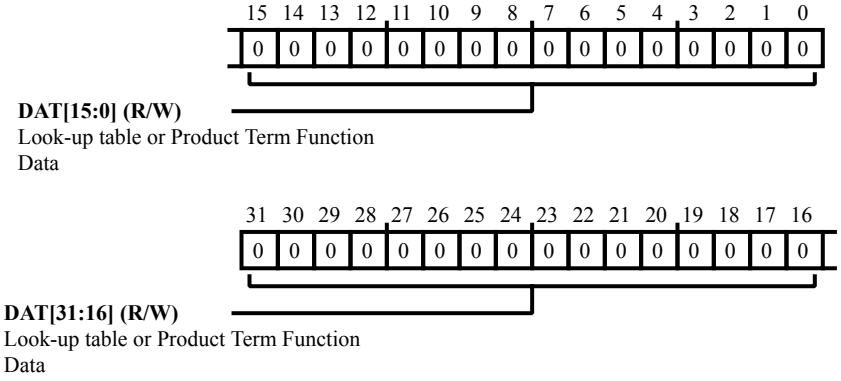


Figure 37-11: LBA0_BLK[n]_FN1 Register Diagram

Table 37-11: LBA0_BLK[n]_FN1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DAT	Look-up table or Product Term Function Data.

Logic Block Function 2 Register

In look-up table mode, the LBA0_BLK[n]_FN2 register represents the output state for a specific combination of the 8 inputs. In this mode, the register is organized similar to a random access memory so that the A7:0 inputs act like address bits to access a specific bit in this register.

In product term mode, the LBA0_BLK[n]_FN2 register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X.

For more information, see the "Programming Examples" section of this chapter.

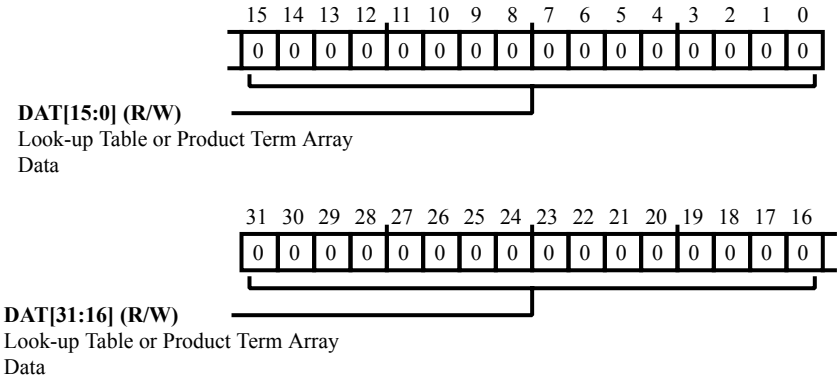


Figure 37-12: LBA0_BLK[n]_FN2 Register Diagram

Table 37-12: LBA0_BLK[n]_FN2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DAT	Look-up Table or Product Term Array Data.

Logic Block Function 3 Register

In look-up table mode, the LBA0_BLK[n]_FN3 register represents the output state for a specific combination of the 8 inputs. In this mode, the register is organized similar to a random access memory so that the A7:0 inputs act like address bits to access a specific bit in this register.

In product term mode, the LBA0_BLK[n]_FN3 register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X.

For more information, see the "Programming Examples" section of this chapter.

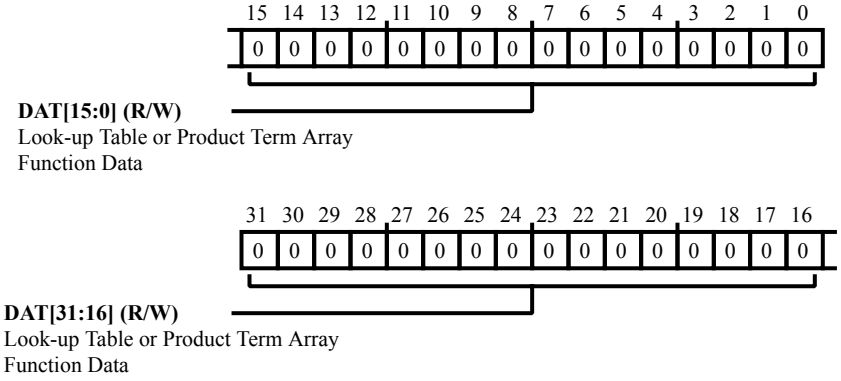


Figure 37-13: LBA0_BLK[n]_FN3 Register Diagram

Table 37-13: LBA0_BLK[n]_FN3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DAT	Look-up Table or Product Term Array Function Data.

Logic Block Function 4 Register

In look-up table mode, the LBA0_BLK[n]_FN4 register represents the output state for a specific combination of the 8 inputs. In this mode, the register is organized similar to a random access memory so that the A7:0 inputs act like address bits to access a specific bit in this register.

In product term mode, the LBA0_BLK[n]_FN4 register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X.

For more information, see the "Programming Examples" section of this chapter.

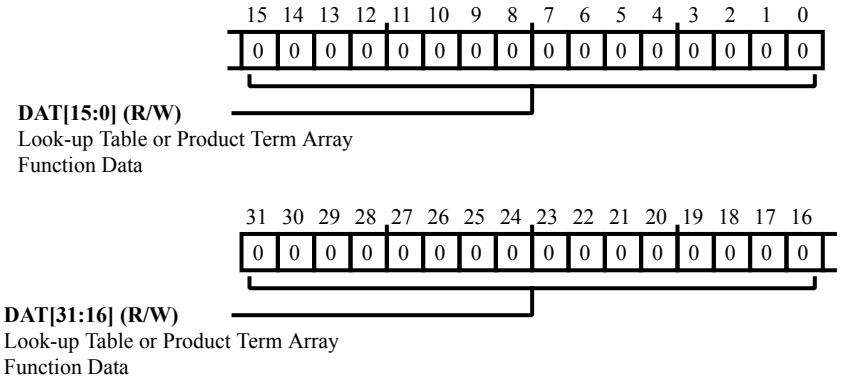


Figure 37-14: LBA0_BLK[n]_FN4 Register Diagram

Table 37-14: LBA0_BLK[n]_FN4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DAT	Look-up Table or Product Term Array Function Data.

Logic Block Function 5 Register

In look-up table mode, the LBA0_BLK[n]_FN5 register represents the output state for a specific combination of the 8 inputs. In this mode, the register is organized similar to a random access memory so that the A7:0 inputs act like address bits to access a specific bit in this register.

In product term mode, the LBA0_BLK[n]_FN5 register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X.

For more information, see the "Programming Examples" section of this chapter.

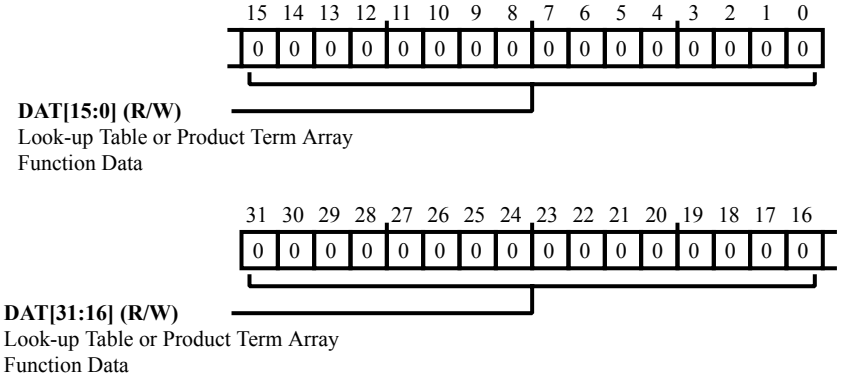


Figure 37-15: LBA0_BLK[n]_FN5 Register Diagram

Table 37-15: LBA0_BLK[n]_FN5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DAT	Look-up Table or Product Term Array Function Data.

Logic Block Function 6 Register

In look-up table mode, the LBA0_BLK[n]_FN6 register represents the output state for a specific combination of the 8 inputs. In this mode, the register is organized similar to a random access memory so that the A7:0 inputs act like address bits to access a specific bit in this register.

In product term mode, the LBA0_BLK[n]_FN6 register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X.

For more information, see the "Programming Examples" section of this chapter.

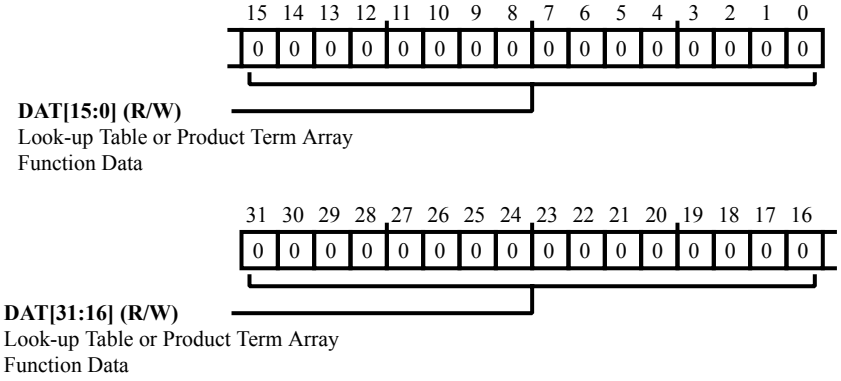


Figure 37-16: LBA0_BLK[n]_FN6 Register Diagram

Table 37-16: LBA0_BLK[n]_FN6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DAT	Look-up Table or Product Term Array Function Data.

Logic Block Function 7 Register

In look-up table mode, the LBA0_BLK[n]_FN7 register represents the output state for a specific combination of the 8 inputs. In this mode, the register is organized similar to a random access memory so that the A7:0 inputs act like address bits to access a specific bit in this register.

In product term mode, the LBA0_BLK[n]_FN7 register defines one product term. The connectivity for each of the 16 inputs is described through a 2-bit field. This 2-bit field can be programmed as: 0, A, ~A or X.

For more information, see the "Programming Examples" section of this chapter.

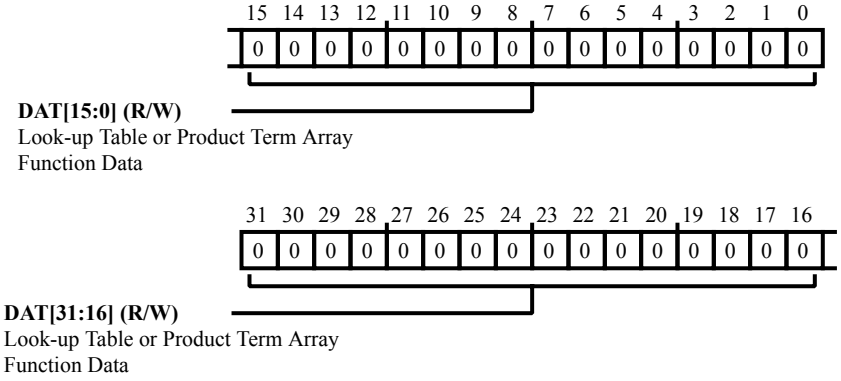


Figure 37-17: LBA0_BLK[n]_FN7 Register Diagram

Table 37-17: LBA0_BLK[n]_FN7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DAT	Look-up Table or Product Term Array Function Data.

Logic Block Array Configuration Register

The LBA0_CFG register is used to select a particular data source. The 2-bit fields control individual 4:1 input muxes for each A or B input.

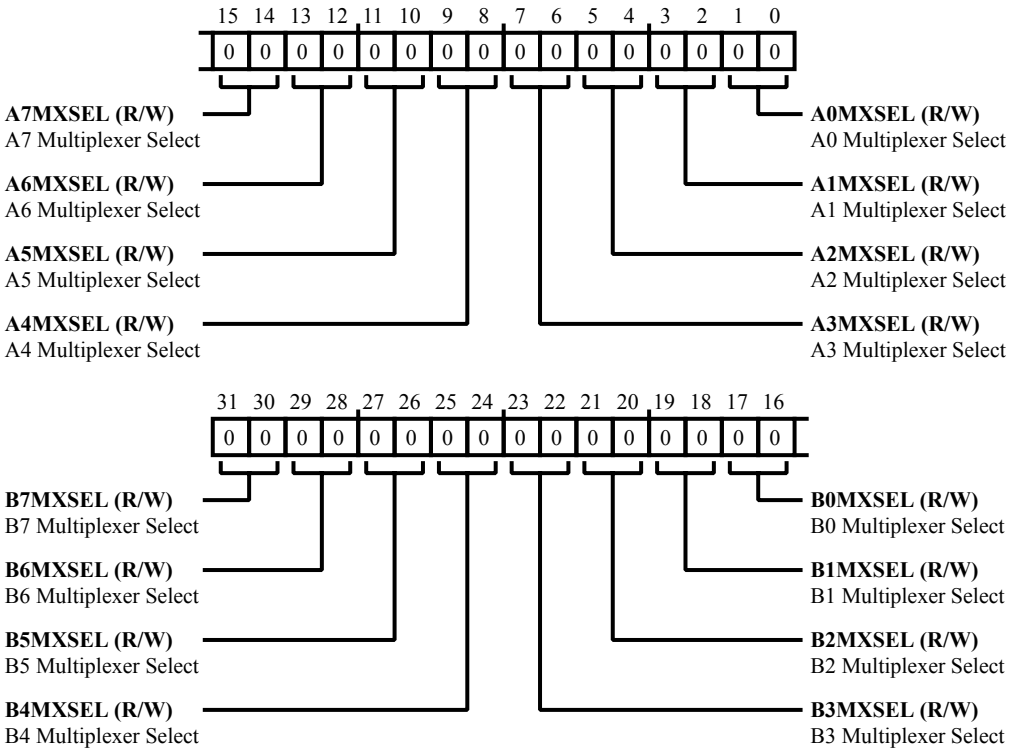


Figure 37-18: LBA0_CFG Register Diagram

Table 37-18: LBA0_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	B7MXSEL	B7 Multiplexer Select. The LBA0_CFG.B7MXSEL bit field selects the 4:1 input multiplexer for the B7 input.
29:28 (R/W)	B6MXSEL	B6 Multiplexer Select. The LBA0_CFG.B6MXSEL bit field selects the 4:1 input multiplexer for the B6 input.
27:26 (R/W)	B5MXSEL	B5 Multiplexer Select. The LBA0_CFG.B5MXSEL bit field selects the 4:1 input multiplexer for the B5 input.

Table 37-18: LBA0_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
25:24 (R/W)	B4MXSEL	B4 Multiplexer Select. The LBA0_CFG.B4MXSEL bit field selects the 4:1 input multiplexer for the B4 input.
23:22 (R/W)	B3MXSEL	B3 Multiplexer Select. The LBA0_CFG.B3MXSEL bit field selects the 4:1 input multiplexer for the B3 input.
21:20 (R/W)	B2MXSEL	B2 Multiplexer Select. The LBA0_CFG.B2MXSEL bit field selects the 4:1 input multiplexer for the B2 input.
19:18 (R/W)	B1MXSEL	B1 Multiplexer Select. The LBA0_CFG.B1MXSEL bit field selects the 4:1 input multiplexer for the B1 input.
17:16 (R/W)	B0MXSEL	B0 Multiplexer Select. The LBA0_CFG.B0MXSEL bit field selects the 4:1 input multiplexer for the B0 input.
15:14 (R/W)	A7MXSEL	A7 Multiplexer Select. The LBA0_CFG.A7MXSEL bit field selects the 4:1 input multiplexer for the A7 input.
13:12 (R/W)	A6MXSEL	A6 Multiplexer Select. The LBA0_CFG.A6MXSEL bit field selects the 4:1 input multiplexer for the A6 input.
11:10 (R/W)	A5MXSEL	A5 Multiplexer Select. The LBA0_CFG.A5MXSEL bit field selects the 4:1 input multiplexer for the A5 input.
9:8 (R/W)	A4MXSEL	A4 Multiplexer Select. The LBA0_CFG.A4MXSEL bit field selects the 4:1 input multiplexer for the A4 input.
7:6 (R/W)	A3MXSEL	A3 Multiplexer Select. The LBA0_CFG.A3MXSEL bit field selects the 4:1 input multiplexer for the A3 input.
5:4 (R/W)	A2MXSEL	A2 Multiplexer Select. The LBA0_CFG.A2MXSEL bit field selects the 4:1 input multiplexer for the A2 input.

Table 37-18: LBA0_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
3:2 (R/W)	A1MXSEL	A1 Multiplexer Select. The `LBA0_CFG.A1MXSEL` bit field selects the 4:1 input multiplexer for the A1 input.
1:0 (R/W)	A0MXSEL	A0 Multiplexer Select. The `LBA0_CFG.A0MXSEL` bit field selects the 4:1 input multiplexer for the A0 input.

Logic Block Array Clock Divisor Register

The LBA0_CLK_CTL register contains bits that enable the clock divisor and configure the frequency divide ratio of SYSCLK to the logic block array clock LFA_CLK.

NOTE: Do not change the CLK divisor while the enable is asserted.

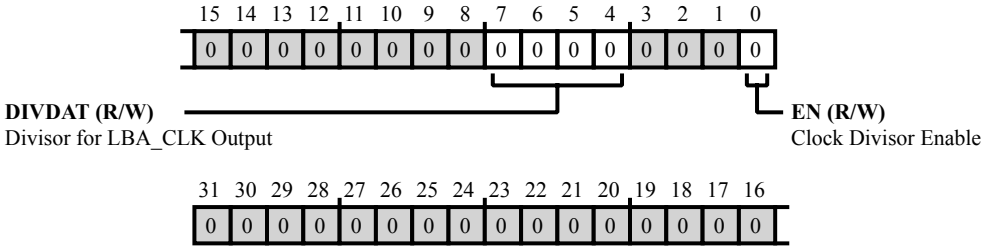


Figure 37-19: LBA0_CLK_CTL Register Diagram

Table 37-19: LBA0_CLK_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	DIVDAT	Divisor for LBA_CLK Output. The LBA0_CLK_CTL.DIVDAT bit field divisor sets the frequency divide ratio of SYSCLK to the logic block array clock LFA_CLK.
0 (R/W)	EN	Clock Divisor Enable. The LBA0_CLK_CTL.EN bit enables the clock divisor function.

Logic Block Array Data Input Register

The LBA0_DIN register is the optional data source which can be selected by the input multiplexer.

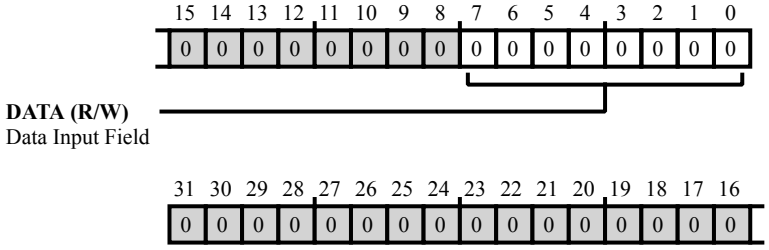


Figure 37-20: LBA0_DIN Register Diagram

Table 37-20: LBA0_DIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	DATA	Data Input Field. The LBA0_DIN.DATA bit field is the optional data source which can be selected by the input multiplexer.

Logic Block Array Pin Direction Register

The LBA0_DIR register configures the direction for the LBA0 pin signals as LBA0 data input or the logic output.

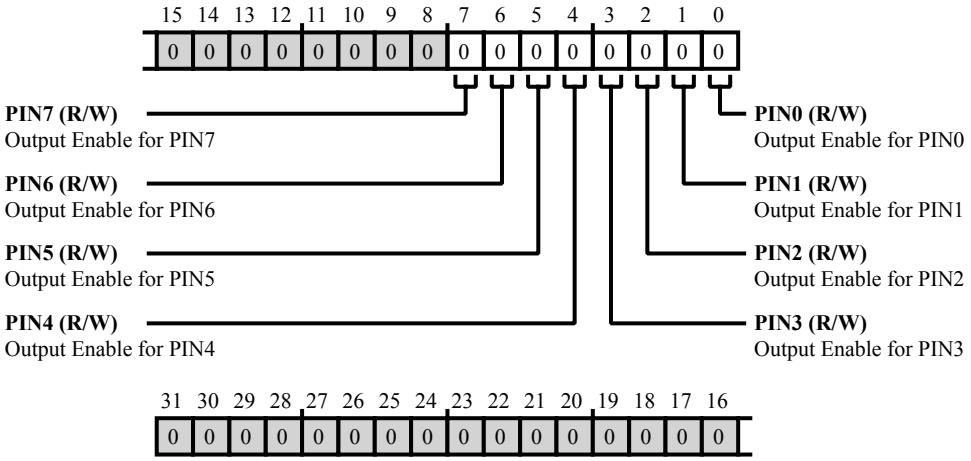


Figure 37-21: LBA0_DIR Register Diagram

Table 37-21: LBA0_DIR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/W)	PIN7	Output Enable for PIN7. The LBA0_DIR.PIN7 configures PIN7 as an output enable. Note that if output is enabled, input is disabled.
6 (R/W)	PIN6	Output Enable for PIN6. The LBA0_DIR.PIN6 configures PIN6 as an output enable. Note that if output is enabled, input is disabled.
5 (R/W)	PIN5	Output Enable for PIN5. The LBA0_DIR.PIN5 configures PIN5 as an output enable. Note that if output is enabled, input is disabled.
4 (R/W)	PIN4	Output Enable for PIN4. The LBA0_DIR.PIN4 configures PIN4 as an output enable. Note that if output is enabled, input is disabled.
3 (R/W)	PIN3	Output Enable for PIN3. The LBA0_DIR.PIN3 configures PIN3 as an output enable. Note that if output is enabled, input is disabled.
2 (R/W)	PIN2	Output Enable for PIN2. The LBA0_DIR.PIN2 configures PIN2 as an output enable. Note that if output is enabled, input is disabled.

Table 37-21: LBA0_DIR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W)	PIN1	Output Enable for PIN1. The `LBA0_DIR.PIN1` configures PIN1 as an output enable. Note that if output is enabled, input is disabled.
0 (R/W)	PIN0	Output Enable for PIN0. The `LBA0_DIR.PIN0` configures PIN0 as an output enable. Note that if output is enabled, input is disabled.

Logic Block Array Data Output Register

The LBA0_DOUT register contains the output states of the N Logic Block Array registered outputs in real time. When reading this register, the application must take into consideration that the outputs are dynamic and can change every system clock cycle.

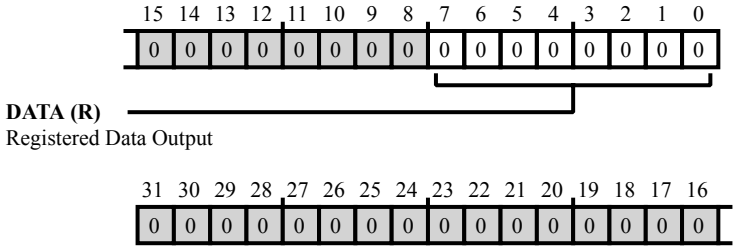


Figure 37-22: LBA0_DOUT Register Diagram

Table 37-22: LBA0_DOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	DATA	Registered Data Output. The LBA0_DOUT.DATA bit field contains the registered data outputs from each logic block.

Logic Block Array GPIO Input Synchronization Control Register

The LBA0_SYNC_CTL register configures the GPIO logic inputs to be used asynchronously as opposed to individually synchronized to the system clock. This selection applies to all LBA0 blocks globally.

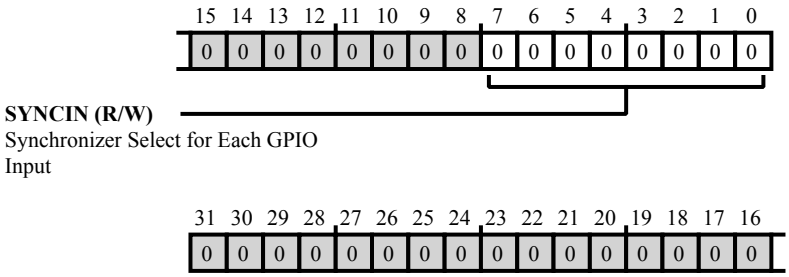


Figure 37-23: LBA0_SYNC_CTL Register Diagram

Table 37-23: LBA0_SYNC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SYNCIN	Synchronizer Select for Each GPIO Input.

38 Mailbox (MBOX)

The MBOX (mailbox) is a shared system resource, which is used to establish communication between two masters. The MBOX block has two access ports. Each access port is connected to a master block in a system, typically a processor core.

MBOX Features

The MBOX module includes the following features.

- The MBOX contains decode logic to alternate between the two masters. The preference of the master will toggle after the preferred master is granted access to the memory.
- 4 KB memory
- Access ports in the same clock domain.
- Access ports are in the same clock domain.

In the ADSP-CM41x processor the connections are:

- PORT1 is connected to the Cortex-M4.
- PORT0 is connected to the Cortex-M0.

The processor contains two register blocks

- Register block for PORT1, which contains:
    - Control registers for PORT1, auto-refresh logic, and ECC test logic.
    - Status registers for PORT1, and auto-refresh logic.
- Register block for PORT0, which contains:
    - Control fields for PORT0.
    - Status registers for PORT0.

MBOX Functional Description

The following sections describe the function of the MBOX block. Port 0 connects to master 0 which is the Cortex M0 and port 1 connects to master 1 which is the Cortex M4.

Port 0 Peripheral Bus MMR Interface

This slave interface performs register access on behalf of master 0.

Port 0 Peripheral Bus Memory Interface

This slave interface performs regular memory accesses on behalf of master 0.

Port 0 Peripheral Bus Exclusive Memory Interface

This slave interface performs exclusive memory accesses on behalf of master 0.

Port 1 Peripheral Bus MMR Interface

This slave interface performs register access on behalf of master 1.

Port 1 Peripheral Bus Memory Interface

This slave interface performs memory access on behalf of master 1.

Port 1 Peripheral Bus Exclusive Extension Interface

This interface is an extension to the peripheral bus interface that contains signals for exclusive requests and responses. These requests and responses are related to the signals defined for the ARM Cortex-M4 processor's core bus extensions for exclusive accesses.

CM41X_M0 MBOX_PORT0 Register List

Table 38-1: CM41X_M0 MBOX_PORT0 Register List

Name	Description
MBOX_PORT0_CTL	Port 0 Control Register
MBOX_PORT0_ESTAT	Port 0 ECC Status Register
MBOX_PORT0_FORCEIRQ	Port 0 Force IRQ Register
MBOX_PORT0_STAT	Port 0 Status Register

CM41X_M4 MBOX_PORT1 Register List

Table 38-2: CM41X_M4 MBOX_PORT1 Register List

Name	Description
MBOX_PORT1_CTL	Port 1 Control Register
MBOX_PORT1_ESTAT	Port 1 ECC Status Register
MBOX_PORT1_FORCEIRQ	Port 1 Force IRQ Register
MBOX_PORT1_RFRADDR	Auto-refresh Address Register
MBOX_PORT1_RFRCNT	Auto-refresh Counter Register
MBOX_PORT1_RFRPER	Auto-refresh Period Register
MBOX_PORT1_STAT	Port 1 Status Register

CM41X_M0 MBOX Interrupt List

Table 38-3: CM41X_M0 MBOX Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
4	MBOX0_PORT0_EERR	MBOX0 Port 0 ECC error	Level	
5	MBOX0_PORT1_EERR	MBOX0 Port 1 ECC error	Level	
30	MBOX0_PORT0_MSG	MBOX0 Port 0 software interrupt	Level	

CM41X_M4 MBOX Interrupt List

Table 38-4: CM41X_M4 MBOX Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
20	MBOX0_PORT1_EERR	MBOX0 Port 1 ECC error	Level	
21	MBOX0_PORT0_EERR	MBOX0 Port 0 ECC error	Level	
155	MBOX0_PORT1_MSG	MBOX0 Port 1 software interrupt	Level	
156	MBOX0_PORT0_MSG	MBOX0 Port 0 software interrupt	Level	

CM41X_M4 MBOX Trigger List

Table 38-5: CM41X_M4 MBOX Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 38-6: CM41X_M4 MBOX Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
52	MBOX0_TMR_TRIG	MBOX0 Self refresh timer trigger input	Pulse

MBOX Block Diagram

The MBOX block consists of two register blocks. The register block for port 1 includes control registers for port1, auto-refresh logic, and ECC test logic. The register block for port 0 includes control registers for port 0, auto-refresh logic, and ECC test logic. Each port can only access its own register block. The ***MBOX Block Diagram*** shows the functional blocks within the MBOX.

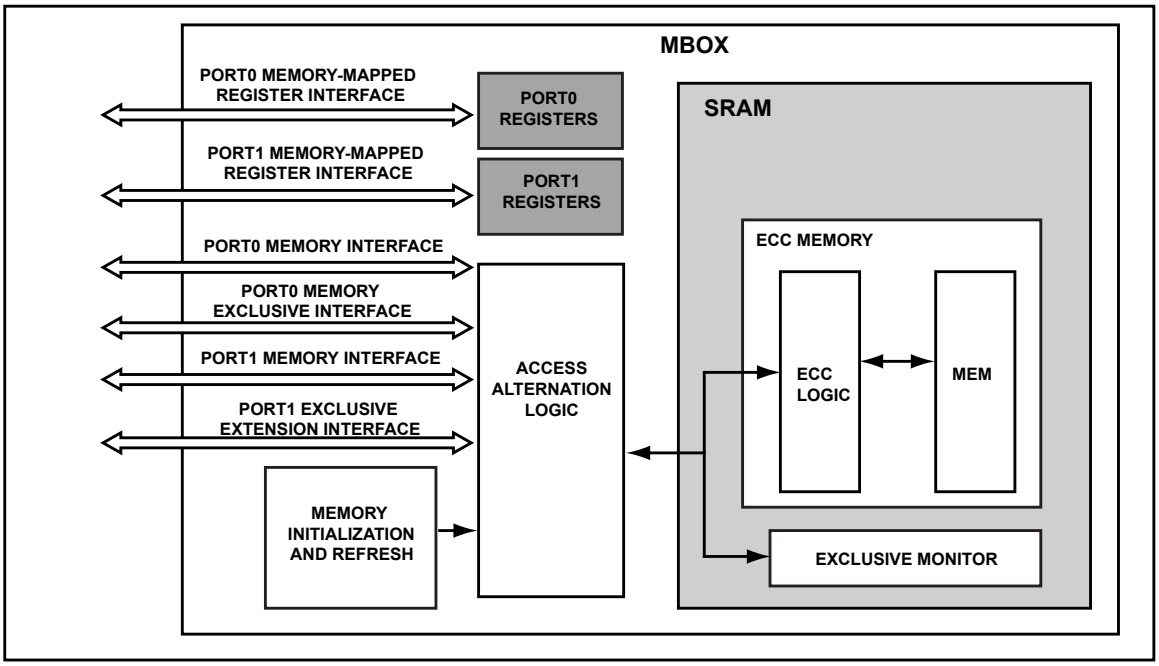


Figure 38-1: MBOX Block Diagram

MBOX Architectural Concepts

The following sections describe the blocks appearing in the MBOX Block Diagram.

- Memory Initialization
- Memory Refresh
- Memory Access
- Exclusive Memory Access
- Error Correction Code (ECC)

Memory Initialization

Memory initialization is triggered by writing 0x1 to theMBOX_PORT1_CTL.INIT bit. While memory initialization is in-progess, the MBOX_PORT1_STAT.INITPND bit is asserted until memory initialization completes. After memory initialization completes, the MBOX_PORT1_STAT.INITPND bit is cleared, and the MBOX_PORT1_STAT.INITDONE bit is asserted. Writing 0x1 to the MBOX_PORT1_STAT.INITDONE bit clears it. Otherwise, the MBOX_PORT1_STAT.INITDONE bit is always asserted, even if another memory initialization has started and is in-progress.

A write operation can write only certain bytes in a memory word. A read-modify-write operation calculates the ECC parity bits. If the read word is uninitialized, the read operation can result in an ECC error status being asserted. Then, all memory addresses must be initialized to 0x0000 for the data bits and the correct ECC parity bits.

Triggering memory initialization while a previously triggered memory initiation is in-progress is ignored and has no effect on the current memory initialization. Memory initialization blocks all memory accesses from both ports (port 0 and port 1).

Memory Refresh

Memory content can be corrupted for various reasons (for example, alpha or gamma particles). Performing a memory refresh corrects an ECC single bit error before it becomes an ECC multi-bit error. Memory is refreshed by sequentially performing a read-modify-write operation to all addresses in the memory. The data read in the read phase is checked for a single ECC bit error. The error is corrected and then written back to the memory. If multiple ECC bit errors are detected in the read data, the write operation is aborted, the port 1 MBOX_PORT1_ESTAT.EERR bit is set to 0x1, and the port 1 MBOX_PORT1_ESTAT.EERRADDR bit field is updated.

To enable memory refresh, the port 1 MBOX_PORT1_CTL.RFR control field is set to 0x1, and the refresh period MBOX_PORT1_RFRPER is set to a non-zero value.

When refresh is enabled, the following actions occur.

- The refresh logic loads the refresh period to the refresh counter register MBOX_PORT1_RFRCNT. Then, on every tmr_trig external event, the refresh counter is decremented by one.
- Once the refresh counter reaches 0x1, and the tmr_trig external event is asserted, the refresh logic reloads the refresh counter with the refresh period, and a refresh request is asserted. Note that the refresh request may have already been asserted due to a previously unserviced refresh request.
- The refresh request attempts to perform a read-modify-write operation to the address contained in the MBOX_PORT1_RFRADDR register. Read accesses caused by refresh operations have the lowest priority, and are blocked by any other access.
- Once the read access takes place, the write access occurs in the next cycle. The value in the MBOX_PORT1_RFRADDR register is incremented, and the refresh request is cleared.
- If the read access caused by the refresh operation did not take place because of collisions, the refresh address does not change, even if a new refresh request occurs (refresh counter is 0x1, and tmr_trig event is asserted).

The refresh period MBOX_PORT1_RFRPER can be changed only when the refresh function is disabled. Changing the refresh period at other times causes unspecified behaviour. Programs can overwrite the refresh counter MBOX_PORT1_RFRCNT only when the refresh function is enabled. Otherwise, writes to MBOX_PORT1_RFRCNT are ignored. The refresh address can be overwritten with the MBOX_PORT1_RFRADDR at any time. If the refresh address reaches the top of the memory (maximum value of the address) and a refresh request is being serviced, the refresh address wraps back to 0x0. Disabling the refresh operation clears any pending refresh request.

Memory Access

The MBOX SRAM memory has three address ranges in the system address map that point to the same memory array:

1. The address range for port 0 regular (non-exclusive) access
2. The address range for port 0 exclusive access
3. The address range for port 1 (regular and exclusive) access

Port 0 regular memory accesses are presented at the port 0 memory interface and port 0 exclusive memory accesses are presented at the port 0 exclusive memory interface. Both interface transactions combine internally as one master and therefore cannot have a valid transaction in the same cycle. Only one access can be active at a time. In cases of contention, there is an alternating state for which one port is preferred. This state initializes to give preference to port 0 at reset. If no access contention takes place, there is no wait state.

ECC parity bits protect every word in memory. If a write operation occurs on parts of the memory word, the whole word must be read from the memory to calculate the new ECC parity bits. Then the updated word and the new ECC bits must be written to the memory.

The ECC read-modify-write operation is uninterruptible (atomic).

Exclusive Memory Access

There is an exclusive monitor assigned to each exclusive master in port 0 and port 1. When any of the exclusive masters issue an exclusive read, the exclusive read address is stored inside the corresponding exclusive monitor, and the state of the monitor is locked.

If the state of the exclusive monitor is locked, then the exclusive monitor watches all memory write accesses (exclusive and non-exclusive) that originate in all of the masters (port 0 and port 1). If any write access matches the locked address in the monitor, the state of the monitor is unlocked.

Exclusive writes from any exclusive master check the corresponding exclusive monitor to determine:

- If the monitor is locked, and
- If the exclusive write address matches the address of the exclusive monitor

If any of these conditions fail, the exclusive write fails.

Port 0 must use an exclusive address range to perform exclusive accesses (read/write). Port 0 exclusive write accesses report the exclusive errors in the MBOX_PORT0_STAT.EXWR status field of the port 0 register block.

Port 1 uses only one address range to access (regular or exclusive) the memory. Port 1 must assert the exclusive request pin in the peripheral bus exclusive extension interface to perform exclusive accesses. The port 1 exclusive write error is reported only on the exclusive response pin in the peripheral bus exclusive extension interface.

Exclusive Access from the ARM Cortex-M4

A typical, simplified exclusive access from Cortex M4 would look like below set of pseudo code. Lock and un-lock are typical operations done utilizing the exclusive memory access, but the memory location can be called flag or variable. Exclusive operations need to be done via specific load and store operations.

1. Read the memory via `__LDREX()` operations.
2. Assume that the lock value of the memory = 1 and unlock = 0.
3. If return of `_LDREX()` = 0 it can be locked. Otherwise keep reading until it = 0
4. Write to the memory (lock) via `__STREXW()` operation.
5. `__STREXW()` returns a value = 0 if the operation is success.
6. `__STREXW()` returns a value = 1 if the operation is fail.

For further information on exclusive access and state, please visit ARM documentation.

Exclusive Access from the ARM Cortex-M0

The ARM Cortex-M0 does not have in-built exclusive instructions. Exclusive memory regions must be accessed from the ARM Cortex-M0 if exclusive accesses are preferred. Typical, simplified access would look like below set of pseudo code.

1. Read the memory via M0 exclusive memory region via normal LDR operations.
2. Assume that the lock value of the memory is 1 and unlock = 0.
3. If return of above read = 0 it can be locked. Otherwise keep reading until it = 0.
4. Write to the memory (lock) via normal write operation.
5. The `MBOX_PORT0_STAT.EXWR` bit returns a value = 0 if the operation is success.
6. The `MBOX_PORT0_STAT.EXWR` bit returns a value = 1 if the operation is fail.

Error Correction Code (ECC)

Seven parity bits protect every 32-bit word in memory. These parity bits detect single and double bit errors and can correct single bit errors.

The ECC algorithm used by the MBOX block is the Hsaio algorithm. In the Hsiao algorithm, a unique combination of three parity bits protect every data bit. Any single bit error caused by a *data bit flip* results in three error syndrome bits asserted. Any single bit error due to a *parity bit flip* results in one error syndrome bit asserted. The Hsiao algorithm is used to correct single bit errors and detect double bit errors. The Hsiao algorithm has an implementation advantage over the Hamming code in detecting double bit errors (fewer xor gates are needed).

The port 1 MBOX_PORT1_CTL.INIT control bit is used to start memory initialization with the right ECC parity bits. Any read operation checks for ECC errors. In normal mode (not ECC or data map modes), the following operations occur.

- Any single ECC error bit is corrected before sending the read data to the requested master. The data inside the SRAM array is not corrected.
- Any double ECC bit error and some multi-bit ECC errors are detected and the corresponding port MBOX_PORT1_ESTAT.EERR is set to 0x1.

The following bit fields control error reporting:

- Port 1 MBOX_PORT1_CTL.MAP control bits are used to bypass the ECC logic to read and write the raw data bits. ECC parity bits test the ECC detection and correction logic.
- Port 0 MBOX_PORT0_CTL.EERR control field is used to control the reporting of double and multi-bit errors detected during port 0 accesses.
- Port 1 MBOX_PORT1_CTL.EERR control field is used to control the reporting for double and multi-bit errors detected during port 1 accesses, or the memory refresh operation.
- Port 1 MBOX_PORT1_CTL.RFR control field is used to enable the refresh operation.

MBOX Event Control and Interrupts

Each MBOX master (port 0/port 1) can send a software interrupt request to the other master.

- Port 0 sends software interrupt request to port 1 by writing 0x1 to the port 0 MBOX_PORT0_FORCEIRQ.SWIRQ bit. This asserts the port 1 MBOX_PORT1_STAT.SWINT status bit.
- Port 1 sends software interrupt request to port 0 by writing 0x1 to the port 1 MBOX_PORT1_FORCEIRQ.SWIRQ bit. This asserts the port 0 MBOX_PORT0_STAT.SWINT status bit.
- The port 0/1 status bits are cleared by writing 0x1 to them.
- The port 0/1 initiate software interrupt request bits always read as zero (RAZ).

Interrupts

The following are the interrupt requests triggered by the MBOX module.

Table 38-7: MBOX_PORT0_CTL Register Fields

Interrupt Name	Description/Enumeration
port0_eerr	Double or multi ECC bit error was detected during port 0 memory access. The register field MBOX_PORT1_CTL.EERR must be assert to enable this interrupt request.
port1_eerr	Double or multi ECC bit bit error was detected during port 1 memory access or memory refresh operation. The register field MBOX_PORT1_CTL.EERR must be asserted to enable this interrupt request.

Table 38-7: MBOX_PORT0_CTL Register Fields (Continued)

Interrupt Name	Description/Enumeration
port0_msg	Software interrupt request to port 0 by port 1.
port1_msg	Either a software interrupt request to port 1 by port 0 or end of memory initialization.

CM41X_M0 MBOX_PORT0 Register Descriptions

MBOX Port 0 register map (MBOX_PORT0) contains the following registers.

Table 38-8: CM41X_M0 MBOX_PORT0 Register List

Name	Description
MBOX_PORT0_CTL	Port 0 Control Register
MBOX_PORT0_ESTAT	Port 0 ECC Status Register
MBOX_PORT0_FORCEIRQ	Port 0 Force IRQ Register
MBOX_PORT0_STAT	Port 0 Status Register

Port 0 Control Register

The MBOX_PORT0_CTL register contains bits that control interrupt configuration and lock the module.

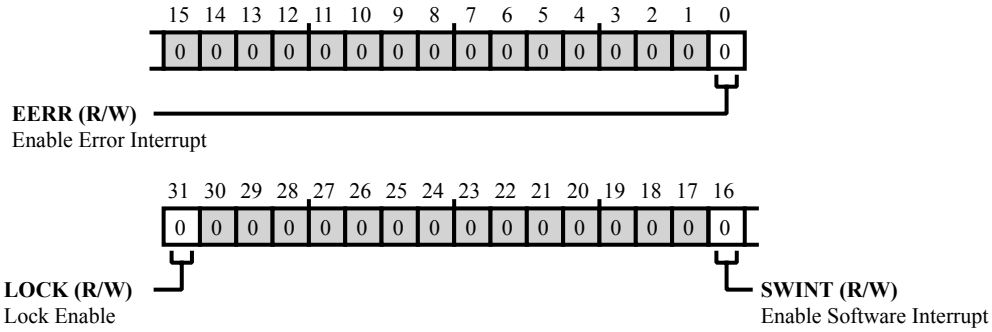


Figure 38-2: MBOX_PORT0_CTL Register Diagram

Table 38-9: MBOX_PORT0_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Enable. When both the MBOX_PORT0_CTL.LOCK bit and the global lock SPU_CTL.GLCK bit is set to 0x1, all writes to the MBOX_PORT0_CTL register are ignored.
16 (R/W)	SWINT	Enable Software Interrupt. The MBOX_PORT0_CTL.SWINT bit enables the software interrupt request to port 0.
0 (R/W)	EERR	Enable Error Interrupt. The MBOX_PORT0_CTL.EERR bit enables an interrupt request upon the detection of an ECC multi-bit (> 1 bit) error for port 0 accesses.

Port 0 ECC Status Register

The MBOX_PORT0_ESTAT register reports the status of the ECC multi-bit errors.

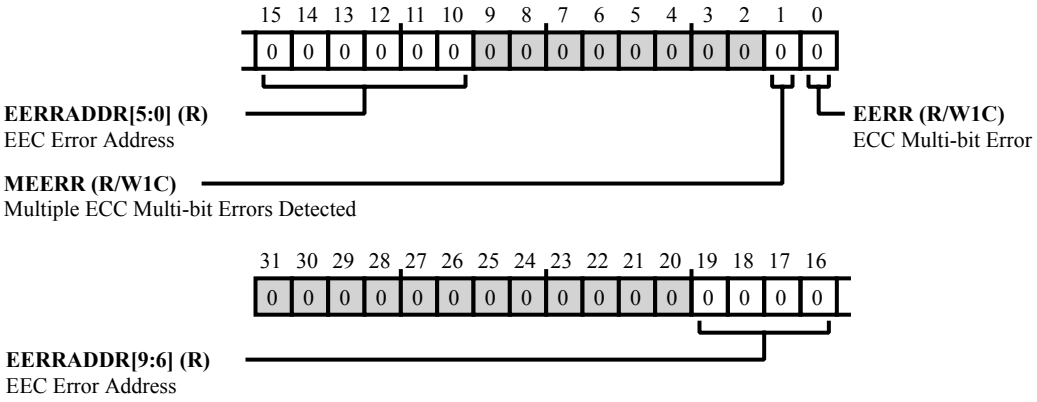


Figure 38-3: MBOX_PORT0_ESTAT Register Diagram

Table 38-10: MBOX_PORT0_ESTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:10 (R/NW)	EERRADDR	EEC Error Address. The MBOX_PORT0_ESTAT.EERRADDR bit field contains bits [11:2] of the address of the 32-bit location containing the detected ECC error event.
1 (R/W1C)	MEERR	Multiple ECC Multi-bit Errors Detected. The MBOX_PORT0_ESTAT.MEERR bit indicates that multiple ECC multi-bit errors are detected (> 1 bit), indicating more than one detection event due to port 0 accesses. Specifically, a detection event was observed while the MBOX_PORT0_ESTAT.EERR bit was already set to 1. On any detection event, the MBOX_PORT0_ESTAT.EERRADDR, MBOX_PORT0_ESTAT.EERR, MBOX_PORT0_ESTAT.MEERR, and TYPE fields are updated at the same time. If the MBOX_PORT0_ESTAT.EERR bit is cleared during the same cycle as a new detection event is being recorded, the MBOX_PORT0_ESTAT.MEERR bit is not asserted.) While the MBOX_PORT0_ESTAT.MEERR bit =0x1, the port0_eerr level interrupt request is asserted.
0 (R/W1C)	EERR	ECC Multi-bit Error. The MBOX_PORT0_ESTAT.EERR bit indicates an ECC multi-bit error (> 1 bit), indicating one detection event due to port 0 accesses. While the MBOX_PORT0_ESTAT.EERR bit =0x1, the port0_eerr level interrupt request is asserted.

Port 0 Force IRQ Register

The MBOX_PORT0_FORCEIRQ register always reads as zero.

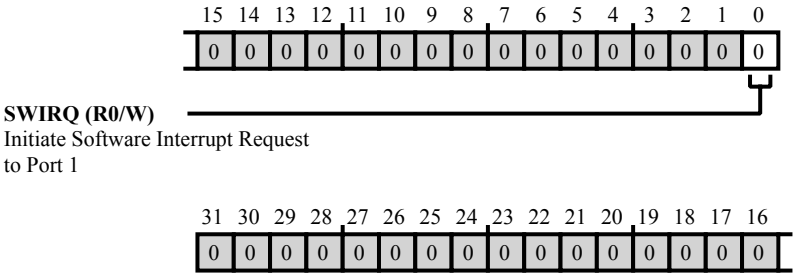


Figure 38-4: MBOX_PORT0_FORCEIRQ Register Diagram

Table 38-11: MBOX_PORT0_FORCEIRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R0/W)	SWIRQ	Initiate Software Interrupt Request to Port 1. The MBOX_PORT0_FORCEIRQ.SWIRQ bit always reads as zero.

Port 0 Status Register

The MBOX_PORT0_STAT register indicates the port 0 status.

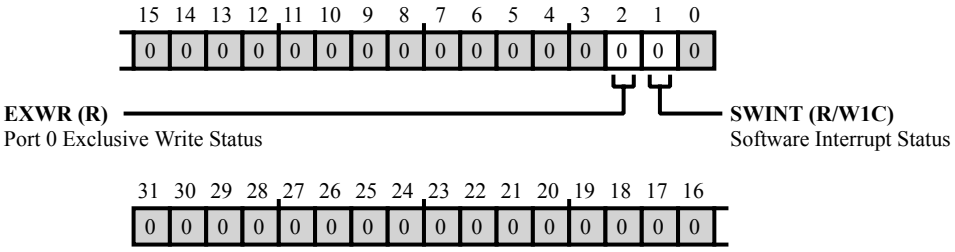


Figure 38-5: MBOX_PORT0_STAT Register Diagram

Table 38-12: MBOX_PORT0_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/NW)	EXWR	Port 0 Exclusive Write Status. The MBOX_PORT0_STAT.EXWR bit indicates the port 0 write status.	
		0	Exclusive write success, memory location was updated (equivalent to system EXOKAY status)
		1	Exclusive write fail, memory location was not updated (equivalent to system OKAY status)
1 (R/W1C)	SWINT	Software Interrupt Status. When the MBOX_PORT0_STAT.SWINT bit =0x1, the port0_msg level interrupt request is asserted.	

CM41X_M4 MBOX_PORT1 Register Descriptions

MBOX Port 1 register map (MBOX_PORT1) contains the following registers.

Table 38-13: CM41X_M4 MBOX_PORT1 Register List

Name	Description
MBOX_PORT1_CTL	Port 1 Control Register
MBOX_PORT1_ESTAT	Port 1 ECC Status Register
MBOX_PORT1_FORCEIRQ	Port 1 Force IRQ Register
MBOX_PORT1_RFRADDR	Auto-refresh Address Register
MBOX_PORT1_RFRCNT	Auto-refresh Counter Register
MBOX_PORT1_RFRPER	Auto-refresh Period Register
MBOX_PORT1_STAT	Port 1 Status Register

Port 1 Control Register

The MBOX_PORT1_CTL register contains bits that enable interrupts and control ECC function.

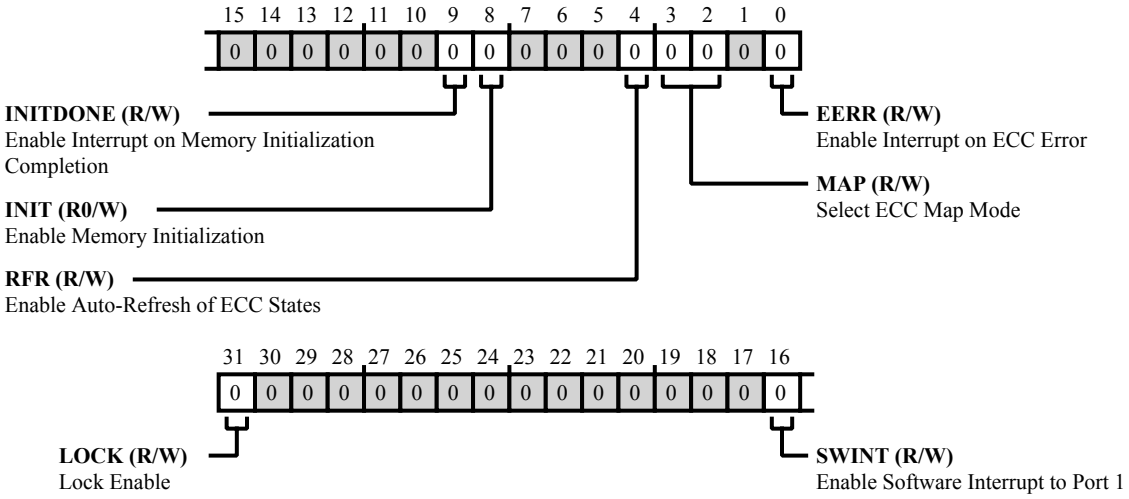


Figure 38-6: MBOX_PORT1_CTL Register Diagram

Table 38-14: MBOX_PORT1_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Enable. While both the MBOX_PORT1_CTL.LOCK bit and the global lock (SPU_CTL.GLCK) are set to 0x1, all writes to the CTL1 register are ignored.
16 (R/W)	SWINT	Enable Software Interrupt to Port 1. The MBOX_PORT1_CTL.SWINT bit enables the software interrupt request to port 1.
9 (R/W)	INITDONE	Enable Interrupt on Memory Initialization Completion. The MBOX_PORT1_CTL.INITDONE bit enables an interrupt request upon memory initialization completion.
8 (R0/W)	INIT	Enable Memory Initialization. The MBOX_PORT1_CTL.INIT bit starts initialization of the MBOX_PORT1 memory to all zeros and initializes the ECC states. This bit always reads as zero (RAZ). Writing 0x1 to this field while memory initialization is in-progress does not have any effect. Memory initialization has the highest priority, blocking port 0 and port 1 core accesses (causing access stalls). Writing 0x1 to the MBOX_PORT1_CTL.INIT bit clears all memory content (all data that existed in the memory before the initialization started).

Table 38-14: MBOX_PORT1_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	RFR	Enable Auto-Refresh of ECC States. The `MBOX_PORT1_CTL.RFR` bit enables an auto-refresh of the ECC states. No auto-refresh operation takes place while `MBOX_PORT1_RFRPER.VALUE` = 0x0, or the `MBOX_PORT1_CTL.MAP` bit field is set to map data or to map ECC.	
3:2 (R/W)	MAP	Select ECC Map Mode. The `MBOX_PORT1_CTL.MAP` bit field enables access to the raw data and ECC parity bits in memory.	
		0	Enable ECC normal mode. ECC parity generated on writes are deposited in memory with the raw write data; corrected data is returned on reads.
		2	Enable ECC Map Data Mode. ECC generation is bypassed; writes are stored to the 32-bit data only. Prior stored ECC bits are not updated. The field can be used to deposit data indicating single- or multiple-bit errors. On reads, the ECC is disabled, and the raw 32-bits array data is returned without correction; no errors or warnings are generated. Note: only a 32-bit access size is supported.
		3	Enable ECC map parity mode. ECC generation is bypassed; bits [6:0] of writes are stored to the ECC array; the data array is not updated. The field can be used to deposit ECC states indicating single or multiple-bit errors. On reads, the ECC bits are returned on data bits [6:0] of the read data values.
0 (R/W)	EERR	Enable Interrupt on ECC Error. The `MBOX_PORT1_CTL.EERR` bit enables the interrupt request on detection of ECC multi-bit (> 1 bit) error for port 1 accesses and auto-refresh accesses.	

Port 1 ECC Status Register

The MBOX_PORT1_ESTAT register reports information related to ECC errors and interrupts.

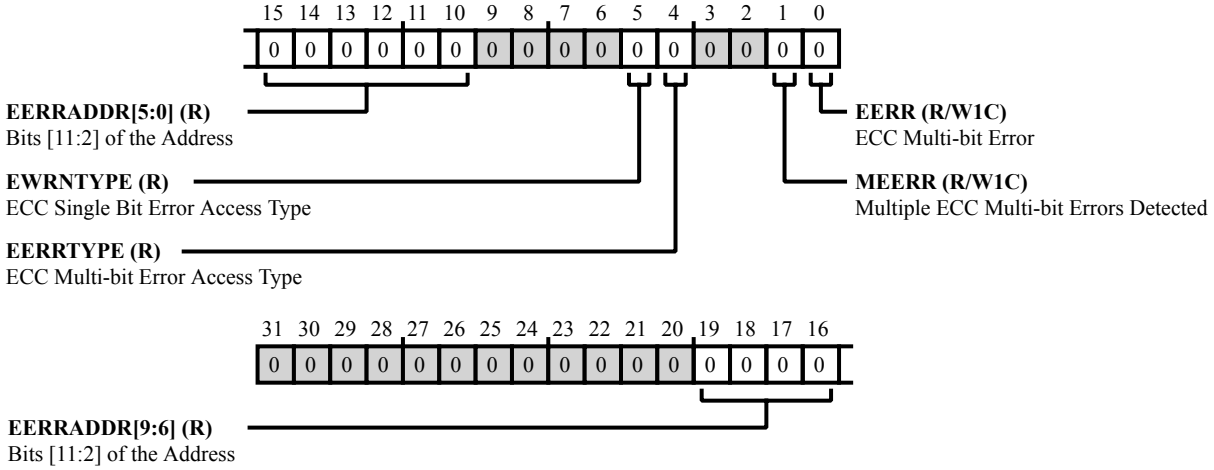


Figure 38-7: MBOX_PORT1_ESTAT Register Diagram

Table 38-15: MBOX_PORT1_ESTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19:10 (R/NW)	EERRADDR	Bits [11:2] of the Address. The MBOX_PORT1_ESTAT.EERRADDR bit field is bits [11:2] of the address of the 32-bit location containing the detected ECC error event.	
5 (R/NW)	EWRNTYPE	ECC Single Bit Error Access Type. The MBOX_PORT1_ESTAT.EWRNTYPE bit indicates the access type that caused the most recent pending ECC multi-bit error detection event.	
		0	Port 1 access
		1	Auto-refresh access
4 (R/NW)	EERRTYPE	ECC Multi-bit Error Access Type. The MBOX_PORT1_ESTAT.EERRTYPE bit indicates the access type that caused the most recent pending ECC single bit error detection event.	
		0	Port 1 access
		1	Auto refresh access

Table 38-15: MBOX_PORT1_ESTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	MEERR	Multiple ECC Multi-bit Errors Detected. The MBOX_PORT1_ESTAT.MEERR bit reports that multiple ECC multi-bit errors have been detected (> 1 bit). This bit indicates that more than one detection event occurred due to either a port 1 access or an auto-refresh access. Specifically, a detection event was observed while the MBOX_PORT1_ESTAT.EERR bit was already set to 1. On any detection event, the MBOX_PORT1_ESTAT.EERRADDR, MBOX_PORT1_ESTAT.EERR, MBOX_PORT1_ESTAT.MEERR, and MBOX_PORT1_ESTAT.EERRTYPE fields are updated at the same time. If the MBOX_PORT1_ESTAT.EERR bit is cleared at the same cycle as a new detection event is being recorded, this bit is not asserted. While this bit = 0x1, the port1_eerr level interrupt request is asserted.
0 (R/W1C)	EERR	ECC Multi-bit Error. The MBOX_PORT1_ESTAT.EERR bit reports an ECC multi-bit error (> 1 bit), indicating one detection event due to either a port 1 access or an auto-refresh access. While the MBOX_PORT1_ESTAT.EERR bit = 0x1, the port1_eerr level interrupt request is asserted.

Port 1 Force IRQ Register

The MBOX_PORT1_FORCEIRQ register reports information related to port 1 force interrupt requests.

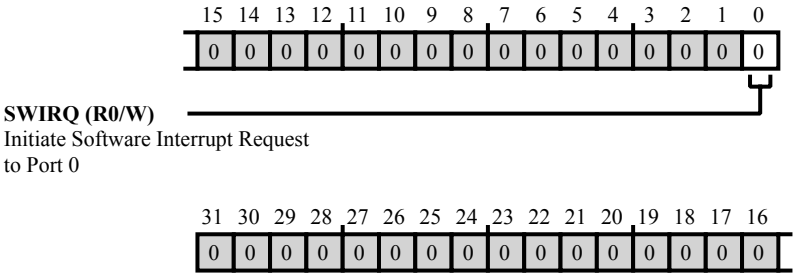


Figure 38-8: MBOX_PORT1_FORCEIRQ Register Diagram

Table 38-16: MBOX_PORT1_FORCEIRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R0/W)	SWIRQ	Initiate Software Interrupt Request to Port 0. The MBOX_PORT1_FORCEIRQ.SWIRQ bit always reads as zero.

Auto-refresh Address Register

The MBOX_PORT1_RFRADDR register indicates the offset from the base of the MBOX_PORT1 address space to the address of the next 32-bit location to be refreshed (bits[1:0] are always 'b00). The auto-refresh address is incremented by 4 after a refresh operation completes, wrapping back to offset address 0x000 after the maximum address offset is refreshed (0xffc > 0x000).

If a refresh operation is unable to complete during the refresh period defined by MBOX_PORT1_RFRPER.VALUE due to total saturation by the traffic on the two ports port 0 and port 1, then the MBOX_PORT1_RFRADDR.VALUE is observed to not increment during the refresh period.

Programs can overwrite the current auto-refresh address value. If auto-refresh MBOX_PORT1_CTL.RFR is disabled and then enabled, the initial auto-refresh address does not change, and the last value of the auto-refresh address is used as the initial auto-refresh address.

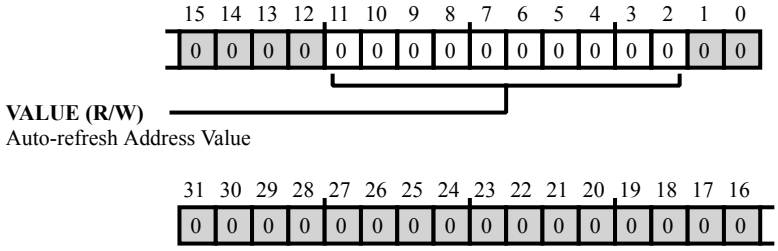


Figure 38-9: MBOX_PORT1_RFRADDR Register Diagram

Table 38-17: MBOX_PORT1_RFRADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:2 (R/W)	VALUE	Auto-refresh Address Value. The MBOX_PORT1_RFRADDR.VALUE bit field indicates the offset from the base of the MBOX_PORT1 address space to the address of the next 32-bit location to be refreshed.

Auto-refresh Counter Register

The MBOX_PORT1_RFRCNT register holds the current 32-bit ECC auto-refresh count down counter value. This register is loaded with the MBOX_PORT1_RFRPER.VALUE value when the MBOX_PORT1_CTL.RFR bit is set to 0x1. The MBOX_PORT1_RFRCNT.VALUE value decrements by 1 every external tick event which originates from a system resource (for example, a timer).

When the counter reaches 0x1, on the next tick event, it is reloaded with the MBOX_PORT1_RFRPER.VALUE value. On the cycle when the MBOX_PORT1_RFRCNT.VALUE wraps around (0x1 -> MBOX_PORT1_RFRPER.VALUE), a refresh request is asserted. An MBOX_PORT1 refresh cycle is made on the next idle cycle (cycle in which no port access is made by either port) starting with the cycle in which refresh request is asserted. If the auto-refresh access fails to occur before the next auto-refresh request, the program can ignore it.

The MBOX_PORT1_RFRCNT.VALUE value stops updating the cycle after the MBOX_PORT1_CTL.RFR bit is disabled, or the value 0x0 is written to the MBOX_PORT1_RFRPER.VALUE bit field. Programs can overwrite the auto-refresh counter current value. If the MBOX_PORT1_CTL.RFR bit is disabled, or the value 0x0 is written to the MBOX_PORT1_RFRPER.VALUE bit field, writes to MBOX_PORT1_RFRCNT.VALUE bit field are ignored.

If the MBOX_PORT1_CTL.RFR bit is disabled and then enabled, and the auto-refresh period is not equal to 0x0, load the auto-refresh counter with the refresh period MBOX_PORT1_RFRPER.VALUE value.

Writing 0x0 to the auto-refresh counter is illegal. It is ignored.

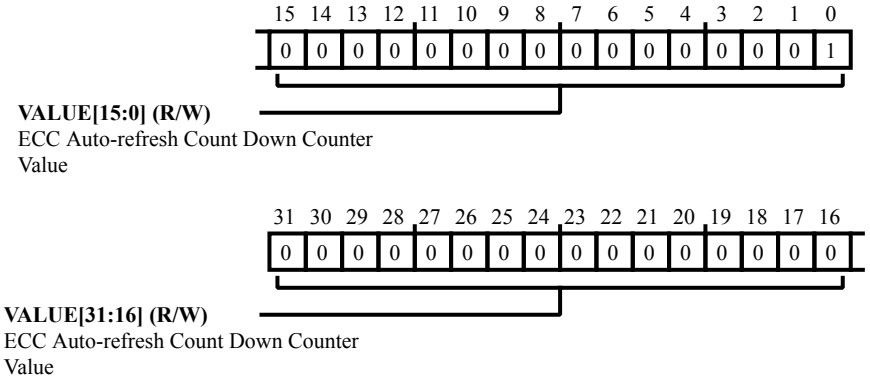


Figure 38-10: MBOX_PORT1_RFRCNT Register Diagram

Table 38-18: MBOX_PORT1_RFRCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	ECC Auto-refresh Count Down Counter Value. The MBOX_PORT1_RFRCNT bit field holds the current 32-bit ECC auto-refresh count down counter value.

Auto-refresh Period Register

The MBOX_PORT1_RFRPER register provides the 32-bit ECC refresh counter period. This is the number of external pulse counts between successive requests for refresh, when the refresh enable bit (MBOX_PORT1_CTL.RFR) is set to 1. Writing the value 0x0 in the auto-refresh period, immediately stops auto-refresh operation.

Disable the auto-refresh enable control bit field (MBOX_PORT1_CTL.RFR) before changing the auto-refresh period, then enable the MBOX_PORT1_CTL.RFR bit after the change. The behavior of changing the auto-refresh period while the MBOX_PORT1_CTL.RFR bit is enabled results in unspecified behavior.

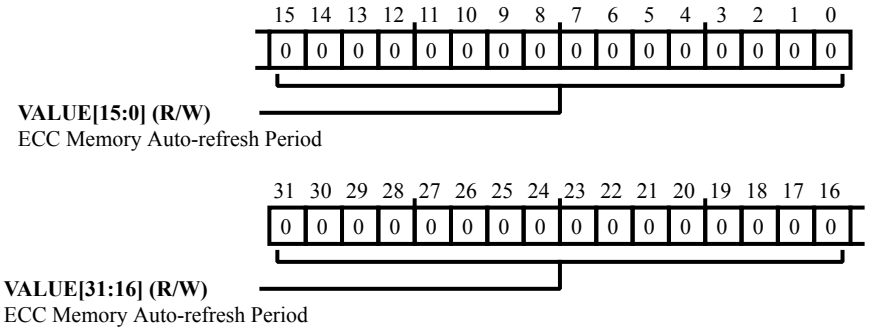


Figure 38-11: MBOX_PORT1_RFRPER Register Diagram

Table 38-19: MBOX_PORT1_RFRPER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	ECC Memory Auto-refresh Period. The MBOX_PORT1_RFRPER.VALUE bit field provides the 32-bit ECC refresh counter period.

Port 1 Status Register

The MBOX_PORT1_STAT register provides information about the status of port 1.

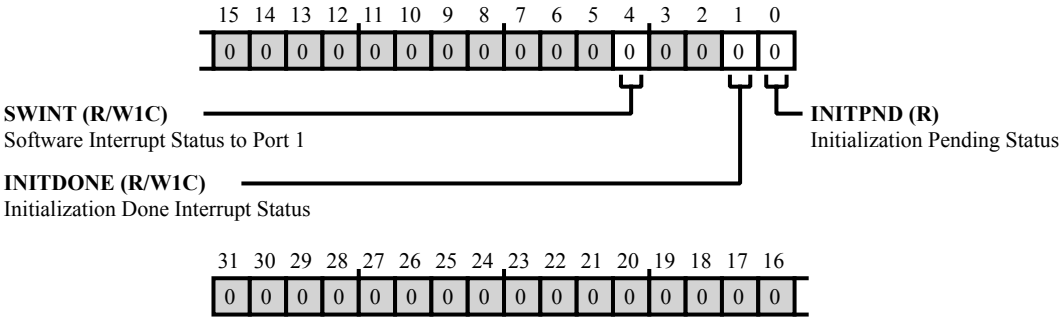


Figure 38-12: MBOX_PORT1_STAT Register Diagram

Table 38-20: MBOX_PORT1_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W1C)	SWINT	Software Interrupt Status to Port 1. While the MBOX_PORT1_STAT.SWINT bit = 0x1, the port1_msg level interrupt request is asserted. This status bit is cleared to 0x0 by writing 0x1 to it.
1 (R/W1C)	INITDONE	Initialization Done Interrupt Status. The MBOX_PORT1_STAT.INITDONE bit is set to 0x1 on the cycle after the last initialization write is done. This status field is cleared to 0x0 by writing 0x1 to this bit. If the MBOX_PORT1_CTL.INITDONE bit = 0x1 and the MBOX_PORT1_STAT.INITDONE bit = 0x1, the port1_msg level interrupt request is asserted.
0 (R/NW)	INITPND	Initialization Pending Status. The next cycle after writing 0x1 to the MBOX_PORT1_CTL.INIT bit, initialization starts and the MBOX_PORT1_STAT.INITPND bit is set to 0x1. The cycle after the last memory initialization write takes place, this bit is cleared (0x0), and the MBOX_PORT1_STAT.INITDONE bit is set (0x1) if the (MBOX_PORT1_CTL.INITDONE bit == 0x1).

39 Voltage Monitoring Unit (VMU)

The Voltage Monitoring Unit (VMU) provides over-voltage and under-voltage detection by monitoring the VDD_INT and VDD_EXT and generates asynchronous signals if the voltages exceed or drop below the programmable limits. The VMU also generates a control signal for the power sequencing requirements of the embedded flash.

VMU Features

The VMU module supports the following safety features:

- Over-voltage and under-voltage detection on the VDD_INT and VDD_EXT
- Disable feature for over-voltage detection for both VDD_INT and VDD_EXT supplies
- Programmable under-voltage thresholds
- Generates flash power down signal during power event
- Programmable (1 µs to 17 µs) fault delay signal
- GPIO Pin Safe States

VMU Functional Description

The VMU module provides voltage monitoring on the the VDD_INT and VDD_EXT power rails when the processor is powered up. When enabled, the VMU triggers an interrupt for a power event in either the VDD_INT or VDD_EXT power domain. The VMU (whether enabled or not) also provides a control signal to the internal flash to facilitate the flash power-down requirements in case of an over-voltage or under-voltage power event.

The VMU provides the ability to program the Low Voltage thresholds for VDD_INT or VDD_EXT monitoring via the REF_PRG [1:0] bits. Over-voltage threshold is a fixed value.

Refer to the ***ADSP-CM41x Processor Data Sheet*** for exact voltage threshold values.

The table shows the thresholds for VDD_EXT (external / IO power supply).

Table 39-1: Thresholds for VDD_EXT

Threshold	Description	REF_PRG[1]
OVLO_E	Over-Voltage threshold detection	-
UVLO_E1	Under-Voltage threshold detection, Level 1	0
UVLO_E2	Under-Voltage threshold detection, Level 2	1

The table shows the thresholds for VDD_INT (internal/core).

Table 39-2: Thresholds for VDD_INT

Threshold	Description	REF_PRG[1]
OVLO_I	Over-Voltage threshold detection	-
UVLO_I1	Under-Voltage threshold detection, Level 1	0
UVLO_I2	Under-Voltage threshold detection, Level 2	1

VMU0_VDDINT_EVT is asserted when VDD_EXT is outside the threshold window. VMU0_VDDEXT_EVT is asserted when VDD_INT is outside the threshold window. The VMU tolerates/ignores power supply transients of short durations (~1 us) to avoid premature tripping.

NOTE: The specified precision of VMU trip levels (threshold) are valid only when the feature is used in conjunction with the part's internal LDO regulator. Using an external supply for VDD_INT can result in inaccurate trip levels for the VMU's comparators.

CM41X_M4 PADS Register List

The PADS module controls system interface signal features for a number of module interfaces.

Table 39-3: CM41X_M4 PADS Register List

Name	Description
PADS_DBC[n]_CTL	Debounce Control Register(s)
PADS_DBC_PRESCALE	Debounce Prescale Register
PADS_FOCP_DIV	Fast Over Current Protection Clock Divisor Register
PADS_MONOSC_CFG	Monitor Oscillator Control Register
PADS_NVWR_RSTCTL	Non-Volatile Write Reset Control Register
PADS_PCFG0	Peripheral Configuration0 Register
PADS_PORT[n]_DS	Multi Port Drive Strength Control Register
PADS_PORT[n]_RCTL	Multi Port Pull-up/Pull-down Resistor Control Register
PADS_PORT[n]_TRIPSEL	Multi Port Trip Select Register
PADS_PORT[n]_TRIPST	Multi Port Trip State Register

Table 39-3: CM41X_M4 PADS Register List (Continued)

Name	Description
PADS_VMU_CTL	Voltage Monitor Unit Control Register
PADS_VMU_TRIM	Voltage Monitor Unit Trim Register
PADS_VMU_TRIPEN	Voltage Monitor Unit Trip Enable Register

CM41X_M0 VMU Interrupt List

Table 39-4: CM41X_M0 VMU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
0	VMU0_VDDEXT_EVT	VMU0 External voltage management fault	Level	
0	VMU0_VDDINT_EVT	VMU0 Internal voltage management fault	Level	

CM41X_M4 VMU Interrupt List

Table 39-5: CM41X_M4 VMU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
0	VMU0_VDDINT_EVT	VMU0 Internal voltage management fault	Level	
1	VMU0_VDDEXT_EVT	VMU0 External voltage management fault	Level	

Input Supply Ramp Rate and Bypassing Requirements

Ideally, bypass caps must be connected at multiple IC pins for a safe operation in case of a pin lift or capacitor failure.

Refer to the ***ADSP-CM41x Processor Data Sheet*** for information about power down timing with VMU.

If the power down timing requirement is met, the VMU triggers a fault while device logic is still operational, and a controlled shutdown of the embedded flash can be performed.

If the power down timing is violated, the VMU trips after logic voltages are out of specification. Controlled device shutdown is not guaranteed and the flash may be corrupted.

Programmable Fault Delay

The VMU provides a programmable (1 µs to 17 µs) delay between the early and late fault outputs. The OSCWD auxillary clock is used for this purpose. See the Oscillator Watchdog section for more details on this monitor clock. This is controlled with the `PADS_VMU_CTL.FAULTDLY` bits [11:9].

Table 39-6: Fault Delay Programming Values

FAULTDLY<11:8>	Delay Min/Max
0	1 µs
1	2 µs / 3 µs
2	3 µs / 4 µs
3	4 µs / 5 µs
4	5 µs / 6 µs
5	6 µs / 7 µs
6	7 µs / 8 µs
7	8 µs / 9 µs
8	9 µs / 10 µs
9	10 µs / 11 µs
10	11 µs / 12 µs
11	12 µs / 13 µs
12	13 µs / 14 µs
13	14 µs / 15 µs
14	15 µs / 16 µs
15	16 µs / 17 µs

Fault Conditioning

The VMU provides a FAULT_TRIG_IN signal that allows fault input from the OCU and Oscillator Watchdog. When enabled, this allows the VMU to shut down the flash in case of a bad clocking event. The FAULT_TRIG_IN also triggers a system fault—SYS_FAULT. This also triggers the GPIO pin safe states to be set to their fail safe configuration for the application.

The PADS_VMU_TRIPEN register is responsible for configuring the FAULT_TRIG_IN for faults generated by the OCU and OSCWD. The register contains the following bits: PADS_VMU_TRIPEN.OSC0FAULT, PADS_VMU_TRIPEN.OSC0CLK, PADS_VMU_TRIPEN.OCU0FAULT and PADS_VMU_TRIPEN.OCU0CLK.

For more information, see the CM41X_M0 PADS Register Descriptions chapter.

The VMU supports the following faults from the OCU and OSCWD in the PADS_VMU_TRIPEN.

Table 39-7: Faults from OCU and OSCWD

Bit Name	Description
OSC0FAULT	System Oscillator Asynchronous Fault Trip enable
OSC0CLK	System Oscillator Clock Not Good Trip Enable

Table 39-7: Faults from OCU and OSCWD (Continued)

OCU0FAULT	Oscillator Comparator Unit Asynchronous Fault Trip enable
OCU0CLK	Oscillator Comparator Unit Dead Good Trip Enable

Once GPIO pin safe states (and $\overline{\text{SYS\_FAULT}}$) have been triggered, a full system reset is required to restart the application.

GPIO Safe Pin States

When configured, the VMU can set GPIO pins to a programmed safe state when the VMU detects a power event. As an example, the feature can be used in PWM to create a safe state for the signal to set to in case of a power failure event.

While the VMU controls the Pin Safe State, the Oscillator Comparator Unit as well as Oscillator Watchdog Unit in CGU can indirectly utilize the Safe State up on detecting faults in clock.

Individual GPIO Pin Safe State Programming

The safe states (HIGH, LOW, THREESTATE, and HOLD) are user programmable through the PADS_PORT[n]_TRIPST register.

Two-stage Pin Safe State timing Selection

The Pin Safe State is signaled by two events, separated by a programmable interval. The delay is generated using an internal 1 MHz clock. The two event signals are PIN_SAFE_EARLY and PIN_SAFE_LATE. The safe timing selection is chosen by programming the PADS_PORT[n]_TRIPSEL register.

PORTn_TRIPST.TSm[1:0]	GPIO Response to **FORCE_PIN_SAFE**
00	No response – pass/hold previous state
01	Threestate (disable driver)
10	Force low
11	Force high

PORTn_TRIPSEL.SEL[m]	GPIO Safe Timing Selection
0	Responds to FORCE_PIN_SAFE_EARLY
1	Responds to FORCE_PIN_SAFE_LATE

GPIO Pin Safe Triggers

Table 39-8: Trigger Masters

Triggers Masters	Description
FRC_PIN_SAFE_SLV_S0	One of the supervisor sources to the Pin Safe State GPIO hardware

Table 39-8: Trigger Masters (Continued)

Triggers Masters	Description
FRC_PIN_SAFE_SLV_S1	One of the supervisor sources to the Pin Safe State GPIO hardware

Table 39-9: Trigger Slaves

Triggers Slaves	Description
FRC_PIN_SAFE_SLV0	Force Pin Safe trigger 0
FRC_PIN_SAFE_SLV1	Force Pin Safe trigger 1
FRC_PIN_SAFE_SLV2	Force Pin Safe trigger 2
FRC_PIN_SAFE_SLV3	Force Pin Safe trigger 3
FRC_PIN_SAFE_SLV4	Force Pin Safe trigger 4
FRC_PIN_SAFE_SLV5	Force Pin Safe trigger 5
FRC_PIN_SAFE_SLV6	Force Pin Safe trigger 6

Configuring the VMU

Here is an example on how to setup and configure the VMU to monitor both the VDD_INT and VDD_EXT.

1. Reset the VMU.
2. Enable VDD_INT monitoring.
3. Set the VDD_INT reference value.
4. Enable VDD_EXT monitoring.
5. Set the VDD_EXT reference value.
6. Set the desired fault delay.
7. Enable the VMU.

Register Descriptions

The registers for the voltage monitoring unit are located in the System and PADs chapter. See System Block (SYSBLK) and PADS.

40 FFT Accelerator Block (FFTB)

The FFT Accelerator Block (FFTB) provides background input signal spectrum analysis. It includes built-in data conversion for various sensor input formats, spectrum averaging, square magnitude computation, and band-power limit detection.

The FFTB signal spectrum monitor unit provides a simple, low-overhead accelerator. It monitors the spectrum of an input sample stream to detect anomalous conditions that are best detected in the frequency domain. It can be logically connected to an input source by configuring DMA directly from a sensor peripheral. Or, it can operate on a data buffer written by the processor or MDMA.

The FFTB module can automatically apply a windowing profile to the input data. The FFTB module collects single or averaged spectra either continuously, or whenever triggered (for example, by a timer or a processor command). DMA can then write the output spectrum to a memory buffer, if desired. An interrupt request is available to signal the completion of the FFT calculation and any DMA transfer of the output. The FFTB module calculates the power spectrum (squared magnitude). It can compare the spectrum point-by-point against a power limit profile to detect out-of-range signal conditions and generate an FFT limit event. This event can cause a processor interrupt, a TRU event, or a system fault.

FFTB Features

The FFTB unit provides up to 512-point, 16-bit FFT on the input signal data provided by memory or DMA. It optionally provides input format conversion, comb filtering, windowing, programmable FFT size, squared-magnitude computation, spectrum averaging, and spectrum limit checking.

The FFTB signal monitor unit provides the following features:

- Up to 512-point fixed-point FFT accelerator engine
- Data input to memory-mapped buffer IBUFF
- Continuous or one-time spectrum capture operation
- Input format conversion (scaling, offsetting, signed and unsigned conversion) to allow direct input DMA feed from sensor peripherals
- Input windowing through a programmable window buffer, WINBUFF
- Programmable input comb filter for periodic noise rejection

- Twiddle factors in logical ROM, requiring no initialization from user flash memory
- Square-magnitude computation of the output spectrum
- Spectrum averaging of up to 16x spectra (programmable)
- Power spectrum limit comparison versus a power spectrum limit in the limit buffer, LBUFF
- Spectrum data output delivery using:
  - A read of the appropriate buffer
  - The FFTB configuration to transfer the square magnitude data from the magnitude buffer, MBUFF, through the system master interface
- A signal spectrum monitor event. The event can be generated upon the detection of an out-of-bounds power spectrum at any frequency point when the FFTB is programmed for power spectrum comparison.
- Support for up to four FFT channels

FFTB Functional Description

The FFTB captures the spectrum of a number of points in the input signal stream. The input signal stream can come from different data sources or sensors. An optional format converter changes the input signal stream to a form suitable to the FFTB. Input data can be processed before calculating the spectrum by applying a comb filter, and a windowing function.

The processed data is stored in the input buffer (IBUFF). The FFT engine calculates the spectrum for the processed input data. The FFT engine results are complex data which are stored in the real and imaginary working buffers (WBUFF). After storage, the square magnitude of the spectrum is calculated and stored in the magnitude buffer (MBUFF), where the square magnitude spectrum of multiple frames in the input stream can be averaged. The averaged spectrum can then be compared to pre-defined values stored in the limit buffer (LBUFF). If the averaged spectrum data point is greater than the corresponding value in the limit buffer, an error status is asserted.

CM41X_M4 FFTB Register List

A set of registers govern FFTB operations. For more information on FFTB functionality, see the FFTB register descriptions.

Table 40-1: CM41X_M4 FFTB Register List

Name	Description
FFTB_COMB	FFTB Input Comb Filter Control Register
FFTB_CTL	FFTB Control Register
FFTB_DMABASE	DMA Output Base Address Register
FFTB_DMAWR	DMA Output Write Address Register
FFTB_FMTCTL	Format Converter Control Register

Table 40-1: CM41X_M4 FFTB Register List (Continued)

Name	Description
FFTB_INOFST	Input Offset Register
FFTB_LIMSTAT[n]	Limit Status Registers
FFTB_MAGSEL[n]	Magnitude Pin Select Registers
FFTB_MAXMAG	Maximum Magnitude Register
FFTB_MINMAG	Minimum Magnitude Register
FFTB_STAT	FFTB Status Register

CM41X_M0 FFTB Interrupt List

Table 40-2: CM41X_M0 FFTB Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
6	FFTB0_DPERR	FFTB0 Data parity error	Level	
6	FFTB0_SPERR	FFTB0 Parity error in the limit buffer (LBUFF), or the window buffer (WNDBUFF). If this interrupt request is asserted, ignore the result of this spectrum capture sequence and rewrite both LBUFF and WNDBUFF.	Level	
22	FFTB0_DONE	FFTB0 Non-continuous FFT averaging has completed. In addition, the interrupt request is asserted after a channel frame is processed (FFT, PSD, Averaging, PSD comparison) in multichannel mode.	Level	
22	FFTB0_LIMERR	FFTB0 One or more points in the power spectrum output exceeded the predefined limit in the limit buffer.	Level	

CM41X_M4 FFTB Interrupt List

Table 40-3: CM41X_M4 FFTB Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
22	FFTB0_SPERR	FFTB0 Parity error in the limit buffer (LBUFF), or the window buffer (WNDBUFF). If this interrupt request is asserted, ignore the result of this spectrum	Level	

Table 40-3: CM41X_M4 FFTB Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
		capture sequence and rewrite both LBUFF and WNDBUFF.		
23	FFTB0_DPERR	FFTB0 Data parity error	Level	
60	FFTB0_LIMERR	FFTB0 One or more points in the power spectrum output exceeded the predefined limit in the limit buffer.	Level	
128	FFTB0_DONE	FFTB0 Non-continuous FFT averaging has completed. In addition, the interrupt request is asserted after a channel frame is processed (FFT, PSD, Averaging, PSD comparison) in multichannel mode.	Level	

CM41X_M4 FFTB Trigger List

Table 40-4: CM41X_M4 FFTB Trigger List Masters

Trigger ID	Name	Description	Sensitivity
28	FFTB0_DONE	FFTB0 FFT done	Level

Table 40-5: CM41X_M4 FFTB Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
27	FFTB0_TRIG_IN	FFTB0 Spectrum Capture Trigger Input	Pulse

FFTB Definitions

To make the best use of the FFTB, it is useful to understand the following terms.

Comb Filter

A feed-forward comb filter is used to eliminate periodic noise from an observed signal. The difference equation of the feed-forward comb filter is:

```
y[n] = x[n] + a * x[n-K];
```

Where `K` is the delay length (measured in samples), and `a` is a scaling factor applied to the delay signal. The scaling factor `a` takes the values +/- 1.

Windowing

Windowing is a point-to-point multiplication of an input vector and windowing function.

Spectrum Capture

Spectrum Capture refers to FFT engine performing FFT on the input vector and calculating magnitude spectrum.

Accumulator

Accumulator is the single MAC (multiplier and accumulator) inside the FFT Engine.

Spectrum Averaging

Spectrum averaging is used to refine a squared-magnitude spectrum by averaging several successive spectra. Only the squared-magnitude value of the spectrum is averaged (not the real or imaginary components individually).

Spectrum Power Limit

A defined power spectrum values. If at any frequency point, the resulting power spectrum is greater than the corresponding limit value, an event is asserted.

FFTB Block Diagram

The figure shows the functional blocks and data flow of the FFTB.

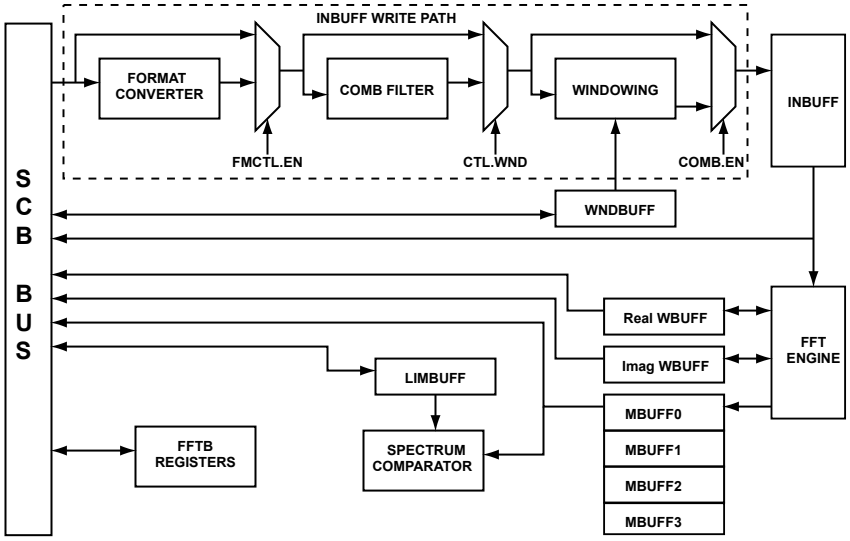


Figure 40-1: FFTB Functional Block Diagram

FFTB Architectural Concepts

The FFTB is a single MAC (multiplier accumulator) spectrum monitoring block. It uses FFT radix-2 DIT (decimation in time) to calculate the spectrum. The ***FFTB Data Widths*** table shows the data widths and effective format used for FFTB operations.

Table 40-6: FFTB Data Widths

Data Use	Data width	Effective format
FFT engine input	16 bits	1.15 signed fractional
FFT twiddle factor	23 bits	0.23 unsigned fractional
Windowing function	16 bits	0.16 unsigned fractional
Intermediate and complex	24 bits	1.23 signed fractional
Square magnitude	32 bits	0.32 unsigned fractional

When using the FFTB module,

- The output of the accumulator is normalized to prevent overflow and is divided by 2. However, when magnitude averaging is enabled, the FFT magnitude output is divided by the number of frames to be averaged, before being passed to the accumulator.
- During any FFT operation, read accesses to any buffer return zero data and write accesses are ignored.
- After capturing the input samples and writing them in the IBUFF, write accesses to the FFT IBUFF are ignored.

Input Format Converter

The FFTB includes a standard input format converter for adapting data, without software overhead. The converter unit performs simple numeric format conversions from diverse data sources to the standard numerical format used for input to the FFTB unit. (The format is a signed fractional 16-bit data, or 1.15). The `FFTB_FMTCTL` and `FFTB_INOFST` registers specify the format conversion settings.

The input converter can accommodate the following input data scenarios:

- Signed or unsigned data, sign-extended or zero-filled, as necessary, using the `FFTB_FMTCTL.SGN` bit
- Data of variable width, from 1 to 16 bits using the `FFTB_FMTCTL.FLDSZ` bit field
- Data MSB is placed in a variable position in the input word as specified by the `FFTB_FMTCTL.IMSB` bit field. The processed data MSB can be placed at any position specified by the `FFTB_FMTCTL.OMSB` bit field.
- Data with a DC offset using the `FFTB_INOFST.VALUE` bit field
- Data that is packed using the `FFTB_FMTCTL.UNPK` bit field
    - two 16-bit samples or four 8-bit samples in one 32-bit data transfer
    - two 8-bit samples in one 16-bit transfer

The input converter cannot handle all situations. Examples of input streams that the input converter cannot handle unassisted are:

- Data that is interleaved in a multiplexed data stream
- Data that is written with a stride (memory offset between samples) of other than 1, 2, or 4 bytes

The input bandwidth limitation applies if unpacking data.

Logical input data size (DSZ) is the smallest number of bytes (in power of 2) that is larger than the input data size (in bytes). The input data size is programmed in the FFTB_FMTCTL.FLDSZ bit field. The input format converter follows this sequence of operations:

1. If the FFTB_FMTCTL.EN bit =0x0, no format conversion takes place. Data is taken from the MSB-aligned 16 bits of the physical data transferred. If the physical data is 8-bit, then it is MSB-aligned and LSB-filled with zeros. Otherwise If the FFTB_FMTCTL.EN bit =0x1, the FFTB unit performs the next steps (in order).
2. The size of the data transfer on the physical bus is compared to the logical data size (DSZ). If the values match, then step 3 manages the physically transferred data. Otherwise, if the physical size is greater than DSZ:
    a. If the FFTB_FMTCTL.UNPK bit =0x1, then the write data is divided into multiple (2 or 4) data samples, LSB-first. The samples are processed independently, one per FFTB clock cycle.
    b. If the FFTB_FMTCTL.UNPK bit =0x0, then the data transfer size does not match the logical data size. The mismatch is tolerated as a programming error or as an unintended widening operation performed in the system bus fabric. So, the data of the size encoded by FFTB_FMTCTL.FLDSZ is taken from the LSB-aligned position in the data physically written. When a 16-bit physical write in the high half of a 32-bit word and DSZ is 8-bit data, bits [23:16] are used.
3. The MSB of the input data that is specified by the FFTB_FMTCTL.IMSB bit field is shifted to bit 15 of the processed data. Then, the MSB of the processed data is masked to the FFTB_FMTCTL.FLDSZ bits wide value.
4. The MSB of masked data is shifted to the position configured by the FFTB_FMTCTL.OMSB bit field.
5. If the FFTB_FMTCTL.SGN bit =0x1, and the FFTB_FMTCTL.FLDSZ value is less than 16, then the data is sign-extended from the bit identified as FFTB_FMTCTL.OMSB + 1 to bit 15. In other words, the bit (in the masked data) at index FFTB_FMTCTL.OMSB is replicated in bits 15 down to FFTB_FMTCTL.OMSB + 1.
6. The 16-bit value in the FFTB_INOFST.VALUE bit field (regarded as 1.15 signed fractional format) is added to the previous result. If a signed overflow occurs and the saturation control is enabled (the FFTB_FMTCTL.SAT bit is set), the value is saturated to a positive or negative full scale.
7. The resulting 16-bit signed, fractional value is deposited in the input buffer, IBUFF, at the appropriate position, k.

IBUFF_WO Input Buffer Write

The data stored in the IBUFF represents the result of a write to the IBUFF with:

- The format converter processing raw input data, when enabled (FFTB_FMTCTL.EN bit =0x1)
- A comb filter processing data, when enabled (FFTB_COMB.EN bit =0x1)
- A windowing function processing data, when enabled (FFTB_CTL.WND bit =0x1)

To perform a spectrum capture operation, a stream of data is transferred into the FFTB input by writing raw data into the FFTB IBUFF_WO address range, starting with address offset 0. These data writes are processed by the format converter and comb filter, if enabled, and then the resulting formatted and filtered samples are written into the IBUFF_RW memory in the order they were processed.

As a convenience for data transfer, the write address offset into IBUFF_WO isn't considered significant, except for the case of address offset 0. When the FFTB is waiting for the start of a spectrum capture, a write to IBUFF_WO[0] is taken as the first sample of the input data, while other writes are ignored. Each subsequent write into the IBUFF_WO range (whether to offset 0 or not) is taken as the next successive data sample. As the comb filter and input formatter process these samples, the resulting formatted data (in standard signed fractional 1.15 notation) is filled into the IBUFF_RW memory via internal write pointer that starts at 0. This pointer increments by 2 bytes for each (unpacked, filtered, and formatted) 16-bit data sample.

This flexible mechanism works smoothly with all data input sources, regardless of their addressing capability. It works, for example, to arrange to have a source peripheral handling a continuous data stream (for example audio data) to always write to the same target address at IBUFF_WO offset 0. The FFTB then extracts the exact proper length of data from the stream as soon as it is enabled to do so. Alternatively, a signal source with more complex DMA capability might write using circular-buffer addressing into a buffer of length N samples based at IBUFF_WO, so that the FFTB starts its spectrum capture window only on a sample index that is a multiple of N.

If no spectrum monitoring is triggered, all write accesses to the IBUFF FFT address range are ignored. If a spectrum monitoring is started, input data capturing does not occur until data is written to IBUFF FFT base address. Consecutive writes update the IBUFF until the last data point associated with the FFT size (`FFTB_CTL.SZ` bit field) is processed. Then, all write accesses are ignored until a new spectrum capture is started.

IBUFF_RW Input Buffer Read

The IBUFF_RW buffer allows inspection of the input data after all pre-processing by the input format converter, comb filter, and/or windowing function, as enabled respectively. The data in IBUFF_RW is then processed directly by the FFTB engine, and so should appear in the standard format (signed fractional, 1.15 – radix format).

Data samples written to a particular address in the IBUFF_WO during input data capture can appear at a different read address in the IBUFF_RW. This action is due to unpacking and the logical data size, DSZ. See IBUFF_WO Input Buffer Write section for the details of which data sample point, `[k]`, is written by a given write transfer. The processed value for point `k` is always a read from the byte address [(IBUFF_RW direct access base address) + `2k`]. If the IBUFF_RW is declared as an array of packed 16-bit values, then the expression `IBUFF_RW[k]` returns the proper value for data sample `[k]`.

The IBUFF_RW buffer may only be read when the FFTB is not running. During any FFT operation, the IBUFF_RW read accesses return zero data.

Comb Filter

The feed forward comb filter is used to reject periodic noise in the input signal. The comb filter feature pre-processes the input signal to the FFTB to reject certain periodic content at a specific frequency. It is enabled by `FFTB_COMB.EN` and controlled by the `FFTB_COMB.LEN`, and `FFTB_COMB.SGN` fields. The comb filter has the following signal response:

```
output[n] = input[n] ± input[n - LEN];
```

The comb filter with negative sign rejects frequencies that are multiples of f_{samp}/LEN, as well as at f = 0 (DC). To compute LEN from the desired comb notch frequency:

$$LEN = (f_{samp}/f_{noise}) - 1;$$

Changing the sign of a comb filter from negative to positive results in rotating the zeroes of the comb filter by [π/LEN]. For example: if f_{samp} is 1 MHz, and there is periodic switching noise at multiples of f_{noise} = 20 kHz, then the comb filter can be configured as follows.

```
ADI_FFTB_COMB.LEN = ((1000 / 20) - 1); // value is 49
ADI_FFTB_COMB.EN = 1;
```

This configuration removes noise at 0, 20 kHz, 40 kHz, 60 kHz, and up to 500 kHz frequencies.

NOTE: When the Comb filter is enabled, write IBUFF with (N + LEN) samples from the signal, where N is the FFT length.

Input Data Windowing

The FFTB provides a windowing buffer (WNDBUFF) to store windowing function values. Windowing is point-to-point multiplication of an input vector and windowing function. Extracting N points from an input data stream is equivalent to multiplying the input data stream with a rectangular window by the width of N.

A rectangular window has the following attributes:

- A narrow mainlobe (which is an advantage)
- The largest sidelobe drop of rate 13 dB, and sidelobe drop off rate 6 dB/octave (which is a disadvantage)

To improve the output spectrum of a window function, the mainlobe must be kept narrow (a wide mainlobe smears sinusoids together) and the sidelobe power must be smaller. (Large sidelobes obscure small sinusoids and cause sidelobe interference).

Lowering the sidelobe power (compare to rectangular window) causes the widening of the mainlobe. Many window functions exist that can lower the sidelobe power with acceptable mainlobe, for example Hanning and Hamming windows. Windowing functions are symmetric, so only half of the points must be stored.

In practice, windowing functions are symmetrical (`wf[k] = wf[N1k]`), so the WNDBUFF is a half-sequence wide (only the first N/2 locations are used, from index 0 to N/21.) This arrangement means that:

- Input points x[k] for k = 0 to (N/2)-1 are multiplied by WNDBUFF[k]
- Input points for k = N/2 to N-1 are multiplied by WNDBUFF[N-1-k], respectively

For the same windowing function, the values written to the WNDBUFF are specific to the FFT size. Changing the FFT size requires that the WNDBUFF be rewritten with the appropriate values.

A write to a WNDBUFF location takes effect immediately on any following writes to either of the two matching IBUFF locations. Writes to the WNDBUFF do not have a retroactive effect. The write operations do not apply to

data values that have previously been written to IBUFF (as defined by the time of data arrival on the SCB write data bus).

During any FFT operation, write accesses to the windowing buffer (WNDBUFF) are ignored and read accesses return 0x0.

Transform Accelerator

The FFTB transform accelerator automates the calculation of the FFT algorithm on the contents of the IBUFF, according to the size specified by `FFTB_CTL.SZ` (which is a power of 2). The complex spectrum is stored to the working buffers (WBUFF). The WBUFF contains two memory arrays (real and imaginary).

An in-place FFT is performed to save area. The WBUFF represents the working storage.

During the initial stage of radix2 FFT, the data is read from the IBUFF (in reverse bit order), and the results are written to the WBUFF (real and imaginary buffers) in-order.

Magnitude Computation Unit

The FFTB contains a magnitude computation unit which computes the square magnitude (power spectrum density) of complex spectrum stored in the WBUFF (real and imaginary buffers) at each frequency point. It writes the result in the magnitude buffer (MBUFF). Because of the symmetry of the power spectrum density, the square magnitude is calculated for half the spectrum plus one. Which means, for the number of samples N (even number), the number of spectrum points produced is (N/2 + 1). The FFTB unit contains one MAC. So, one square magnitude calculation completes every two cycles. The width of the square magnitude data is 32 bits (effectively, a 0.32 unsigned-fractional format).

FFT Performance Calculation

FFT Spectrum Computation Cycles:

The FFT engine performs a nested loop, where the inner loop executes N/2 radix2 butterfly operations. The outer loop is executed log2(N) times. The FFT engine can finish a butterfly operation every four FFTB clock cycles. So, the time required for FFT computation is:

Cycles = $4 \times (N/2) \times \log_2(N) + C = 2N \times \log_2(N) + C$;

where:

C is depth of the butterfly pipeline and is equal to 4.

N is the FFT size

FFT Magnitude Computation Cycles:

Cycles = N + 2, where N is the FFT size.

NOTE: If `FFTB_CTL.DMAOUT` == 0x0, the FFTB DONE interrupt request is asserted after the computing the last FFT data point's squared magnitude. Therefore, the total FFT Performance Cycle is the sum of both FFT Spectrum Computation Cycles and FFT Magnitude Computation Cycles.

Spectrum Averaging

The FFTB contains spectrum averaging logic where multiple (up to 16) successive Power Spectrum Densities (PSDs) are averaged. If (FFTB_CTL.AVG ≠ 0x0), then spectrum averaging is selected.

For example, *A* is the number of averaged spectra specified by the FFTB_CTL.AVG bit setting. The results of successive spectra are divided by *A*, then added into the MBUFF buffer. Each time a spectrum is added, the output of the accumulator is not normalized to have linear averaging for the spectrum. After a complete set of *A* spectra is collected and one of the trigger modes initiate a new spectrum capture sequence, the contents of the magnitude buffer (MBUFF) are discarded and a new sum is initiated. This sequence does not mean that the magnitude buffer MBUFF is reset to 0 upon the described events. Instead, the events set a state condition (FFTB_STAT.AVGPND = *A*) which directs the averaging operation to ignore (treat as zero) the prior contents of MBUFF, when processing the next spectrum results.

When the power spectrum averaging completes, the minimum and maximum values of the resulting power spectrum averaging are recorded in the FFTB_MAXMAG and FFTB_MINMAG registers. For each power spectrum data sample, there is a bit in the register array FFTB_MAGSEL[n]. The bit controls when the power spectrum data sample is used in the calculation of the minimum and maximum power spectrum values.

Spectrum Comparator

The FFTB supports an optional point-to-point power spectrum density comparison. It compares the output power spectrum or averaged power spectrum to the limit buffer (LBUFF) that contains a programmable power spectrum limit value. This value is the maximum power for each frequency point in a half-spectrum. An out-of-bounds power detected at any frequency point can be programmed to cause a signal spectrum monitor event.

To enable spectrum comparison, set the FFTB_CTL.CMP bit to 0x1.

For each power spectrum data sample, there is a bit in the FFTB_MAGSEL[n] register that controls when the limit comparison occurs.

The result of each comparison of the power spectrum data sample has a corresponding bit in the FFTB_LIMSTAT[n] register.

If spectrum comparison is enabled, and at least one limit comparison fails (for example, the power spectrum data sample exceeds the power limit), the FFTB_STAT.LIMERR status bit is asserted. More FFT averaging does not start until the FFTB_STAT.LIMERR bit is cleared. In multichannel mode, the FFT operations are stopped only after the spectrum comparison completes for all the channels. The FFTB_LIMSTAT[n] registers for each channel can be reviewed and the FFTB_STAT.LIMERR bit can be cleared by software for each channel before changing the channel.

Buffer Memory Maps

The ***FFTB Memory Mapped Buffers List*** identifies the base addresses, sizes, and access types for the different buffers in the FFTB.

Table 40-7: FFTB Memory Mapped Buffers List

Buffer Name	Start Address	Size	Access Type	Notes
IBUFF_WO	0x7000 0000	1K Byte	WO	This address range is used only to capture input data samples used in the FFT calculation. Used only for FFT capture operation.
IBUFF_RW	0x7000 0800	1K Byte	R/W	Access is ignored while any FFT operation is in-progress. Used only for direct access.
LBUFF	0x7000 1000	1025 Bytes	R/W	Access is ignored while any FFT operation is in-progress. Used only for direct access.
MBUFF0	0x7000 2000	1028 Bytes	R/W	Access is ignored while any FFT operation is in-progress. Used only for direct access.
MBUFF1	0x7000 3000	1028 Bytes	R/W	Access is ignored while any FFT operation is in-progress. Used only for direct access.
MBUFF2	0x7000 4000	1028 Bytes	R/W	Access is ignored while any FFT operation is in-progress. Used only for direct access.
MBUFF3	0x7000 5000	1028 Bytes	R/W	Access is ignored while any FFT operation is in-progress. Used only for direct access.
Real WBUFF	0x7000 6000	2K Bytes	R/W	Access is ignored while any FFT operation is in progress. Used only for direct access.
Imag WBUFF	0x7000 7000	2K Bytes	R/W	Access is ignored while any FFT operation is in progress. Used only for direct access.
WNDBUFF	0x7000 8000	512 Bytes	R/W	Access is ignored while any FFT operation is in progress. Used only for direct access.

FFTB Spectrum Monitor Triggering Modes

The spectrum monitor starts with capturing the input data stream, and ends with the last operation needed for the current data frame. This sequence depends on the configuration in the `FFTB_CTL` register. Input data capture always starts with writing location 0x0 of IBUFF. When the spectrum capture process starts, all input stream writes prior to writing the base address of the input buffer are ignored.

The FFTB can operate in the following modes:

- ***Manual.*** In this mode, only one spectrum is captured. The spectrum capture process starts after writing 0x1 to the `FFTB_CTL.START` bit. No trigger is required in this mode.
- ***Auto.*** In this mode, the FFTB continuously captures the spectrum. The first spectrum capture starts after writing 0x1 to the `FFTB_CTL.START` bit. No trigger is required in this mode. After the current spectrum capture completes, a new spectrum capture starts.

- ***Trigger One Shot.*** In this mode, only one spectrum average is captured. Spectrum averaging starts after writing 0x1 to the FFTB_CTL.START bit. Then, a trigger is needed on the FFTB_TRIG_IN for each frame in the spectrum averaging. After all frames are captured, the spectrum capture process does not start, even if another trigger is sent on the FFTB_TRIG_IN.
- ***Trigger Continuous.*** In this mode, The FFTB continuously captures spectrum. The first frame capture starts after writing 0x1 to the FFTB_CTL.START bit. Then, a trigger is needed on the FFTB_TRIG_IN for each frame in the spectrum averaging. After the current spectrum averaging completes, a new spectrum capture does not start, unless another pulse is sent on the selected FFTB_TRIG_IN.

NOTE: In both Trigger One Shot and Trigger Continuous modes, any write to IBUFF is ignored before the trigger.

Multichannel Operation

Up to four channels can be enabled in multichannel mode. Once enabled, the spectrum analyzer performs all the enabled operations for an individual channel before moving to the next channel in a frame. The enabled operations can be capture, FFT, PSD, averaging in second and subsequent frames, and optional PSD comparison in the last frame. After all the channels in a frame are processed, the spectrum analyzer advances to the next frame. In the last frame, if enabled, the DMA operation is initiated only after all channels are processed.

Setting the FFTB_CTL.CHEN bit field enables the multichannel operation. Setting the FFTB_CTL.CHDONEIRQ bit triggers the DONE interrupt request after each channel is processed so that the input source of the spectrum analyzer can be switched to the next channel. The FFTB_STAT.ACTCH bit field indicates which channel is being processed. Hardware sets the FFTB_STAT.CHDONE bit after each channel is processed. It indicates that the spectrum analyzer is waiting for a channel change. If the FFTB_STAT.CHDONE bit is set, writes to the capture buffer are ignored. To resume operation, software must clear this bit after the channel is changed.

Each channel uses its own magnitude buffer to store the averaged PSD data. All PSD data from channel 0 is transferred, followed by PSD data from channel 1 and so on. The data is transferred to consecutive destination locations. For example, for a 32 point FFT, 17 PSD words for channel 0 are transferred first and the 18th destination location contains the first PSD data for channel 1.

NOTE: After all the spectrum operations and optional DMA are complete, the status registers contain information about the last channel only. Software should review status information before changing the channel.

The limit error interrupt stops the FFTB operation only after all the channels are processed.

Launching and Stopping FFT Monitor

To launch the FFT monitor, enable the module by setting the FFTB_CTL.EN bit =0x1 and the FFTB_CTL.START bit =0x1. The FFT trigger mode is configured using the FFTB_CTL.TRIGMODE bit.

When the FFT is launched, all FFT control configurations are captured internally. The captured configurations are used for all FFT operations. Changing the configuration after launching the FFT does not alter the FFT calculation.

The only configurations that immediately take effect are:

- Parity error test bit (FFTB_CTL.PETST)
- Interrupt control bits:
  - FFTB_CTL.DONEIRQ
  - FFTB_CTL.LIMIRQ
  - FFTB_CTL.DPEIRQ
  - FFTB_CTL.SPEIRQ

The status bits associated with these interrupt requests are always updated.

While the FFT monitor is in-progress, writing 0x1 to the FFTB_CTL.START bit has no effect. If any parity error or limit error status is asserted, writes to the FFTB_CTL.START bit are ignored.

There are two ways to stop the FFT monitoring operation:

- Abrupt stop. Disable the FFTB unit by writing 0x0 to the FFTB_CTL.EN bit.
- Clean stop. This operation is only useful when the FFTB unit is operating in auto-mode or trigger-continuous mode. Write 0x1 to the FFTB_CTL.STOP bit for a clean stop. This operation causes the FFTB unit to wait until the in-progress FFT averaging function completes, then transition to idle and wait for another FFT monitor launch. In multichannel mode, the FFT operation stops after the averaging operation for all the channels is complete.

NOTE: Setting both the FFTB_CTL.START and FFTB_CTL.STOP bits to 0x1 at the same time is equivalent to setting the FFTB_CTL.STOP bit to 0x1.

Parity Protection

Every byte in the FFTB memory is protected with one parity bit and every memory read checks for parity errors. If a parity error is detected, the appropriate parity error status bit is asserted. If a parity error is detected while an FFT averaging operation is in-progress, all FFT operations in single and multichannel mode are stopped immediately. No FFT operation is launched until all asserted parity errors status bits are cleared. The done_int and limerr_int interrupt requests cannot be asserted in this case. Enable the sperr_int and dperr_int interrupt requests for timely software intervention.

If the FFTB_STAT.DPERR bit is asserted, just clearing the status bit should be enough.If FFTB_STAT.SPERR bit is asserted, the program must refill both the windowing buffer (WNDBUFF), and the limit buffer (LIMBUFF).

The FFTB_CTL.PETST bit is used to test parity error logic. While this bit is asserted, any write operation to any buffer results in depositing the wrong parity. Once the FFTB_CTL.PETST bit is cleared, all write operations deposit the correct parity.

FFTB Event Control

The FFTB can signal status events when certain conditions occur. All FFTB output events are sticky. To clear these events, write the value 0x1 to the event status bit in the status register. The FFTB also has control event input ports (`FFTB_TRIG_IN`).

FFTB Status and Error Signals

The FFTB has the following status and error signals.

Table 40-8: Interrupt Signals

Status Name	Description
SPERR	Parity error in the limit buffer (LBUFF) or the window buffer (WNDBUFF). If this interrupt request is asserted, the program should ignore the result of this spectrum capture sequence and rewrite both LBUFF and WNDBUFF.
DPERR	Parity error in the input buffer (IBUFF), the complex working buffer (WBUFF), or magnitude buffer (MBUFF). If this interrupt request is asserted, the program should ignore the result of this spectrum capture sequence.
LIMERR	One or more points in the power spectrum output exceeded the predefined limit in the limit buffer.
IPND	Input buffer (IBUFF) filling is in-progress.
SPND	FFT, PSD, averaging, or PSD comparison are in-progress.
OPND	DMA output is in-progress.
DONE	Non-continuous FFT averaging has completed.
CHDONE	FFT, PSD, averaging, or PSD comparison completed for a channel frame.

CM41X_M4 FFTB Register Descriptions

FFTB register map. (FFTB) contains the following registers.

Table 40-9: CM41X_M4 FFTB Register List

Name	Description
FFTB_COMB	FFTB Input Comb Filter Control Register
FFTB_CTL	FFTB Control Register
FFTB_DMABASE	DMA Output Base Address Register
FFTB_DMAWR	DMA Output Write Address Register
FFTB_FMTCTL	Format Converter Control Register
FFTB_INOFST	Input Offset Register
FFTB_LIMSTAT[n]	Limit Status Registers
FFTB_MAGSEL[n]	Magnitude Pin Select Registers

Table 40-9: CM41X_M4 FFTB Register List (Continued)

Name	Description
FFTB_MAXMAG	Maximum Magnitude Register
FFTB_MINMAG	Minimum Magnitude Register
FFTB_STAT	FFTB Status Register

FFTB Input Comb Filter Control Register

The FFTB_COMB register controls various aspects of the Comb filter.

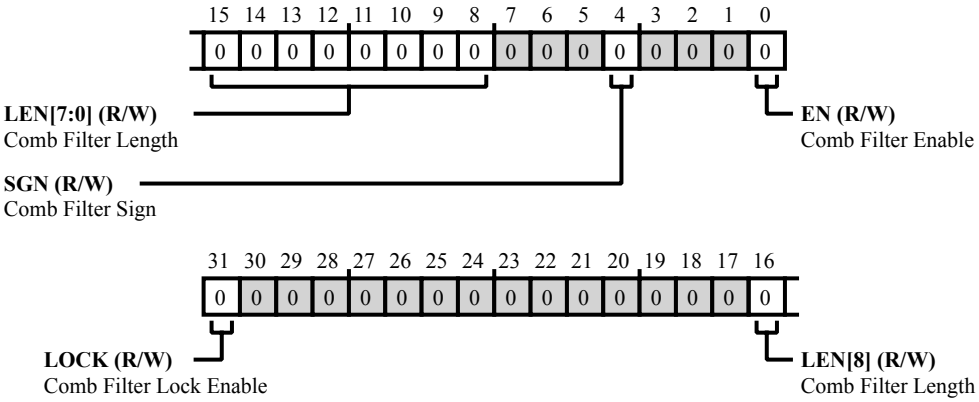


Figure 40-2: FFTB_COMB Register Diagram

Table 40-10: FFTB_COMB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Comb Filter Lock Enable. When both the FFTB_COMB.LOCK bit and the SPU global lock (SPU_CTL.GLCK) are set to 0x1, then all writes to the FFTB_COMB register are ignored.
16:8 (R/W)	LEN	Comb Filter Length. When the FFTB_COMB.LEN bit =1, then the input sample buffer element INBUF[n] receives the sum or the difference of two sample values: INBUF[n] = x[n] +/- x[n-LEN]; Writing the value 0x0 to the comb filter length is equivalent to disable the comb filter.

4 (R/W)	SGN	Comb Filter Sign.
		If (FFTB_COMB.SGN =1'b0), then y[n] = x[n]+x[n-len].
		If (FFTB_COMB.SGN =1'b1), then y[n] = x[n]-x[n-len].
		Where x[n] is the input of the comb filter, and y[n] the output of the comb filter.

0 (R/W)	EN	Comb Filter Enable. The FFTB_COMB.EN bit enables the Comb filter on the signal input.

FFTB Control Register

The FFTB_CTL register contains all the field needed to control the functionality of the FFTB block.

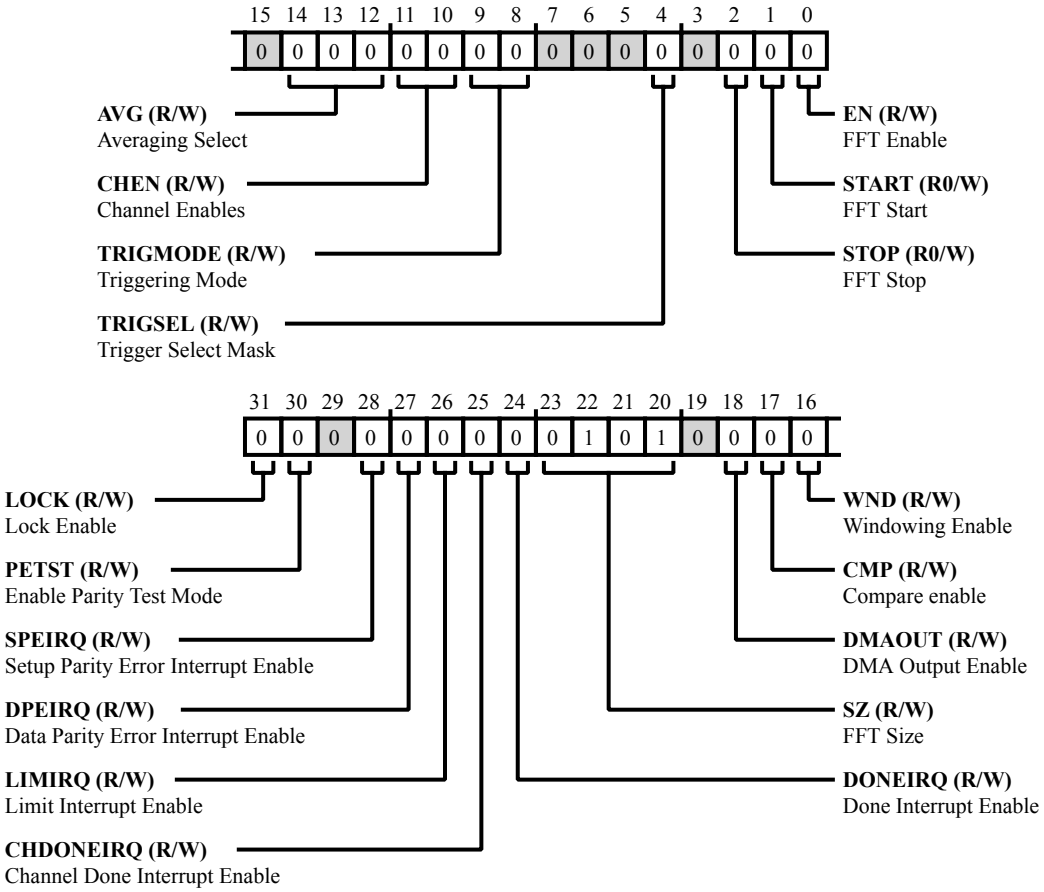


Figure 40-3: FFTB_CTL Register Diagram

Table 40-11: FFTB_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Lock Enable. When both the FFTB_CTL.LOCK bit and the global lock (SPU_CTL.GLCK) bit are set to 0x1, all writes to the CTL register are ignored.
30 (R/W)	PETST	Enable Parity Test Mode. When the FFTB_CTL.PETST bit is set (=1) writes to SRAM deposit invalid parity into the SRAM arrays for each byte in the targeted data location. This test mode should be used only to initialize the arrays for testing parity error.

Table 40-11: FFTB_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	SPEIRQ	Setup Parity Error Interrupt Enable. The FFTB_CTL.SPEIRQ bit enables the parity error interrupt request when detected in any of the setup buffers (windowing buffer, and limit buffer).	
27 (R/W)	DPEIRQ	Data Parity Error Interrupt Enable. The FFTB_CTL.DPEIRQ bit enables the parity error interrupt request when detected in any of the data buffers (input buffer, real buffer, imaginary buffer, averaging buffer, and compare buffer).	
26 (R/W)	LIMIRQ	Limit Interrupt Enable. The FFTB_CTL.LIMIRQ bit enables the limit interrupt request when an output point larger than its corresponding value in the limit buffer is detected. If FFTB_CTL.LIMIRQ is 1, then the interrupt request is asserted whenever FFTB_STAT.LIMERR is 1.	
25 (R/W)	CHDONEIRQ	Channel Done Interrupt Enable. The FFTB_CTL.CHDONEIRQ bit enables the channel done interrupt request. It can be used to switch the FFT input source. Set the FFTB_CTL.CHEN bit to a value >0 to enable this interrupt request.	

24 (R/W)	DONEIRQ	Done Interrupt Enable.
		The FFTB_CTL.DONEIRQ bit enables an FFT done interrupt on the completion of an FFT computation.
		If (FFTB_CTL.DMAOUT == 0x1), the interrupt request is asserted after the DMA output is done.
		If (FFTB_CTL.DMAOUT == 0x0), the interrupt request is asserted after the computation of the last FFT data point's squared magnitude.	

23:20 (R/W)	SZ	FFT Size. The FFTB_CTL.SZ bit field configures an enumerated value for the FFT input vector size. The FFT size must be power of 2. The minimum allowed input vector size is 32 points, and the maximum allowed input vector size is 512 points. If this field is programmed with any value outside the enumerated values, it is not updated.	
		5	The size of the input vector is 32 points.
		6	The size of the input vector is 64 points.
		7	The size of the input vector is 128 points.
		8	The size of the input vector is 256 points.
		9	The size of the input vector is 512 points.
18 (R/W)	DMAOUT	DMA Output Enable. The FFTB_CTL.DMAOUT bit enables DMA to output the averaging results through the DMAWRPTR to the address programmed in the FFTB_DMABASE register.	

Table 40-11: FFTB_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	CMP	Compare enable. The `FFTB_CTL.CMP` bit enables comparing the averaged output spectrum with the limit spectrum in the limit buffer.	
16 (R/W)	WND	Windowing Enable. The `FFTB_CTL.WND` bit enables windowing of the input vector. If set to 0x1, the input vector is multiplied by the appropriate window function value in the window buffer; otherwise the input vector is not multiplied by the windowing function.	
14:12 (R/W)	AVG	Averaging Select. The `FFTB_CTL.AVG` bit field selects spectrum output averaging.	
		0	Spectrum averaging is disabled.
		1	Average the spectrum of 2 input vectors.
		2	Average the spectrum of 4 input vectors.
		3	Average the spectrum of 8 input vectors.
		4	Average the spectrum of 16 input vectors.
11:10 (R/W)	CHEN	Channel Enables. The `FFTB_CTL.CHEN` bit field specifies the channels enabled.	
		0	Channel 0 is enabled
		1	Channel 0 and 1 are enabled
		2	Channel 0, 1 and 2 are enabled
		3	Channel 0, 1, 2 and 3 are enabled

Table 40-11: FFTB_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	TRIGMODE	Triggering Mode. The FFTB_CTL.TRIGMODE bit field configures the trigger mode.	
		0	Manual Triggering The capture of the current signal buffer is initiated manually, when the FFTB_CTL.START bit is written to 1.
		1	Auto Triggering The capture of the signal buffer occurs continuously, whenever FFTB_CTL.START =1. Signal capture repeats continuously, re-starting as soon as the previous spectrum processing (and DMA output, if applicable) has completed.
		2	Triggered One Shot (TRIG_ONE) In this mode, if FFTB_CTL.START is set and FFTB_CTL.TRIGMODE = TRIG_ONE, capture of a single input signal buffer is initiated on the next pulse on any FFTB0_TRIG_IN signal enabled by the corresponding bit mask FFTB_CTL.TRIGSEL. Only one spectrum capture averaging is initiated. One trigger pulse is required to capture each single frame in the input signal stream.
		3	Triggered Continuous (TRIG_CONT) In this mode, if FFTB_CTL.START is set, the capture of the input signal buffers is initiated at each pulse on the FFTB0_TRIG_IN signal enabled by the FFTB_CTL.TRIGSEL. Signal buffers are captured continuously. One trigger pulse is required to capture each single frame in the input signal stream.
4 (R/W)	TRIGSEL	Trigger Select Mask. The FFTB_CTL.TRIGSEL bit selects the input trigger source on the FFTB_TRIG_IN port to initiate spectrum capture sequence. A code of 0x0 means no trigger input is enabled. The FFTB_CTL.TRIGSEL and FFTB_TRIG_IN are only applicable when FFTB_CTL.TRIGMODE = TRIG_ONE or TRIG_CONT.	
2 (R0/W)	STOP	FFT Stop. The FFTB_CTL.STOP bit stops any spectrum capture in-progress and prevents any new spectrum capture from starting until the FFTB_CTL.START bit is written to 1 again.	
1 (R0/W)	START	FFT Start. The FFTB_CTL.START bit starts a spectrum capture operation. If trigger mode requires an external trigger, a spectrum capture does not start capturing a data sample until an external trigger event is received.	

Table 40-11: FFTB_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	FFT Enable. The `FFTB_CTL.EN` bit enables the FFTB block. When the `FFTB_CTL.EN` bit == 0x0, the the whole FFT block is disabled and all error status bits are preserved.	
		0	FFTB disabled
		1	FFTB enabled

DMA Output Base Address Register

When the FFTB_CTL.DMAOUT bit field =0x1, the FFTB_DMABASE.ADDR bit field is used as the initial value of the FFTB_DMAWR.ADDR bit field for DMA out.

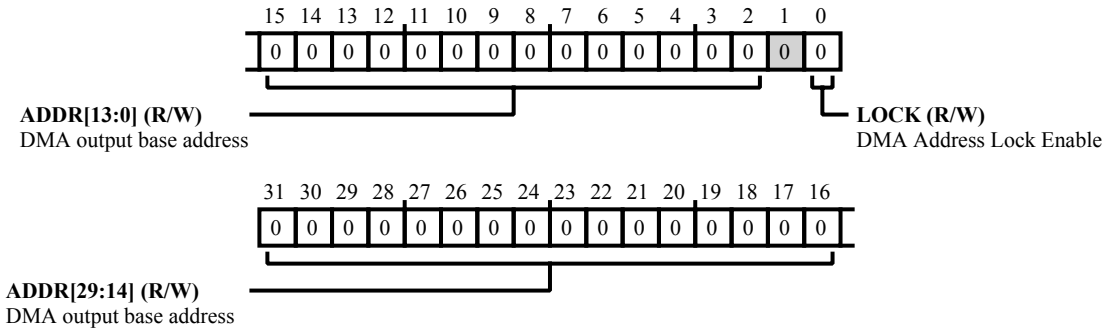


Figure 40-4: FFTB_DMABASE Register Diagram

Table 40-12: FFTB_DMABASE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:2 (R/W)	ADDR	DMA output base address. An initial address for DMA output[31:2].
0 (R/W)	LOCK	DMA Address Lock Enable. When both the FFTB_DMABASE.LOCK bit and the SPU global lock (SPU_CTL.GLCK) bit are set to 0x1, then all writes to the FFTB_DMABASE register are ignored.

DMA Output Write Address Register

The FFTB_DMAWR register is initialized with the value in the FFTB_DMABASE.ADDR bit field.

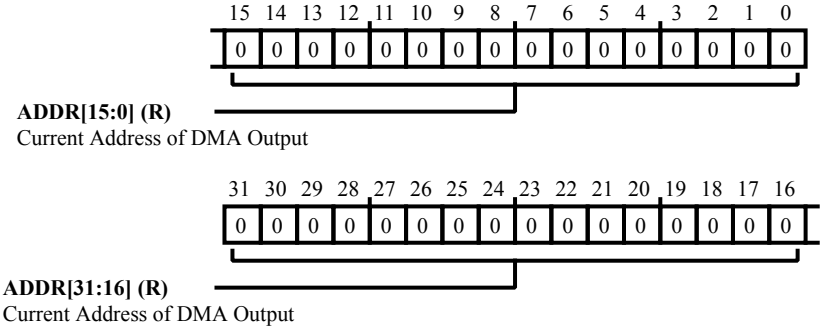


Figure 40-5: FFTB_DMAWR Register Diagram

Table 40-13: FFTB_DMAWR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	ADDR	Current Address of DMA Output. In the beginning of DMA output, this field is initialized with the value in the FFTB_DMABASE.ADDR bit field.

Format Converter Control Register

The FFTB_FMTCTL register contains control fields to guide in processing the input stream before writing it to the input buffer.

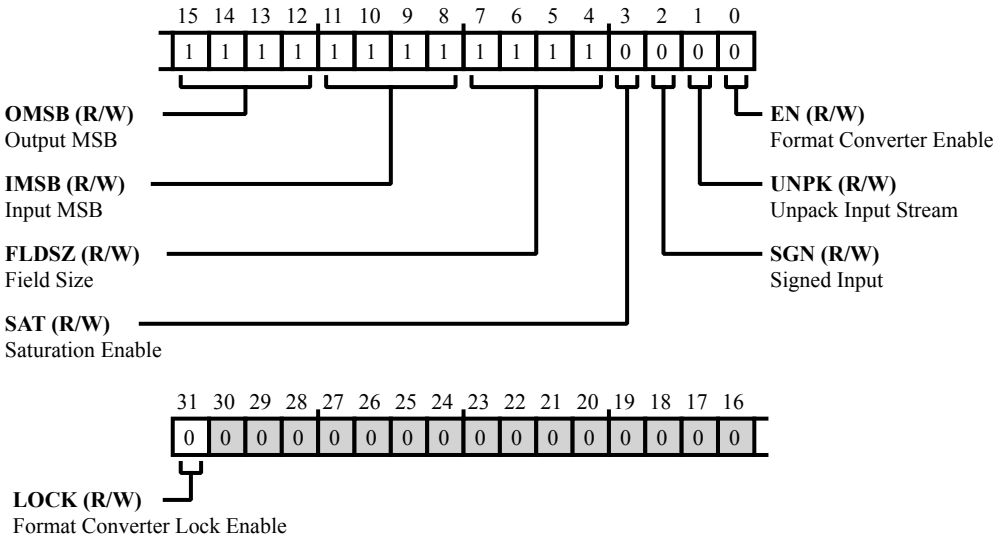


Figure 40-6: FFTB_FMTCTL Register Diagram

Table 40-14: FFTB_FMTCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	LOCK	Format Converter Lock Enable. When both the FFTB_FMTCTL.LOCK bit and the SPU global lock (SPU_CTL.GLCK) bit are set to 0x1, then all writes to the FFTB_FMTCTL and FFTB_INOFST registers are ignored.
15:12 (R/W)	OMSB	Output MSB. The legal values of the FFTB_FMTCTL.OMSB bit field are [0xF down to FFTB_FMTCTL.FLDSZ]. If FFTB_FMTCTL.OMSB < FFTB_FMTCTL.FLDSZ, then the FFTB_FMTCTL.OMSB bit field is overwritten with FFTB_FMTCTL.FLDSZ.

Table 40-14: FFTB_FMTCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration

11:8 (R/W)	IMSB	Input MSB.
		For (FFTB_FMTCTL.FLDSZ <= 0x7), the legal values of FFTB_FMTCTL.IMSB are [0x7 down to FFTB_FMTCTL.FLDSZ]. For (FFTB_FMTCTL.FLDSZ > 0x7), the legal values of FFTB_FMTCTL.IMSB are [0xF down to FFTB_FMTCTL.FLDSZ].
		If (FFTB_FMTCTL.FLDSZ <= 0x7) and (FFTB_FMTCTL.IMSB > 0x7), then the FFTB_FMTCTL.IMSB bit field is overwritten with the value 0x7. If (FFTB_FMTCTL.IMSB < FFTB_FMTCTL.FLDSZ), then the FFTB_FMTCTL.IMSB bit field is overwritten with the value FFTB_FMTCTL.FLDSZ.

7:4 (R/W)	FLDSZ	Field Size. The FFTB_FMTCTL.FLDSZ bit field specifies the field size of the input data transfer in bits. The input field size in the input stream = FFTB_FMTCTL.FLDSZ + 1

3 (R/W)	SAT	Saturation Enable.
		If (FFTB_FMTCTL.SAT =0x1), saturate negative values (that are too large to represent in 16-bits 1.15 signed format) to 0x8000.
		If (FFTB_FMTCTL.SAT =0x1), saturate positive values (that are too large to represent in 16-bits 1.15 signed format) to 0x7FFF.

2 (R/W)	SGN	Signed Input. If (FFTB_FMTCTL.SGN == 0x1), the input is signed and will be sign-extended, if necessary. If (FFTB_FMTCTL.SGN == 0x0), the input is unsigned and will be zero-filled, if necessary.
1 (R/W)	UNPK	Unpack Input Stream. Enables unpacking each data write into multiple data samples. If enabled (FFTB_FMTCTL.UNPK == 0x1), each data write of size larger than FFTB_FMTCTL.FLDSZ is considered to hold multiple packed data samples. If disabled (FFTB_FMTCTL.UNPK == 0x0), each data data write (of any size) is considered to hold a single data sample.
0 (R/W)	EN	Format Converter Enable. Enables the format converter logic while writing the data to the input buffer. If (FFTB_FMTCTL.EN == 0x0), bypass the format converter logic. If (FFTB_FMTCTL.EN == 0x1), process the input stream before attempting to write to the input buffer.

Input Offset Register

The FFTB_INOFST register contains the offset value to be used by the format converter. If the format converter is enabled (FFTB_FMTCTL.EN bit =1'b1), this offset value is added to the processed input value.

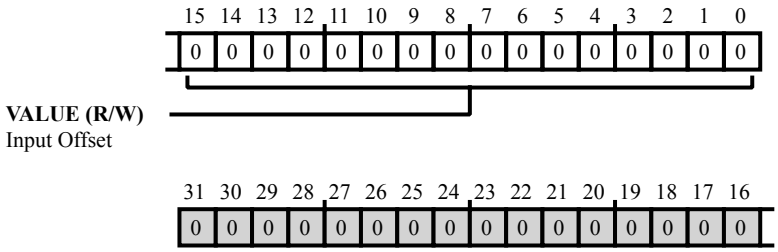


Figure 40-7: FFTB_INOFST Register Diagram

Table 40-15: FFTB_INOFST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Input Offset. The FFTB_INOFST.VALUE bit field contains the 16 bits signed offset that is added to unpacked, shifted data. This offset is only added to the processed data if the FFTB_FMTCTL.EN bit is set to 0x1.

Limit Status Registers

The FFTB_LIMSTAT[n] register contains one status bit for each power spectrum density point comparison in half the FFT spectrum.

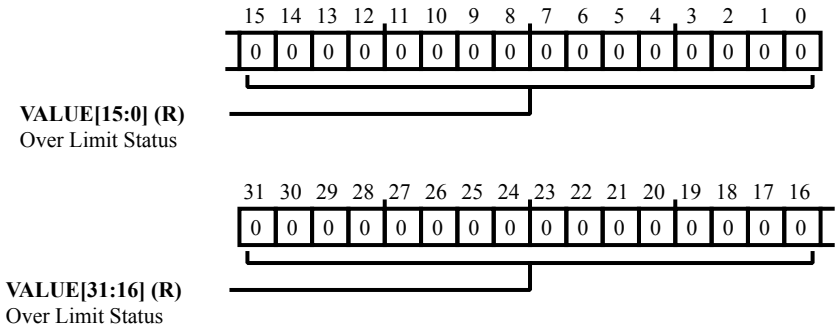


Figure 40-8: FFTB_LIMSTAT[n] Register Diagram

Table 40-16: FFTB_LIMSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Over Limit Status. Each bit in the FFTB_LIMSTAT[n].VALUE register array is set to 0x1, if the corresponding power spectrum value in half the FFT spectrum exceeds the corresponding value in the limit buffer LIMBUFF.

Magnitude Pin Select Registers

The FFTB_MAGSEL[n] registers contain one control bit for each power spectrum density point, if that bit is set the corresponding point will be used in Power Spectrum Density comparison, and finding the minimum, and maximum values.

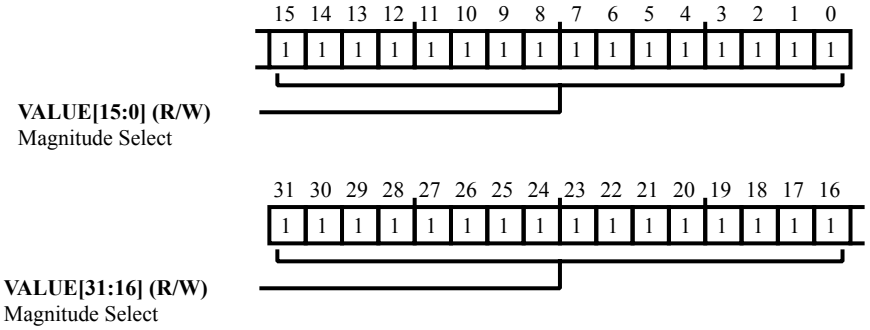


Figure 40-9: FFTB_MAGSEL[n] Register Diagram

Table 40-17: FFTB_MAGSEL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Magnitude Select. Each bit in the FFTB_MAGSEL[n].VALUE register array corresponds to power spectrum value in half the FFT spectrum. If any bit is set to 0x1 the corresponding magnitude value is used in magnitude comparison, and finding the minimum and maximum magnitude values.

Maximum Magnitude Register

The FFTB_MAXMAG register contains the maximum value of the spectrum magnitude square (Power Spectrum Density).

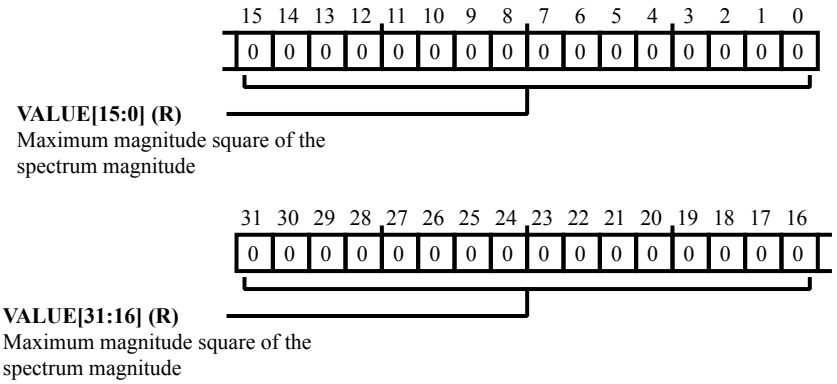


Figure 40-10: FFTB_MAXMAG Register Diagram

Table 40-18: FFTB_MAXMAG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Maximum magnitude square of the spectrum magnitude.

Minimum Magnitude Register

The FFTB_MINMAG register contains the minimum value of the spectrum magnitude square (Power Spectrum Density).

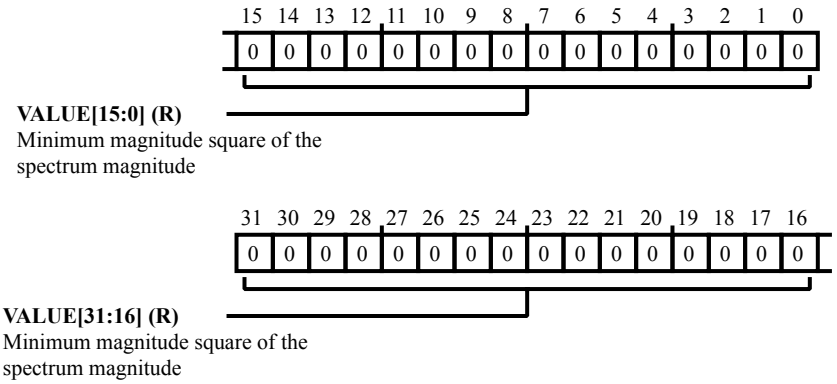


Figure 40-11: FFTB_MINMAG Register Diagram

Table 40-19: FFTB_MINMAG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Minimum magnitude square of the spectrum magnitude.

FFTB Status Register

The FFTB_STAT register contains all fields that report the status of the FFTB functionality.

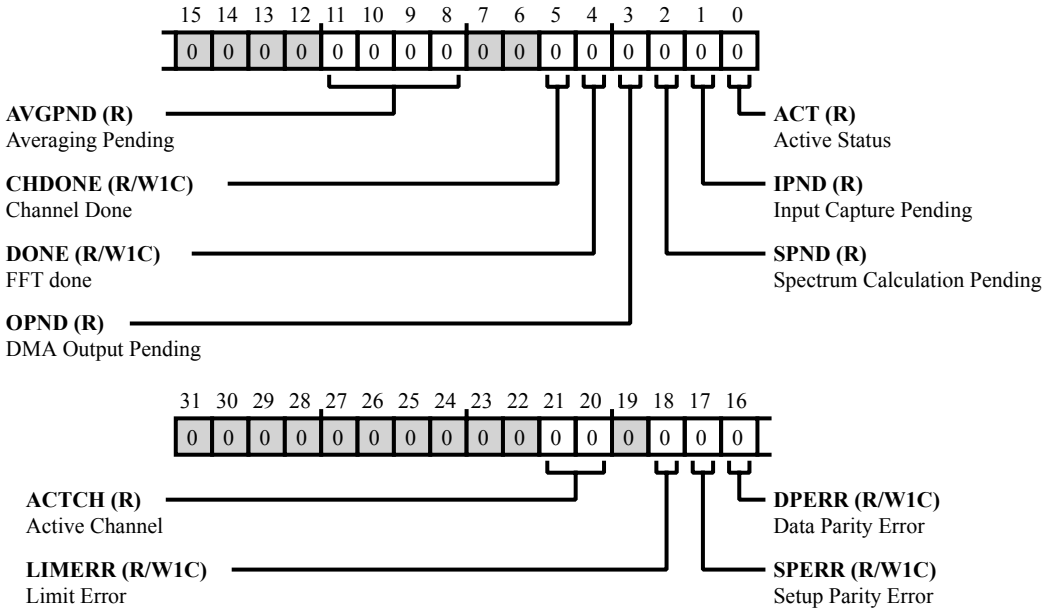


Figure 40-12: FFTB_STAT Register Diagram

Table 40-20: FFTB_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/NW)	ACTCH	Active Channel. The FFTB_STAT.ACTCH bit indicates that the FFTB is working on this channel.	
		0	Channel 0 active
		1	Channel 1 active
		2	Channel 2 active
		3	Channel 3 active
18 (R/W1C)	LIMERR	Limit Error. The FFTB_STAT.LIMERR bit indicates that one or more values in the averaging buffer exceeded the corresponding limit in the limit buffer.	
17 (R/W1C)	SPERR	Setup Parity Error. The FFTB_STAT.SPERR bit indicates that a Parity error was detected in any of the setup buffer (windowing buffer, and limit buffer).	

Table 40-20: FFTB_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W1C)	DPERR	Data Parity Error. The FFTB_STAT.DPERR bit indicates that a Parity error was detected in any of the data buffer (input buffer, real buffer, imaginary buffer, averaging buffer, and compare buffer).
11:8 (R/NW)	AVGPND	Averaging Pending. The FFTB_STAT.AVGPND bit indicates the remaining number of input vectors needed for averaging calculation.
5 (R/W1C)	CHDONE	Channel Done. The FFTB_STAT.CHDONE bit is asserted on the completion of an FFT computation. If (FFTB_CTL.DMAOUT =0x1), the FFT done status is asserted after the DMA output is done writing the last transfer. If (FFTB_CTL.DMAOUT =0x0), the FFT done status is asserted after the computation of the last FFT data point.
4 (R/W1C)	DONE	FFT done. The FFTB_STAT.DONE bit is asserted on the completion of an FFT computation. If (FFTB_CTL.DMAOUT == 0x1), the FFT done status is asserted after the DMA output is done writing the last transfer. If (FFTB_CTL.DMAOUT == 0x0), the FFT done status is asserted after the computation of the last FFT data point.
3 (R/NW)	OPND	DMA Output Pending. The FFTB_STAT.OPND bit indicates that a DMA output is in progress.
2 (R/NW)	SPND	Spectrum Calculation Pending. The FFTB_STAT.SPND bit indicates that the FFT, magnitude, and averaging calculations are in progress.
1 (R/NW)	IPND	Input Capture Pending. The FFTB_STAT.IPND bit indicates that filling an input buffer is in progress.
0 (R/NW)	ACT	Active Status. The FFTB_STAT.ACT bit indicates that spectrum capture is activated by writing 0x1 to the FFTB_CTL.START bit, while the FFTB_CTL.EN bit is 0x1, and the FFTB_CTL.STOP bit is 0x0. The FFTB_STAT.ACT status field is cleared at the assertion of any error status or spectrum capture is finished and no more capture is needed.

41 Reset Control Unit (RCU)

Reset is the initial state of the processor at power-on or the run-time state of any core, as controlled by another core in the device via the RCU or as a a result of a hardware or software triggered event. In this state, all control registers are set to their default values and functional units are idle.

Additional information on reset and booting can be found in the *Boot ROM and Booting the Processor* chapter.

The Reset Control Unit (RCU) controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. Programs must guarantee that none of the reset functions puts the system into an undefined state or causes resources to stall. This functionality is important when only one of the cores is reset (programs must ensure that there is no pending system activity involving the core that is being reset). While core resets and software system resets are controlled directly in the RCU, hardware resets can come from the TRU, SEC, or CGU Oscillator Watchdog.

The following table identifies the core IDs for the processor.

Core ID	Core Description
Core 0	Cortex-M4
Core 1	Cortex-M0

RCU Features

The RCU module supports the following features:

- Hardware reset through the $\overline{\texttt{SYS\_HWRST}}$ pin
- Software system reset through the RCU control (RCU_CTL) register
- Hardware system reset through:
  - TRU module
  - SEC module (System Fault Unit , see System Fault Interface (SFI) and the NVIC)
  - ARM core reset request from the SDCTRL_AICR register.
- A clock not good reset state (safe state of chip under reset) from the Oscillator Watchdog.

- Core reset through RCU Core Reset Output (RCU_CRCTL) register

RCU Functional Description

This section provides information on the function of RCU module.

CM41X_M4 RCU Register List

The Reset Control Unit (RCU) controls how all the functional units in the processor enter and exit Reset. Differences in functional requirements and clocking constraints exist (units in different clock domains have to enter reset asynchronously, but units exit reset in a deterministic way), and these differences define how reset signals are generated. Reset signals propagate through all functional units asynchronously. For more information on RCU functionality, see the RCU register descriptions.

Table 41-1: CM41X_M4 RCU Register List

Name	Description
RCU_BCODE	Boot Code Register
RCU_CRCTL	Core Reset Outputs Control Register
RCU_CRSTAT	Core Reset Outputs Status Register
RCU_CTL	Control Register
RCU_MSG	Message Register
RCU_MSG_CLR	Message Clear Bits Register
RCU_MSG_SET	Message Set Bits Register
RCU_SRRQSTAT	System Reset Request Status Register
RCU_STAT	Status Register

CM41X_M4 RCU Trigger List

Table 41-2: CM41X_M4 RCU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 41-3: CM41X_M4 RCU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
0	RCU0_SYSRST0	RCU0 System Reset 0	Pulse
1	RCU0_SYSRST1	RCU0 System Reset 1	Pulse
2	RCU0_SYSRST2	RCU0 System Reset 2	Pulse

RCU Definitions

To make the best use of the RCU, it is useful to understand the terms in this section.

The target or source defines the following are types of resets.

Hardware Reset (by target)

All functional units except a small subsection of debug interfaces are set to their default states. State is lost in all non-volatile storage.

System Reset (by target)

All functional units except the RCU, flash interface, and debug are set to their default states.

Core n Only Reset (by target)

Affects Core n only. The system software must guarantee that a bus master cannot access the core in reset state.

Hardware Reset (by source)

The $\overline{\texttt{SYS\_HWRST}}$ input signal is asserted active (pulled low).

System Reset (by source)

Software can trigger the reset by writing to the RCU_CTL register or by another functional unit such as the TRU or any of the generic reset inputs.

RCU Architectural Concepts

To understand the architecture of the RCU, it is important to consider the reset sources and how differing resets affect the functional units of the processor.

The RCU provides the hardware that controls how all the functional units enter and exit reset. Differences in functional requirements and clocking constraints define how reset signals are generated. For example, units in different clock domains must enter reset asynchronously but exit reset in a deterministic way.

The program must guarantee that none of the reset functions put the system in an undefined state or cause resources to stall. This functionality is important when only one of the cores is reset. The program must guarantee that there is no pending system activity involving Core n before it is reset. For example, there must be no pending transactions to core 0 when the core 0 is reset and vice-versa.

Reset Sources

The following sections provide details on the various reset options and their effects on the processor.

The ***RCU Reset Sources*** figure and table define how reset sources affect the different functional units.

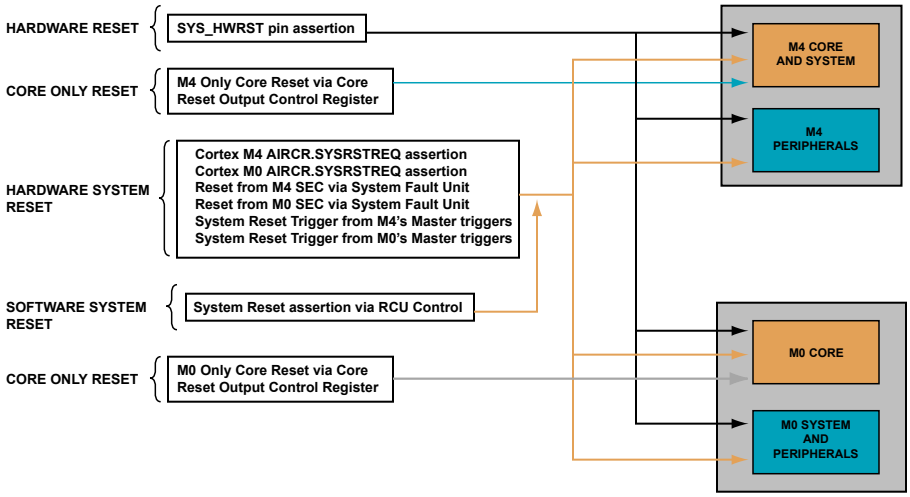


Figure 41-1: RCU Reset Sources

Table 41-4: RCU Reset Sources

Reset Type	Affected Functional Units
Hardware Reset	All functional units
Hardware System Reset	All functional units, except: • RCU_CRCTL.CR[n] • RCU_STAT • the units on the V_{DD_EXT} power domain
Software System Reset	All functional units, except: • RCU_CRCTL.CR[n] • RCU_STAT • the units on the V_{DD_EXT} power domain
Core Only Reset	Core 0 or Core 1 only

Status of Reset Assertions

Following registers are available to read the status of reset assertions by various sources:

- The RCU_SRRQSTAT register reflects the status of various system resets.
- The RCU_STAT register reflects the status of various reset types in the chip.

Reset Out Pin

The SYS_RESOUT pin is asserted while a hardware rest or system reset is asserted and continues to be automatically asserted until the application deasserts the pin. Software can control the pin by programming RCU_CTL.RSTOUTASRT and RCU_CTL.RSTOUTDSRT

Hardware Reset

A hardware reset affects the entire chip, all cores and all system and peripherals and is asserted as soon as a valid assertion is detected on the SYS_HWRST pin.

Hardware System Reset

Hardware System reset A hardware system reset can be asserted from multiple sources and it also affects all the components in the chip

- ARM Cortex-M4 SDCTRL_AICR.SYSRESETREQ assertion: Refer to the ARM Cortex-M4 User Guide for more details
- ARM Cortex-M0 SDCTRL_AICR.SYSRESETREQ assertion: Refer to the ARM Cortex-M0 User Guide for more details
- Reset from ARM Cortex-M4 SEC via System Fault Unit. An interrupt request to SEC1 can trigger a reset directly, by programming the SEC_FCTL.RESET and SEC_SCTL[n].FEN bits.
- Reset from ARM Cortex-M0 SEC via System Fault Unit. An interrupt request to SEC2 can trigger a reset directly, by programming the SEC_FCTL.RESET and SEC_SCTL[n].FEN bits.
- System Reset Trigger from ARM Cortex-M4's Master triggers. A Trigger Master can directly send a trigger pulse to Trigger slaves through the RCU_SYSRST0 signal, initiating a system reset by trigger events.
- System Reset Trigger from ARM Cortex-M0's Master triggers. A Trigger Master can directly send a trigger pulse to Trigger slaves through the RCU_SYSRST0 signal, initiating a system reset by trigger events.
- The RCU_SRRQSTAT register reflects the status of various system resets.

  The bits in this register are as follows.

  - Bit 7 source is M4_SYSRST_REQ. This is the System Reset request initiated using the M4 SDCTRL_AICR.SYSRESETREQ control bit.
  - Bit 6 source is M0_SYSRST_REQ. This is the System Reset request initiated using the M0 SDCTRL_AICR.SYSRESETREQ control bit.
  - Bit 5 source is TRU0, Assignable RCU[1].SYSRST[0] trigger slave.
  - Bit 4:2 source is TRU1 [2:0], Assignable RCU[0].SYSRST[2:0] trigger slave.
  - Bit 1 source is SEC1_RESET_REQ. This is the SEC1 event/fault controllers reset request output.
  - Bit 0 source is SEC0_RESET_REQ. This is the SEC0 event/fault controllers reset request output.

The RCU_CTL.SRSTREQEN bit controls whether System Reset sources listed above can asserted a system reset or not. Optionally, it is also possible to assert a core only reset from the system reset sources, by programming the RCU_CTL.CRSTREQEN bit.

Software System Reset

Software can also issue a system reset by writing to the RCU_CTL.SYSRST bit.

Core only System Reset

Either of the ARM Cortex cores can be separately and independently reset by writing to the RCU_CRCTL.CR[n] bit field, where, n = 0 implies the ARM Cortex-M4 core and n = 1 implies ARM Cortex-M0 core. Note that core only reset is a case where the intended core is put under reset and therefore it must be asserted and de-asserted by the other Core. Once asserted, the other core can check the status by reading the RCU_CRSTAT.CR[n] bit field. The system software must guarantee that a bus master cannot access the core in reset state.

RCU Status and Error Signals

The RCU_STAT register reflects status and error information. There are three kinds of errors that can occur in the RCU. The reset out error is triggered when RSTOUT is both asserted and deasserted at the same time. The lock write error occurs if an attempt is made to write a lock RCU register. The address error occurs if a read-only register is written to or if an attempt is made to a reserved address within the RCU MMR address range.

Resetting a Core Through a System Master

The RCU allows reset of a given core n using a system master.

A core can be reset by software, either by setting any of the RCU_CRCTL.CR[n] bits. A core that resets itself cannot guarantee that all the system transactions to or from it have completed. Although a core reset can be triggered by the core itself, it is recommended that a system master trigger the reset. The core can then be reset to restore its proper operation when it cannot execute software. The following steps provide the suggested sequence to reset a core.

1. Clear the RCU_CRSTAT.CR[n] bit.
2. Disable interrupts to the core.
3. Set the RCU_CRCTL.CR[n] bit to reset the core.
4. Poll the RCU_CRSTAT.CR[n] bit until the core is in reset.
5. Clear the RCU_CRCTL.CR[n] bit to take the core out of reset.
6. Poll the RCU_CRSTAT.CR[n] bit until the core is out of reset.

CM41X_M4 RCU Register Descriptions

Reset Control Unit (RCU) contains the following registers.

Table 41-5: CM41X_M4 RCU Register List

Name	Description
RCU_BCODE	Boot Code Register
RCU_CRCTL	Core Reset Outputs Control Register
RCU_CRSTAT	Core Reset Outputs Status Register
RCU_CTL	Control Register
RCU_MSG	Message Register
RCU_MSG_CLR	Message Clear Bits Register
RCU_MSG_SET	Message Set Bits Register
RCU_SRRQSTAT	System Reset Request Status Register
RCU_STAT	Status Register

Boot Code Register

The RCU_BCODE register can be used to determine if and how core boots. This register is set to its default values by Hard Reset.

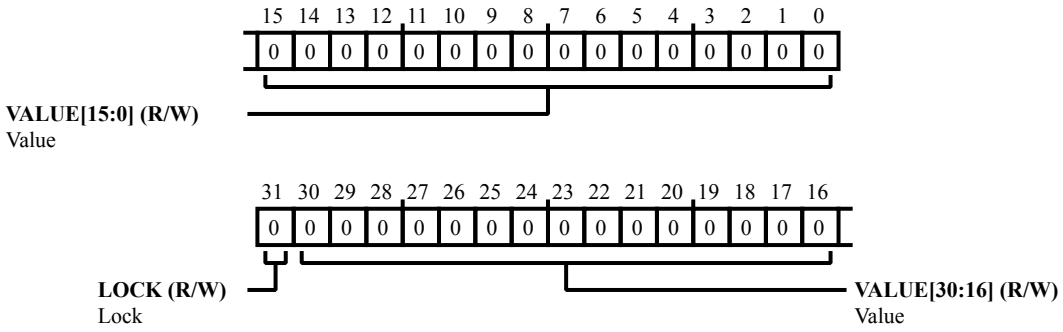


Figure 41-2: RCU_BCODE Register Diagram

Table 41-6: RCU_BCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the RCU_BCODE.LOCK bit is set, the RCU_BCODE register is read only (locked).	
		0	Unlock
		1	Lock
30:0 (R/W)	VALUE	Value. The RCU_BCODE.VALUE is a reserved field and may be used for boot software implementations.	

Core Reset Outputs Control Register

The RCU core reset control n registers (RCU_CRCTL) include a lock bit (RCU_CRCTL.LOCK) and a core reset bit (RCU_CRCTL.CR[n]) for each core reset signal on the product.

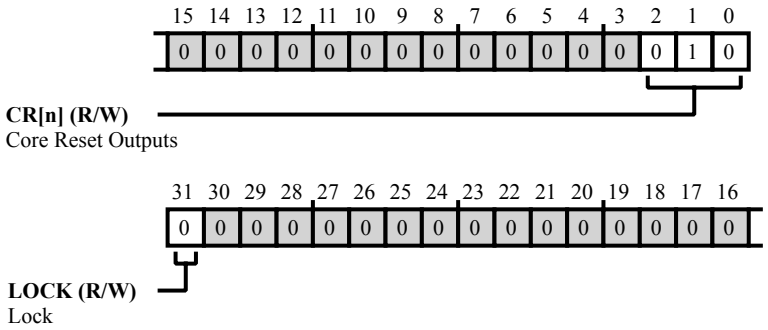


Figure 41-3: RCU_CRCTL Register Diagram

Table 41-7: RCU_CRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the RCU_CRCTL.LOCK bit is set, the RCU_CRCTL register is read only (locked).	
		0	Unlock
		1	Lock
2:0 (R/W)	CR[n]	Core Reset Outputs. The RCU_CRCTL.CR[n] bits control CRES[1:0] core reset signals. The RCU_CRES[n] signals can be individually controlled. They are reset to their default value by a hard reset or a system reset. For each RCU_CRES[n], the selected RCU0_CRMSKi[n] bit is cleared.	
		0	RCU_CRES[2:0] Deasserted
		7	RCU_CRES[2:0] Asserted

Core Reset Outputs Status Register

The RCU core reset status register (RCU_CRSTAT) contains status bits, indicating which core reset signals have been asserted.

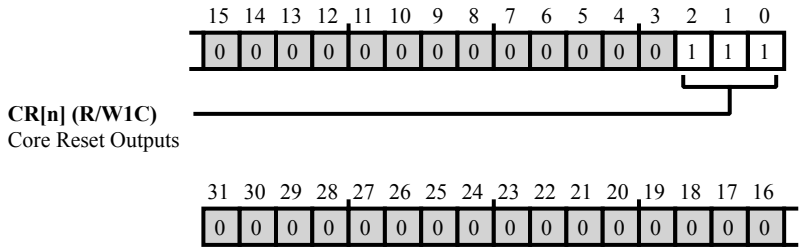


Figure 41-4: RCU_CRSTAT Register Diagram

Table 41-8: RCU_CRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W1C)	CR[n]	Core Reset Outputs. The RCU_CRSTAT.CR[n] bits indicate which cores have been reset since the last time the bit was cleared. Bits masked by CORE_DISABLE_MASK[15:0] are permanently disabled and the corresponding CR bits set. CR bits are sticky, they need to be cleared by software.	
		0	RCU_CRES[1:0] deasserted. CR[n] corresponds to RCU_CRES[n].
		7	RCU_CRES[2:0] were asserted since the last time bits were cleared. CR[n] corresponds to RCU_CRES[n].

Control Register

The RCU control register (RCU_CTL) provides a register lock, enables for the core and system reset requests inputs and control for the Reset Output pin.

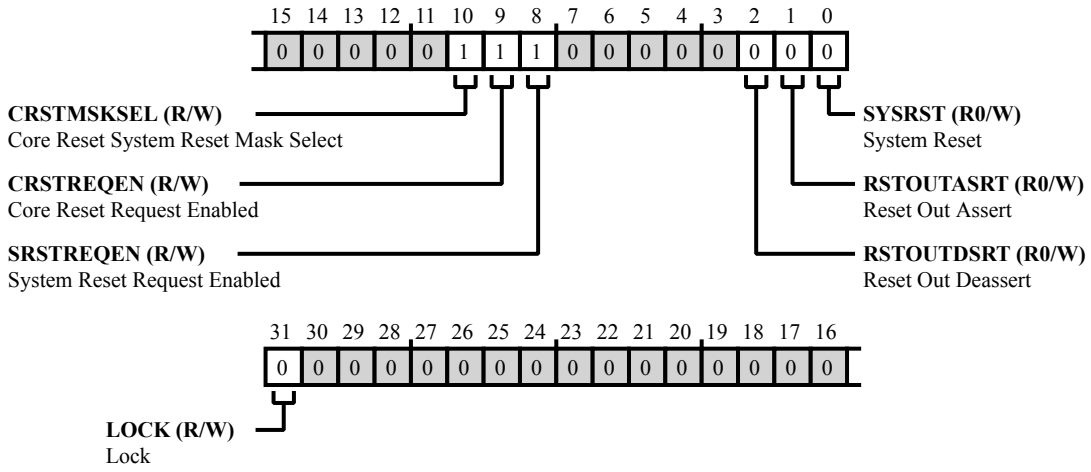


Figure 41-5: RCU_CTL Register Diagram

Table 41-9: RCU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the RCU_CTL.LOCK bit is set, the RCU_CTL register is read only (locked). This bit is cleared by a hard reset or any system reset event.	
		0	Unlock
		1	Lock
10 (R/W)	CRSTMSKSEL	Core Reset System Reset Mask Select. The RCU_CTL.CRSTMSKSEL bit selects the core reset system reset mask. This bit is cleared by a hard reset.	
9 (R/W)	CRSTREQEN	Core Reset Request Enabled. The RCU_CTL.CRSTREQEN bit controls whether the SYSCLK domain source(s) of reset is/are enabled to reset the core(s) when asserted. This bit is cleared by hard reset or any system reset event.	
		0	Disabled
		1	Enabled

Table 41-9: RCU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	SRSTREQEN	System Reset Request Enabled. The RCU_CTL.SRSTREQEN bit controls whether the SYSCLK domain sources of reset are enabled to do a system reset when asserted. This bit is cleared by a hard reset.	
		0	Disabled
		1	Enabled
2 (R0/W)	RSTOUTDSRT	Reset Out Deassert. The RCU_CTL.RSTOUTDSRT bit controls the deassertion of the system reset pin.	
		0	No Action
		1	Deassert RSTOUT
1 (R0/W)	RSTOUTASRT	Reset Out Assert. The RCU_CTL.RSTOUTASRT bit controls assertion of the system reset pin.	
		0	No Action
		1	Assert RSTOUT
0 (R0/W)	SYSRST	System Reset. The RCU_CTL.SYSRST bit provides reset for all system units.	
		0	No Action
		1	System Reset

Message Register

The RCU_MSG is a general-purpose register. It is intended to provide flexibility for Boot ROM code and to pass predefined variables to the debugger. Please see the Booting chapter for product-specific details.

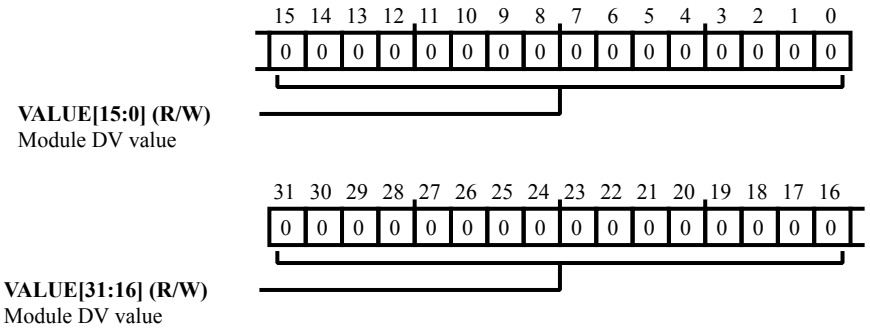


Figure 41-6: RCU_MSG Register Diagram

Table 41-10: RCU_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Module DV value. The RCU_MSG.VALUE is a reserved field and may be used for boot software implementations.

Message Clear Bits Register

The RCU_MSG_CLR register is used to clear bits in RCU_MSG register. Reading this register returns 0x00000000.

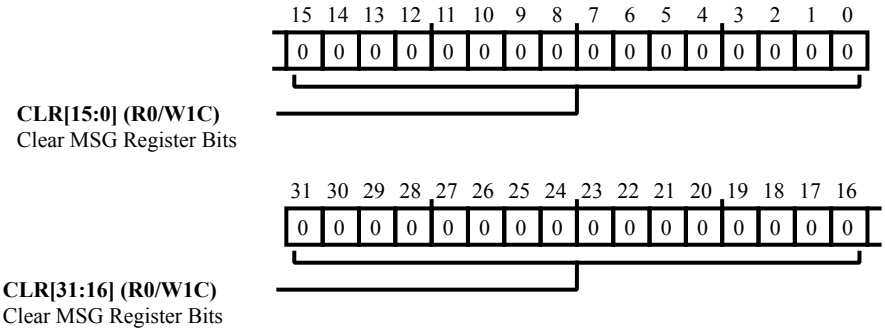


Figure 41-7: RCU_MSG_CLR Register Diagram

Table 41-11: RCU_MSG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1C)	CLR	Clear MSG Register Bits. The RCU_MSG_CLR.CLR bit resets MSG bit n.

Message Set Bits Register

The RCU_MSG_SET register is used to set bits in RCU_MSG register. Reading this register returns 0x00000000.

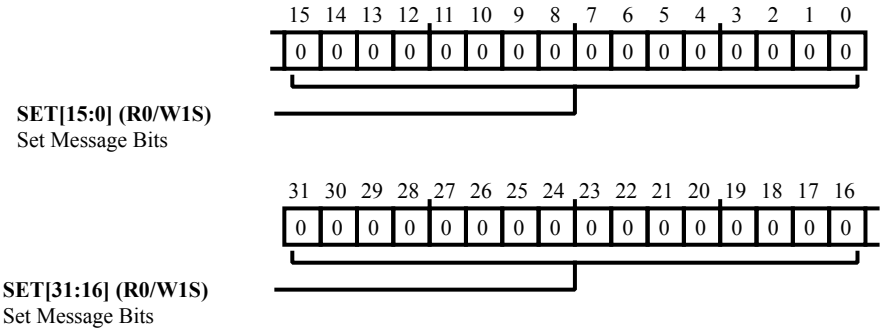


Figure 41-8: RCU_MSG_SET Register Diagram

Table 41-12: RCU_MSG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W1S)	SET	Set Message Bits. The RCU_MSG_SET.SET bit sets MSG bit n.

System Reset Request Status Register

The RCU system reset request status register (RCU_SRRQSTAT) contains status bits, indicating which system reset request input triggered a system reset. This register is set to its default values by a hard reset.

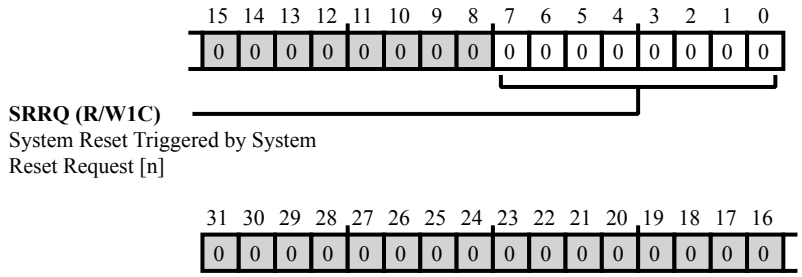


Figure 41-9: RCU_SRRQSTAT Register Diagram

Table 41-13: RCU_SRRQSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W1C)	SRRQ	System Reset Triggered by System Reset Request [n]. The RCU_SRRQSTAT.SRRQ bits are set by the assertion of the corresponding system reset request input and deasserted by writing "1" to the bit. The RCU_SRRQSTAT register is cleared by a hard reset. The bits in this register are described as follows. Bit 7 source is M4_SYSRST_REQ. This is the System Reset request initiated using the M4 AIRCR.SYSRESETREQ control bit. Bit 6 source is M0_SYSRST_REQ. This is the System Reset request initiated using the M0 AIRCR.SYSRESETREQ control bit. Bit 5 source is TRU0, Assignable RCU[1].SYSRST[0] trigger slave. Bit 4:2 source is TRU1 [2:0], Assignable RCU[0].SYSRST[2:0] trigger slave. Bit 1 source is SEC1_RESET_REQ. This is the SEC1 event/fault controllers reset request output. Bit 0 source is SEC0_RESET_REQ. This is the SEC0 event/fault controllers reset request output.

Status Register

The RCU status register (RCU_STAT) contains status bits for all RCU reset sources, reset status, and boot mode inputs. Status bits for reset sources are sticky and can cleared by software. Error status bits are cleared by any reset event.

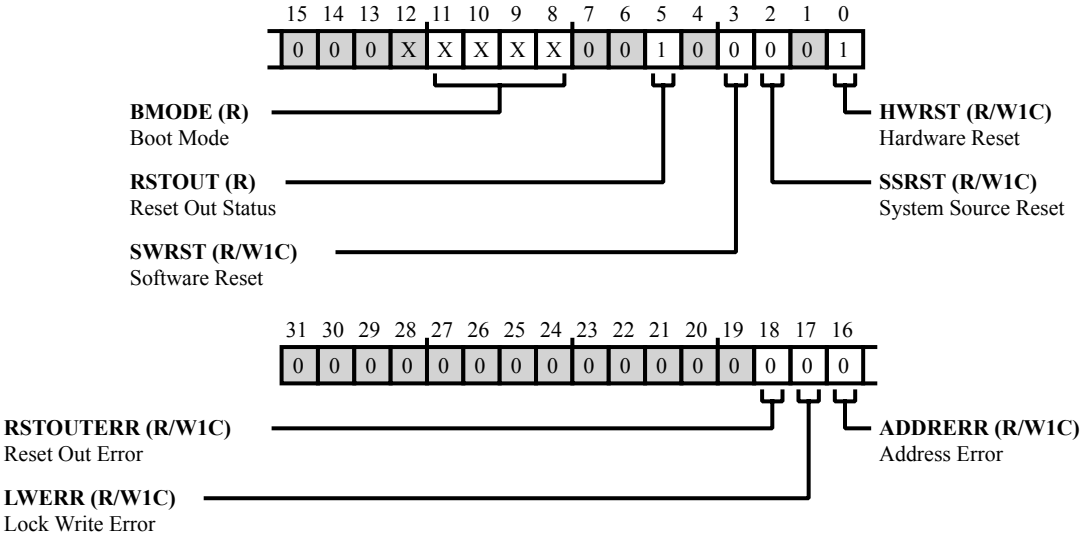


Figure 41-10: RCU_STAT Register Diagram

Table 41-14: RCU_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W1C)	RSTOUTERR	Reset Out Error. The RCU_STAT.RSTOUTERR bit indicates (if set) that a write attempted to set the RCU_CTL.RSTOUTASRT and RCU_CTL.RSTOUTDSRT simultaneously. This condition triggers a bus error.	
		0	No Error
		1	Error Occurred
17 (R/W1C)	LWERR	Lock Write Error. The RCU_STAT.LWERR bit indicates (when set) there was an attempted write to an RCU register while the RCU_CTL.LOCK bit was set and the global lock bit is enabled (SPU_CTL.GLCK bit =1). This status bit is sticky; write-1-to-clear	
		0	No Error
		1	Error Occurred

Table 41-14: RCU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	ADDRERR	Address Error. The RCU_STAT.ADDRERR bit indicates that the RCU generated an address error. This status bit is sticky; write-1-to-clear it.	
		0	No Error
		1	Error Occurred
11:8 (R/NW)	BMODE	Boot Mode. The RCU_STAT.BMODE bits indicate the input on the boot mode pins.	
5 (R/NW)	RSTOUT	Reset Out Status. The RCU_STAT.RSTOUT bit indicates the assertion status of the system reset pin.	
		0	RSTOUT Deasserted
		1	RSTOUT Asserted
3 (R/W1C)	SWRST	Software Reset. The RCU_STAT.SWRST bit indicates that a system reset (which was triggered by software) has occurred since the last time a hardware reset occurred or since the RCU_STAT.SWRST bit was cleared by software.	
		0	Inactive
		1	Reset Occurred
2 (R/W1C)	SSRST	System Source Reset. The RCU_STAT.SSRST bit indicates that a system reset triggered by hardware in the system clock domain, clock A domain, or clock B domain has occurred since the last time a hardware reset occurred or since the RCU_STAT.SSRST bit was cleared by software.	
		0	Inactive
		1	Reset Occurred
0 (R/W1C)	HWRST	Hardware Reset. The RCU_STAT.HWRST bit indicates that a hardware reset has occurred.	
		0	Inactive
		1	Reset Occurred

42 System Design and Safety

The ADSP-CM41x family of processors provide a rich variety of interface modules for construction of real-time signal chains.

Signal Chain Peripherals

The principal signal chain interfaces are shown in the following list

- Inputs
  - ADC converters and ADCC converter controllers
  - SINC filters for use with external isolated ADC converters, for example AD740x
  - Digital level or edge inputs to PORTs
  - Digital pulse or PWM digital inputs to Timers
  - Rotary counter encoders (quadrature encoded)
- Outputs
  - DAC converter and DACC controller
  - ePWM and high-precision HPPWM outputs
  - Quadrature encoded frequency output
  - Digital levels and pulses from GPIO Ports and Timers.

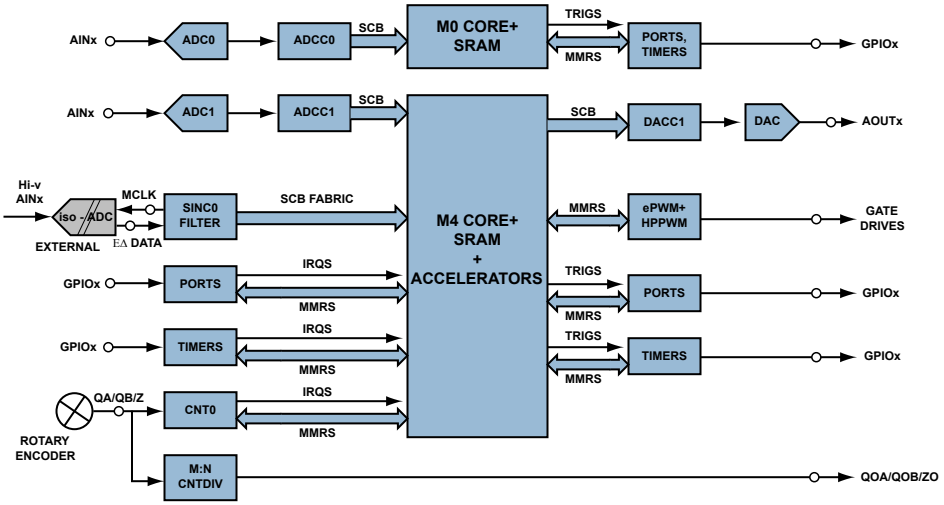


Figure 42-1: Signal Chain Peripherals

Built-In Modules for Functional Safety

The ADSP-CM41x provides a comprehensive set of interlocking functional safety features which support safety-critical applications. These safety features include:

- Voltage Monitoring Unit. Monitors V_{DD_EXT} and V_{DD_INT}.
- System Oscillator and Watchdog. Monitors SYS_CLKIN0 for dead-clock and harmonic-multiple clock faults (wrong crystal harmonic mode faults).
- Monitor Oscillator. Optional oscillator for auxiliary frequency reference source SYS_CLKIN1.
- Oscillator Comparator Unit. Monitors SYS_CLKIN0 compared to an auxiliary frequency reference SYS_CLKIN1 with high accuracy.
- WDOG Watchdog Timer. Enables a software watchdog function.
- Fault Management Unit. Monitors events designated as faults, and controls asserting the $\overline{\text{SYS\_FAULT}}$ pin.
- PWM Trip Emergency Shutdown controls.
- ECC Error-Correcting-Code Memories and Parity-Protected memories.
- The SYSBLK0 Supervisor timeout mechanism.
- System Protection Units for peripheral Control Bus Access. Controls access by any system master (ARM Cortex-M4, M0, DMA) to each peripheral block's MMRs.
- System Memory Protection Units for AXI data bus access. Controls access by any system master (ARM Cortex-M4, ARM Cortex-M0, DMA) to user-defined regions within the ARM Cortex-M4 main SRAM and the ARM Cortex-M0 supervisor SRAM system Watchpoint Unit for monitoring control and data access.
- User MEMST Memory Built-In Self-Test

- CRC for Memory Contents Validation

43 System Block (SYSBLK) and PADS

The SYSBLK and PADS registers are a group of registers that are spread across various peripherals for some specific features. This chapter is a reference for the register descriptions for each of these modules. Please consult the individual chapters for detailed functionality.

CM41X_M4 SYSBLK Register List

The SYSBLK module controls system interface signal features for a number of module interfaces.

Table 43-1: CM41X_M4 SYSBLK Register List

Name	Description
SYSBLK_SCB_RESP_CFG	SCB Response Configuration Register
SYSBLK_SCB_TIMEOUT_VALUE	SCB Timeout Value Register
SYSBLK_CLKNG_TRIPEN	Clock Not Good Trip Register
SYSBLK_DMA_MUXCTL	Peripheral DMA Multiplexer Control
SYSBLK_ENG_MODE_CFG0	Engineering Mode Configuration Register 0
SYSBLK_FAULT_TRIPEN	Fault Trip Register
SYSBLK_IRQ_LATENCY	M0 IRQ Latency Register
SYSBLK_LROM_STAT	Logic ROM Status Register
SYSBLK_M0_VTOR	Vector Table Base Offset Register
SYSBLK_MEMST_CTL	Memory Self-Test Control Register
SYSBLK_PWM_SYS_CFG	PWM System Configuration Register
SYSBLK_ROT_UPDN_CFG	Rotary Counter Up/Down Configuration Register
SYSBLK_SINC_TEST	SINC Test Register
SYSBLK_SISTAT0	Shared Interrupt 0 Status Register
SYSBLK_SISTAT10	Shared Interrupt 10 Status Register
SYSBLK_SISTAT11	Shared Interrupt 11 Status Register
SYSBLK_SISTAT12	Shared Interrupt 12 Status Register

Table 43-1: CM41X_M4 SYSBLK Register List (Continued)

Name	Description
SYSBLK_SISTAT15	Shared Interrupt 15 Status Register
SYSBLK_SISTAT16	Shared Interrupt 16 Status Register
SYSBLK_SISTAT17	Shared Interrupt 17 Status Register
SYSBLK_SISTAT18	Shared Interrupt 18 Status Register
SYSBLK_SISTAT19	Shared Interrupt 19 Status Register
SYSBLK_SISTAT22	Shared Interrupt 22 Status Register
SYSBLK_SISTAT25	Shared Interrupt 25 Status Register
SYSBLK_SISTAT28	Shared Interrupt 28 Status Register
SYSBLK_SISTAT3	Shared Interrupt 3 Status Register
SYSBLK_SISTAT5	Shared Interrupt 5 Status Register
SYSBLK_SISTAT6	Shared Interrupt 6 Status Register
SYSBLK_SISTAT7	Shared Interrupt 7 Status Register
SYSBLK_SISTAT8	Shared Interrupt 8 Status Register
SYSBLK_SRAM0_ECC	M0 SRAM ECC Register
SYSBLK_SYSSTAT	System Status Register
SYSBLK_SYS_STCALIB	System Register

CM41X_M0 SYSBLK Register List

The SYSBLK module controls system interface signal features for a number of module interfaces.

Table 43-2: CM41X_M0 SYSBLK Register List

Name	Description
SYSBLK_SCB_RESP_CFG	SCB Response Configuration Register
SYSBLK_SCB_TIMEOUT_VALUE	SCB Timeout Value Register
SYSBLK_CLKNG_TRIPEN	Clock Not Good Trip Register
SYSBLK_DMA_MUXCTL	Peripheral DMA Multiplexer Control
SYSBLK_ENG_MODE_CFG0	Engineering Mode Configuration Register 0
SYSBLK_FAULT_TRIPEN	Fault Trip Register
SYSBLK_IRQ_LATENCY	M0 IRQ Latency Register
SYSBLK_LROM_STAT	Logic ROM Status Register
SYSBLK_M0_VTOR	Vector Table Base Offset Register
SYSBLK_MEMST_CTL	Memory Self-Test Control Register

Table 43-2: CM41X_M0 SYSBLK Register List (Continued)

Name	Description
SYSBLK_PWM_SYS_CFG	PWM System Configuration Register
SYSBLK_ROT_UPDN_CFG	Rotary Counter Up/Down Configuration Register
SYSBLK_SINC_TEST	SINC Test Register
SYSBLK_SISTAT0	Shared Interrupt 0 Status Register
SYSBLK_SISTAT10	Shared Interrupt 10 Status Register
SYSBLK_SISTAT11	Shared Interrupt 11 Status Register
SYSBLK_SISTAT12	Shared Interrupt 12 Status Register
SYSBLK_SISTAT15	Shared Interrupt 15 Status Register
SYSBLK_SISTAT16	Shared Interrupt 16 Status Register
SYSBLK_SISTAT17	Shared Interrupt 17 Status Register
SYSBLK_SISTAT18	Shared Interrupt 18 Status Register
SYSBLK_SISTAT19	Shared Interrupt 19 Status Register
SYSBLK_SISTAT22	Shared Interrupt 22 Status Register
SYSBLK_SISTAT25	Shared Interrupt 25 Status Register
SYSBLK_SISTAT28	Shared Interrupt 28 Status Register
SYSBLK_SISTAT3	Shared Interrupt 3 Status Register
SYSBLK_SISTAT5	Shared Interrupt 5 Status Register
SYSBLK_SISTAT6	Shared Interrupt 6 Status Register
SYSBLK_SISTAT7	Shared Interrupt 7 Status Register
SYSBLK_SISTAT8	Shared Interrupt 8 Status Register
SYSBLK_SRAM0_ECC	M0 SRAM ECC Register
SYSBLK_SYSSTAT	System Status Register
SYSBLK_SYS_STCALIB	System Register

CM41X_M4 PADS Register List

The PADS module controls system interface signal features for a number of module interfaces.

Table 43-3: CM41X_M4 PADS Register List

Name	Description
PADS_DBC[n]_CTL	Debounce Control Register(s)
PADS_DBC_PRESCALE	Debounce Prescale Register
PADS_FOCP_DIV	Fast Over Current Protection Clock Divisor Register

Table 43-3: CM41X_M4 PADS Register List (Continued)

Name	Description
PADS_MONOSC_CFG	Monitor Oscillator Control Register
PADS_NVWR_RSTCTL	Non-Volatile Write Reset Control Register
PADS_PCFG0	Peripheral Configuration0 Register
PADS_PORT[n]_DS	Multi Port Drive Strength Control Register
PADS_PORT[n]_RCTL	Multi Port Pull-up/Pull-down Resistor Control Register
PADS_PORT[n]_TRIPSEL	Multi Port Trip Select Register
PADS_PORT[n]_TRIPST	Multi Port Trip State Register
PADS_VMU_CTL	Voltage Monitor Unit Control Register
PADS_VMU_TRIM	Voltage Monitor Unit Trim Register
PADS_VMU_TRIPEN	Voltage Monitor Unit Trip Enable Register

CM41X_M0 PADS Register List

The PADS module controls system interface signal features for a number of module interfaces.

Table 43-4: CM41X_M0 PADS Register List

Name	Description
PADS_DBC[n]_CTL	Debounce Control Register(s)
PADS_DBC_PRESCALE	Debounce Prescale Register
PADS_FOCP_DIV	Fast Over Current Protection Clock Divisor Register
PADS_MONOSC_CFG	Monitor Oscillator Control Register
PADS_NVWR_RSTCTL	Non-Volatile Write Reset Control Register
PADS_PCFG0	Peripheral Configuration0 Register
PADS_PORT[n]_DS	Multi Port Drive Strength Control Register
PADS_PORT[n]_RCTL	Multi Port Pull-up/Pull-down Resistor Control Register
PADS_PORT[n]_TRIPSEL	Multi Port Trip Select Register
PADS_PORT[n]_TRIPST	Multi Port Trip State Register
PADS_VMU_CTL	Voltage Monitor Unit Control Register
PADS_VMU_TRIM	Voltage Monitor Unit Trim Register
PADS_VMU_TRIPEN	Voltage Monitor Unit Trip Enable Register

CM41X_M4 TESYS Interrupt List

Table 43-5: CM41X_M4 TESYS Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
13	C1_BUS_TIMEOUT	SYSBLK1 Bus Timeout	Level	
28	C0_BUS_TIMEOUT	SYSBLK0 Bus Timeout	Level	
29	SYSBLK0_POSTWR_ERR	SYSBLK0 Posted Write Error	Level	

CM41X_M0 TESYS Interrupt List

Table 43-6: CM41X_M0 TESYS Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
3	C0_BUS_TIMEOUT	SYSBLK0 Bus Timeout	Level	
3	SYSBLK0_POSTWR_ERR	SYSBLK0 Posted Write Error	Level	
5	C1_BUS_TIMEOUT	SYSBLK1 Bus Timeout	Level	

CM41X_M4 SYSBLK Register Descriptions

Pads Controller (SYSBLK) contains the following registers.

Table 43-7: CM41X_M4 SYSBLK Register List

Name	Description
SYSBLK_SCB_RESP_CFG	SCB Response Configuration Register
SYSBLK_SCB_TIMEOUT_VALUE	SCB Timeout Value Register
SYSBLK_CLKNG_TRIPEN	Clock Not Good Trip Register
SYSBLK_DMA_MUXCTL	Peripheral DMA Multiplexer Control
SYSBLK_ENG_MODE_CFG0	Engineering Mode Configuration Register 0
SYSBLK_FAULT_TRIPEN	Fault Trip Register
SYSBLK_IRQ_LATENCY	M0 IRQ Latency Register
SYSBLK_LROM_STAT	Logic ROM Status Register
SYSBLK_M0_VTOR	Vector Table Base Offset Register
SYSBLK_MEMST_CTL	Memory Self-Test Control Register
SYSBLK_PWM_SYS_CFG	PWM System Configuration Register
SYSBLK_ROT_UPDN_CFG	Rotary Counter Up/Down Configuration Register
SYSBLK_SINC_TEST	SINC Test Register

Table 43-7: CM41X_M4 SYSBLK Register List (Continued)

Name	Description
SYSBLK_SISTAT0	Shared Interrupt 0 Status Register
SYSBLK_SISTAT10	Shared Interrupt 10 Status Register
SYSBLK_SISTAT11	Shared Interrupt 11 Status Register
SYSBLK_SISTAT12	Shared Interrupt 12 Status Register
SYSBLK_SISTAT15	Shared Interrupt 15 Status Register
SYSBLK_SISTAT16	Shared Interrupt 16 Status Register
SYSBLK_SISTAT17	Shared Interrupt 17 Status Register
SYSBLK_SISTAT18	Shared Interrupt 18 Status Register
SYSBLK_SISTAT19	Shared Interrupt 19 Status Register
SYSBLK_SISTAT22	Shared Interrupt 22 Status Register
SYSBLK_SISTAT25	Shared Interrupt 25 Status Register
SYSBLK_SISTAT28	Shared Interrupt 28 Status Register
SYSBLK_SISTAT3	Shared Interrupt 3 Status Register
SYSBLK_SISTAT5	Shared Interrupt 5 Status Register
SYSBLK_SISTAT6	Shared Interrupt 6 Status Register
SYSBLK_SISTAT7	Shared Interrupt 7 Status Register
SYSBLK_SISTAT8	Shared Interrupt 8 Status Register
SYSBLK_SRAM0_ECC	M0 SRAM ECC Register
SYSBLK_SYSSTAT	System Status Register
SYSBLK_SYS_STCALIB	System Register

SCB Response Configuration Register

The SYSBLK_SCB_RESP_CFG register provides SCB bus monitoring capabilities from the ARM Cortex-M4 system entering the main system scoreboard (SCB).

If the SYSBLK_SCB_RESP_CFG.TOENA bit is enabled the block checks the response time from the AHB block. If the response time exceeds the timeout value (configured in the SYSBLK_SCB_TIMEOUT_VALUE register) this block generates a default error response back to the ARM Cortex-M4 system and indicates an error.

Accessing the SCB from the ARM Cortex-M4 through the system scoreboard may result in a minor multi-cycle stall. The timeout is a safety feature that ensures the ARM Cortex-M4 system has a known maximum response time to a SCB system access. Note that system MMR accesses are handled on a separate peripheral bus chain and use different timeout hardware to monitor those accesses.

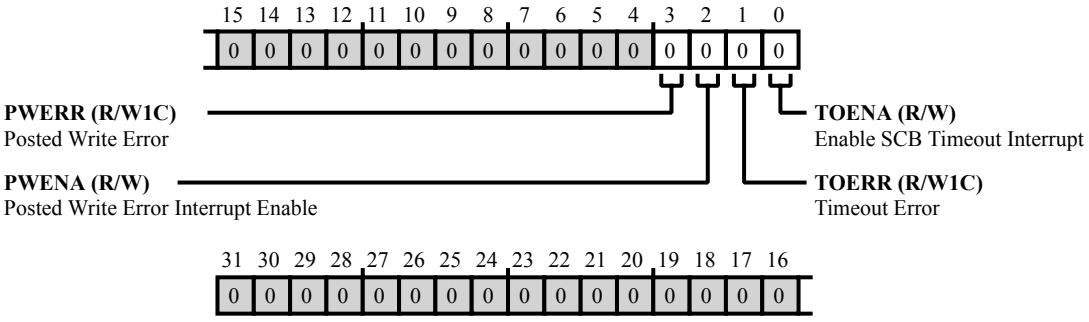


Figure 43-1: SYSBLK_SCB_RESP_CFG Register Diagram

Table 43-8: SYSBLK_SCB_RESP_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1C)	PWERR	Posted Write Error. The SYSBLK_SCB_RESP_CFG.PWERR bit indicates an error has occurred on a previously executed Posted Write to the system fabric. Since writes were posted, this error is not associated with the interrupted instruction.	
		0	No error
		1	Error occurred
2 (R/W)	PWENA	Posted Write Error Interrupt Enable. The SYSBLK_SCB_RESP_CFG.PWENA bit enables read/write error indications to the system fabric. The ARM M0 process does not wait for the read/write to complete before executing the subsequent instructions. Any errors caused by the read/write set the PWERR bit and cause an inexact interrupt to the ARM processor.	
		0	Interrupt disabled
		1	Interrupt enabled

Table 43-8: SYSBLK_SCB_RESP_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	TOERR	Timeout Error. The SYSBLK_SCB_RESP_CFG.TOERR indicates that the SCB bus has timed out.	
		0	No error
		1	Error occurred
0 (R/W)	TOENA	Enable SCB Timeout Interrupt. The SYSBLK_SCB_RESP_CFG.TOENA bit enables the the SCB timeout interrupt. When enabled, the value in the SYSBLK_SCB_TIMEOUT_VALUE register is loaded into a counter at the start of a each new SCB transaction. The counter decrements once for each cycle that the SCB bus is not ready (stalled). If the counter decrements to the value one, an error interrupt request is generated.	
		0	Disable interrupt
		1	Enable interrupt

SCB Timeout Value Register

The SYSBLK_SCB_TIMEOUT_VALUE register contains the timeout value. This value is loaded into a counter at the start of each new SCB transaction. The counter decrements once for each cycle that the SCB bus is not ready (stalled).

If the counter decrements to the value =1, an error interrupt request is produced. If DATA =0, no error produced. If DATA =1, an error is produced for any SCB bus transaction.

The Timeout Value should be set to a value larger than the number of cycles that it takes for the slowest SCB transaction to complete to prevent spurious IRQs.

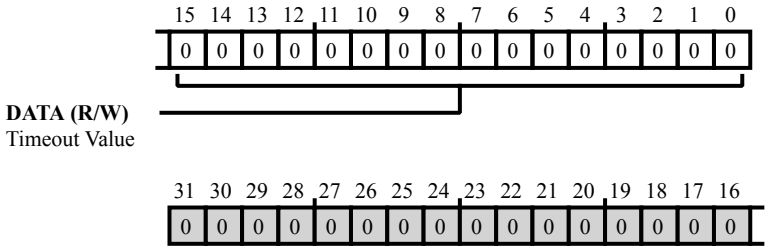


Figure 43-2: SYSBLK_SCB_TIMEOUT_VALUE Register Diagram

Table 43-9: SYSBLK_SCB_TIMEOUT_VALUE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DATA	Timeout Value. The SYSBLK_SCB_TIMEOUT_VALUE.DATA bit field contains the timeout value. The Timeout value should be set to a value larger than the number of cycles that it takes for the slowest SCB transaction to complete to prevent spurious IRQs.

Clock Not Good Trip Register

The SYSBLK_CLKNG_TRIPEN register (OCU) has 2 error outputs for clock monitoring. These 2 control bits allow the OCU0 error outputs to cause the safety pin-trip feature to activate.

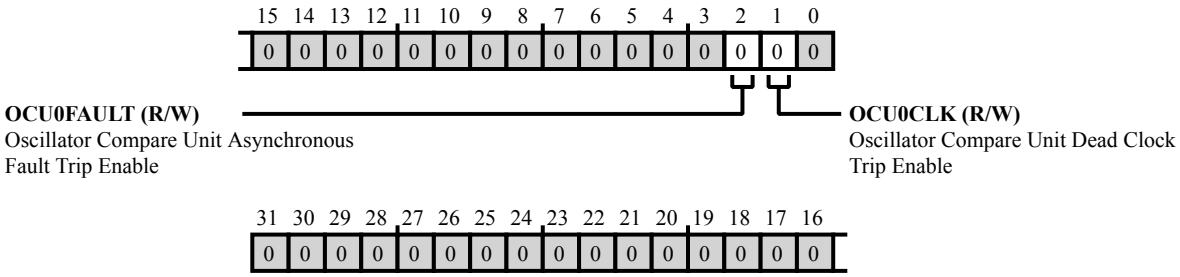


Figure 43-3: SYSBLK_CLKNG_TRIPEN Register Diagram

Table 43-10: SYSBLK_CLKNG_TRIPEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	OCU0FAULT	Oscillator Compare Unit Asynchronous Fault Trip Enable.
1 (R/W)	OCU0CLK	Oscillator Compare Unit Dead Clock Trip Enable.

Peripheral DMA Multiplexer Control

The SYSBLK_DMA_MUXCTL register allows peripherals to share pins and the same DMA ports. This sharing is not "in-parallel", users must configure the MDMA blocks for the desired operating peripherals. DMA channels 6-11 need to be configured to support the desired user peripheral.

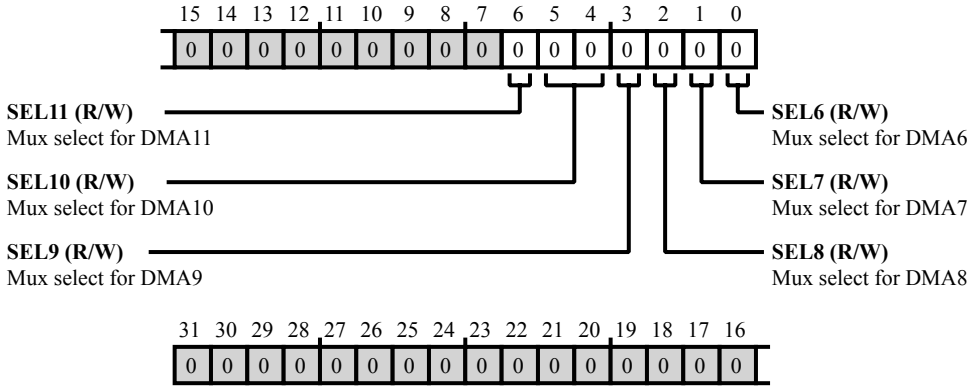


Figure 43-4: SYSBLK_DMA_MUXCTL Register Diagram

Table 43-11: SYSBLK_DMA_MUXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SEL11	Mux select for DMA11.	
		0	Select SPORT[0] B_DMA port
		1	Select UART[4] RXDMA port
5:4 (R/W)	SEL10	Mux select for DMA10.	
		0	Select HAE[0] RXDMA_CH1 port
		1	Select SPORT[0] A_DMA port
		2	Select UART[4] TXDMA port
3 (R/W)	SEL9	Mux select for DMA9.	
		0	Select HAE[0] TXDMA port
		1	Select UART[3] RXDMA port
2 (R/W)	SEL8	Mux select for DMA8.	
		0	Select HAE[0] RXDMA_CH0 port
		1	Select UART[3] TXDMA port
1 (R/W)	SEL7	Mux select for DMA7.	
		0	Select SPI[1] RXDMA port
		1	Select UART[2] RXDMA port

Table 43-11: SYSBLK_DMA_MUXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	SEL6	Mux select for DMA6.	
		0	Select SPI[1] TXDMA port
		1	Select UART[2] TXDMA port

Engineering Mode Configuration Register 0

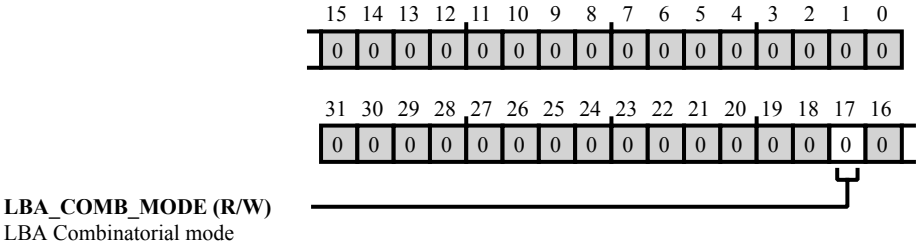


Figure 43-5: SYSBLK_ENG_MODE_CFG0 Register Diagram

Table 43-12: SYSBLK_ENG_MODE_CFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	LBA_COMB_MODE	LBA Combinatorial mode.

Fault Trip Register

The SYSBLK_FAULT_TRIPEN register controls various Trip functions of the Oscillator Comparator Unit (OCU). See the OCU chapter for complete information.

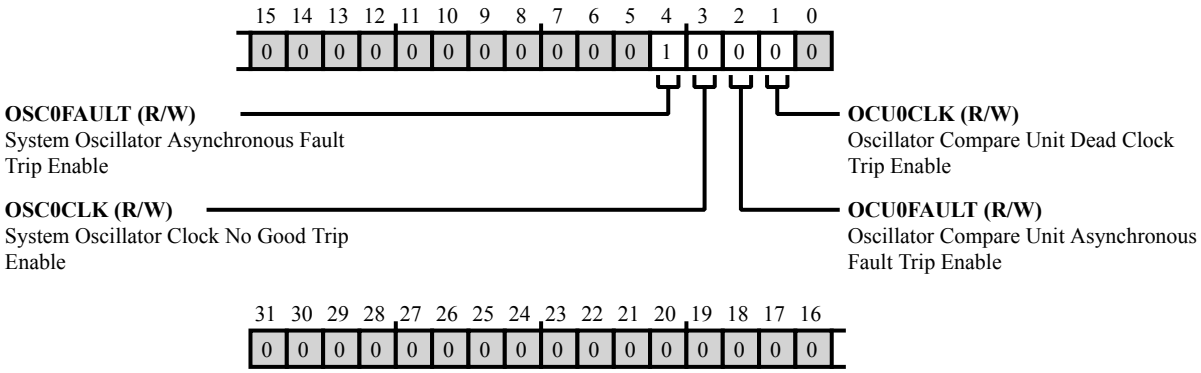


Figure 43-6: SYSBLK_FAULT_TRIPEN Register Diagram

Table 43-13: SYSBLK_FAULT_TRIPEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	OSC0FAULT	System Oscillator Asynchronous Fault Trip Enable. The SYSBLK_FAULT_TRIPEN.OSC0FAULT bit enables asserting the SYS_FAULTb pin asynchronously on OSC_WDOG detection of a clock fault, as asserted on the OSC_WDOG's 'fault' pin. Note that this enable is true by default at power-on. This allows detection of dead-clock or wrong-harmonic-mode faults on SYS_CLKIN0 to be signaled on the SYS_FAULTb pin at power-on.
3 (R/W)	OSC0CLK	System Oscillator Clock No Good Trip Enable. The SYSBLK_FAULT_TRIPEN.OSC0CLK bit enables asserting the SYS_FAULTb pin asynchronously on assertion of the OSC_WDOG clk_not_good output signal. clk_not_good can be asserted by the OSC_WDOG approximately 1 ms after a fault detection.
2 (R/W)	OCU0FAULT	Oscillator Compare Unit Asynchronous Fault Trip Enable. The SYSBLK_FAULT_TRIPEN.OCU0FAULT bit enables asserting the SYS_FAULTb pin asynchronously on OCU0 detection of either a clock frequency error or a dead clock (OCU port ocu_fault_async).
1 (R/W)	OCU0CLK	Oscillator Compare Unit Dead Clock Trip Enable. The SYSBLK_FAULT_TRIPEN.OCU0CLK bit enables asserting the SYS_FAULTb pin asynchronously on OCU0 detection of a dead SYSCLK (OCU port ocu_dead_clock).

M0 IRQ Latency Register

The SYSBLK_IRQ_LATENCY register configures the wait state memory system.

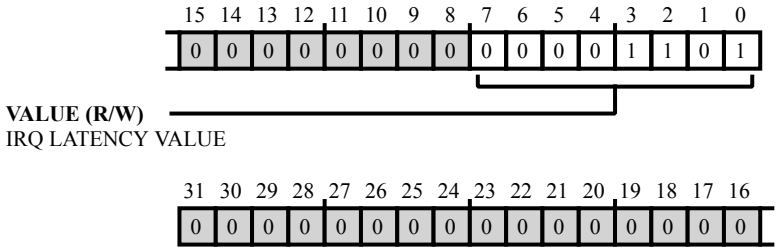


Figure 43-7: SYSBLK_IRQ_LATENCY Register Diagram

Table 43-14: SYSBLK_IRQ_LATENCY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	IRQ LATENCY VALUE. The SYSBLK_IRQ_LATENCY.VALUE bit field configures the wait state memory system. For zero jitter in a zero wait state memory system, set this bus to at least a decimal value of 13.

Logic ROM Status Register

The SYSBLK_LROM_STAT register provides indications of various errors in logic ROM boot code.

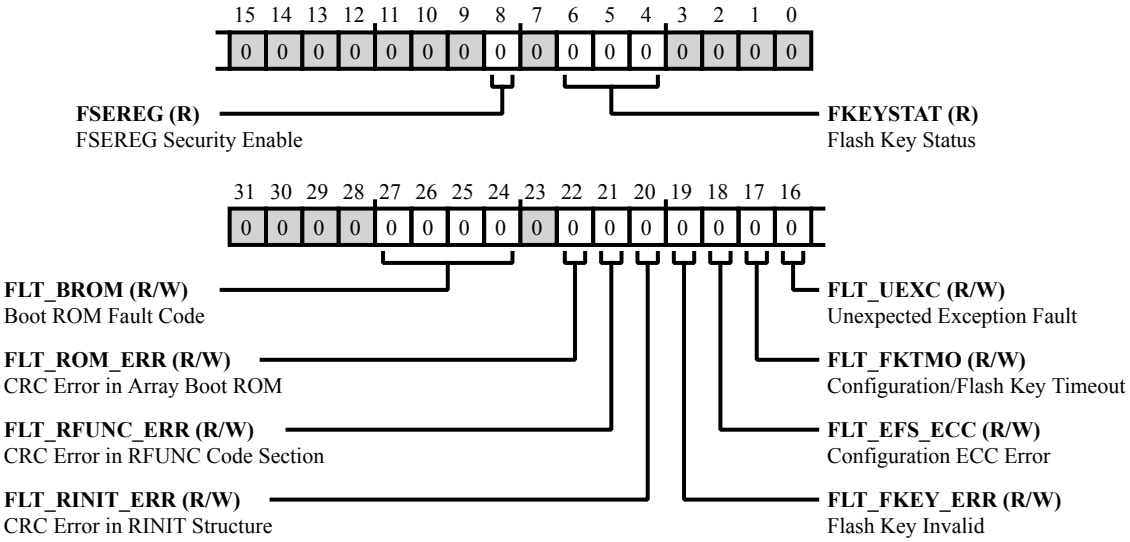


Figure 43-8: SYSBLK_LROM_STAT Register Diagram

Table 43-15: SYSBLK_LROM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:24 (R/W)	FLT_BROM	Boot ROM Fault Code. The SYSBLK_LROM_STAT.FLT_BROM bit indicates a Boot ROM fault code.
22 (R/W)	FLT_ROM_ERR	CRC Error in Array Boot ROM. The SYSBLK_LROM_STAT.FLT_ROM_ERR bit indicates that the logic ROM code detected a CRC error was detected in the Array Boot ROM. The Array Boot ROM code was not executed.
21 (R/W)	FLT_RFUNC_ERR	CRC Error in RFUNC Code Section. The SYSBLK_LROM_STAT.FLT_RFUNC_ERR bit indicates a CRC error was detected in the RFUNC structure described by RFUNC_PTR and RFUNC_CSIZ in Flash Info Block 0.
20 (R/W)	FLT_RINIT_ERR	CRC Error in RINIT Structure. The SYSBLK_LROM_STAT.FLT_RINIT_ERR bit indicates a CRC error was detected in the RINIT structure described by RINIT_PTR and RINIT_CSIZ in Flash Info Block 0.
19 (R/W)	FLT_FKEY_ERR	Flash Key Invalid. The SYSBLK_LROM_STAT.FLT_FKEY_ERR bit indicates the SDBGKEY key stored in Flash Info Block 0 is invalid.

Table 43-15: SYSBLK_LROM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	FLT_EFS_ECC	Configuration ECC Error. The `SYSBLK_LROM_STAT.FLT_EFS_ECC` bit indicates an uncorrectable ECC error was detected while reading the configuration array.	
17 (R/W)	FLT_FKTMO	Configuration/Flash Key Timeout. The `SYSBLK_LROM_STAT.FLT_FKTMO` bit indicates a Configuration/Flash initialization timeout. Either the initial configuration array read operation or the Flash Controller flash key read operation did not complete within the specified number of clocks after the deassertion of M4 core reset.	
16 (R/W)	FLT_UEXC	Unexpected Exception Fault. The `SYSBLK_LROM_STAT.FLT_UEXC` bit indicates an unexpected exception or interrupt occurred via the initial vector table at address 0x0. This results if an exception or interrupt occurs during logic boot ROM operation while VTOR = 0x0.	
8 (R/NW)	FSEREG	FSEREG Security Enable. The `SYSBLK_LROM_STAT.FSEREG` bit indicates the security feature of the device is enabled.	
6:4 (R/NW)	FKEYSTAT	Flash Key Status. Indicates the status of the customer-programmable Flash Password. This is read by the Flash Controller at deassertion of reset. See the Flash Interface Controller specification for details.	
		0	Flash Key read process is pending This code indicates the flash controller is in the process of reading the flash id and key codes following the deassertion of SYS_HWRSTb. The duration of this state is specified in the flash controller spec document. For the CM41x this is approximately 1400 CLKIN periods.
		1	Flash key is invalid The flash key is in an invalid state. Either: 1) the flash key or contrast code contained an uncorrectable ECC error, 2) or the flash contrast code did not match the required value, and was not the special case of the key and contrast code both being all blank (all 1s). The all-blank special case is indicated by KEY_UNINIT.
		2	Flash Key is unprogrammed but contrast code is valid The flash key is blank, and the flash contrast code is valid.
		3	Flash key is programmed and is valid The flash key contains a valid nonblank value (not all 1s and not all 0s), and the contrast code was valid.

Table 43-15: SYSBLK_LROM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	The flash contrast code is blank or invalid. The flash key and contrast code are blank (all 1's). No ECC error was detected. This is the uninitialized state of the flash memory.

Vector Table Base Offset Register

The SYSBLK_M0_VTOR register contains the most significant bits of the offset of the table base from the bottom of the memory map.

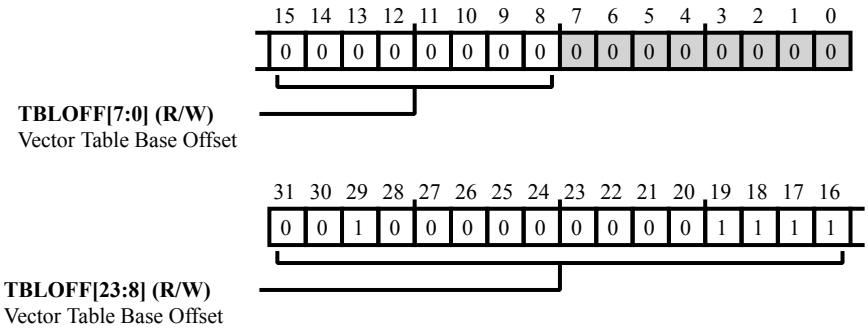


Figure 43-9: SYSBLK_M0_VTOR Register Diagram

Table 43-16: SYSBLK_M0_VTOR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:8 (R/W)	TBLOFF	Vector Table Base Offset. The SYSBLK_M0_VTOR.TBLOFF bit field contains the most significant bits of the offset of the table base from the bottom of the memory map.

Memory Self-Test Control Register

The SYSBLK_MEMST_CTL register controls various memory self-test functions.

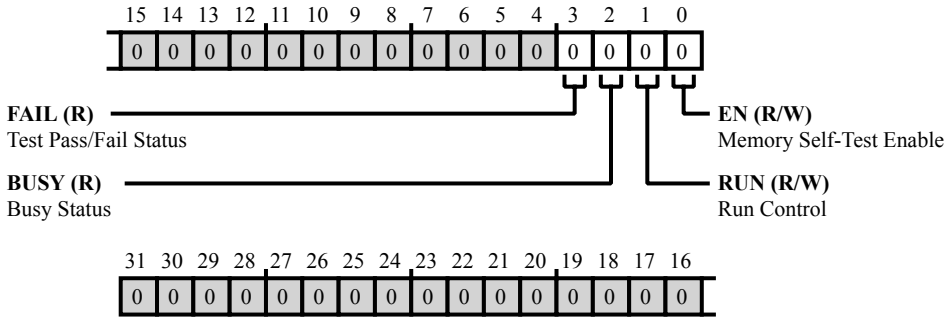


Figure 43-10: SYSBLK_MEMST_CTL Register Diagram

Table 43-17: SYSBLK_MEMST_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	FAIL	Test Pass/Fail Status. The SYSBLK_MEMST_CTL.FAIL bit indicates the Memory self-test result. This bit is not valid while SYSBLK_MEMST_CTL.BUSY is high. FAIL = 1	
		0	Self-test passed
		1	Self-test failed
2 (R/NW)	BUSY	Busy Status. The SYSBLK_MEMST_CTL.BUSY bit indicates that Memory self-test is in progress.	
1 (R/W)	RUN	Run Control. The SYSBLK_MEMST_CTL.RUN bit indicates that Memory Self-Test has started. The SYSBLK_MEMST_CTL.EN bit must be HIGH.	
0 (R/W)	EN	Memory Self-Test Enable. The SYSBLK_MEMST_CTL.EN bit enables user Memory Self-Test mode.	
		0	Memory Self-Test disabled
		1	Memory Self-Test enabled

PWM System Configuration Register

The SYSBLK_PWM_SYS_CFG register controls various functions for the PWM module.

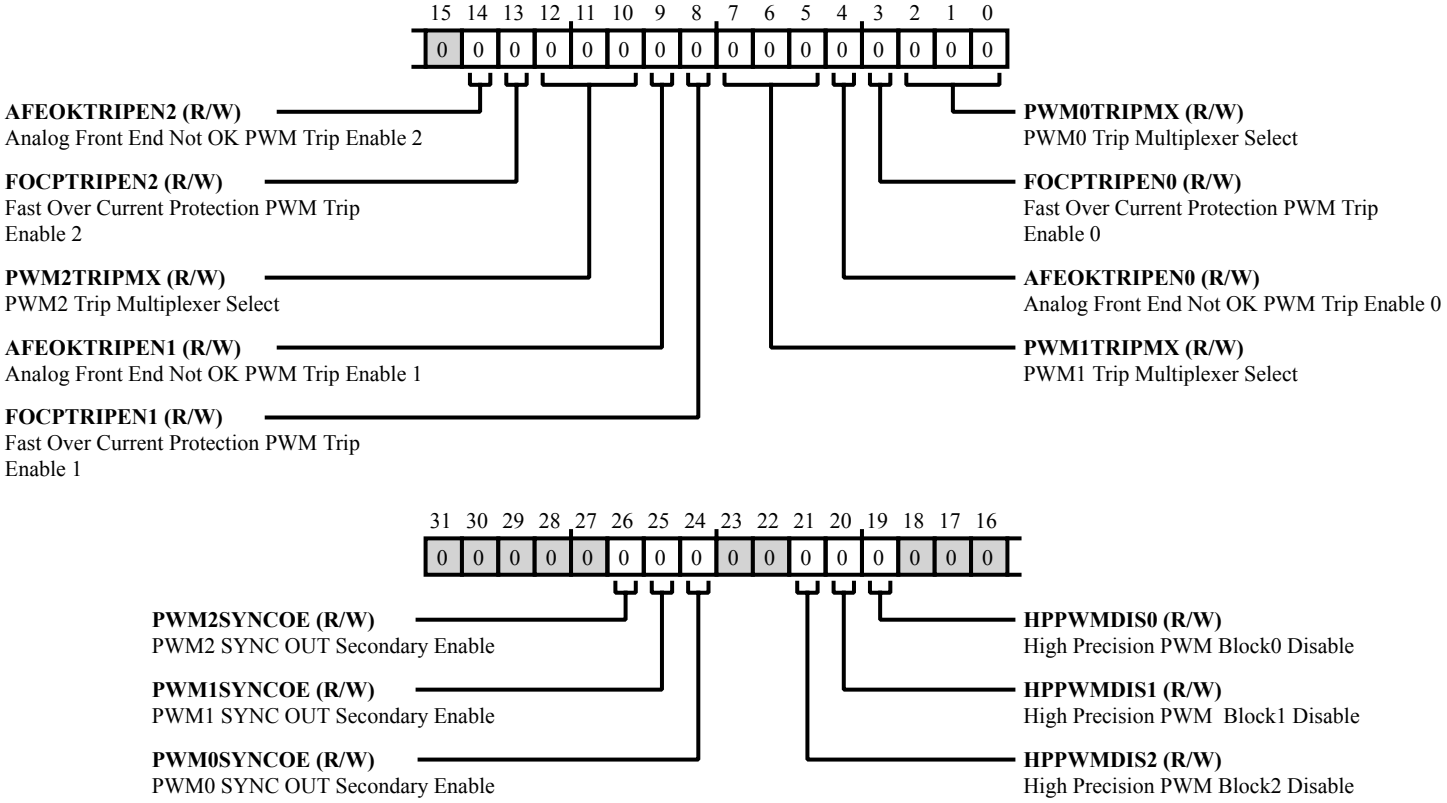


Figure 43-11: SYSBLK_PWM_SYS_CFG Register Diagram

Table 43-18: SYSBLK_PWM_SYS_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	PWM2SYNCOE	PWM2 SYNC OUT Secondary Enable. The SYSBLK_PWM_SYS_CFG.PWM2SYNCOE bit is used when the PWM is being synced by a trigger signal. This requires that PWM external sync bit is set (=1). Additionally, route the PWM's sync-pulse generator to an output pin.
25 (R/W)	PWM1SYNCOE	PWM1 SYNC OUT Secondary Enable. The SYSBLK_PWM_SYS_CFG.PWM1SYNCOE bit is used when the PWM is being synced by a trigger signal. This requires that PWM external sync bit is set (=1). Additionally, route the PWM's sync-pulse generator to an output pin.
24 (R/W)	PWM0SYNCOE	PWM0 SYNC OUT Secondary Enable. The SYSBLK_PWM_SYS_CFG.PWM0SYNCOE bit is used when the PWM is being synced by a trigger signal. This requires that the PWM external sync bit is set (=1). Additionally, route the PWM's sync-pulse generator to an output pin.

Table 43-18: SYSBLK_PWM_SYS_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	HPPWMDIS2	High Precision PWM Block2 Disable. The SYSBLK_PWM_SYS_CFG.HPPWMDIS2 bit reverts high precision PWM block 2 to medium precision.
20 (R/W)	HPPWMDIS1	High Precision PWM Block1 Disable. The SYSBLK_PWM_SYS_CFG.HPPWMDIS1 bit reverts high precision PWM block 1 to medium precision.
19 (R/W)	HPPWMDIS0	High Precision PWM Block0 Disable. The SYSBLK_PWM_SYS_CFG.HPPWMDIS0 bit reverts high precision PWM block 0 to medium precision.
14 (R/W)	AFEOKTRIPEN2	Analog Front End Not OK PWM Trip Enable 2. The SYSBLK_PWM_SYS_CFG.AFEOKTRIPEN2 bit controls if the AFE die output NOT_OK causes a PWM unit trip.
13 (R/W)	FOCPTRIPEN2	Fast Over Current Protection PWM Trip Enable 2. The SYSBLK_PWM_SYS_CFG.FOCPTRIPEN2 bit controls if the AFE die output FOCP causes a PWM unit trip.
12:10 (R/W)	PWM2TRIPMX	PWM2 Trip Multiplexer Select. The SYSBLK_PWM_SYS_CFG.PWM2TRIPMX bit controls if the trip2 chip input causes a PWM unit trip.
9 (R/W)	AFEOKTRIPEN1	Analog Front End Not OK PWM Trip Enable 1. The SYSBLK_PWM_SYS_CFG.AFEOKTRIPEN1 bit controls if the AFE die output NOT_OK causes a PWM unit trip.
8 (R/W)	FOCPTRIPEN1	Fast Over Current Protection PWM Trip Enable 1. The SYSBLK_PWM_SYS_CFG.FOCPTRIPEN1 bit controls if the AFE die output FOCP causes a PWM unit trip.
7:5 (R/W)	PWM1TRIPMX	PWM1 Trip Multiplexer Select. The SYSBLK_PWM_SYS_CFG.PWM1TRIPMX bit controls if the trip1 chip input causes a PWM unit trip.
4 (R/W)	AFEOKTRIPEN0	Analog Front End Not OK PWM Trip Enable 0. The SYSBLK_PWM_SYS_CFG.AFEOKTRIPEN0 bit controls if the AFE die output NOT_OK causes a PWM unit trip.
3 (R/W)	FOCPTRIPEN0	Fast Over Current Protection PWM Trip Enable 0. The SYSBLK_PWM_SYS_CFG.FOCPTRIPEN0 bit controls if the AFE die output FOCP causes a PWM unit trip.

Table 43-18: SYSBLK_PWM_SYS_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	PWM0TRIPMX	PWM0 Trip Multiplexer Select. The `SYSBLK_PWM_SYS_CFG.PWM0TRIPMX` bit controls if the trip0 chip input causes a PWM unit trip.

Rotary Counter Up/Down Configuration Register

The SYSBLK_ROT_UPDN_CFG register selects between rotary counter phase A or B inputs.

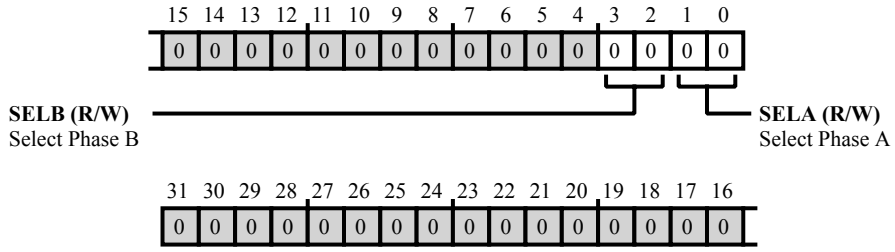


Figure 43-12: SYSBLK_ROT_UPDN_CFG Register Diagram

Table 43-19: SYSBLK_ROT_UPDN_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/W)	SELB	Select Phase B. The SYSBLK_ROT_UPDN_CFG.SELB bit field selects Rotary CNT0 Phase B input.	
		0	Select Pad as input to Rotary Counter
		1	Select Pad as input to Rotary Counter
		2	Select Trigger Slave as input to Rotary Counter
		3	Select LBA as input to Rotary Counter
1:0 (R/W)	SELA	Select Phase A. The SYSBLK_ROT_UPDN_CFG.SELA bit field selects Rotary CNT0 Phase A input.	
		0	Select Pad as input to Rotary Counter
		1	Select Pad as input to Rotary Counter
		2	Select Trigger Slave as input to Rotary Counter
		3	Select LBA as input to Rotary Counter

SINC Test Register

The SYSBLK_SINC_TEST register configures the SINC block 4 data inputs (one per MMR bit) to come from the single SPORT0 data A output. This allows a know data stream to be directed to SINC blocks for verification, development, and self-test code for safety.

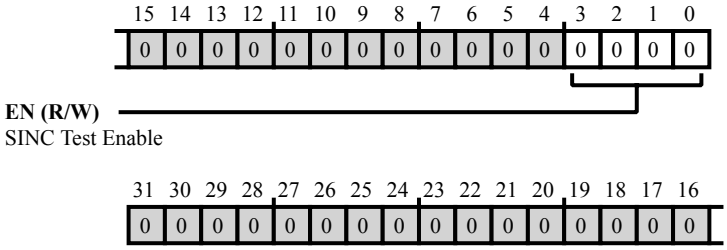


Figure 43-13: SYSBLK_SINC_TEST Register Diagram

Table 43-20: SYSBLK_SINC_TEST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	EN	SINC Test Enable. The SYSBLK_SINC_TEST.EN bit field configures the SINC block 4 data inputs (one per MMR bit) to come from the single SPORT0 data A output

Shared Interrupt 0 Status Register

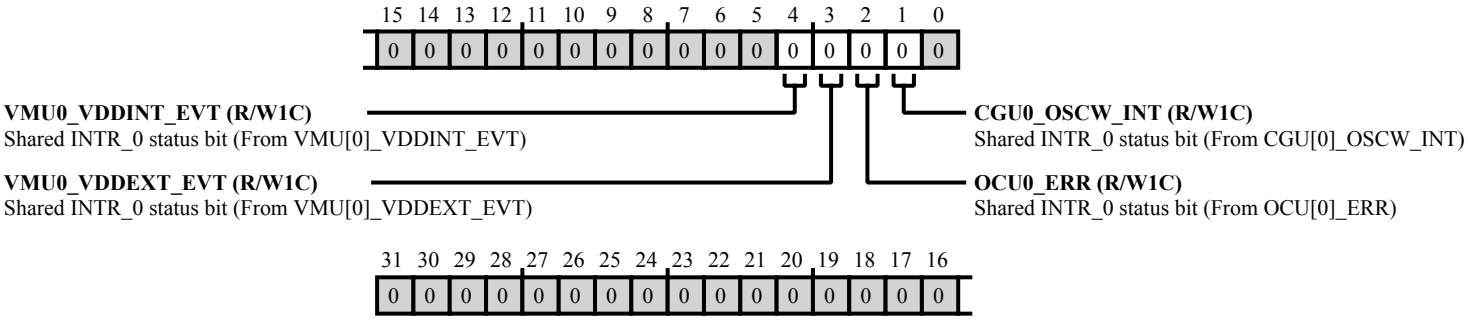


Figure 43-14: SYSBLK_SISTAT0 Register Diagram

Table 43-21: SYSBLK_SISTAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W1C)	VMU0_VDDINT_EVT	Shared INTR_0 status bit (From VMU[0]_VDDINT_EVT).
3 (R/W1C)	VMU0_VDDEXT_EVT	Shared INTR_0 status bit (From VMU[0]_VDDEXT_EVT).
2 (R/W1C)	OCU0_ERR	Shared INTR_0 status bit (From OCU[0]_ERR).
1 (R/W1C)	CGU0_OSCW_INT	Shared INTR_0 status bit (From CGU[0]_OSCW_INT).

Shared Interrupt 10 Status Register

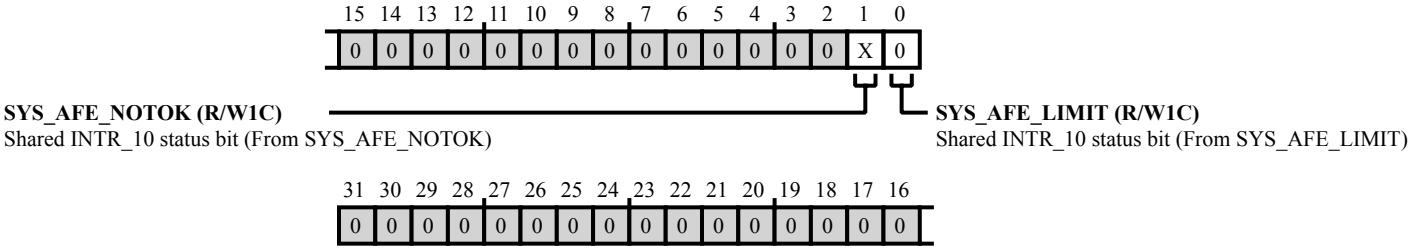


Figure 43-15: SYSBLK_SISTAT10 Register Diagram

Table 43-22: SYSBLK_SISTAT10 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	SYS_AFE_NOTOK	Shared INTR_10 status bit (From SYS_AFE_NOTOK).
0 (R/W1C)	SYS_AFE_LIMIT	Shared INTR_10 status bit (From SYS_AFE_LIMIT).

Shared Interrupt 11 Status Register

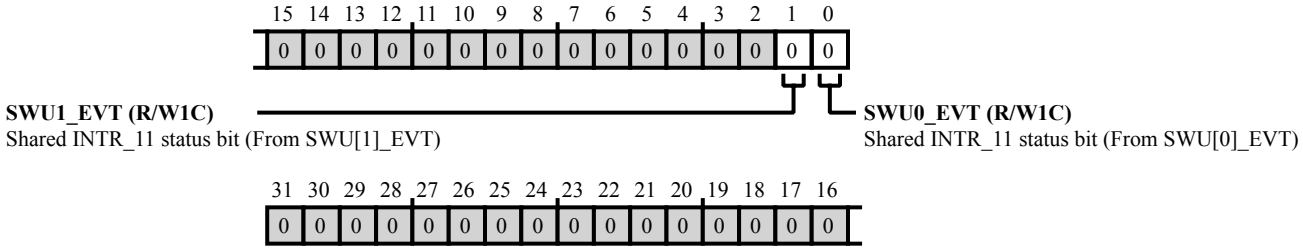


Figure 43-16: SYSBLK_SISTAT11 Register Diagram

Table 43-23: SYSBLK_SISTAT11 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	SWU1_EVT	Shared INTR_11 status bit (From SWU[1]_EVT).
0 (R/W1C)	SWU0_EVT	Shared INTR_11 status bit (From SWU[0]_EVT).

Shared Interrupt 12 Status Register

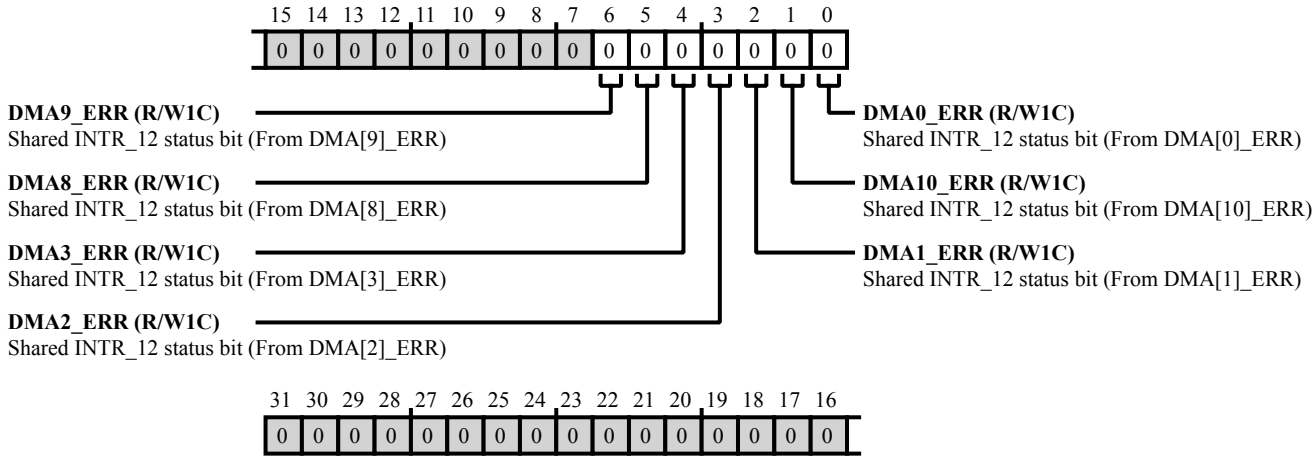


Figure 43-17: SYSBLK_SISTAT12 Register Diagram

Table 43-24: SYSBLK_SISTAT12 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	DMA9_ERR	Shared INTR_12 status bit (From DMA[9]_ERR).
5 (R/W1C)	DMA8_ERR	Shared INTR_12 status bit (From DMA[8]_ERR).
4 (R/W1C)	DMA3_ERR	Shared INTR_12 status bit (From DMA[3]_ERR).
3 (R/W1C)	DMA2_ERR	Shared INTR_12 status bit (From DMA[2]_ERR).
2 (R/W1C)	DMA1_ERR	Shared INTR_12 status bit (From DMA[1]_ERR).
1 (R/W1C)	DMA10_ERR	Shared INTR_12 status bit (From DMA[10]_ERR).
0 (R/W1C)	DMA0_ERR	Shared INTR_12 status bit (From DMA[0]_ERR).

Shared Interrupt 15 Status Register

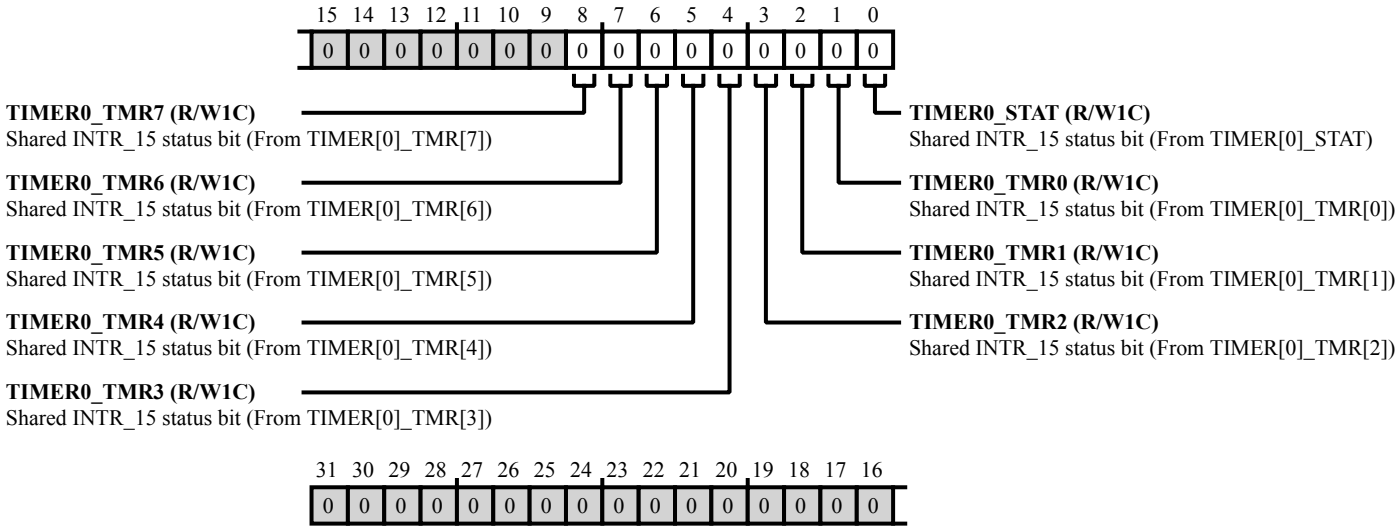


Figure 43-18: SYSBLK_SISTAT15 Register Diagram

Table 43-25: SYSBLK_SISTAT15 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	TIMER0_TMR7	Shared INTR_15 status bit (From TIMER[0]_TMR[7]).
7 (R/W1C)	TIMER0_TMR6	Shared INTR_15 status bit (From TIMER[0]_TMR[6]).
6 (R/W1C)	TIMER0_TMR5	Shared INTR_15 status bit (From TIMER[0]_TMR[5]).
5 (R/W1C)	TIMER0_TMR4	Shared INTR_15 status bit (From TIMER[0]_TMR[4]).
4 (R/W1C)	TIMER0_TMR3	Shared INTR_15 status bit (From TIMER[0]_TMR[3]).
3 (R/W1C)	TIMER0_TMR2	Shared INTR_15 status bit (From TIMER[0]_TMR[2]).
2 (R/W1C)	TIMER0_TMR1	Shared INTR_15 status bit (From TIMER[0]_TMR[1]).
1 (R/W1C)	TIMER0_TMR0	Shared INTR_15 status bit (From TIMER[0]_TMR[0]).

Table 43-25: SYSBLK_SISTAT15 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1C)	TIMER0_STAT	Shared INTR_15 status bit (From TIMER[0]_STAT).

Shared Interrupt 16 Status Register

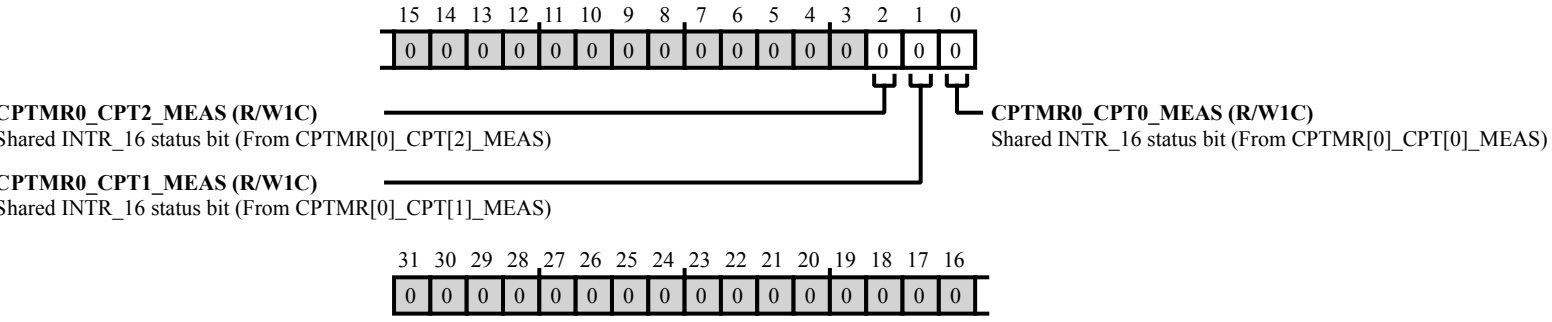


Figure 43-19: SYSBLK_SISTAT16 Register Diagram

Table 43-26: SYSBLK_SISTAT16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	CPTMR0_CPT2_MEAS	Shared INTR_16 status bit (From CPTMR[0]_CPT[2]_MEAS).
1 (R/W1C)	CPTMR0_CPT1_MEAS	Shared INTR_16 status bit (From CPTMR[0]_CPT[1]_MEAS).
0 (R/W1C)	CPTMR0_CPT0_MEAS	Shared INTR_16 status bit (From CPTMR[0]_CPT[0]_MEAS).

Shared Interrupt 17 Status Register

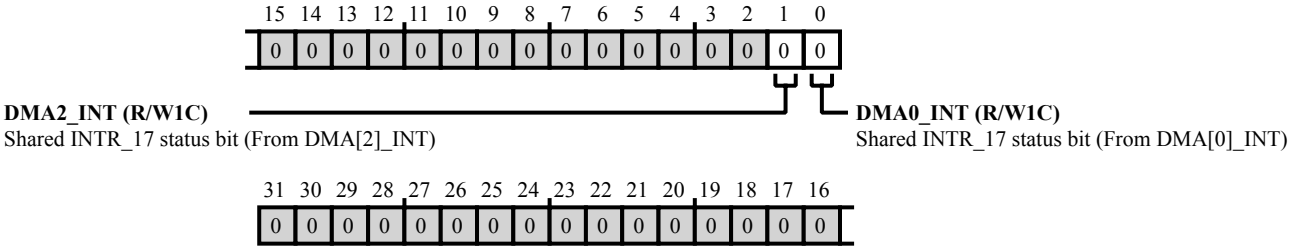


Figure 43-20: SYSBLK_SISTAT17 Register Diagram

Table 43-27: SYSBLK_SISTAT17 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	DMA2_INT	Shared INTR_17 status bit (From DMA[2]_INT).
0 (R/W1C)	DMA0_INT	Shared INTR_17 status bit (From DMA[0]_INT).

Shared Interrupt 18 Status Register

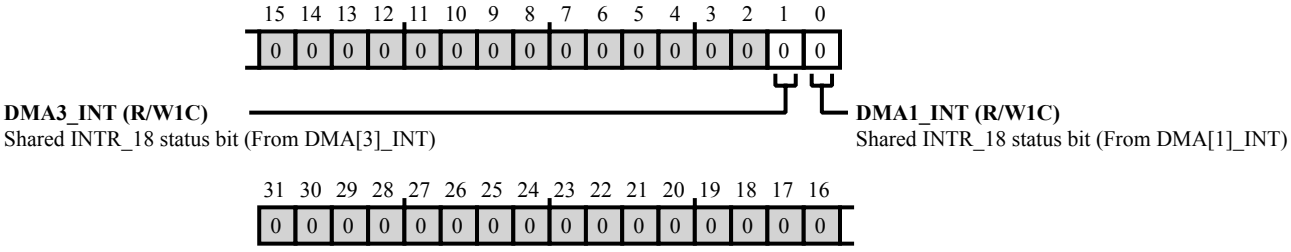


Figure 43-21: SYSBLK_SISTAT18 Register Diagram

Table 43-28: SYSBLK_SISTAT18 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	DMA3_INT	Shared INTR_18 status bit (From DMA[3]_INT).
0 (R/W1C)	DMA1_INT	Shared INTR_18 status bit (From DMA[1]_INT).

Shared Interrupt 19 Status Register

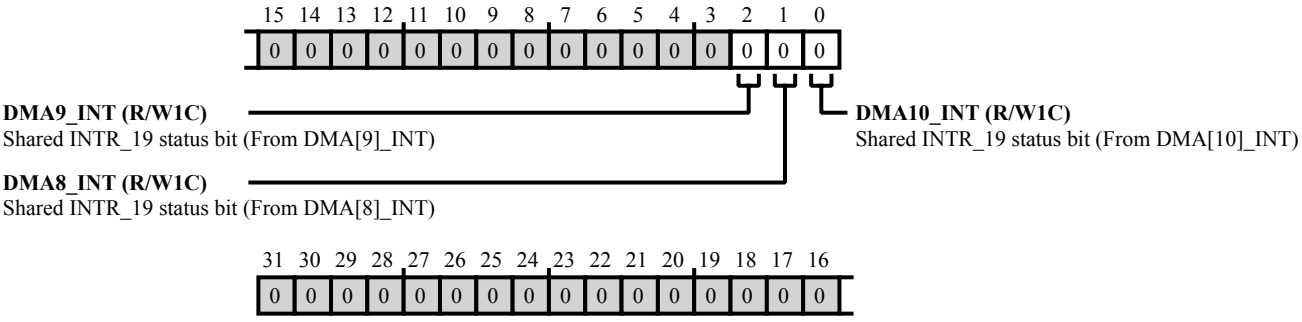


Figure 43-22: SYSBLK_SISTAT19 Register Diagram

Table 43-29: SYSBLK_SISTAT19 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	DMA9_INT	Shared INTR_19 status bit (From DMA[9]_INT).
1 (R/W1C)	DMA8_INT	Shared INTR_19 status bit (From DMA[8]_INT).
0 (R/W1C)	DMA10_INT	Shared INTR_19 status bit (From DMA[10]_INT).

Shared Interrupt 22 Status Register

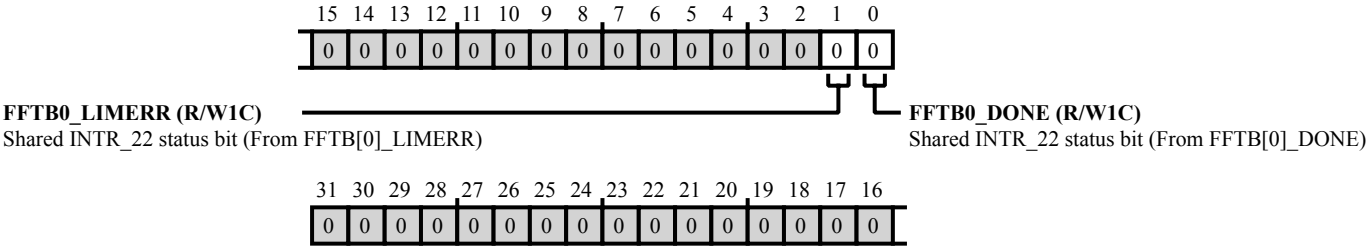


Figure 43-23: SYSBLK_SISTAT22 Register Diagram

Table 43-30: SYSBLK_SISTAT22 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	FFTB0_LIMERR	Shared INTR_22 status bit (From FFTB[0]_LIMERR).
0 (R/W1C)	FFTB0_DONE	Shared INTR_22 status bit (From FFTB[0]_DONE).

Shared Interrupt 25 Status Register

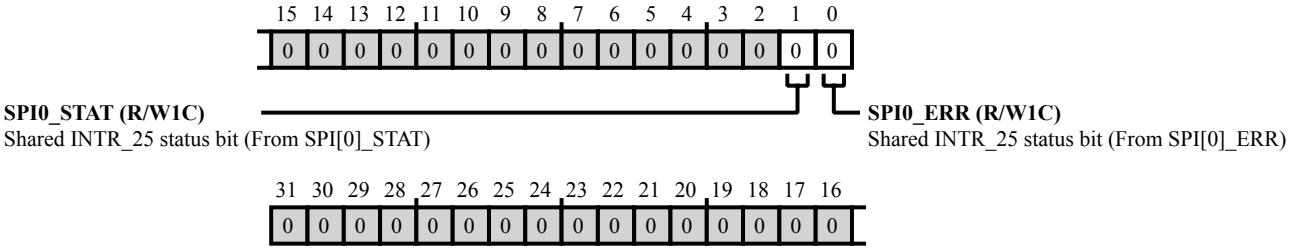


Figure 43-24: SYSBLK_SISTAT25 Register Diagram

Table 43-31: SYSBLK_SISTAT25 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	SPI0_STAT	Shared INTR_25 status bit (From SPI[0]_STAT).
0 (R/W1C)	SPI0_ERR	Shared INTR_25 status bit (From SPI[0]_ERR).

Shared Interrupt 28 Status Register

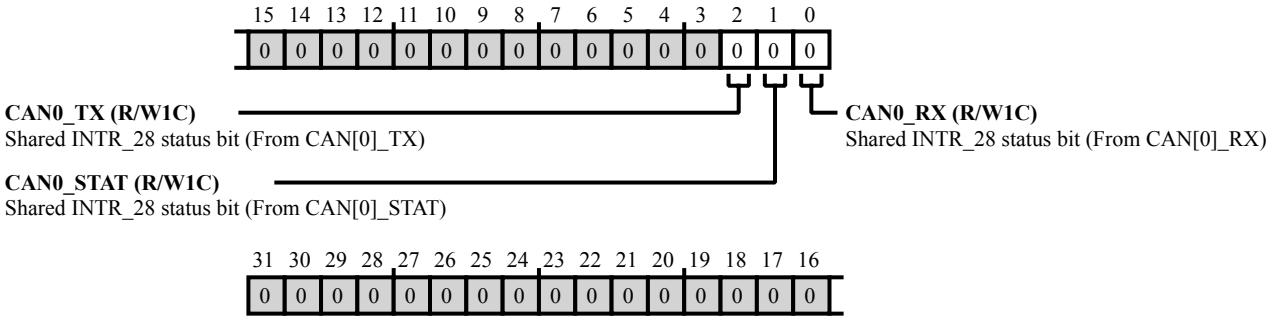


Figure 43-25: SYSBLK_SISTAT28 Register Diagram

Table 43-32: SYSBLK_SISTAT28 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	CAN0_TX	Shared INTR_28 status bit (From CAN[0]_TX).
1 (R/W1C)	CAN0_STAT	Shared INTR_28 status bit (From CAN[0]_STAT).
0 (R/W1C)	CAN0_RX	Shared INTR_28 status bit (From CAN[0]_RX).

Shared Interrupt 3 Status Register

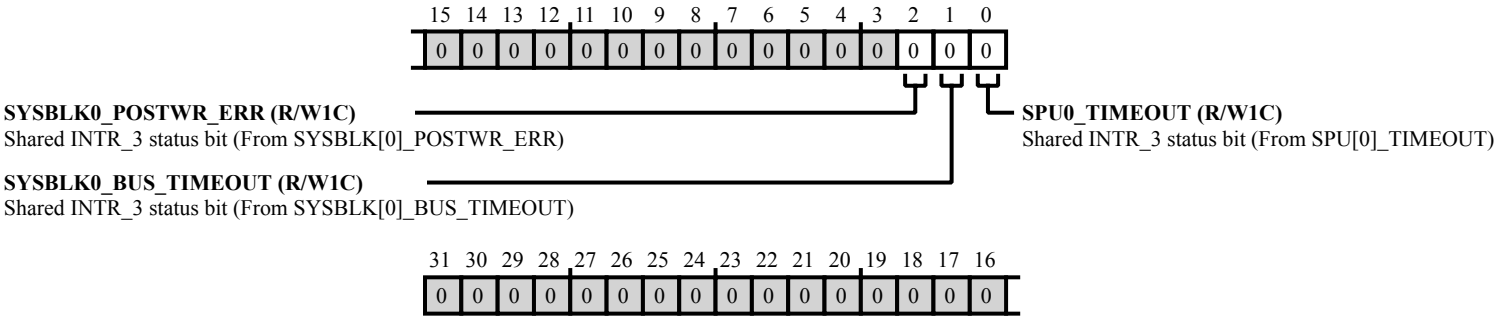


Figure 43-26: SYSBLK_SISTAT3 Register Diagram

Table 43-33: SYSBLK_SISTAT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	SYSBLK0_POSTWR_ERR	Shared INTR_3 status bit (From SYSBLK[0]_POSTWR_ERR).
1 (R/W1C)	SYSBLK0_BUS_TIME-OUT	Shared INTR_3 status bit (From SYSBLK[0]_BUS_TIMEOUT).
0 (R/W1C)	SPU0_TIMEOUT	Shared INTR_3 status bit (From SPU[0]_TIMEOUT).

Shared Interrupt 5 Status Register

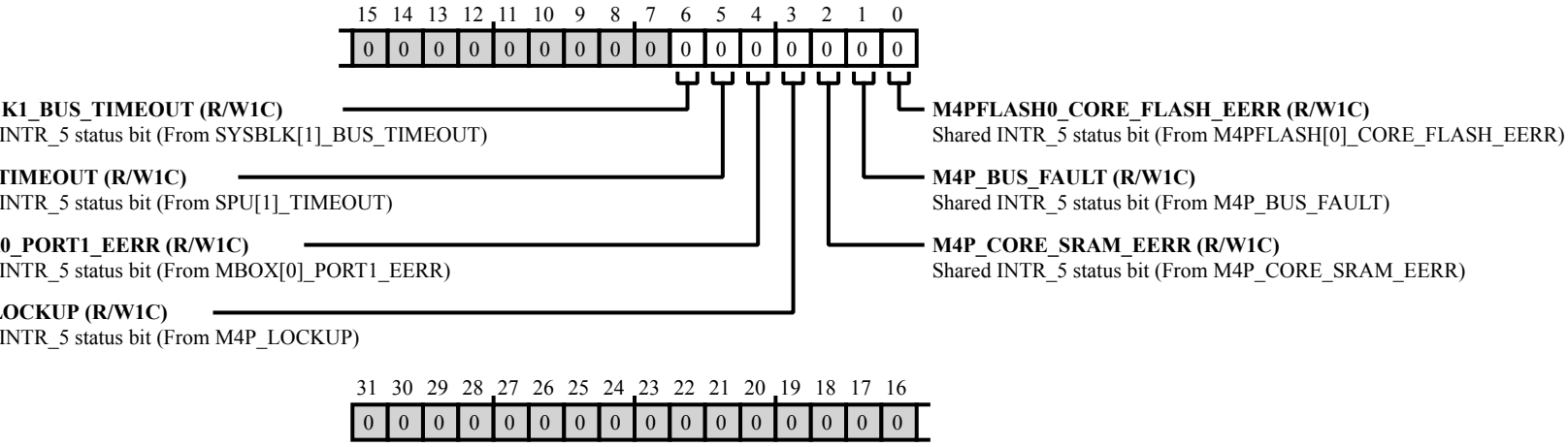


Figure 43-27: SYSBLK_SISTAT5 Register Diagram

Table 43-34: SYSBLK_SISTAT5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	SYSBLK1_BUS_TIME-OUT	Shared INTR_5 status bit (From SYSBLK[1]_BUS_TIMEOUT).
5 (R/W1C)	SPU1_TIMEOUT	Shared INTR_5 status bit (From SPU[1]_TIMEOUT).
4 (R/W1C)	MBOX0_PORT1_EERR	Shared INTR_5 status bit (From MBOX[0]_PORT1_EERR).
3 (R/W1C)	M4P_LOCKUP	Shared INTR_5 status bit (From M4P_LOCKUP).
2 (R/W1C)	M4P_CORE_SRAM_EER R	Shared INTR_5 status bit (From M4P_CORE_SRAM_EERR).
1 (R/W1C)	M4P_BUS_FAULT	Shared INTR_5 status bit (From M4P_BUS_FAULT).
0 (R/W1C)	M4PFLASH0_CORE_FLA SH_EERR	Shared INTR_5 status bit (From M4PFLASH[0]_CORE_FLASH_EERR).

Shared Interrupt 6 Status Register

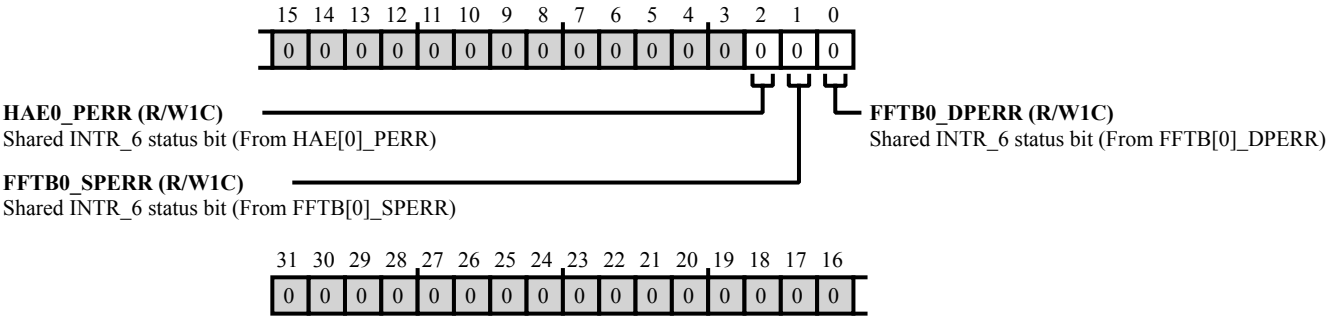


Figure 43-28: SYSBLK_SISTAT6 Register Diagram

Table 43-35: SYSBLK_SISTAT6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	HAE0_PERR	Shared INTR_6 status bit (From HAE[0]_PERR).
1 (R/W1C)	FFTB0_SPERR	Shared INTR_6 status bit (From FFTB[0]_SPERR).
0 (R/W1C)	FFTB0_DPERR	Shared INTR_6 status bit (From FFTB[0]_DPERR).

Shared Interrupt 7 Status Register

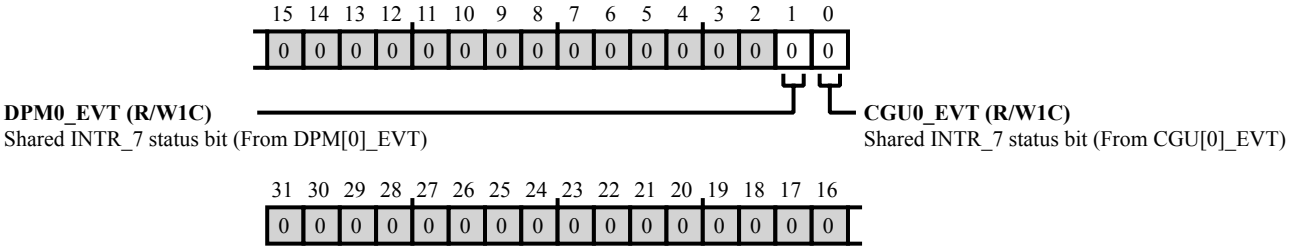


Figure 43-29: SYSBLK_SISTAT7 Register Diagram

Table 43-36: SYSBLK_SISTAT7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	DPM0_EVT	Shared INTR_7 status bit (From DPM[0]_EVT).
0 (R/W1C)	CGU0_EVT	Shared INTR_7 status bit (From CGU[0]_EVT).

Shared Interrupt 8 Status Register

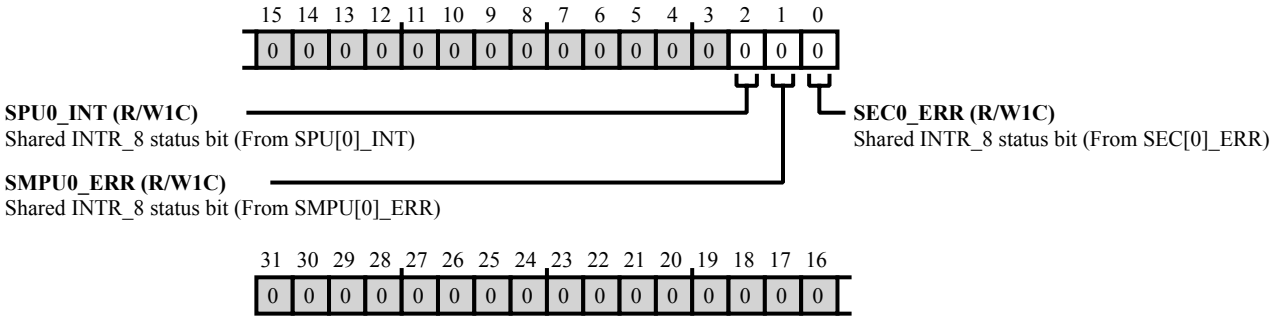


Figure 43-30: SYSBLK_SISTAT8 Register Diagram

Table 43-37: SYSBLK_SISTAT8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	SPU0_INT	Shared INTR_8 status bit (From SPU[0]_INT).
1 (R/W1C)	SMPU0_ERR	Shared INTR_8 status bit (From SMPU[0]_ERR).
0 (R/W1C)	SEC0_ERR	Shared INTR_8 status bit (From SEC[0]_ERR).

M0 SRAM ECC Register

The SYSBLK_SRAM0_ECC register allows direct R/W access of the raw ECC/DATA bits of the memory for error testing and debug. When SYSBLK_SRAM0_ECC.MAPMODE[31] is set, MAPMODE[30] determines whether ECC (SYSBLK_SRAM0_ECC.MAPMODE[30] equal one) or DATA (SYSBLK_SRAM0_ECC.MAPMODE[30] equal zero) bits are read/written. ECC generation/correction is disable in this mode and no ECC IRQs are generated for memory transactions.

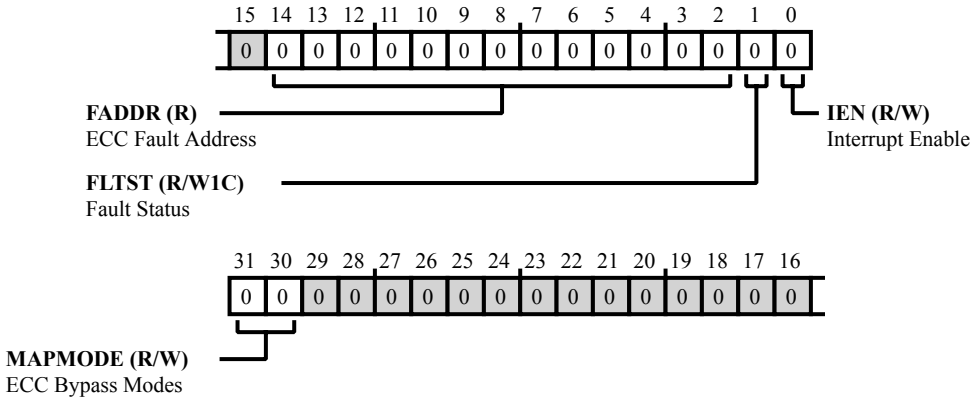


Figure 43-31: SYSBLK_SRAM0_ECC Register Diagram

Table 43-38: SYSBLK_SRAM0_ECC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	MAPMODE	ECC Bypass Modes. When SYSBLK_SRAM0_ECC.MAPMODE[31] is set, SYSBLK_SRAM0_ECC.MAPMODE[30] determines whether ECC (SYSBLK_SRAM0_ECC.MAPMODE[30] equal one) or DATA (SYSBLK_SRAM0_ECC.MAPMODE[30] equal zero) bits are read/written. ECC generation/correction is disable in this mode and no ECC IRQs are generated for memory transactions.
14:2 (R/NW)	FADDR	ECC Fault Address.
1 (R/W1C)	FLTST	Fault Status.
0 (R/W)	IEN	Interrupt Enable.

System Status Register

The SYSBLK_SYSSTAT register bits indicate the AFE status.

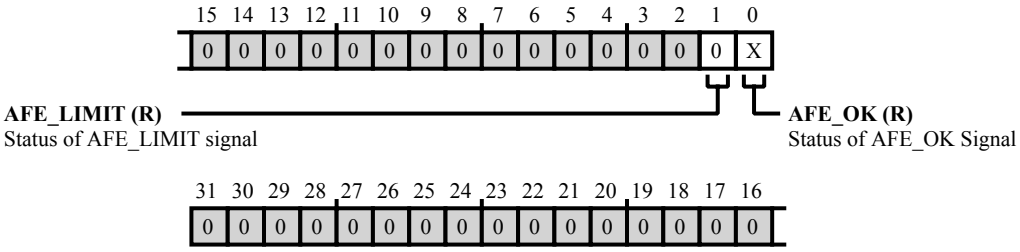


Figure 43-32: SYSBLK_SYSSTAT Register Diagram

Table 43-39: SYSBLK_SYSSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	AFE_LIMIT	Status of AFE_LIMIT signal. The SYSBLK_SYSSTAT.AFE_LIMIT bit indicates the status of AFE_LIMIT signal, indicating at least one of the AFE FOCP comparators has detected an over limit condition.	
0 (R/NW)	AFE_OK	Status of AFE_OK Signal. The SYSBLK_SYSSTAT.AFE_OK bit indicates the status of AFE_OK signal.	
		0	AFE not OK AFE not OK
		1	AFE OK

System Register

It allows the programmer to define the calibration value for the ARM Processor's SysTick timer, according to the frequency of the installed crystal and the divisor settings of the CGU and PLL. The value programmed into this register appears in the ARM SYST_CALIB read-only register.

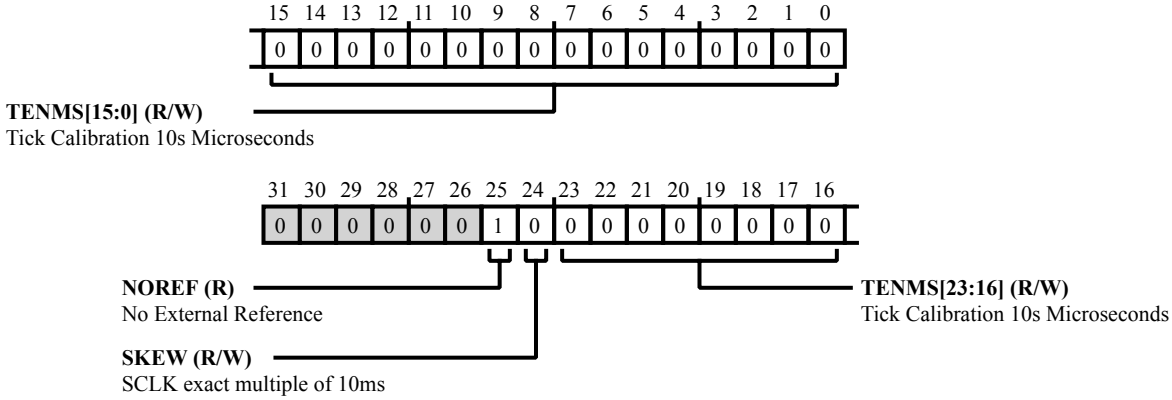


Figure 43-33: SYSBLK_SYS_STCALIB Register Diagram

Table 43-40: SYSBLK_SYS_STCALIB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	NOREF	No External Reference. The `SYSBLK_SYS_STCALIB.NOREF` indicates whether an external SysTick reference clock is implemented. In the ADI M4 Platform, no SysTick external reference clock is implemented. The processor internal reference clock is used for SYSTICK.
24 (R/W)	SKEW	SCLK exact multiple of 10ms. The `SYSBLK_SYS_STCALIB.SKEW` bit is set to 1 if the calibration value specified by `SYSBLK_SYS_STCALIB.TENMS` does not provide an exact multiple of 10ms. Otherwise, set this bit to 0. For example, the frequency of a 166.66 (6 repeating) MHz core clock corresponds to a `SYSBLK_SYS_STCALIB.TENMS` value of 1,666,666.66, which is not an exact integer, so in that case `SYSBLK_SYS_STCALIB.SKEW` is set to 1.
23:0 (R/W)	TENMS	Tick Calibration 10s Microseconds. The `SYSBLK_SYS_STCALIB.TENMS` bit is set to an integer 24-bit value usable to compute a 10ms delay from the user-programmed frequency of the M4P core clock. For example, for a 200MHz core clock, set this value to (200MHz * 10ms) = 24'd2_000_000 = 24'h1e_8480.

CM41X_M4 PADS Register Descriptions

Pads Controller (PADS) contains the following registers.

Table 43-41: CM41X_M4 PADS Register List

Name	Description
PADS_DBC[n]_CTL	Debounce Control Register(s)
PADS_DBC_PRESCALE	Debounce Prescale Register
PADS_FOCP_DIV	Fast Over Current Protection Clock Divisor Register
PADS_MONOSC_CFG	Monitor Oscillator Control Register
PADS_NVWR_RSTCTL	Non-Volatile Write Reset Control Register
PADS_PCFG0	Peripheral Configuration0 Register
PADS_PORT[n]_DS	Multi Port Drive Strength Control Register
PADS_PORT[n]_RCTL	Multi Port Pull-up/Pull-down Resistor Control Register
PADS_PORT[n]_TRIPSEL	Multi Port Trip Select Register
PADS_PORT[n]_TRIPST	Multi Port Trip State Register
PADS_VMU_CTL	Voltage Monitor Unit Control Register
PADS_VMU_TRIM	Voltage Monitor Unit Trim Register
PADS_VMU_TRIPEN	Voltage Monitor Unit Trip Enable Register

Debounce Control Register(s)

The PADS_DBC[n]_CTL register controls how the module operates.

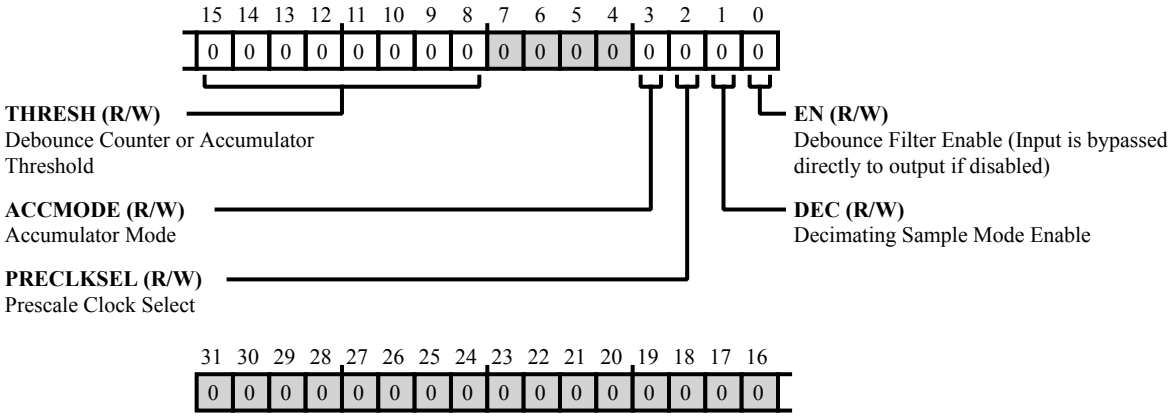


Figure 43-34: PADS_DBC[n]_CTL Register Diagram

Table 43-42: PADS_DBC[n]_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	THRESH	Debounce Counter or Accumulator Threshold. The PADS_DBC[n]_CTL.THRESH bit field contain the value THRESH + 1.	
3 (R/W)	ACCMODE	Accumulator Mode. The PADS_DBC[n]_CTL.ACCMODE bit configures the debounce filter to use accumulator mode or counter mode.	
		0	Counter mode (default)
		1	Accumulator mode
2 (R/W)	PRECLKSEL	Prescale Clock Select. The PADS_DBC[n]_CTL.PRECLKSEL bit configures the debounce filter to use the prescale clock or the system clock.	
		0	System Clock (default)
		1	Prescale clock
1 (R/W)	DEC	Decimating Sample Mode Enable. The PADS_DBC[n]_CTL.DEC bit configures the debounce filter for decimating or continuous sample mode.	
		0	Continuous mode (default)
		1	Sample mode

Table 43-42: PADS_DBC[n]_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Debounce Filter Enable (Input is bypassed directly to output if disabled). The default of the `PADS_DBC[n]_CTL.EN` bit is disabled.	
		0	Filter disabled
		1	Filter enabled

Debounce Prescale Register

The PADS_DBC_PRESCALE register contains the Debounce clock prescale divisor.

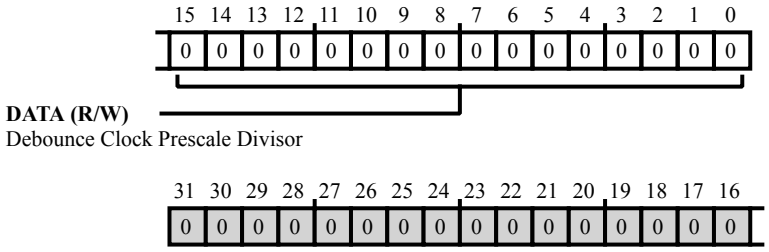


Figure 43-35: PADS_DBC_PRESCALE Register Diagram

Table 43-43: PADS_DBC_PRESCALE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DATA	Debounce Clock Prescale Divisor. The value for the PADS_DBC_PRESCALE.DATA bit field is Filter update pulse = SYSCLK/PRESCALE + 1.

Fast Over Current Protection Clock Divisor Register

The PADS_FOCP_DIV register set the frequency divide ratio.

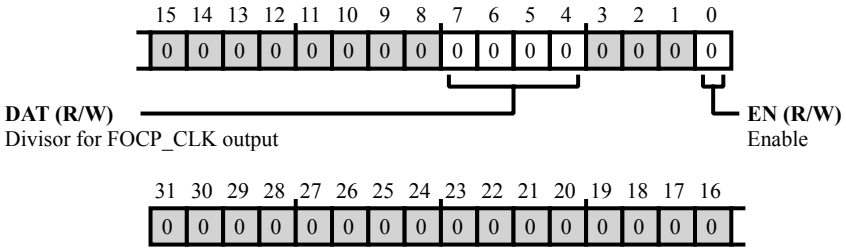


Figure 43-36: PADS_FOCP_DIV Register Diagram

Table 43-44: PADS_FOCP_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	DAT	Divisor for FOCP_CLK output. The FOCP Comparators in the AFE require a free-running clock of roughly 10 MHz. The PADS_FOCP_DIV.DAT bit field sets the frequency divide ratio of DCLK to FOCP_CLK as follows: FOCP_DIV = 1 to 15: $f_{FOCP\_CLK} = f_{DCLK} / (2 * FOCP\_DIV:DAT)$ FOCP_DIV = 0: $f_{FOCP\_CLK} = f_{DCLK} / 32$
0 (R/W)	EN	Enable. The PADS_FOCP_DIV.EN bit enables the over current protection clock divisor.

Monitor Oscillator Control Register

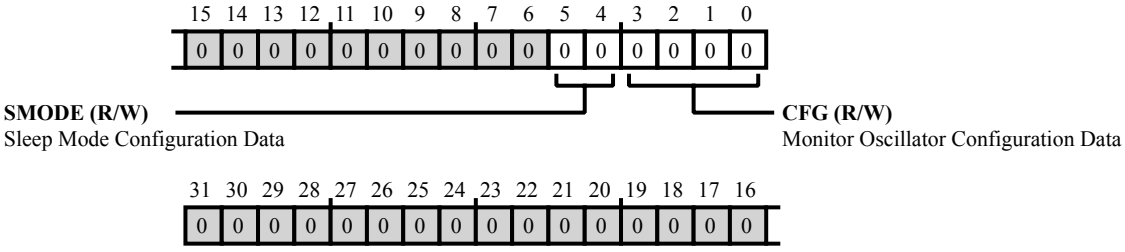


Figure 43-37: PADS_MONOSC_CFG Register Diagram

Table 43-45: PADS_MONOSC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:4 (R/W)	SMODE	Sleep Mode Configuration Data.
3:0 (R/W)	CFG	Monitor Oscillator Configuration Data.

Non-Volatile Write Reset Control Register

There are 2 basic flash unit functions, read and write. Reads occur without any special requirements. Writes (programming) of flash memory require special dedicated hardware and control. The PADS_NVWR_RSTCTL register controls the enable for the flash0/1 write control units. Without enabling the write logic block then no programming to the flash may take place, regardless of the commands written to the flash control MMR registers.

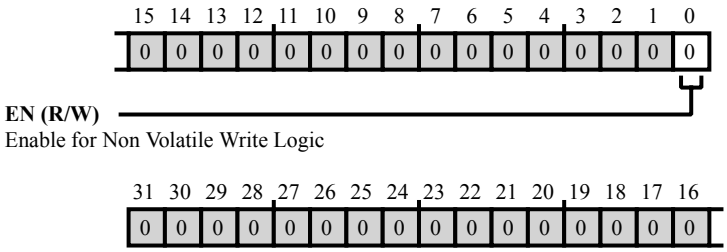


Figure 43-38: PADS_NVWR_RSTCTL Register Diagram

Table 43-46: PADS_NVWR_RSTCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	EN	Enable for Non Volatile Write Logic. The PADS_NVWR_RSTCTL.EN bit controls the enable for the flash0/1 write control units. Without enabling the write logic block then no programming to the flash may take place, regardless of the commands written to the flash control MMR registers. The default state is disabled and in reset.

Peripheral Configuration0 Register

The PADS_PCFG0 register provides several configuration options for various peripherals.

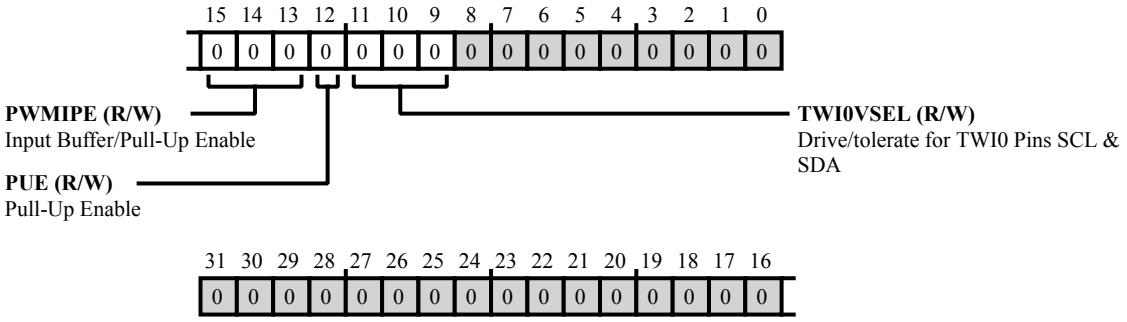


Figure 43-39: PADS_PCFG0 Register Diagram

Table 43-47: PADS_PCFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:13 (R/W)	PWMIPE	Input Buffer/Pull-Up Enable. Different PWM power topographies require different behavior before configuration. The 24 pins associated with the 3 sets of PWM units have a default state different from a generic GPIO pin. The PADS_PCFG0.PWMIPE bits allow the 3 sets of 8 pins associated with PWM units to be quickly programmed back to GPIO functionality for a given use case. Bit0 = 1 restores the defaults for pwm0 pins (PB0 to 7) Bit1 = 1 restores the defaults for pwm1 pins (PB8 to 15) Bit2 = 1 restores the defaults for pwm2 pins (PE0 to 7)	
		0	All PWM units
		1	PWM0 unit
		2	PWM0, 1 units
		3	No PWM units used (111)
12 (R/W)	PUE	Pull-Up Enable. The PADS_PCFG0.PUE bit overrides the input enable and enables the pull-up on the AFE pads.	
11:9 (R/W)	TWI0VSEL	Drive/tolerate for TWI0 Pins SCL & SDA. The PADS_PCFG0.TWI0VSEL sets the voltage requirements for the TWI_SCL and TWI_SDA pins on TWI0.	
		0	VDD_EXT = 3.3 V, VBUS_TWI = 3.3 V
		1	Reserved
		2	Reserved

Table 43-47: PADS_PCFG0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	Reserved
		4	VDD_EXT = 3.3 V, VBUS_TWI = 5 V
		5-7	Reserved

Multi Port Drive Strength Control Register

The PADS_PORT[n]_DS register increases the drive strength of PWM pins. See the product specific data sheet for more information.

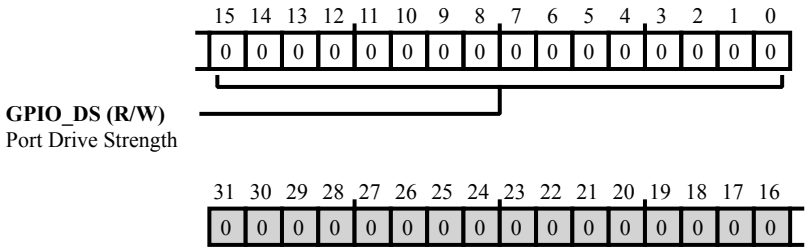


Figure 43-40: PADS_PORT[n]_DS Register Diagram

Table 43-48: PADS_PORT[n]_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	GPIO_DS	Port Drive Strength.

Multi Port Pull-up/Pull-down Resistor Control Register

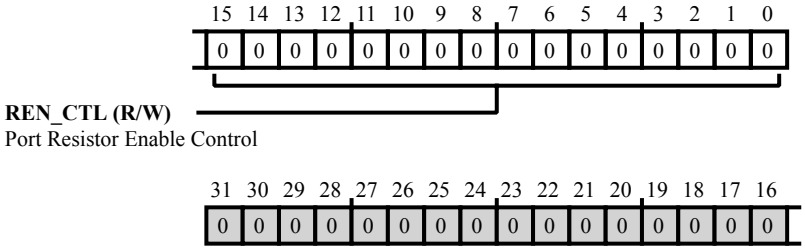


Figure 43-41: PADS_PORT[n]_RCTL Register Diagram

Table 43-49: PADS_PORT[n]_RCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	REN_CTL	Port Resistor Enable Control.

Multi Port Trip Select Register

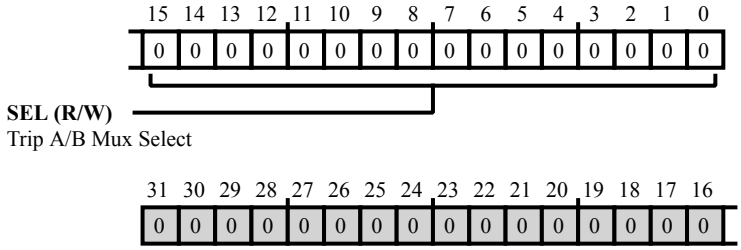


Figure 43-42: PADS_PORT[n]_TRIPSEL Register Diagram

Table 43-50: PADS_PORT[n]_TRIPSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SEL	Trip A/B Mux Select.

Multi Port Trip State Register

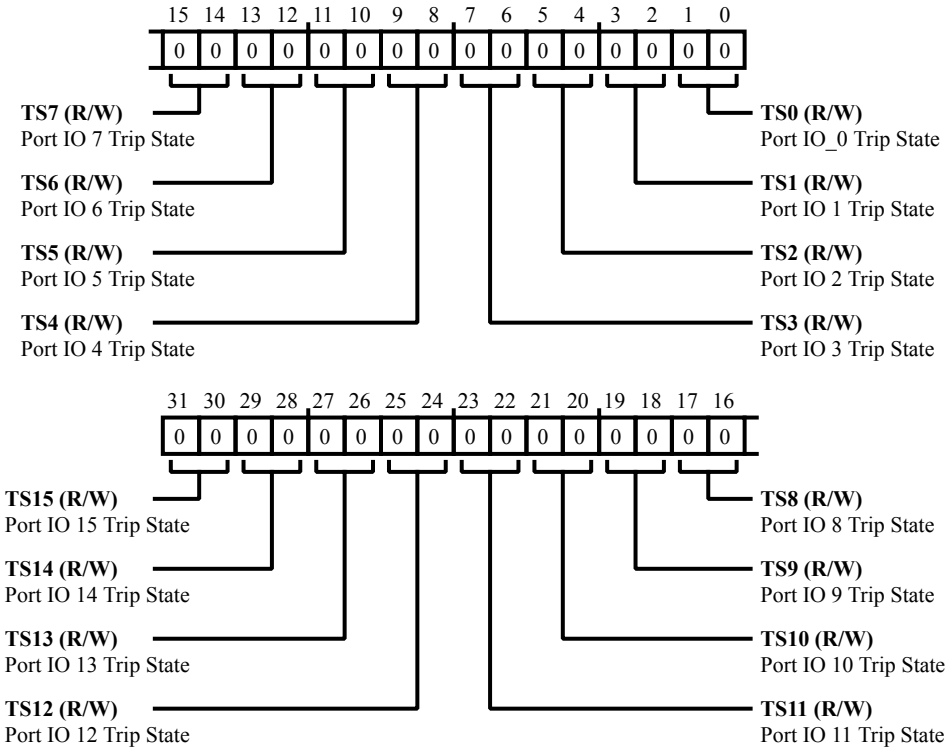


Figure 43-43: PADS_PORT[n]_TRIPST Register Diagram

Table 43-51: PADS_PORT[n]_TRIPST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	TS15	Port IO 15 Trip State.
29:28 (R/W)	TS14	Port IO 14 Trip State.
27:26 (R/W)	TS13	Port IO 13 Trip State.
25:24 (R/W)	TS12	Port IO 12 Trip State.
23:22 (R/W)	TS11	Port IO 11 Trip State.
21:20 (R/W)	TS10	Port IO 10 Trip State.

Table 43-51: PADS_PORT[n]_TRIPST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19:18 (R/W)	TS9	Port IO 9 Trip State.
17:16 (R/W)	TS8	Port IO 8 Trip State.
15:14 (R/W)	TS7	Port IO 7 Trip State.
13:12 (R/W)	TS6	Port IO 6 Trip State.
11:10 (R/W)	TS5	Port IO 5 Trip State.
9:8 (R/W)	TS4	Port IO 4 Trip State.
7:6 (R/W)	TS3	Port IO 3 Trip State.
5:4 (R/W)	TS2	Port IO 2 Trip State.
3:2 (R/W)	TS1	Port IO 1 Trip State.
1:0 (R/W)	TS0	Port IO_0 Trip State.

Voltage Monitor Unit Control Register

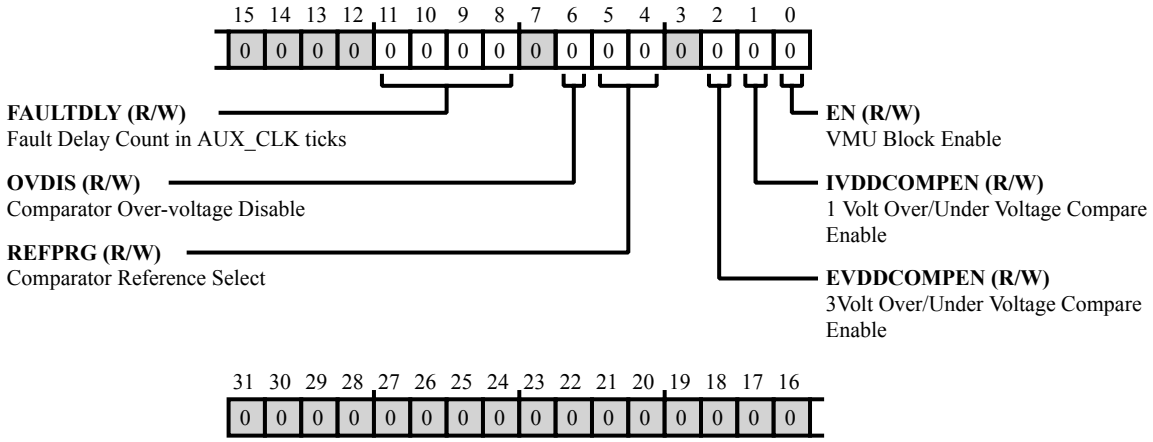


Figure 43-44: PADS_VMU_CTL Register Diagram

Table 43-52: PADS_VMU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:8 (R/W)	FAULTDLY	Fault Delay Count in AUX_CLK ticks.
6 (R/W)	OVDIS	Comparator Over-voltage Disable.
5:4 (R/W)	REFPRG	Comparator Reference Select.
2 (R/W)	EVDDCOMPEN	3Volt Over/Under Voltage Compare Enable.
1 (R/W)	IVDDCOMPEN	1 Volt Over/Under Voltage Compare Enable.
0 (R/W)	EN	VMU Block Enable.

Voltage Monitor Unit Trim Register

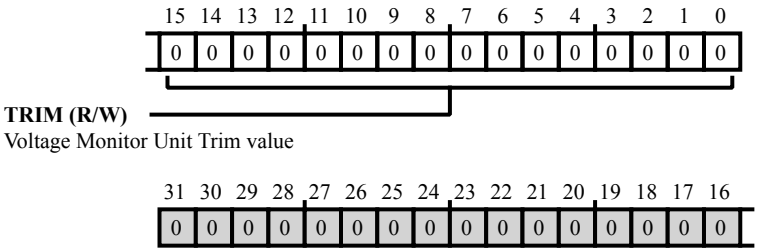


Figure 43-45: PADS_VMU_TRIM Register Diagram

Table 43-53: PADS_VMU_TRIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TRIM	Voltage Monitor Unit Trim value.

Voltage Monitor Unit Trip Enable Register

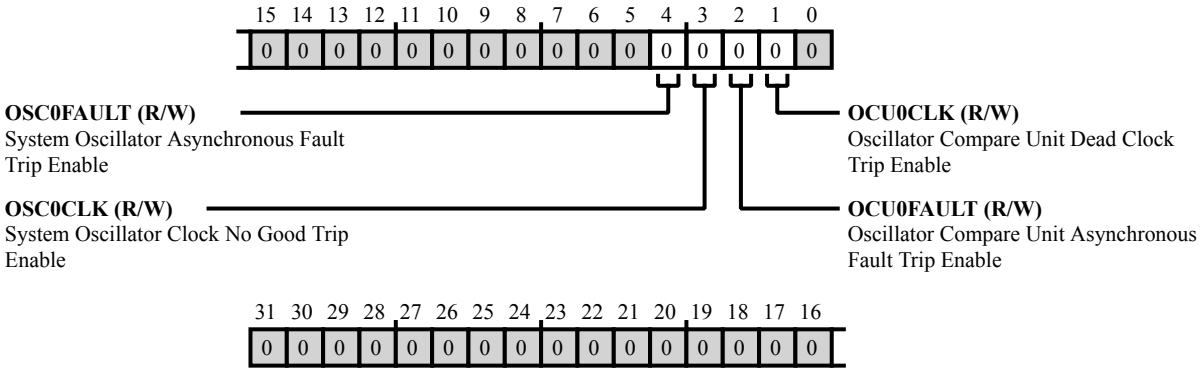


Figure 43-46: PADS_VMU_TRIPEN Register Diagram

Table 43-54: PADS_VMU_TRIPEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	OSC0FAULT	System Oscillator Asynchronous Fault Trip Enable.
3 (R/W)	OSC0CLK	System Oscillator Clock No Good Trip Enable.
2 (R/W)	OCU0FAULT	Oscillator Compare Unit Asynchronous Fault Trip Enable.
1 (R/W)	OCU0CLK	Oscillator Compare Unit Dead Clock Trip Enable.

CM41X_M0 SYSBLK Register Descriptions

Pads Controller (SYSBLK) contains the following registers.

Table 43-55: CM41X_M0 SYSBLK Register List

Name	Description
SYSBLK_SCB_RESP_CFG	SCB Response Configuration Register
SYSBLK_SCB_TIMEOUT_VALUE	SCB Timeout Value Register
SYSBLK_CLKNG_TRIPEN	Clock Not Good Trip Register
SYSBLK_DMA_MUXCTL	Peripheral DMA Multiplexer Control
SYSBLK_ENG_MODE_CFG0	Engineering Mode Configuration Register 0
SYSBLK_FAULT_TRIPEN	Fault Trip Register

Table 43-55: CM41X_M0 SYSBLK Register List (Continued)

Name	Description
SYSBLK_IRQ_LATENCY	M0 IRQ Latency Register
SYSBLK_LROM_STAT	Logic ROM Status Register
SYSBLK_M0_VTOR	Vector Table Base Offset Register
SYSBLK_MEMST_CTL	Memory Self-Test Control Register
SYSBLK_PWM_SYS_CFG	PWM System Configuration Register
SYSBLK_ROT_UPDN_CFG	Rotary Counter Up/Down Configuration Register
SYSBLK_SINC_TEST	SINC Test Register
SYSBLK_SISTAT0	Shared Interrupt 0 Status Register
SYSBLK_SISTAT10	Shared Interrupt 10 Status Register
SYSBLK_SISTAT11	Shared Interrupt 11 Status Register
SYSBLK_SISTAT12	Shared Interrupt 12 Status Register
SYSBLK_SISTAT15	Shared Interrupt 15 Status Register
SYSBLK_SISTAT16	Shared Interrupt 16 Status Register
SYSBLK_SISTAT17	Shared Interrupt 17 Status Register
SYSBLK_SISTAT18	Shared Interrupt 18 Status Register
SYSBLK_SISTAT19	Shared Interrupt 19 Status Register
SYSBLK_SISTAT22	Shared Interrupt 22 Status Register
SYSBLK_SISTAT25	Shared Interrupt 25 Status Register
SYSBLK_SISTAT28	Shared Interrupt 28 Status Register
SYSBLK_SISTAT3	Shared Interrupt 3 Status Register
SYSBLK_SISTAT5	Shared Interrupt 5 Status Register
SYSBLK_SISTAT6	Shared Interrupt 6 Status Register
SYSBLK_SISTAT7	Shared Interrupt 7 Status Register
SYSBLK_SISTAT8	Shared Interrupt 8 Status Register
SYSBLK_SRAM0_ECC	M0 SRAM ECC Register
SYSBLK_SYSSTAT	System Status Register
SYSBLK_SYS_STCALIB	System Register

SCB Response Configuration Register

The SYSBLK_SCB_RESP_CFG register provides SCB bus monitoring capabilities from the ARM Cortex-M4 system entering the main system scoreboard (SCB).

If the SYSBLK_SCB_RESP_CFG.TOENA bit is enabled the block checks the response time from the AHB block. If the response time exceeds the timeout value (configured in the SYSBLK_SCB_TIMEOUT_VALUE register) this block generates a default error response back to the ARM Cortex-M4 system and indicates an error.

Accessing the SCB from the ARM Cortex-M4 through the system scoreboard may result in a minor multi-cycle stall. The timeout is a safety feature that ensures the ARM Cortex-M4 system has a known maximum response time to a SCB system access. Note that system MMR accesses are handled on a separate peripheral bus chain and use different timeout hardware to monitor those accesses.

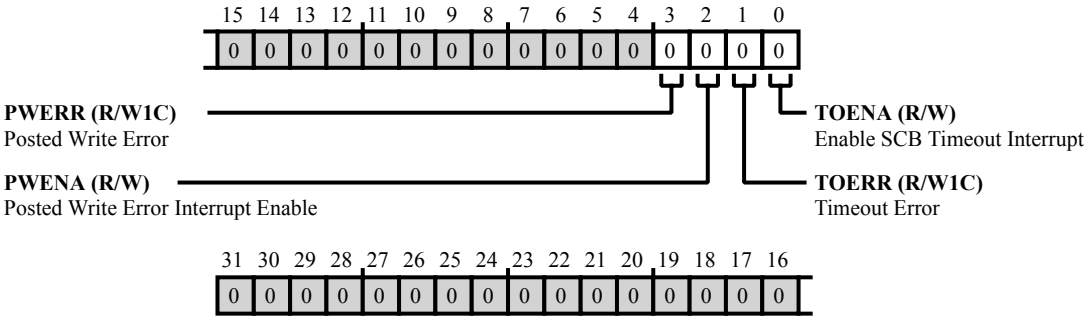


Figure 43-47: SYSBLK_SCB_RESP_CFG Register Diagram

Table 43-56: SYSBLK_SCB_RESP_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W1C)	PWERR	Posted Write Error. The SYSBLK_SCB_RESP_CFG.PWERR bit indicates an error has occurred on a previously executed Posted Write to the system fabric. Since writes were posted, this error is not associated with the interrupted instruction.	
		0	No error
		1	Error occurred
2 (R/W)	PWENA	Posted Write Error Interrupt Enable. The SYSBLK_SCB_RESP_CFG.PWENA bit enables read/write error indications to the system fabric. The ARM M0 process does not wait for the read/write to complete before executing the subsequent instructions. Any errors caused by the read/write set the PWERR bit and cause an inexact interrupt to the ARM processor.	
		0	Interrupt disabled
		1	Interrupt enabled

Table 43-56: SYSBLK_SCB_RESP_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	TOERR	Timeout Error. The SYSBLK_SCB_RESP_CFG.TOERR indicates that the SCB bus has timed out.	
		0	No error
		1	Error occurred
0 (R/W)	TOENA	Enable SCB Timeout Interrupt. The SYSBLK_SCB_RESP_CFG.TOENA bit enables the the SCB timeout interrupt. When enabled, the value in the SYSBLK_SCB_TIMEOUT_VALUE register is loaded into a counter at the start of a each new SCB transaction. The counter decrements once for each cycle that the SCB bus is not ready (stalled). If the counter decrements to the value one, an error interrupt request is generated.	
		0	Disable interrupt
		1	Enable interrupt

SCB Timeout Value Register

The SYSBLK_SCB_TIMEOUT_VALUE register contains the timeout value. This value is loaded into a counter at the start of each new SCB transaction. The counter decrements once for each cycle that the SCB bus is not ready (stalled).

If the counter decrements to the value =1, an error interrupt request is produced. If DATA =0, no error produced. If DATA =1, an error is produced for any SCB bus transaction.

The Timeout Value should be set to a value larger than the number of cycles that it takes for the slowest SCB transaction to complete to prevent spurious IRQs.

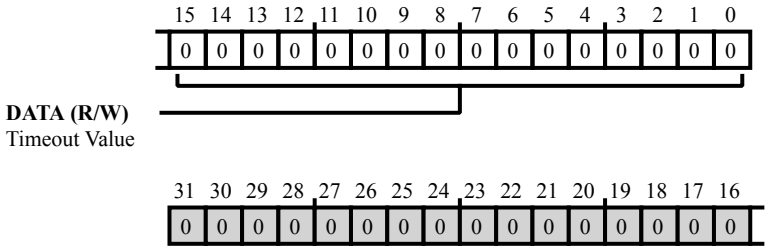


Figure 43-48: SYSBLK_SCB_TIMEOUT_VALUE Register Diagram

Table 43-57: SYSBLK_SCB_TIMEOUT_VALUE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DATA	Timeout Value. The SYSBLK_SCB_TIMEOUT_VALUE.DATA bit field contains the timeout value. The Timeout value should be set to a value larger than the number of cycles that it takes for the slowest SCB transaction to complete to prevent spurious IRQs.

Clock Not Good Trip Register

The SYSBLK_CLKNG_TRIPEN register (OCU) has 2 error outputs for clock monitoring. These 2 control bits allow the OCU0 error outputs to cause the safety pin-trip feature to activate.

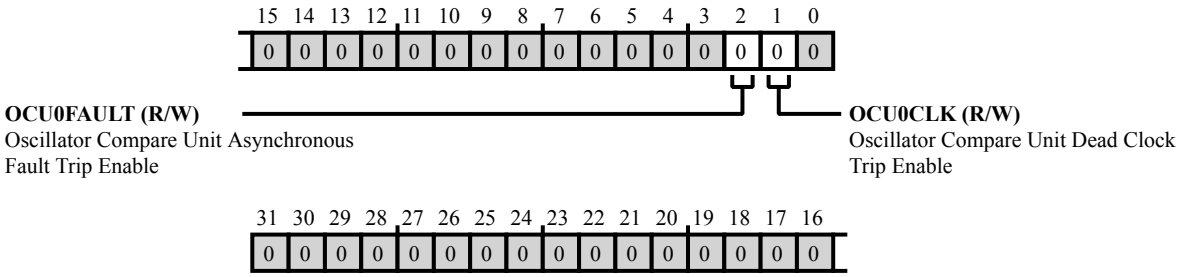


Figure 43-49: SYSBLK_CLKNG_TRIPEN Register Diagram

Table 43-58: SYSBLK_CLKNG_TRIPEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W)	OCU0FAULT	Oscillator Compare Unit Asynchronous Fault Trip Enable.
1 (R/W)	OCU0CLK	Oscillator Compare Unit Dead Clock Trip Enable.

Peripheral DMA Multiplexer Control

The SYSBLK_DMA_MUXCTL register allows peripherals to share pins and the same DMA ports. This sharing is not "in-parallel", users must configure the MDMA blocks for the desired operating peripherals. DMA channels 6-11 need to be configured to support the desired user peripheral.

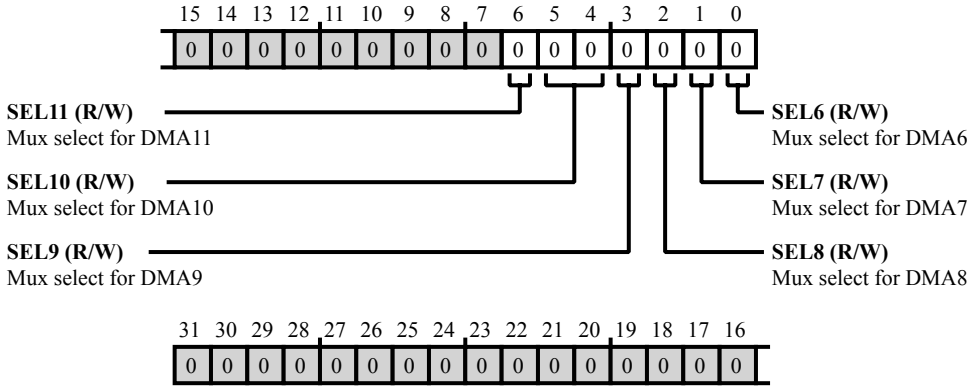


Figure 43-50: SYSBLK_DMA_MUXCTL Register Diagram

Table 43-59: SYSBLK_DMA_MUXCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SEL11	Mux select for DMA11.	
		0	Select SPORT[0] B_DMA port
		1	Select UART[4] RXDMA port
5:4 (R/W)	SEL10	Mux select for DMA10.	
		0	Select HAE[0] RXDMA_CH1 port
		1	Select SPORT[0] A_DMA port
		2	Select UART[4] TXDMA port
3 (R/W)	SEL9	Mux select for DMA9.	
		0	Select HAE[0] TXDMA port
		1	Select UART[3] RXDMA port
2 (R/W)	SEL8	Mux select for DMA8.	
		0	Select HAE[0] RXDMA_CH0 port
		1	Select UART[3] TXDMA port
1 (R/W)	SEL7	Mux select for DMA7.	
		0	Select SPI[1] RXDMA port
		1	Select UART[2] RXDMA port

Table 43-59: SYSBLK_DMA_MUXCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	SEL6	Mux select for DMA6.	
		0	Select SPI[1] TXDMA port
		1	Select UART[2] TXDMA port

Engineering Mode Configuration Register 0

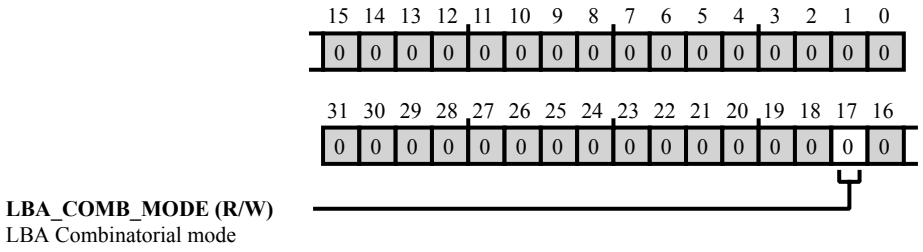


Figure 43-51: SYSBLK_ENG_MODE_CFG0 Register Diagram

Table 43-60: SYSBLK_ENG_MODE_CFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17 (R/W)	LBA_COMB_MODE	LBA Combinatorial mode.

Fault Trip Register

The SYSBLK_FAULT_TRIPEN register controls various Trip functions of the Oscillator Comparator Unit (OCU). See the OCU chapter for complete information.

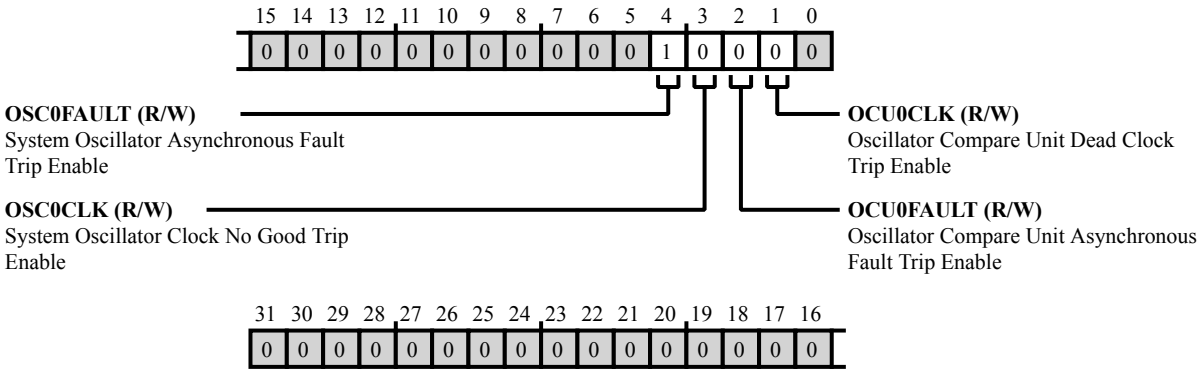


Figure 43-52: SYSBLK_FAULT_TRIPEN Register Diagram

Table 43-61: SYSBLK_FAULT_TRIPEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	OSC0FAULT	System Oscillator Asynchronous Fault Trip Enable. The SYSBLK_FAULT_TRIPEN.OSC0FAULT bit enables asserting the SYS_FAULTb pin asynchronously on OSC_WDOG detection of a clock fault, as asserted on the OSC_WDOG's 'fault' pin. Note that this enable is true by default at power-on. This allows detection of dead-clock or wrong-harmonic-mode faults on SYS_CLKIN0 to be signaled on the SYS_FAULTb pin at power-on.
3 (R/W)	OSC0CLK	System Oscillator Clock No Good Trip Enable. The SYSBLK_FAULT_TRIPEN.OSC0CLK bit enables asserting the SYS_FAULTb pin asynchronously on assertion of the OSC_WDOG clk_not_good output signal. clk_not_good can be asserted by the OSC_WDOG approximately 1 ms after a fault detection.
2 (R/W)	OCU0FAULT	Oscillator Compare Unit Asynchronous Fault Trip Enable. The SYSBLK_FAULT_TRIPEN.OCU0FAULT bit enables asserting the SYS_FAULTb pin asynchronously on OCU0 detection of either a clock frequency error or a dead clock (OCU port ocu_fault_async).
1 (R/W)	OCU0CLK	Oscillator Compare Unit Dead Clock Trip Enable. The SYSBLK_FAULT_TRIPEN.OCU0CLK bit enables asserting the SYS_FAULTb pin asynchronously on OCU0 detection of a dead SYSCLK (OCU port ocu_dead_clock).

M0 IRQ Latency Register

The SYSBLK_IRQ_LATENCY register configures the wait state memory system.

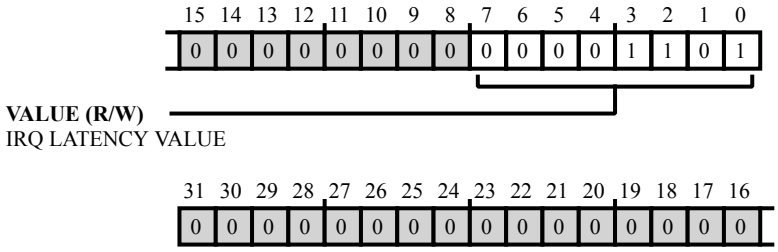


Figure 43-53: SYSBLK_IRQ_LATENCY Register Diagram

Table 43-62: SYSBLK_IRQ_LATENCY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	IRQ LATENCY VALUE. The SYSBLK_IRQ_LATENCY.VALUE bit field configures the wait state memory system. For zero jitter in a zero wait state memory system, set this bus to at least a decimal value of 13.

Logic ROM Status Register

The SYSBLK_LROM_STAT register provides indications of various errors in logic ROM boot code.

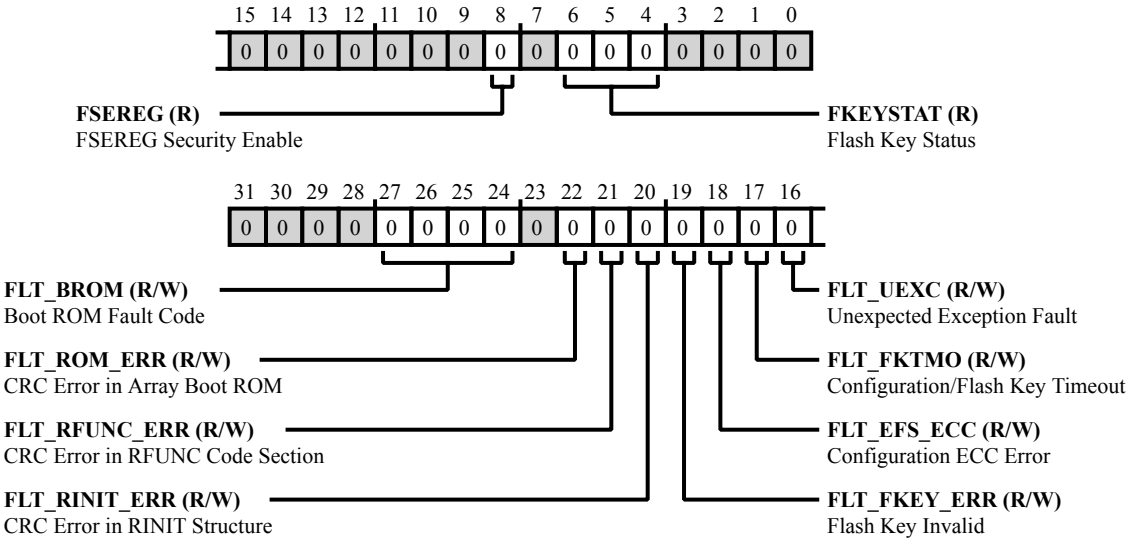


Figure 43-54: SYSBLK_LROM_STAT Register Diagram

Table 43-63: SYSBLK_LROM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:24 (R/W)	FLT_BROM	Boot ROM Fault Code. The SYSBLK_LROM_STAT.FLT_BROM bit indicates a Boot ROM fault code.
22 (R/W)	FLT_ROM_ERR	CRC Error in Array Boot ROM. The SYSBLK_LROM_STAT.FLT_ROM_ERR bit indicates that the logic ROM code detected a CRC error was detected in the Array Boot ROM. The Array Boot ROM code was not executed.
21 (R/W)	FLT_RFUNC_ERR	CRC Error in RFUNC Code Section. The SYSBLK_LROM_STAT.FLT_RFUNC_ERR bit indicates a CRC error was detected in the RFUNC structure described by RFUNC_PTR and RFUNC_CSIZ in Flash Info Block 0.
20 (R/W)	FLT_RINIT_ERR	CRC Error in RINIT Structure. The SYSBLK_LROM_STAT.FLT_RINIT_ERR bit indicates a CRC error was detected in the RINIT structure described by RINIT_PTR and RINIT_CSIZ in Flash Info Block 0.
19 (R/W)	FLT_FKEY_ERR	Flash Key Invalid. The SYSBLK_LROM_STAT.FLT_FKEY_ERR bit indicates the SDBGKEY key stored in Flash Info Block 0 is invalid.

Table 43-63: SYSBLK_LROM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	FLT_EFS_ECC	Configuration ECC Error. The `SYSBLK_LROM_STAT.FLT_EFS_ECC` bit indicates an uncorrectable ECC error was detected while reading the configuration array.	
17 (R/W)	FLT_FKTMO	Configuration/Flash Key Timeout. The `SYSBLK_LROM_STAT.FLT_FKTMO` bit indicates a Configuration/Flash initialization timeout. Either the initial configuration array read operation or the Flash Controller flash key read operation did not complete within the specified number of clocks after the deassertion of M4 core reset.	
16 (R/W)	FLT_UEXC	Unexpected Exception Fault. The `SYSBLK_LROM_STAT.FLT_UEXC` bit indicates an unexpected exception or interrupt occurred via the initial vector table at address 0x0. This results if an exception or interrupt occurs during logic boot ROM operation while VTOR = 0x0.	
8 (R/NW)	FSEREG	FSEREG Security Enable. The `SYSBLK_LROM_STAT.FSEREG` bit indicates the security feature of the device is enabled.	
6:4 (R/NW)	FKEYSTAT	Flash Key Status. Indicates the status of the customer-programmable Flash Password. This is read by the Flash Controller at deassertion of reset. See the Flash Interface Controller specification for details.	
		0	Flash Key read process is pending This code indicates the flash controller is in the process of reading the flash id and key codes following the deassertion of SYS_HWRSTb. The duration of this state is specified in the flash controller spec document. For the CM41x this is approximately 1400 CLKIN periods.
		1	Flash key is invalid The flash key is in an invalid state. Either: 1) the flash key or contrast code contained an uncorrectable ECC error, 2) or the flash contrast code did not match the required value, and was not the special case of the key and contrast code both being all blank (all 1s). The all-blank special case is indicated by KEY_UNINIT.
		2	Flash Key is unprogrammed but contrast code is valid The flash key is blank, and the flash contrast code is valid.
		3	Flash key is programmed and is valid The flash key contains a valid nonblank value (not all 1s and not all 0s), and the contrast code was valid.

Table 43-63: SYSBLK_LROM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	The flash contrast code is blank or invalid. The flash key and contrast code are blank (all 1's). No ECC error was detected. This is the uninitialized state of the flash memory.

Vector Table Base Offset Register

The SYSBLK_M0_VTOR register contains the most significant bits of the offset of the table base from the bottom of the memory map.

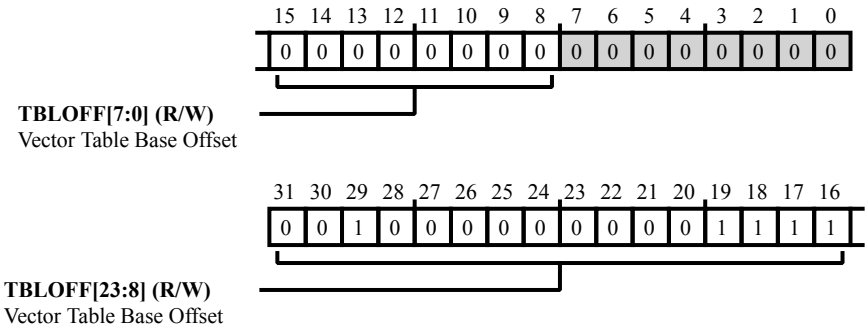


Figure 43-55: SYSBLK_M0_VTOR Register Diagram

Table 43-64: SYSBLK_M0_VTOR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:8 (R/W)	TBLOFF	Vector Table Base Offset. The SYSBLK_M0_VTOR.TBLOFF bit field contains the most significant bits of the offset of the table base from the bottom of the memory map.

Memory Self-Test Control Register

The SYSBLK_MEMST_CTL register controls various memory self-test functions.

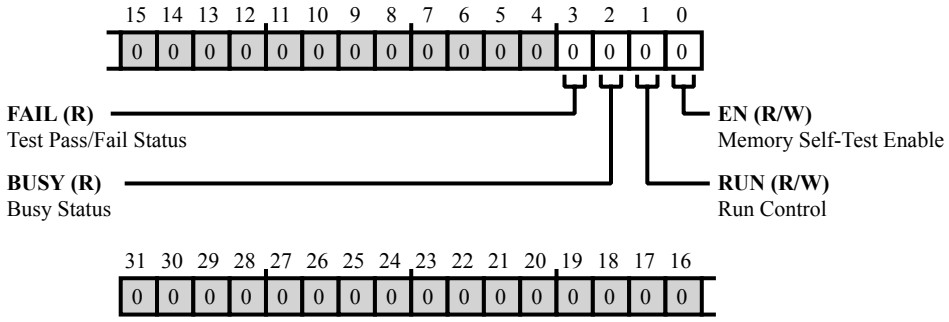


Figure 43-56: SYSBLK_MEMST_CTL Register Diagram

Table 43-65: SYSBLK_MEMST_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	FAIL	Test Pass/Fail Status. The SYSBLK_MEMST_CTL.FAIL bit indicates the Memory self-test result. This bit is not valid while SYSBLK_MEMST_CTL.BUSY is high. FAIL = 1	
		0	Self-test passed
		1	Self-test failed
2 (R/NW)	BUSY	Busy Status. The SYSBLK_MEMST_CTL.BUSY bit indicates that Memory self-test is in progress.	
1 (R/W)	RUN	Run Control. The SYSBLK_MEMST_CTL.RUN bit indicates that Memory Self-Test has started. The SYSBLK_MEMST_CTL.EN bit must be HIGH.	
0 (R/W)	EN	Memory Self-Test Enable. The SYSBLK_MEMST_CTL.EN bit enables user Memory Self-Test mode.	
		0	Memory Self-Test disabled
		1	Memory Self-Test enabled

PWM System Configuration Register

The SYSBLK_PWM_SYS_CFG register controls various functions for the PWM module.

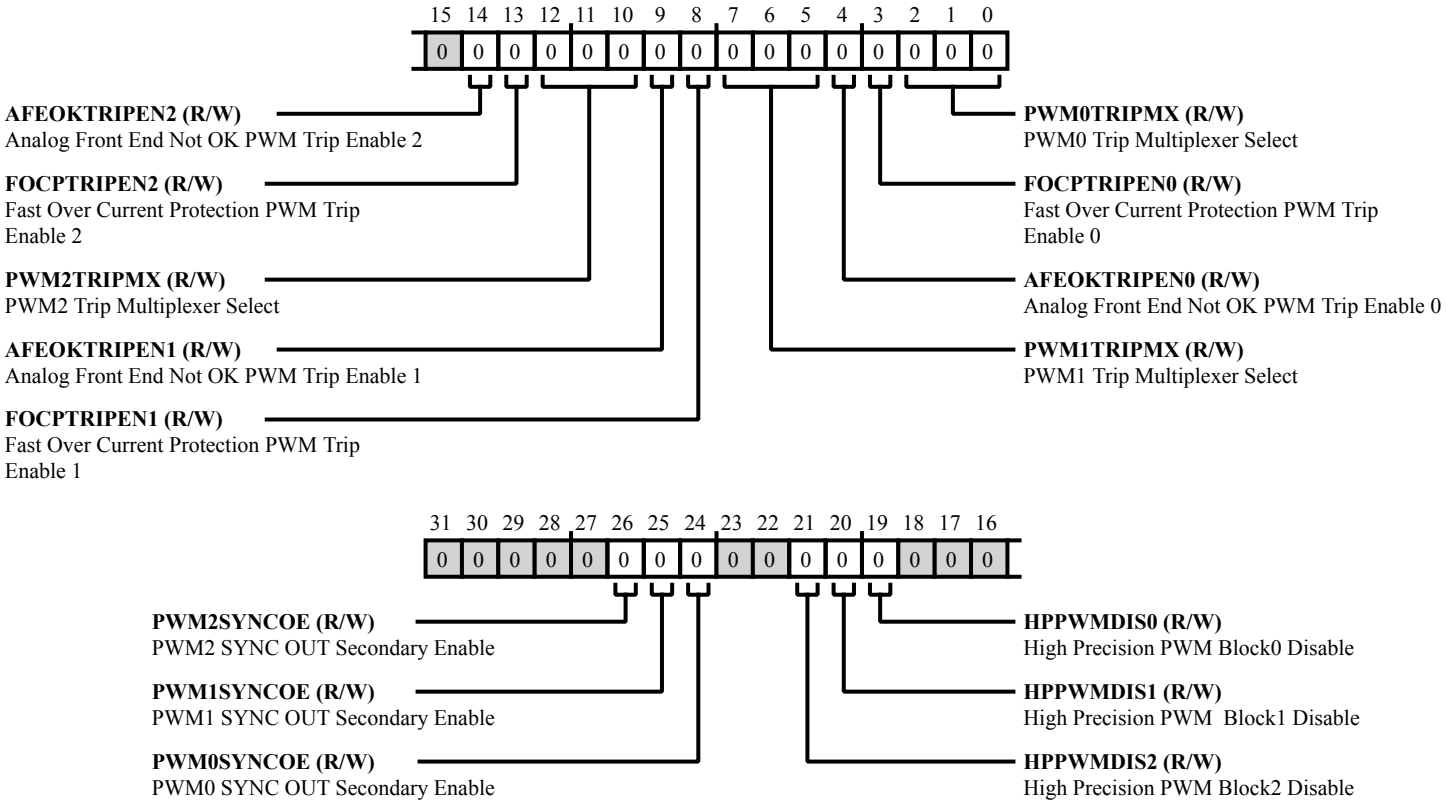


Figure 43-57: SYSBLK_PWM_SYS_CFG Register Diagram

Table 43-66: SYSBLK_PWM_SYS_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
26 (R/W)	PWM2SYNCOE	PWM2 SYNC OUT Secondary Enable. The SYSBLK_PWM_SYS_CFG.PWM2SYNCOE bit is used when the PWM is being synced by a trigger signal. This requires that PWM external sync bit is set (=1). Additionally, route the PWM's sync-pulse generator to an output pin.
25 (R/W)	PWM1SYNCOE	PWM1 SYNC OUT Secondary Enable. The SYSBLK_PWM_SYS_CFG.PWM1SYNCOE bit is used when the PWM is being synced by a trigger signal. This requires that PWM external sync bit is set (=1). Additionally, route the PWM's sync-pulse generator to an output pin.
24 (R/W)	PWM0SYNCOE	PWM0 SYNC OUT Secondary Enable. The SYSBLK_PWM_SYS_CFG.PWM0SYNCOE bit is used when the PWM is being synced by a trigger signal. This requires that the PWM external sync bit is set (=1). Additionally, route the PWM's sync-pulse generator to an output pin.

Table 43-66: SYSBLK_PWM_SYS_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
21 (R/W)	HPPWMDIS2	High Precision PWM Block2 Disable. The SYSBLK_PWM_SYS_CFG.HPPWMDIS2 bit reverts high precision PWM block 2 to medium precision.
20 (R/W)	HPPWMDIS1	High Precision PWM Block1 Disable. The SYSBLK_PWM_SYS_CFG.HPPWMDIS1 bit reverts high precision PWM block 1 to medium precision.
19 (R/W)	HPPWMDIS0	High Precision PWM Block0 Disable. The SYSBLK_PWM_SYS_CFG.HPPWMDIS0 bit reverts high precision PWM block 0 to medium precision.
14 (R/W)	AFEOKTRIPEN2	Analog Front End Not OK PWM Trip Enable 2. The SYSBLK_PWM_SYS_CFG.AFEOKTRIPEN2 bit controls if the AFE die output NOT_OK causes a PWM unit trip.
13 (R/W)	FOCPTRIPEN2	Fast Over Current Protection PWM Trip Enable 2. The SYSBLK_PWM_SYS_CFG.FOCPTRIPEN2 bit controls if the AFE die output FOCP causes a PWM unit trip.
12:10 (R/W)	PWM2TRIPMX	PWM2 Trip Multiplexer Select. The SYSBLK_PWM_SYS_CFG.PWM2TRIPMX bit controls if the trip2 chip input causes a PWM unit trip.
9 (R/W)	AFEOKTRIPEN1	Analog Front End Not OK PWM Trip Enable 1. The SYSBLK_PWM_SYS_CFG.AFEOKTRIPEN1 bit controls if the AFE die output NOT_OK causes a PWM unit trip.
8 (R/W)	FOCPTRIPEN1	Fast Over Current Protection PWM Trip Enable 1. The SYSBLK_PWM_SYS_CFG.FOCPTRIPEN1 bit controls if the AFE die output FOCP causes a PWM unit trip.
7:5 (R/W)	PWM1TRIPMX	PWM1 Trip Multiplexer Select. The SYSBLK_PWM_SYS_CFG.PWM1TRIPMX bit controls if the trip1 chip input causes a PWM unit trip.
4 (R/W)	AFEOKTRIPEN0	Analog Front End Not OK PWM Trip Enable 0. The SYSBLK_PWM_SYS_CFG.AFEOKTRIPEN0 bit controls if the AFE die output NOT_OK causes a PWM unit trip.
3 (R/W)	FOCPTRIPEN0	Fast Over Current Protection PWM Trip Enable 0. The SYSBLK_PWM_SYS_CFG.FOCPTRIPEN0 bit controls if the AFE die output FOCP causes a PWM unit trip.

Table 43-66: SYSBLK_PWM_SYS_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	PWM0TRIPMX	PWM0 Trip Multiplexer Select. The `SYSBLK_PWM_SYS_CFG.PWM0TRIPMX` bit controls if the trip0 chip input causes a PWM unit trip.

Rotary Counter Up/Down Configuration Register

The SYSBLK_ROT_UPDN_CFG register selects between rotary counter phase A or B inputs.

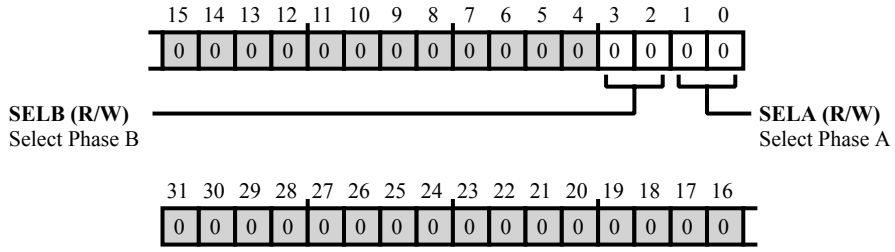


Figure 43-58: SYSBLK_ROT_UPDN_CFG Register Diagram

Table 43-67: SYSBLK_ROT_UPDN_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/W)	SELB	Select Phase B. The SYSBLK_ROT_UPDN_CFG.SELB bit field selects Rotary CNT0 Phase B input.	
		0	Select Pad as input to Rotary Counter
		1	Select Pad as input to Rotary Counter
		2	Select Trigger Slave as input to Rotary Counter
		3	Select LBA as input to Rotary Counter
1:0 (R/W)	SELA	Select Phase A. The SYSBLK_ROT_UPDN_CFG.SELA bit field selects Rotary CNT0 Phase A input.	
		0	Select Pad as input to Rotary Counter
		1	Select Pad as input to Rotary Counter
		2	Select Trigger Slave as input to Rotary Counter
		3	Select LBA as input to Rotary Counter

SINC Test Register

The SYSBLK_SINC_TEST register configures the SINC block 4 data inputs (one per MMR bit) to come from the single SPORT0 data A output. This allows a know data stream to be directed to SINC blocks for verification, development, and self-test code for safety.

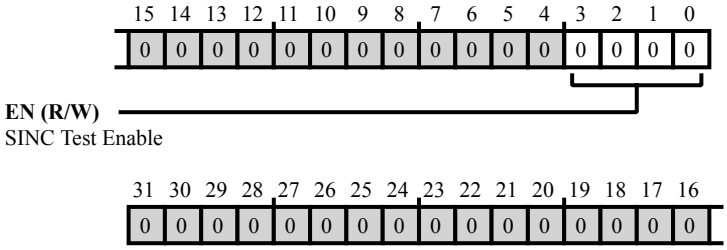


Figure 43-59: SYSBLK_SINC_TEST Register Diagram

Table 43-68: SYSBLK_SINC_TEST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	EN	SINC Test Enable. The SYSBLK_SINC_TEST.EN bit field configures the SINC block 4 data inputs (one per MMR bit) to come from the single SPORT0 data A output

Shared Interrupt 0 Status Register

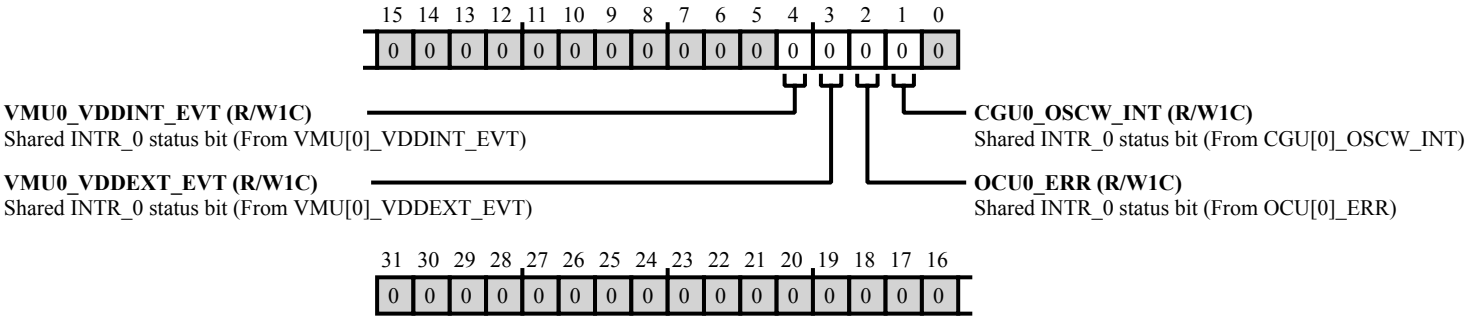


Figure 43-60: SYSBLK_SISTAT0 Register Diagram

Table 43-69: SYSBLK_SISTAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W1C)	VMU0_VDDINT_EVT	Shared INTR_0 status bit (From VMU[0]_VDDINT_EVT).
3 (R/W1C)	VMU0_VDDEXT_EVT	Shared INTR_0 status bit (From VMU[0]_VDDEXT_EVT).
2 (R/W1C)	OCU0_ERR	Shared INTR_0 status bit (From OCU[0]_ERR).
1 (R/W1C)	CGU0_OSCW_INT	Shared INTR_0 status bit (From CGU[0]_OSCW_INT).

Shared Interrupt 10 Status Register

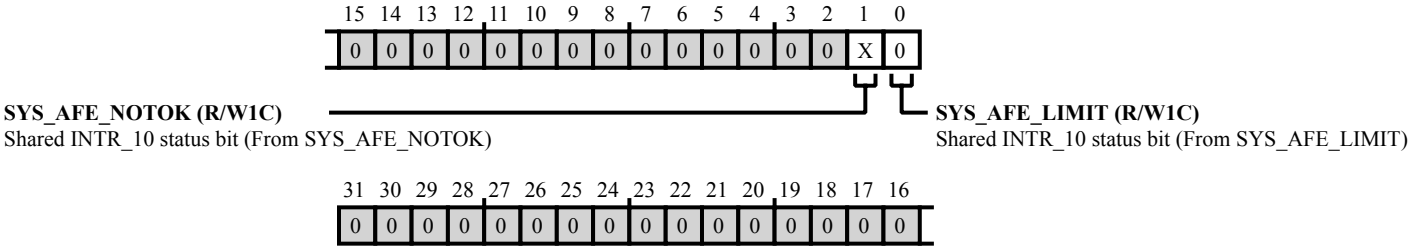


Figure 43-61: SYSBLK_SISTAT10 Register Diagram

Table 43-70: SYSBLK_SISTAT10 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	SYS_AFE_NOTOK	Shared INTR_10 status bit (From SYS_AFE_NOTOK).
0 (R/W1C)	SYS_AFE_LIMIT	Shared INTR_10 status bit (From SYS_AFE_LIMIT).

Shared Interrupt 11 Status Register

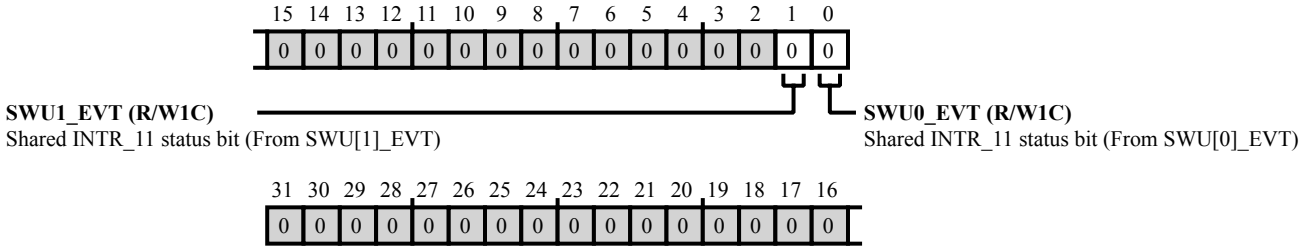


Figure 43-62: SYSBLK_SISTAT11 Register Diagram

Table 43-71: SYSBLK_SISTAT11 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	SWU1_EVT	Shared INTR_11 status bit (From SWU[1]_EVT).
0 (R/W1C)	SWU0_EVT	Shared INTR_11 status bit (From SWU[0]_EVT).

Shared Interrupt 12 Status Register

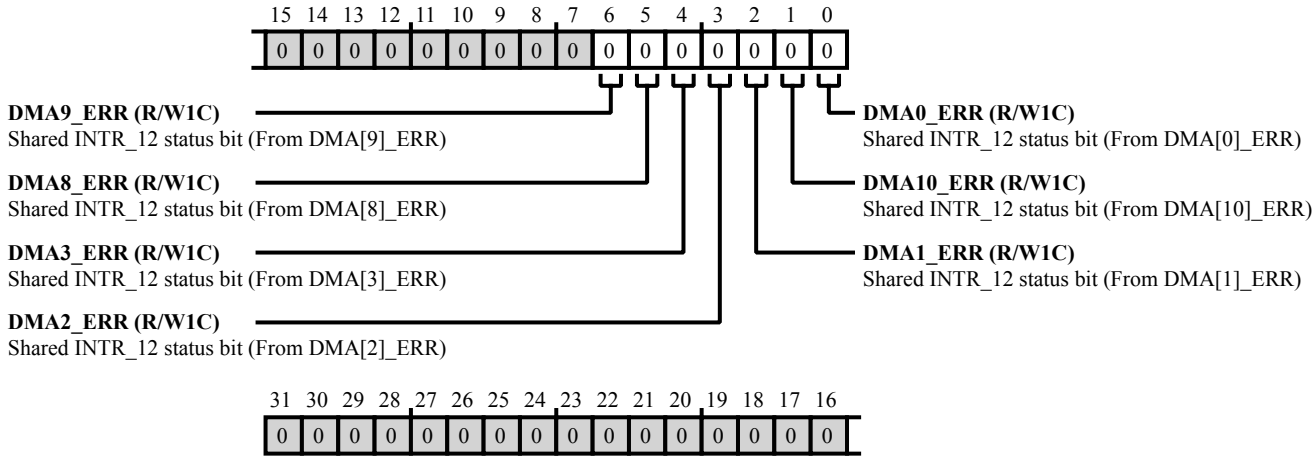


Figure 43-63: SYSBLK_SISTAT12 Register Diagram

Table 43-72: SYSBLK_SISTAT12 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	DMA9_ERR	Shared INTR_12 status bit (From DMA[9]_ERR).
5 (R/W1C)	DMA8_ERR	Shared INTR_12 status bit (From DMA[8]_ERR).
4 (R/W1C)	DMA3_ERR	Shared INTR_12 status bit (From DMA[3]_ERR).
3 (R/W1C)	DMA2_ERR	Shared INTR_12 status bit (From DMA[2]_ERR).
2 (R/W1C)	DMA1_ERR	Shared INTR_12 status bit (From DMA[1]_ERR).
1 (R/W1C)	DMA10_ERR	Shared INTR_12 status bit (From DMA[10]_ERR).
0 (R/W1C)	DMA0_ERR	Shared INTR_12 status bit (From DMA[0]_ERR).

Shared Interrupt 15 Status Register

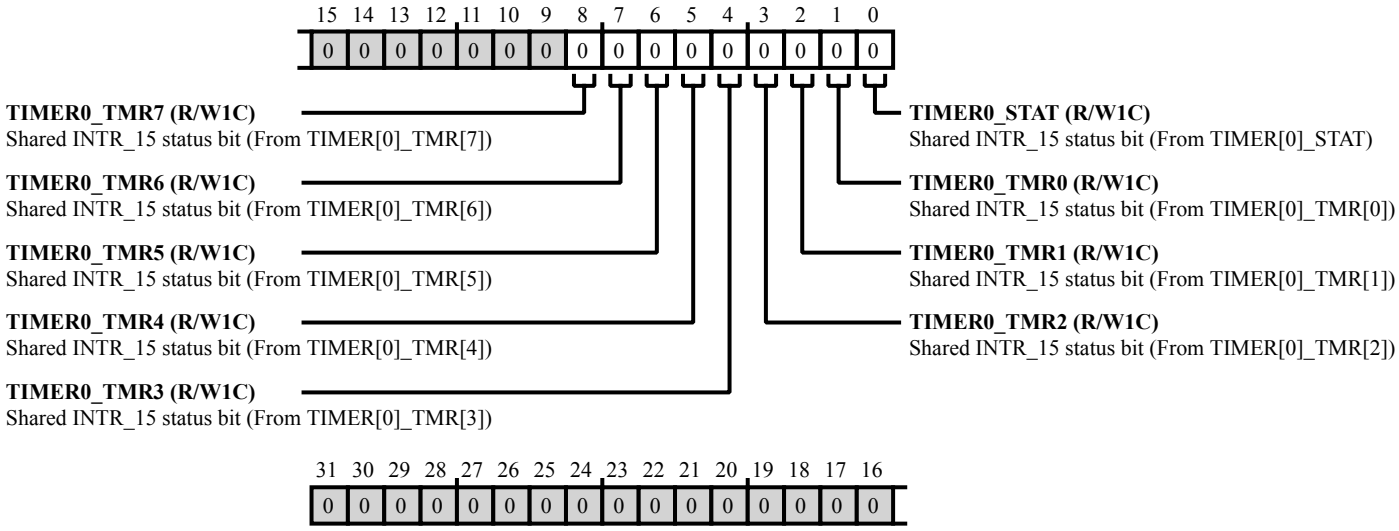


Figure 43-64: SYSBLK_SISTAT15 Register Diagram

Table 43-73: SYSBLK_SISTAT15 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8 (R/W1C)	TIMER0_TMR7	Shared INTR_15 status bit (From TIMER[0]_TMR[7]).
7 (R/W1C)	TIMER0_TMR6	Shared INTR_15 status bit (From TIMER[0]_TMR[6]).
6 (R/W1C)	TIMER0_TMR5	Shared INTR_15 status bit (From TIMER[0]_TMR[5]).
5 (R/W1C)	TIMER0_TMR4	Shared INTR_15 status bit (From TIMER[0]_TMR[4]).
4 (R/W1C)	TIMER0_TMR3	Shared INTR_15 status bit (From TIMER[0]_TMR[3]).
3 (R/W1C)	TIMER0_TMR2	Shared INTR_15 status bit (From TIMER[0]_TMR[2]).
2 (R/W1C)	TIMER0_TMR1	Shared INTR_15 status bit (From TIMER[0]_TMR[1]).
1 (R/W1C)	TIMER0_TMR0	Shared INTR_15 status bit (From TIMER[0]_TMR[0]).

Table 43-73: SYSBLK_SISTAT15 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1C)	TIMER0_STAT	Shared INTR_15 status bit (From TIMER[0]_STAT).

Shared Interrupt 16 Status Register

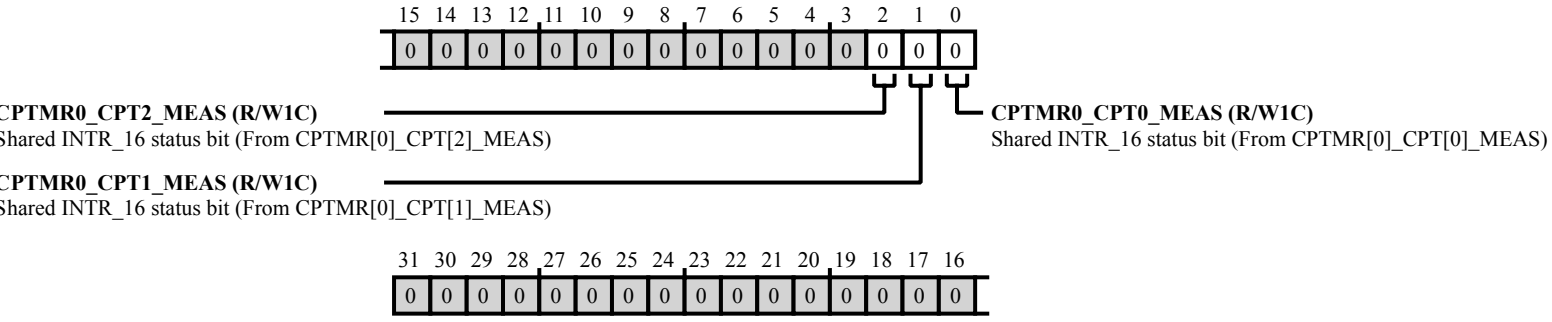


Figure 43-65: SYSBLK_SISTAT16 Register Diagram

Table 43-74: SYSBLK_SISTAT16 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	CPTMR0_CPT2_MEAS	Shared INTR_16 status bit (From CPTMR[0]_CPT[2]_MEAS).
1 (R/W1C)	CPTMR0_CPT1_MEAS	Shared INTR_16 status bit (From CPTMR[0]_CPT[1]_MEAS).
0 (R/W1C)	CPTMR0_CPT0_MEAS	Shared INTR_16 status bit (From CPTMR[0]_CPT[0]_MEAS).

Shared Interrupt 17 Status Register

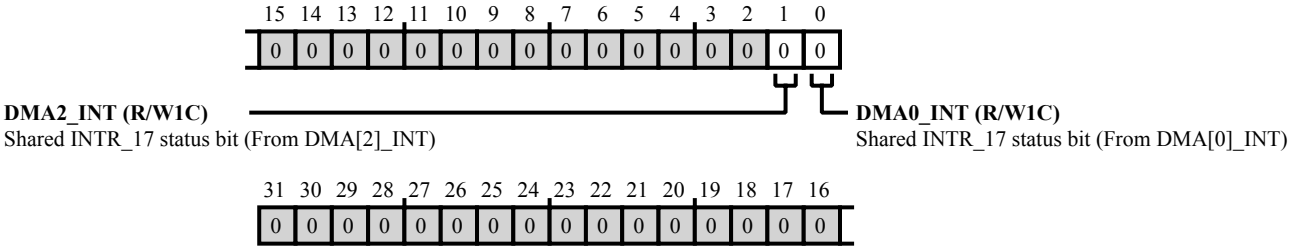


Figure 43-66: SYSBLK_SISTAT17 Register Diagram

Table 43-75: SYSBLK_SISTAT17 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	DMA2_INT	Shared INTR_17 status bit (From DMA[2]_INT).
0 (R/W1C)	DMA0_INT	Shared INTR_17 status bit (From DMA[0]_INT).

Shared Interrupt 18 Status Register

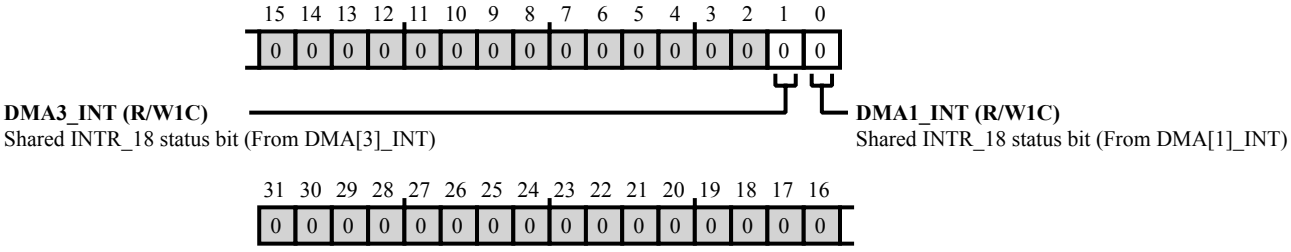


Figure 43-67: SYSBLK_SISTAT18 Register Diagram

Table 43-76: SYSBLK_SISTAT18 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	DMA3_INT	Shared INTR_18 status bit (From DMA[3]_INT).
0 (R/W1C)	DMA1_INT	Shared INTR_18 status bit (From DMA[1]_INT).

Shared Interrupt 19 Status Register

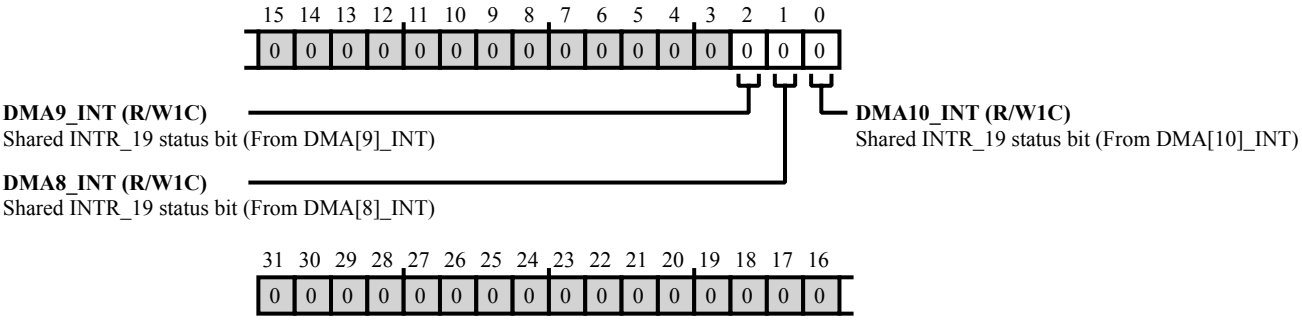


Figure 43-68: SYSBLK_SISTAT19 Register Diagram

Table 43-77: SYSBLK_SISTAT19 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	DMA9_INT	Shared INTR_19 status bit (From DMA[9]_INT).
1 (R/W1C)	DMA8_INT	Shared INTR_19 status bit (From DMA[8]_INT).
0 (R/W1C)	DMA10_INT	Shared INTR_19 status bit (From DMA[10]_INT).

Shared Interrupt 22 Status Register

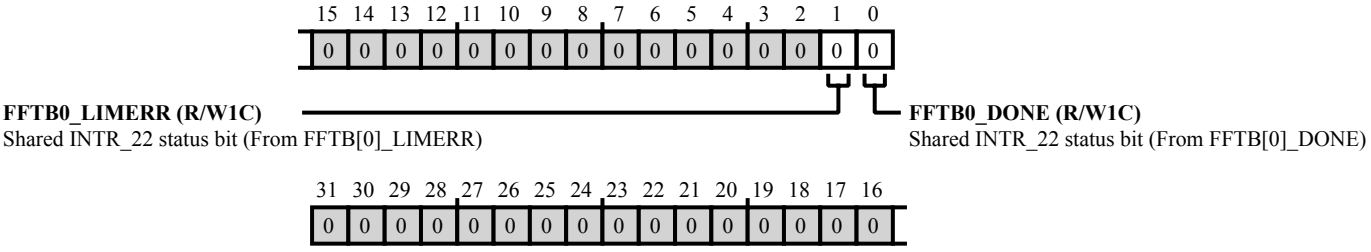


Figure 43-69: SYSBLK_SISTAT22 Register Diagram

Table 43-78: SYSBLK_SISTAT22 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	FFTB0_LIMERR	Shared INTR_22 status bit (From FFTB[0]_LIMERR).
0 (R/W1C)	FFTB0_DONE	Shared INTR_22 status bit (From FFTB[0]_DONE).

Shared Interrupt 25 Status Register

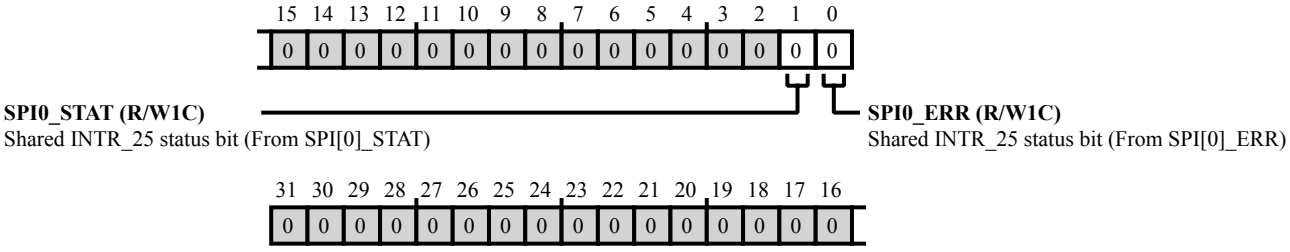


Figure 43-70: SYSBLK_SISTAT25 Register Diagram

Table 43-79: SYSBLK_SISTAT25 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	SPI0_STAT	Shared INTR_25 status bit (From SPI[0]_STAT).
0 (R/W1C)	SPI0_ERR	Shared INTR_25 status bit (From SPI[0]_ERR).

Shared Interrupt 28 Status Register

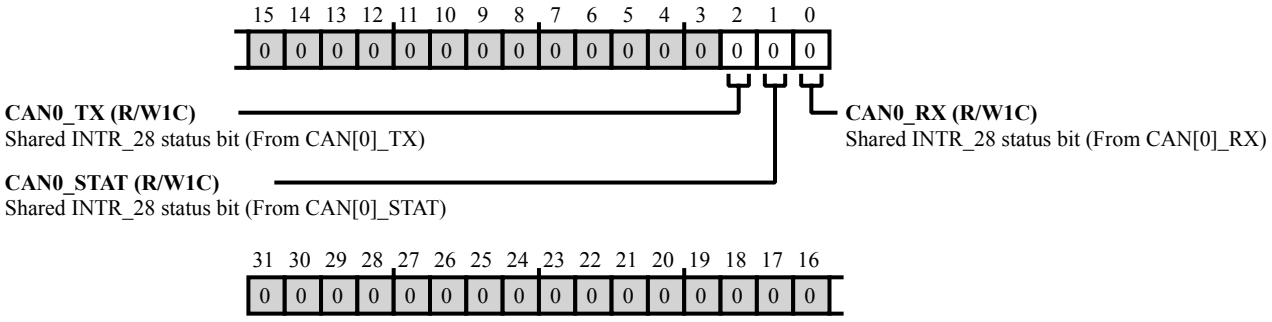


Figure 43-71: SYSBLK_SISTAT28 Register Diagram

Table 43-80: SYSBLK_SISTAT28 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	CAN0_TX	Shared INTR_28 status bit (From CAN[0]_TX).
1 (R/W1C)	CAN0_STAT	Shared INTR_28 status bit (From CAN[0]_STAT).
0 (R/W1C)	CAN0_RX	Shared INTR_28 status bit (From CAN[0]_RX).

Shared Interrupt 3 Status Register

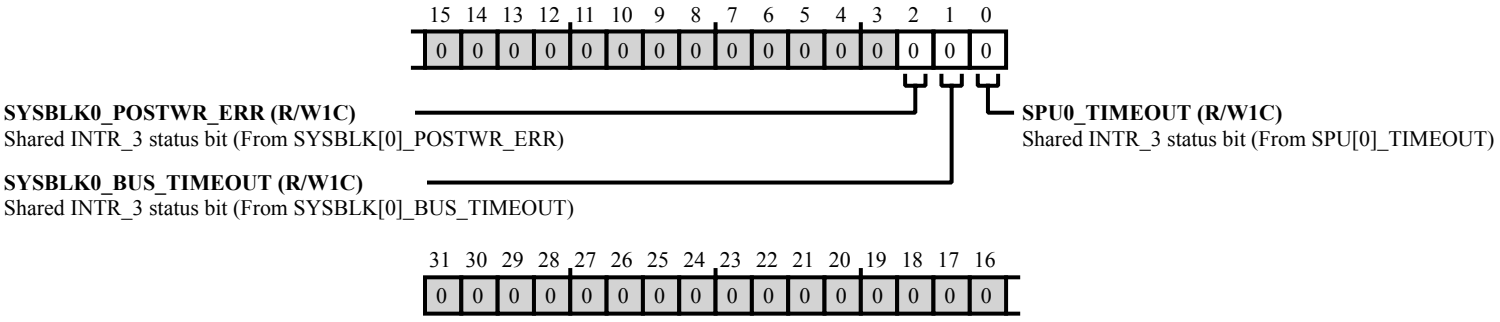


Figure 43-72: SYSBLK_SISTAT3 Register Diagram

Table 43-81: SYSBLK_SISTAT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	SYSBLK0_POSTWR_ERR	Shared INTR_3 status bit (From SYSBLK[0]_POSTWR_ERR).
1 (R/W1C)	SYSBLK0_BUS_TIME-OUT	Shared INTR_3 status bit (From SYSBLK[0]_BUS_TIMEOUT).
0 (R/W1C)	SPU0_TIMEOUT	Shared INTR_3 status bit (From SPU[0]_TIMEOUT).

Shared Interrupt 5 Status Register

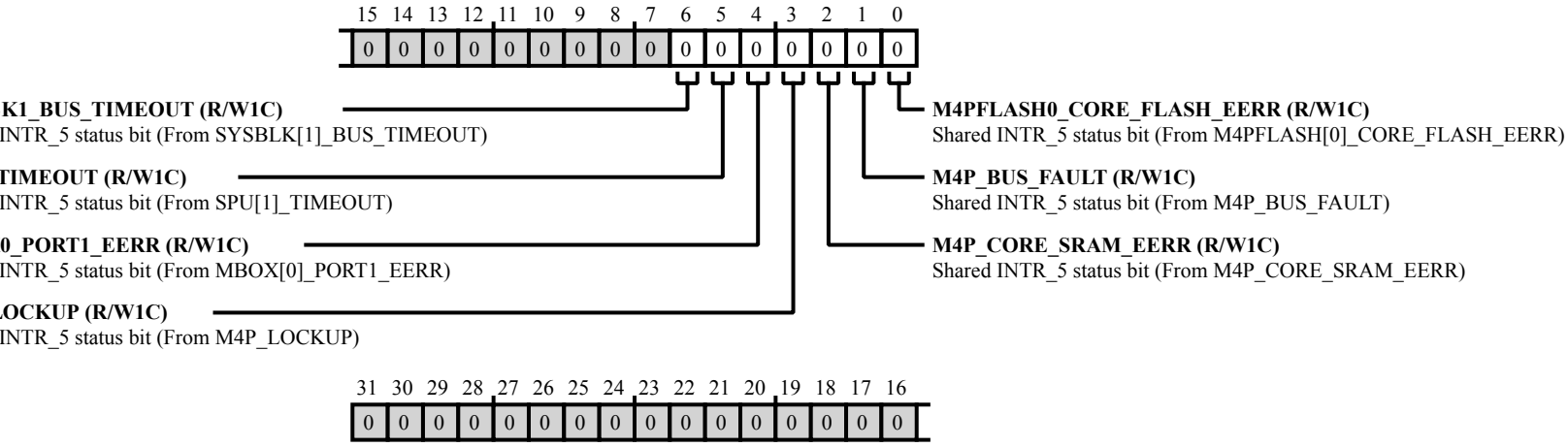


Figure 43-73: SYSBLK_SISTAT5 Register Diagram

Table 43-82: SYSBLK_SISTAT5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W1C)	SYSBLK1_BUS_TIME-OUT	Shared INTR_5 status bit (From SYSBLK[1]_BUS_TIMEOUT).
5 (R/W1C)	SPU1_TIMEOUT	Shared INTR_5 status bit (From SPU[1]_TIMEOUT).
4 (R/W1C)	MBOX0_PORT1_EERR	Shared INTR_5 status bit (From MBOX[0]_PORT1_EERR).
3 (R/W1C)	M4P_LOCKUP	Shared INTR_5 status bit (From M4P_LOCKUP).
2 (R/W1C)	M4P_CORE_SRAM_EER R	Shared INTR_5 status bit (From M4P_CORE_SRAM_EERR).
1 (R/W1C)	M4P_BUS_FAULT	Shared INTR_5 status bit (From M4P_BUS_FAULT).
0 (R/W1C)	M4PFLASH0_CORE_FLA SH_EERR	Shared INTR_5 status bit (From M4PFLASH[0]_CORE_FLASH_EERR).

Shared Interrupt 6 Status Register

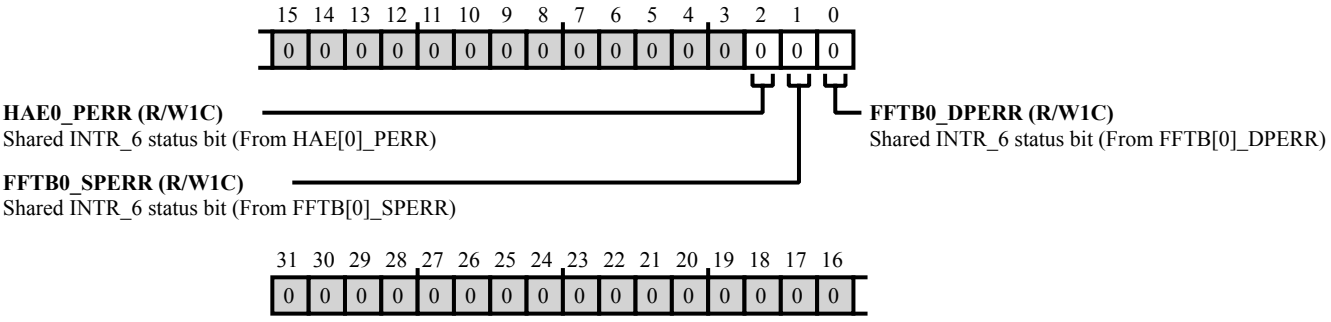


Figure 43-74: SYSBLK_SISTAT6 Register Diagram

Table 43-83: SYSBLK_SISTAT6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	HAE0_PERR	Shared INTR_6 status bit (From HAE[0]_PERR).
1 (R/W1C)	FFTB0_SPERR	Shared INTR_6 status bit (From FFTB[0]_SPERR).
0 (R/W1C)	FFTB0_DPERR	Shared INTR_6 status bit (From FFTB[0]_DPERR).

Shared Interrupt 7 Status Register

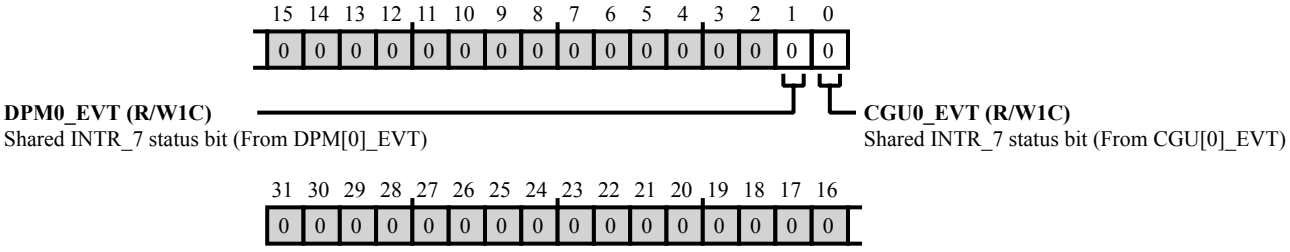


Figure 43-75: SYSBLK_SISTAT7 Register Diagram

Table 43-84: SYSBLK_SISTAT7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/W1C)	DPM0_EVT	Shared INTR_7 status bit (From DPM[0]_EVT).
0 (R/W1C)	CGU0_EVT	Shared INTR_7 status bit (From CGU[0]_EVT).

Shared Interrupt 8 Status Register

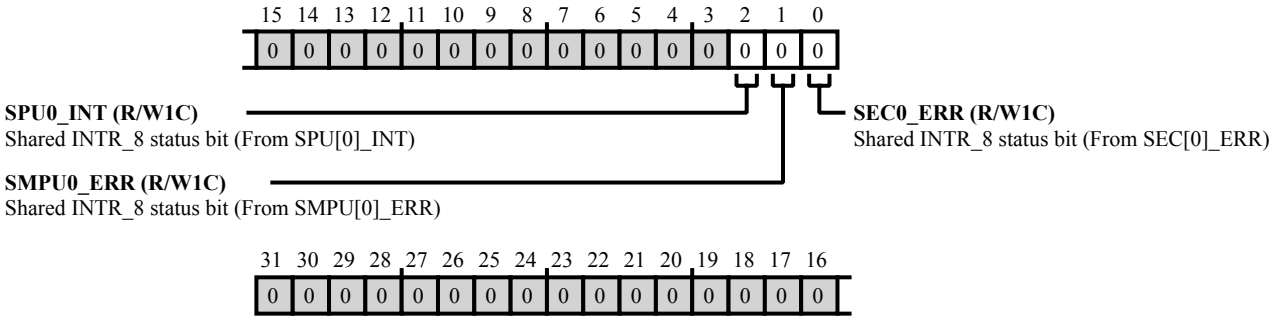


Figure 43-76: SYSBLK_SISTAT8 Register Diagram

Table 43-85: SYSBLK_SISTAT8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/W1C)	SPU0_INT	Shared INTR_8 status bit (From SPU[0]_INT).
1 (R/W1C)	SMPU0_ERR	Shared INTR_8 status bit (From SMPU[0]_ERR).
0 (R/W1C)	SEC0_ERR	Shared INTR_8 status bit (From SEC[0]_ERR).

M0 SRAM ECC Register

The SYSBLK_SRAM0_ECC register allows direct R/W access of the raw ECC/DATA bits of the memory for error testing and debug. When SYSBLK_SRAM0_ECC.MAPMODE[31] is set, MAPMODE[30] determines whether ECC (SYSBLK_SRAM0_ECC.MAPMODE[30] equal one) or DATA (SYSBLK_SRAM0_ECC.MAPMODE[30] equal zero) bits are read/written. ECC generation/correction is disable in this mode and no ECC IRQs are generated for memory transactions.

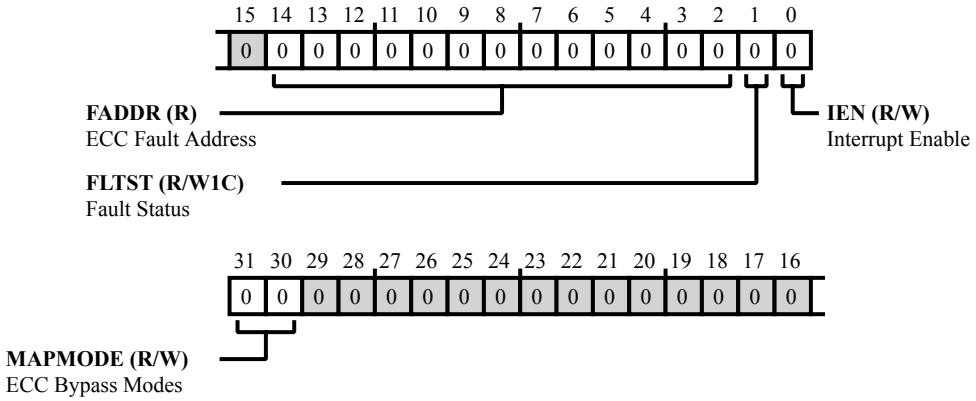


Figure 43-77: SYSBLK_SRAM0_ECC Register Diagram

Table 43-86: SYSBLK_SRAM0_ECC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	MAPMODE	ECC Bypass Modes. When SYSBLK_SRAM0_ECC.MAPMODE[31] is set, SYSBLK_SRAM0_ECC.MAPMODE[30] determines whether ECC (SYSBLK_SRAM0_ECC.MAPMODE[30] equal one) or DATA (SYSBLK_SRAM0_ECC.MAPMODE[30] equal zero) bits are read/written. ECC generation/correction is disable in this mode and no ECC IRQs are generated for memory transactions.
14:2 (R/NW)	FADDR	ECC Fault Address.
1 (R/W1C)	FLTST	Fault Status.
0 (R/W)	IEN	Interrupt Enable.

System Status Register

The SYSBLK_SYSSTAT register bits indicate the AFE status.

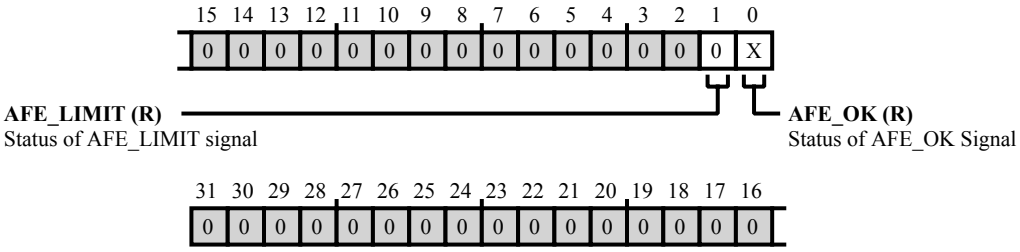


Figure 43-78: SYSBLK_SYSSTAT Register Diagram

Table 43-87: SYSBLK_SYSSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	AFE_LIMIT	Status of AFE_LIMIT signal. The SYSBLK_SYSSTAT.AFE_LIMIT bit indicates the status of AFE_LIMIT signal, indicating at least one of the AFE FOCP comparators has detected an over limit condition.	
0 (R/NW)	AFE_OK	Status of AFE_OK Signal. The SYSBLK_SYSSTAT.AFE_OK bit indicates the status of AFE_OK signal.	
		0	AFE not OK AFE not OK
		1	AFE OK

System Register

It allows the programmer to define the calibration value for the ARM Processor's SysTick timer, according to the frequency of the installed crystal and the divisor settings of the CGU and PLL. The value programmed into this register appears in the ARM SYST_CALIB read-only register.

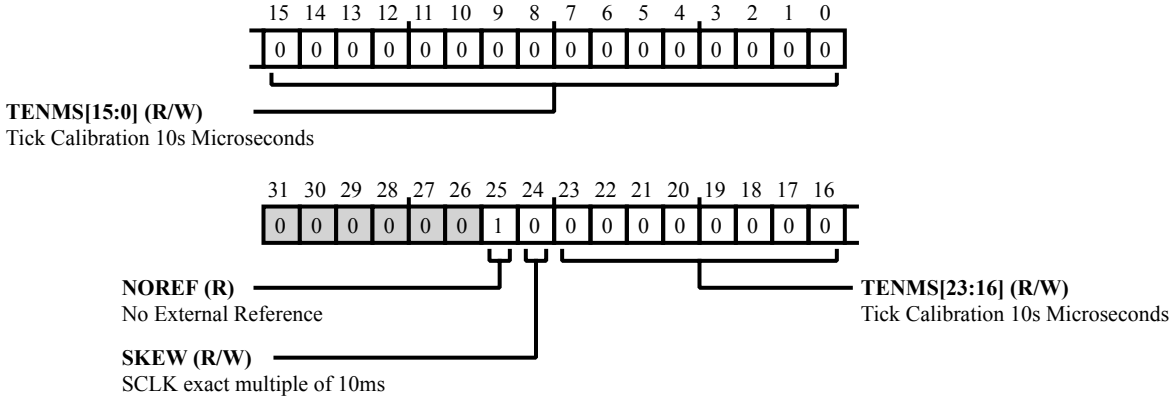


Figure 43-79: SYSBLK_SYS_STCALIB Register Diagram

Table 43-88: SYSBLK_SYS_STCALIB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/NW)	NOREF	No External Reference. The `SYSBLK_SYS_STCALIB.NOREF` indicates whether an external SysTick reference clock is implemented. In the ADI M4 Platform, no SysTick external reference clock is implemented. The processor internal reference clock is used for SYSTICK.
24 (R/W)	SKEW	SCLK exact multiple of 10ms. The `SYSBLK_SYS_STCALIB.SKEW` bit is set to 1 if the calibration value specified by `SYSBLK_SYS_STCALIB.TENMS` does not provide an exact multiple of 10ms. Otherwise, set this bit to 0. For example, the frequency of a 166.66 (6 repeating) MHz core clock corresponds to a `SYSBLK_SYS_STCALIB.TENMS` value of 1,666,666.66, which is not an exact integer, so in that case `SYSBLK_SYS_STCALIB.SKEW` is set to 1.
23:0 (R/W)	TENMS	Tick Calibration 10s Microseconds. The `SYSBLK_SYS_STCALIB.TENMS` bit is set to an integer 24-bit value usable to compute a 10ms delay from the user-programmed frequency of the M4P core clock. For example, for a 200MHz core clock, set this value to (200MHz * 10ms) = 24'd2_000_000 = 24'h1e_8480.

CM41X_M0 PADS Register Descriptions

Pads Controller (PADS) contains the following registers.

Table 43-89: CM41X_M0 PADS Register List

Name	Description
PADS_DBC[n]_CTL	Debounce Control Register(s)
PADS_DBC_PRESCALE	Debounce Prescale Register
PADS_FOCP_DIV	Fast Over Current Protection Clock Divisor Register
PADS_MONOSC_CFG	Monitor Oscillator Control Register
PADS_NVWR_RSTCTL	Non-Volatile Write Reset Control Register
PADS_PCFG0	Peripheral Configuration0 Register
PADS_PORT[n]_DS	Multi Port Drive Strength Control Register
PADS_PORT[n]_RCTL	Multi Port Pull-up/Pull-down Resistor Control Register
PADS_PORT[n]_TRIPSEL	Multi Port Trip Select Register
PADS_PORT[n]_TRIPST	Multi Port Trip State Register
PADS_VMU_CTL	Voltage Monitor Unit Control Register
PADS_VMU_TRIM	Voltage Monitor Unit Trim Register
PADS_VMU_TRIPEN	Voltage Monitor Unit Trip Enable Register

Debounce Control Register(s)

The PADS_DBC[n]_CTL register controls how the module operates.

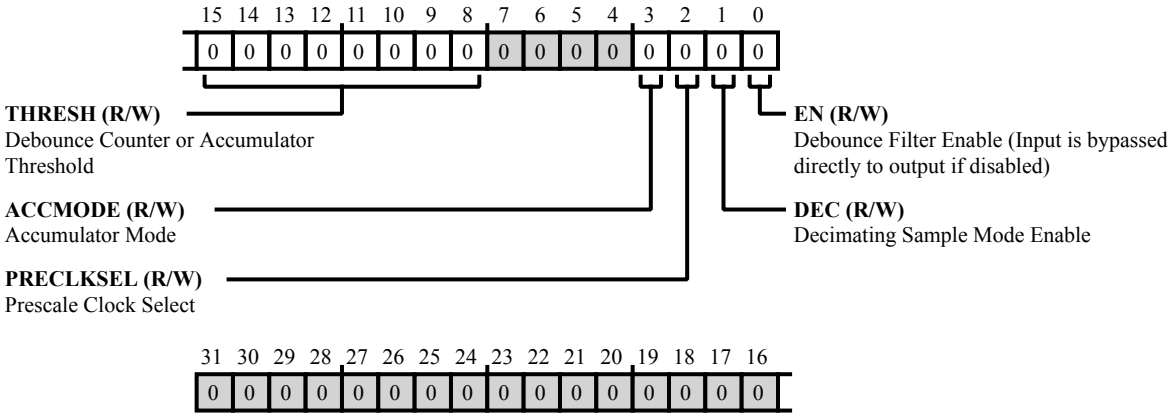


Figure 43-80: PADS_DBC[n]_CTL Register Diagram

Table 43-90: PADS_DBC[n]_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:8 (R/W)	THRESH	Debounce Counter or Accumulator Threshold. The PADS_DBC[n]_CTL.THRESH bit field contain the value THRESH + 1.	
3 (R/W)	ACCMODE	Accumulator Mode. The PADS_DBC[n]_CTL.ACCMODE bit configures the debounce filter to use accumulator mode or counter mode.	
		0	Counter mode (default)
		1	Accumulator mode
2 (R/W)	PRECLKSEL	Prescale Clock Select. The PADS_DBC[n]_CTL.PRECLKSEL bit configures the debounce filter to use the prescale clock or the system clock.	
		0	System Clock (default)
		1	Prescale clock
1 (R/W)	DEC	Decimating Sample Mode Enable. The PADS_DBC[n]_CTL.DEC bit configures the debounce filter for decimating or continuous sample mode.	
		0	Continuous mode (default)
		1	Sample mode

Table 43-90: PADS_DBC[n]_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	Debounce Filter Enable (Input is bypassed directly to output if disabled). The default of the `PADS_DBC[n]_CTL.EN` bit is disabled.	
		0	Filter disabled
		1	Filter enabled

Debounce Prescale Register

The PADS_DBC_PRESCALE register contains the Debounce clock prescale divisor.

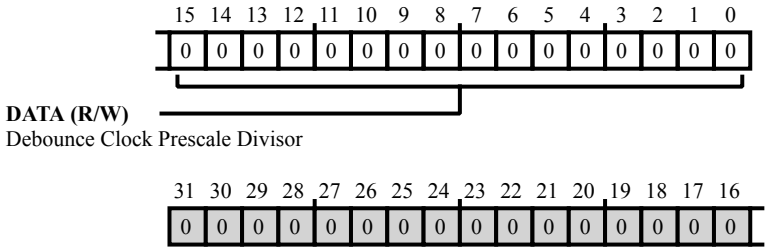


Figure 43-81: PADS_DBC_PRESCALE Register Diagram

Table 43-91: PADS_DBC_PRESCALE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	DATA	Debounce Clock Prescale Divisor. The value for the PADS_DBC_PRESCALE.DATA bit field is Filter update pulse = SYSCLK/PRESCALE + 1.

Fast Over Current Protection Clock Divisor Register

The PADS_FOCP_DIV register set the frequency divide ratio.

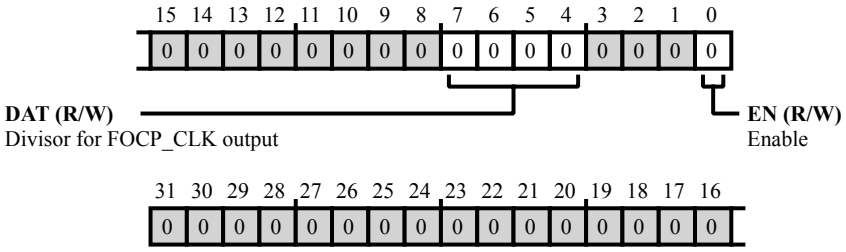


Figure 43-82: PADS_FOCP_DIV Register Diagram

Table 43-92: PADS_FOCP_DIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	DAT	Divisor for FOCP_CLK output. The FOCP Comparators in the AFE require a free-running clock of roughly 10 MHz. The PADS_FOCP_DIV.DAT bit field sets the frequency divide ratio of DCLK to FOCP_CLK as follows: FOCP_DIV = 1 to 15: $f_{FOCP\_CLK} = f_{DCLK} / (2 * FOCP\_DIV:DAT)$ FOCP_DIV = 0: $f_{FOCP\_CLK} = f_{DCLK} / 32$
0 (R/W)	EN	Enable. The PADS_FOCP_DIV.EN bit enables the over current protection clock divisor.

Monitor Oscillator Control Register

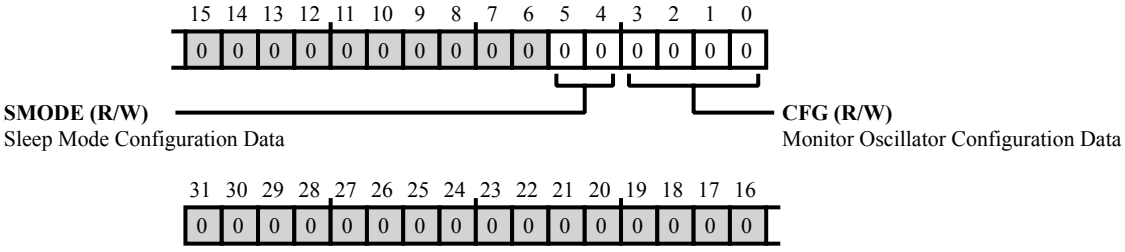


Figure 43-83: PADS_MONOSC_CFG Register Diagram

Table 43-93: PADS_MONOSC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
5:4 (R/W)	SMODE	Sleep Mode Configuration Data.
3:0 (R/W)	CFG	Monitor Oscillator Configuration Data.

Non-Volatile Write Reset Control Register

There are 2 basic flash unit functions, read and write. Reads occur without any special requirements. Writes (programming) of flash memory require special dedicated hardware and control. The PADS_NVWR_RSTCTL register controls the enable for the flash0/1 write control units. Without enabling the write logic block then no programming to the flash may take place, regardless of the commands written to the flash control MMR registers.

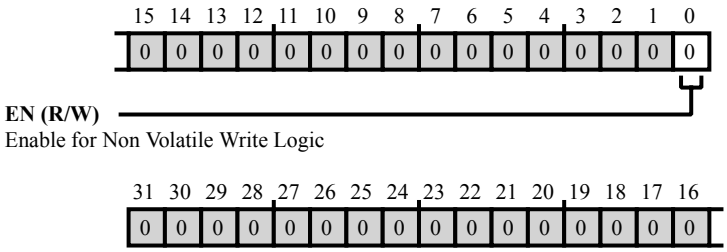


Figure 43-84: PADS_NVWR_RSTCTL Register Diagram

Table 43-94: PADS_NVWR_RSTCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	EN	Enable for Non Volatile Write Logic. The PADS_NVWR_RSTCTL.EN bit controls the enable for the flash0/1 write control units. Without enabling the write logic block then no programming to the flash may take place, regardless of the commands written to the flash control MMR registers. The default state is disabled and in reset.

Peripheral Configuration0 Register

The PADS_PCFG0 register provides several configuration options for various peripherals.

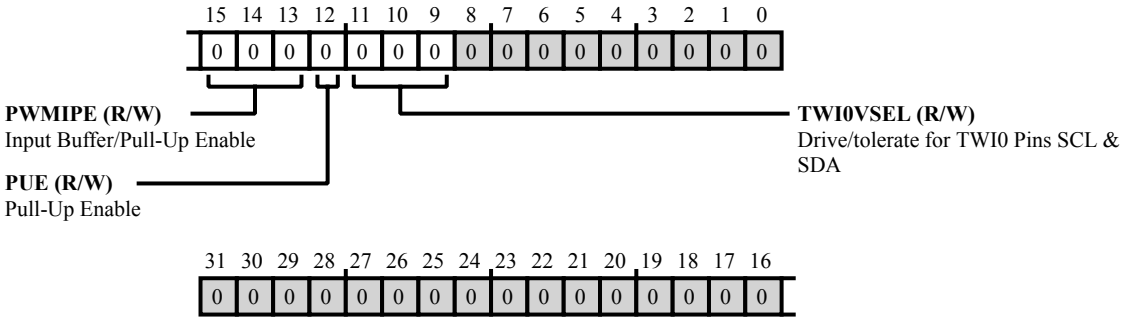


Figure 43-85: PADS_PCFG0 Register Diagram

Table 43-95: PADS_PCFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:13 (R/W)	PWMIPE	Input Buffer/Pull-Up Enable. Different PWM power topographies require different behavior before configuration. The 24 pins associated with the 3 sets of PWM units have a default state different from a generic GPIO pin. The PADS_PCFG0.PWMIPE bits allow the 3 sets of 8 pins associated with PWM units to be quickly programmed back to GPIO functionality for a given use case. Bit0 = 1 restores the defaults for pwm0 pins (PB0 to 7) Bit1 = 1 restores the defaults for pwm1 pins (PB8 to 15) Bit2 = 1 restores the defaults for pwm2 pins (PE0 to 7)	
		0	All PWM units
		1	PWM0 unit
		2	PWM0, 1 units
		3	No PWM units used (111)
12 (R/W)	PUE	Pull-Up Enable. The PADS_PCFG0.PUE bit overrides the input enable and enables the pull-up on the AFE pads.	
11:9 (R/W)	TWI0VSEL	Drive/tolerate for TWI0 Pins SCL & SDA. The PADS_PCFG0.TWI0VSEL sets the voltage requirements for the TWI_SCL and TWI_SDA pins on TWI0.	
		0	VDD_EXT = 3.3 V, VBUS_TWI = 3.3 V
		1	Reserved
		2	Reserved

Table 43-95: PADS_PCFG0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	Reserved
		4	VDD_EXT = 3.3 V, VBUS_TWI = 5 V
		5-7	Reserved

Multi Port Drive Strength Control Register

The PADS_PORT[n]_DS register increases the drive strength of PWM pins. See the product specific data sheet for more information.

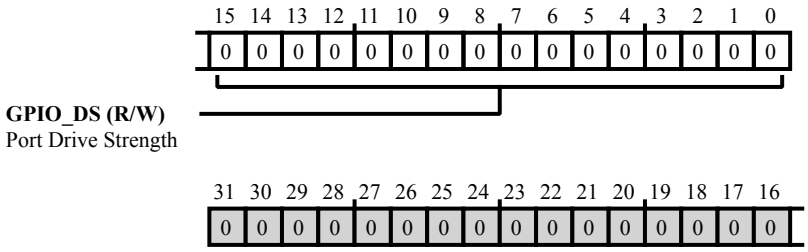


Figure 43-86: PADS_PORT[n]_DS Register Diagram

Table 43-96: PADS_PORT[n]_DS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	GPIO_DS	Port Drive Strength.

Multi Port Pull-up/Pull-down Resistor Control Register

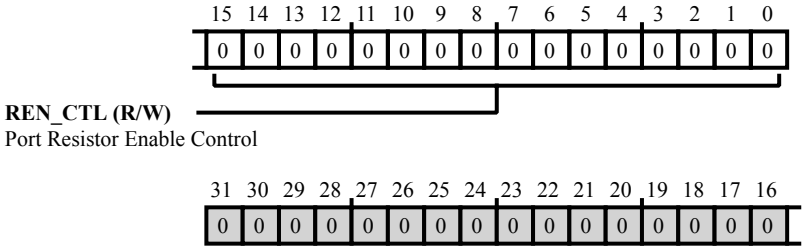


Figure 43-87: PADS_PORT[n]_RCTL Register Diagram

Table 43-97: PADS_PORT[n]_RCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	REN_CTL	Port Resistor Enable Control.

Multi Port Trip Select Register

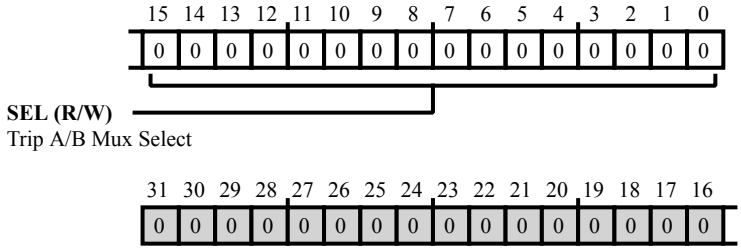


Figure 43-88: PADS_PORT[n]_TRIPSEL Register Diagram

Table 43-98: PADS_PORT[n]_TRIPSEL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	SEL	Trip A/B Mux Select.

Multi Port Trip State Register

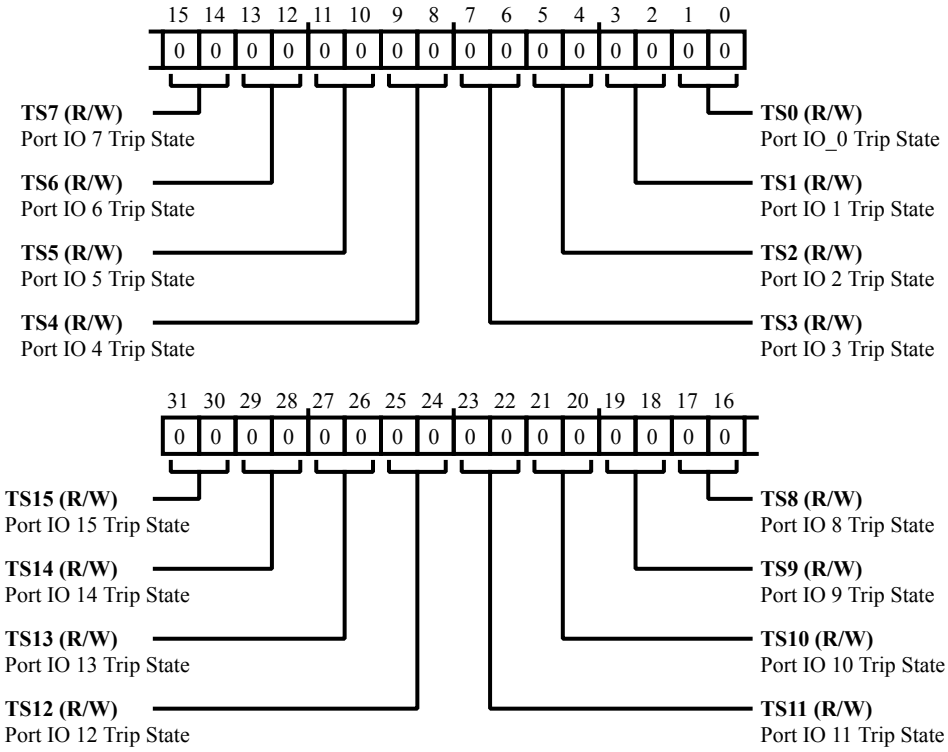


Figure 43-89: PADS_PORT[n]_TRIPST Register Diagram

Table 43-99: PADS_PORT[n]_TRIPST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:30 (R/W)	TS15	Port IO 15 Trip State.
29:28 (R/W)	TS14	Port IO 14 Trip State.
27:26 (R/W)	TS13	Port IO 13 Trip State.
25:24 (R/W)	TS12	Port IO 12 Trip State.
23:22 (R/W)	TS11	Port IO 11 Trip State.
21:20 (R/W)	TS10	Port IO 10 Trip State.

Table 43-99: PADS_PORT[n]_TRIPST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
19:18 (R/W)	TS9	Port IO 9 Trip State.
17:16 (R/W)	TS8	Port IO 8 Trip State.
15:14 (R/W)	TS7	Port IO 7 Trip State.
13:12 (R/W)	TS6	Port IO 6 Trip State.
11:10 (R/W)	TS5	Port IO 5 Trip State.
9:8 (R/W)	TS4	Port IO 4 Trip State.
7:6 (R/W)	TS3	Port IO 3 Trip State.
5:4 (R/W)	TS2	Port IO 2 Trip State.
3:2 (R/W)	TS1	Port IO 1 Trip State.
1:0 (R/W)	TS0	Port IO_0 Trip State.

Voltage Monitor Unit Control Register

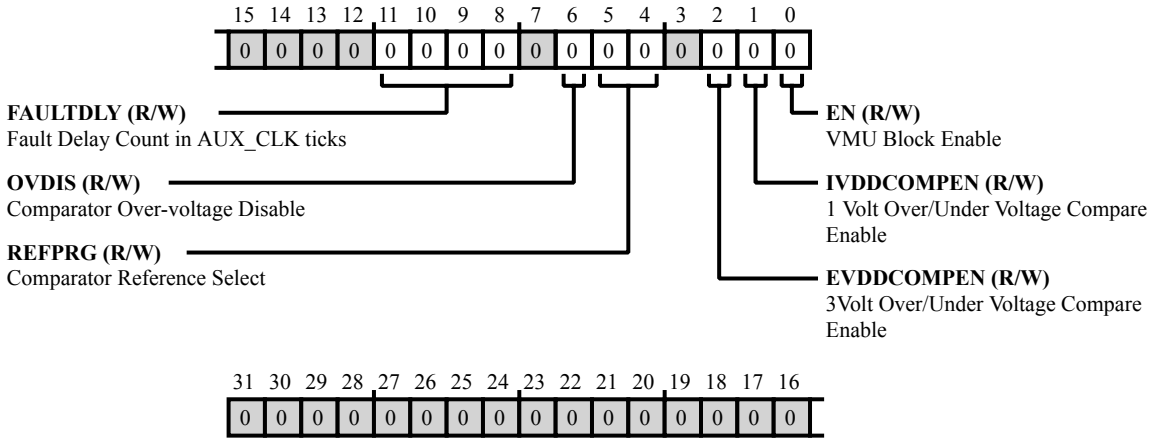


Figure 43-90: PADS_VMU_CTL Register Diagram

Table 43-100: PADS_VMU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:8 (R/W)	FAULTDLY	Fault Delay Count in AUX_CLK ticks.
6 (R/W)	OVDIS	Comparator Over-voltage Disable.
5:4 (R/W)	REFPRG	Comparator Reference Select.
2 (R/W)	EVDDCOMPEN	3Volt Over/Under Voltage Compare Enable.
1 (R/W)	IVDDCOMPEN	1 Volt Over/Under Voltage Compare Enable.
0 (R/W)	EN	VMU Block Enable.

Voltage Monitor Unit Trim Register

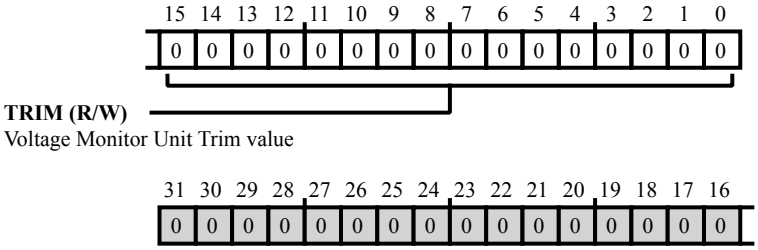


Figure 43-91: PADS_VMU_TRIM Register Diagram

Table 43-101: PADS_VMU_TRIM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	TRIM	Voltage Monitor Unit Trim value.

Voltage Monitor Unit Trip Enable Register

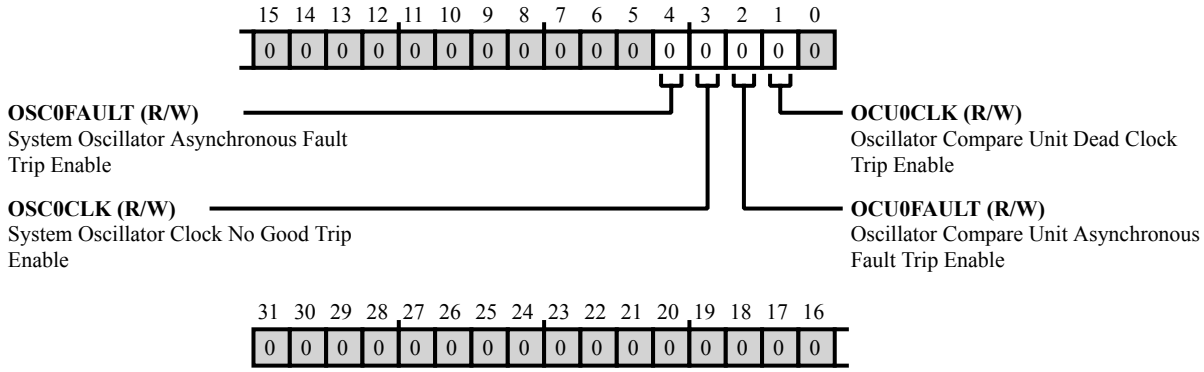


Figure 43-92: PADS_VMU_TRIPEN Register Diagram

Table 43-102: PADS_VMU_TRIPEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	OSC0FAULT	System Oscillator Asynchronous Fault Trip Enable.
3 (R/W)	OSC0CLK	System Oscillator Clock No Good Trip Enable.
2 (R/W)	OCU0FAULT	Oscillator Compare Unit Asynchronous Fault Trip Enable.
1 (R/W)	OCU0CLK	Oscillator Compare Unit Dead Clock Trip Enable.

44 Boot ROM and Booting the Processor

Bootstrapping or booting is the series of events that occur when the system applies power to the processor or when the processor enters a hardware or system reset state. This section gives an in-depth description of these events and how to integrate an application effectively.

The ADSP-CM41x processor implements a Logic ROM and a Boot ROM that executes from within the Cortex-M4 Platform. On reset, the control execution begins at the Logic ROM, which performs vital security functions. The logic ROM transfers control to the Boot ROM after performing an (optional) CRC of the ROM. The 8KB on chip ROM supports two boot modes and performs some initial device configuration. Two boot modes are supported- one executes previously loaded code from flash (note that the ARM Cortex-M0 application must be loaded by the user program), the other enables a flash recovery mode, enabling communication via UART in order to reprogram flash.

Booting an ARM Cortex-M0 Application

The ARM Cortex-M0 core does not have an in-built ROM similar to M4 for initiating a boot sequence to load an application. It is therefore the ARM Cortex-M4 application's responsibility to boot the M0's application.

On power-up, the ARM Cortex-M0 is in reset while code from the ARM Cortex-M4 flash memory is copied to the ARM Cortex-M0 RAM by the ARM Cortex-M4.When directed by the ARM Cortex-M4, the ARM Cortex-M0 leaves reset and runs its code from local ARM Cortex-M0 system RAM, which has been placed and verified under control of the ARM Cortex-M4.

A typical set of actions include the following:

- A utility to convert the ARM Cortex-M0 application image into a data file for inclusion into the ARM Cortex-M4 project can be found in CrossCore Utilities from Analog Devices. This utility can be used to automatically generate a hex file containing the complete dump and memory address of the application executable.
- The hex file should be included as part of ARM Cortex-M4's application. It can either be stored in the ARM Cortex-M4 SRAM or can be separately flashed so that ARM Cortex-M4 SRAM space is not used.
- Put the ARM Cortex-M0 under reset if needed, though after reset ARM Cortex-M0 is in the reset state. Load the ARM Cortex-M0 application to its memory from the Hex Image, in the example mentioned as M0_code. Release the ARM Cortex-M0 out of reset.

```
*pREG_RCU0_CRCTL = 0x00000002;
                adi_libldr_Move(M0_code);
                *pREG_RCU0_CRCTL = 0x00000000;
```

Boot Features

Key Features of the boot implementation include:

- Security facilities to protect sensitive IP located in the on-chip flash
- Secure programmer for provisioning new content to the on-chip flash via the UART
- CRC-32 protected Array Boot ROM
- Initialization of all on-chip ECC protected SRAM
- Support for early initialization of MMRs for time sensitive operations
- Support for early function execution for more complex time sensitive operations
- Optional CRC-32 verification of the first 4KB of on-chip flash

Boot Process Overview

The ***ADSP-CM41x Boot Process Flow Diagram*** figure shows the booting process with both BMODE options.

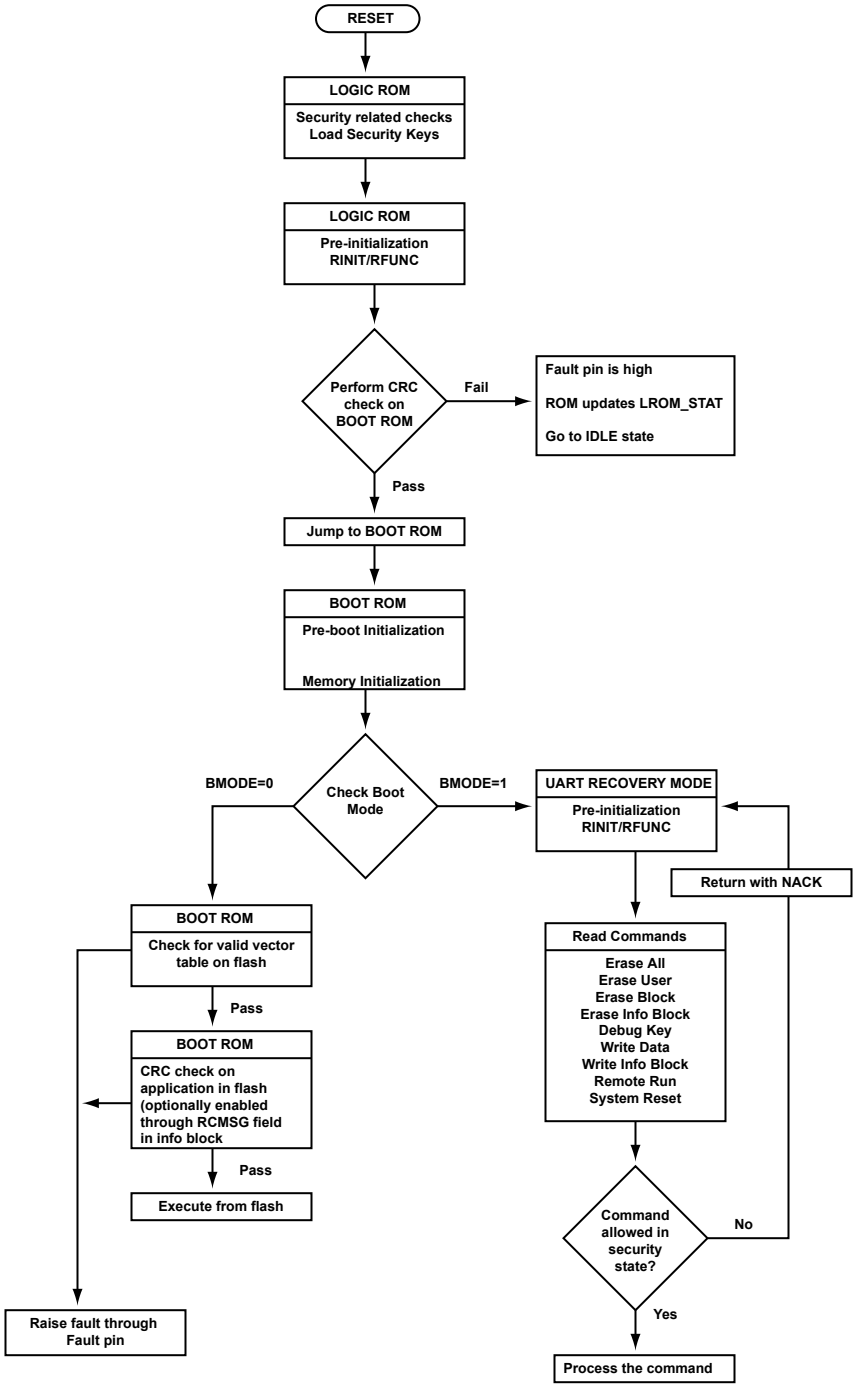


Figure 44-1: ADSP-CM41x Boot Process Flow Diagram

Logic ROM

The purpose of the logic ROM is to ensure that the first few instructions following a deassertion of reset can perform vital security functions, even in the presence of conditions that may disrupt the ROM memory.

- The logic ROM defines basic vector handlers for any non maskable interrupts such as NMI and hard fault.

- Prior to passing control to the boot ROM the logic ROM performs a CRC32 check on the boot ROM to validate the integrity of the ROM. The CRC32 check of the boot ROM image can be disabled by setting the TAPC_RCMSG.NOCRC bit in the info block of the flash.
- The logic ROM, in the event of a failure, branches to an error routine to signal a fault on the SYS_FAULT pin. The error routine writes an error code to the LROM_STATUS register as a means of indicating the source of the error.
- The logic ROM supports early-initialization features which allow either a list of MMRs to be loaded with user defined values (for example to configure the logic block or to set GPIO states), or to call a small user function (if more complex operations are required.) This allows very low latency initialization after reset (less than 100 µs). See Pre-initialization with Logic ROM for more information.

Boot Modes

Two boot modes are supported. One executes previously loaded code from flash, the other enables a flash recovery mode, enabling communication via UART in order to reprogram flash.

Boot Modes Table

Table 44-1: Booting Modes

BMODE[0]	Boot Source	Description
0	Execute from Flash	The processor executes some initial start-up checks from the Logic Boot ROM and the Array Boot ROM. If successful, reads application entry from the vector table located at the base of the flash memory then vectors to the entry location.
1	UART Flash Programmer	Allows for programming of the on-chip programmable flash via the UART peripheral

Boot ROM Pre-boot Routine for Device Initialization

The following steps are performed when execution is passed to the Boot ROM.

1. The Boot ROM performs SRAM configuration. Code and data banks are set to the default configuration of 2 code banks and the remaining data banks.
2. The Boot ROM initializes the MBOX memory and also initializes all the SRAM memory. This feature may be disabled by the user by setting a flag in the TAPC_RCMSG field in the info block of the flash.
3. The Boot code does not release SYS_RESOUT via the RCU. It is the responsibility of the user to release SYS_RESOUT from within their application stored in flash.
4. The boot code does not perform any Memory Built in Test operations.
5. Interrupts are not supported during the boot process therefore any interrupt that may occur and is enabled will result in the Boot ROM entering the error handler and a fault notification signaled via the SYS_FAULT pin.

6. Errors detected by Boot code during the booting process are captured in the`SYSBLK_LROM_STAT` register as the boot error handler is passed an error code and writes to a field in the MMR.
7. Boot code performs all device calibration. This step can be bypassed by programming the `TAPC_RCMSG` field in the info block of the flash.
8. The program can halt boot by setting `TAPC_RCMSG.HALTONENTRY` bit. This results in execution of a WFI instruction.
9. On completing the boot process successfully, the Boot ROM indicates the same by setting the `TAPC_RCMSG.BOOT_DONE` bit. (TAPC: JTAG TAP Controller)
10. The CGU operates in the bypass mode by default, and this is not altered by the boot ROM.
11. The boot code error handler raises a fault to indicate any errors occurring during boot. The errors are also indicated through the $\overline{\texttt{SYS\_FAULT}}$ pin.
12. The Boot ROM does not enable the $\overline{\texttt{SYS\_FAULT}}$ pin to be a fault input. Hence the processor cannot be used to detect incoming faults until enabled by user software.
13. The boot software applies the trim values to the Oscillator Watchdog. Also the boot ROM enables the Oscillator Watchdog fault.
14. Recovering from uncorrectable ECC errors in Security Keys

    If an ECC error occurs when the code located in the Logic Boot ROM reads the initial key from the flash info block and error handler is immediately entered and a fault raised. The UART recovery mode cannot be used to recover from this type of event as a result as execution does not continue in the Boot ROM. The only means to recover from an uncorrectable ECC error when reading the keys from flash is to perform a mass erase of the flash via the debug port.

Pre-initialization with Logic ROM

The Logic ROM is tightly connected to the BOOT ROM and supports early-initialization features which allow either a list of MMRs to be loaded (for example to configure the logic block array or to set GPIO states), or to call a small user function (if more complex operations are required.) This allows very low latency to initialization after reset.

Table 44-2: RINIT and RFUNC in Flash Info Blocks

Flash Block 0	Address Offset			
Address Base	0xC	0x8	0x4	0x0
0x1180_0FD0	Reserved	reserved	RCMSG	RCMSG_KEY
0x1180_0FC0	Reserved	RFUNC_CSIZ	RFUNC_PTR	RFUNC_KEY
0x1180_0FB0	RINIT_CNT	RINIT_CSIZ	RINIT_PTR	RINIT_KEY

The functions in the ***RINIT and RFUNC in Flash Info Blocks*** table are described below.

RCMSG_KEY. If this location is programmed to the value RCMSG_KEY_VAL, then run-control options are taken from bits in the RCMSG location. The TAPC_RCMSG register is described in the JTAG debug and Serial Wire Debug Port (SWJ-DP) Chapter. The RCMSG_KEY_VAL = 0x544D 5347

RCMSG. This word contains run-control option bits which include whether to immediately halt, to initialize memories and other hardware. The function of this word is identical to the TAPC_RCMSG register (the two locations are OR'ed together).

RINIT_KEY. If this word is set to the value RINIT_KEY_VAL, then the early register-initialization operation is selected. This feature is explained in following sections. The RINIT_KEY_VAL = 0x3744 CA0C.

RINIT_PTR. Points to an array of address-value pairs stored anywhere in flash memory (main arrays or info block 0). The pointer must be 32-bit aligned. Access violations resulting from bad pointer or count cause a HardFault exception, which asserts SYS_FAULT and wait forever in IDLE.

```
typedef struct {
    uinit32_t *mmr_ptr;
    uinit32_t mmr_val;
    } rinit_pair;
```

RINIT_CNT. Specifies the length of the array of rinit_pairs pointed to by RINIT_PTR. Must be nonzero.

RINIT_CSZ. Specifies whether the rinit_pair array should be CRC-checked. If 0, CRC checking is not performed. If nonzero, then the specified number of 32-bit words starting at RINIT_PTR is CRC checked.

RFUNC_KEY. If this word is set to the value RFUNC_KEY_VAL, then the early initialization-function operation is selected. The RFUNC_KEY_VAL = 0x3049 470D.

RFUNC_PTR. Points to a function stored anywhere in flash memory (main arrays or info block 0). The function pointer must be 32-bit aligned. The function must restrict stack usage to a 256 byte region which is initialized by the logic ROM. The rest of main SRAM is not initialized, and access to it may cause ECC errors. Access violations resulting from bad pointer or by exceptions from memory accesses within the function will cause a HardFault exception, which asserts the SYS_FAULT signal and wait forever in IDLE. The function must match the following prototype:

```
void rfunc_function(void);
```

The pointer value must have the bit[0] =1, as the function is executed as a jump operation to that address to conform to the Thumb2 requirement from ARM on executing a branch instruction.

RFUNC_CSZ: Specifies whether the rfunc_function body should be CRC-checked. If 0, CRC checking is not performed. If nonzero, then the specified number of 32-bit words starting at RFUNC_PTR is CRC checked.

Programming Example RINIT

As an example for the use case of RINIT feature, RINIT can be used to perform SRAM configuration as required by the application. Note that by default Boot Rom does SRAM configuration, and hence it will override the configuration done by RINIT. Hence one should also set the NOL2CONFIG bit in the RCMSG field in flash. (Please refer to section Control Flags for Booting and the section TAPC_RCMSG for more details.)

```
unsigned int RCMSG_KEY_ADDR = 0x11800FD0;
unsigned int RCMSG_KEY_VAL = 0x544D5347;
unsigned int RCMSG_ADDR =  0x11800FD4;

unsigned int RINIT_KEY_ADDR = 0x11800FB0;
unsigned int RINIT_PTR_ADDR = 0x11800FB4;
unsigned int RINIT_CSIZ_ADDR = 0x11800FB8;
unsigned int RINIT_CNT_ADDR  = 0x11800FBC;

unsigned int RINIT_KEY_VAL = 0x3744CA0C;
unsigned int RINIT_PTR_VAL = BUFFER_BASE;
unsigned int RINIT_CNT_VAL = REGISTER_COUNT;
unsigned int RINIT_CSIZ_VAL = NO_CRC_CHECK;

unsigned int* pRINIT_KEY_VAL = &RINIT_KEY_VAL;
unsigned int* pRINIT_PTR_VAL = &RINIT_PTR_VAL;
unsigned int* pRINIT_CNT_VAL = &RINIT_CNT_VAL;
unsigned int* pRINIT_CSIZ_VAL = &RINIT_CSIZ_VAL;

//define the srtucture of {MMR pointer, MMR value} to be placed in the specific
location in flash
#pragma location = ".flash_reserved"
   struct FLASH_RINIT_PAIR{
    volatile uint32_t * ptr;
    int value;
   } RINT_PAIR[REGISTER_COUNT] = {
          //{MMR pointer, MMR value     }
           { pREG_M4P_SRAM_CFG, SRAM_CFG_MSK}
          };

//The following function writes the RNIT structure and Key values to the
respective flash locations.
Void Func()
{
//Write to info block RCMSG field and RCMSG key to set the NOL2CONFIG bit in
RCMSG.
CM41xFlash_WriteBuffer(FLASH_CONTROLLER_A, RCMSG_KEY_ADDR,1, RCMSG_KEY_VAL);
CM41xFlash_WriteBuffer(FLASH_CONTROLLER_A, RCMSG_ADDR,1, RCMSG_VALUE_MASK);

//Write to the info block0 fields {RINIT_KEY, RINIT_PTR, RINIT_CSIZ, RINIT_CNT}
CM41xFlash_WriteBuffer(FLASH_CONTROLLER_A, RINIT_KEY_ADDR,1, pRINIT_KEY_VAL);
CM41xFlash_WriteBuffer(FLASH_CONTROLLER_A, RINIT_PTR_ADDR,1, pRINIT_PTR_VAL);
CM41xFlash_WriteBuffer(FLASH_CONTROLLER_A, RINIT_CSIZ_ADDR,1, pRINIT_CSIZ_VAL);
CM41xFlash_WriteBuffer(FLASH_CONTROLLER_A, RINIT_CNT_ADDR, 1,pRINIT_CNT_VAL);
}
```

NOTES:

1. `CM41xFlash_WriteBuffer()` is only a sample API to write to flash location. For the actual API, please refer to the BSP (Board Support Package) for the processor.
2. `SRAM_CFG_MSK` is the value required to be programmed into the M4P_SRAM_CFG register.
3. RCMSG_VALUE_MASK is the value required to be programmed into the RCMSG field in flash.

Restrictions on RINIT and RFUNC early runtime Initialization

The RINIT MMR-list operations and the user-defined RFUNC function operate in a severely restricted environment, as almost no system initialization takes place when they are processed.

- The LDO and OSCWDOG have not yet been trimmed to factory tolerances. This is performed later by the boot ROM.
- The power state of the CGU and DPU must not be changed
- The state of the Reset Control Unit (RCU) must not be changed
- The Oscillator Watchdog may not be configured or enabled
- No interrupts may be enabled in the NVIC
- The ARM Cortex-M0 supervisor may not be brought out of reset
- No system memory (main SRAM) may be accessed in a manner which might cause ECC errors. Full-width aligned 32-bit writes may be performed, and reads from locations may be performed if they have been previously initialized. Note that for convenience, 256 bytes of stack are initialized for the use of local variables and temporaries in RFUNC functions.

Flash Execution Mode

The Flash Execution Mode allows for application execution from the on-chip flash. Before executing from the on-chip flash the boot process performs a validation value check and optionally perform a CRC-32 of the first 4KB of the on-chip flash.

In order to support execution for the on-chip flash, the core compatible vector table must be located at `0x11000000`. The second entry of the vector table specifies the reset vector address for the application. Prior to branching to the applications reset vector address, a basic value check is performed to help ensure that the on-chip flash is provisioned with a valid vector table and user application. There is also an additional CRC-32 operation that can be enabled for increased protection. If no valid code is found, or the CRC-32 fails when enabled, the error handler is entered.

Value Check

The basic value check is always performed. The requirement is that the eighth 32-bit entry in the vector table must be programmed with the value 0xAD12 21AD. If the boot process reads the value as required, the reset vector address is then read from the second entry in the vector table and then a branch to that address takes place.

Error On no Valid Vector Table

If there is no valid vector table in flash, an error occurs that generates the BCODE_ERR fault. This fault also occurs if the flash is blank or uninitialized. (Flash states are described in the Security chapter).

Optional CRC-32

An optional CRC check is performed on the first 4096 bytes of Flash memory. This feature is enabled by setting `TAPC_RCMSG.ACRC`. The calculated CRC value must match the expected compare value. The application image must be patched to ensure the expected value is calculated.

The CRC-32 feature requires the use of the stack and as such memory initialization should not be disabled during the preboot phase. The expected CRC-32 result is always the same value, thus a 32-bit patch value must be placed within the first 4096 bytes of memory such that the CRC-32 result of the 4096 byte block of data equals the expected value as described below.

The CRC peripheral is configured to implement bit reversing and byte mirroring when calculating the CRC and as such this needs to be taken into account when calculating the CRC-32 patch value.

CRC-32 Polynomial loaded to CRC peripheral	0xBA0DC66B
CRC-32 Polynomial as used by host software (bit reversed byte mirror)	0xD663B05D
CRC-32 Seed	0x00000000
CRC-32 Expected Result	0xCBECE537

IAR Electronic Workbench includes a utility for generating and patching a CRC-32 checksum into an application image. The following command line will embed the required 32-bit patch value at location 0x1100 0FFC such that it is compliant with the CRC functionality as implemented during the boot process.

```
ielftool.exe application.out application_patched.out
--fill 0x00;0x11000000-0x11000FFF
--checksum 0x11000FFC:4,crc=0xD663B05D:1mi,0x00000000;0x11000000-0x11000FFB
```

UART Recovery Mode

The UART Recovery Mode allows for the on-chip flash to be programmed from data sent over the UART peripheral from a host.

The data received is required to be in a specific format that allows data to be consumed via the UART peripheral and processed by the flash programmer. In addition to supporting programming of the on-chip flash, the implementation also accommodates programming of content to the on-chip SRAM. This boot mode is intended to assist with point to point firmware updates between the host and the platform without the need for connecting JTAG/SWD debug tools.

NOTE: In order to preserve the security aspects of the processor the UART Recovery Mode is limited in some of the functionality it can perform. In particular it does not support any operations that allow for the

contents of the on-chip flash to be read out via the UART and a limited command set is initially supported until the required secure key is received. There is no encryption supported on the key transfer via the UART and thus precautions are required to ensure the security of the connection between the host and the processor.

The UART requires an initial autobaud detection character to be sent from the host in order to establish connection with the processor. The host may then continue to issue the supported command sequences to the device to perform a number of erase and programming operations to load firmware onto the processor for execution. The command set supported is determined by the security state of the product.

The recovery mode is capable of loading application code and data to both the erasable and programmable flash memory and /or to internal SRAM. This can be leveraged to implement a secondary bootloader executed from SRAM that can program the flash with data accessed via another peripheral other than the UART or via a user defined protocol from an alternative boot source.

The UART packet transfers are all protected with CRC-32 checksums.

Hardware Configuration

Describes the UART peripheral configuration for communication.

The UART3 peripheral mode of operations is configured as follows once the initial auto-baud detection process has completed.

- CTS/RTS handshaking disabled
- Active low CTS/RTS
- Parity disabled
- Single stop bit
- 8-bit word length

Autobaud Detection

Autobaud detection establishes initial communication and synchronizes the host with the processor for communication via the UART. In order for the UART peripheral to synchronize to the host baud rate. The host is required to initially send a 0x40 byte ("@") over the serial communication channel.

ID Packet

In response to the autobaud character being received, a 25-byte ID packet is sent back to the host. This ID packet is of the following form:

- 10-byte product ID code 'ADSP-CM41x'
- 2-byte line feed and carriage return
- 4-byte firmware version number 'V100'

- 2-byte line feed and carriage return
- 6-bytes reserved for future use
- 1-byte XOR checksum of the previous 24-bytes

Data Transport Packet Format

The data transport packet describes the command, address, associated payload and the CRC-32 checksum format required for a host to communicate with the UART recovery mode.

The data transport packet consists of the following fields:

- 2-byte Start ID field
- 2-byte Number of Bytes field
- 1 to 4101-byte Data Bytes field that can support a single byte command, 4 byte address field (optional depending on the command) and up to 4096 data bytes.
- 4-byte CRC32 field

ATTENTION: In order to meet alignment requirements of the flash programmer which supports ECC protection, data targeted for the flash should be supplied in a multiple of 8 bytes and issued on an 8-byte aligned address. Data targeted towards SRAM has no byte or address alignment requirements.

Start ID Field

The start ID consist of two bytes that indicate the start of a valid data packet. These bytes are constant and must be supplied at the beginning of every packet transfer. The start ID field is received in little endian form, with Start ID byte 0 issued first followed by Start ID byte 1.

- Start ID byte 0 = 0x07
- Start ID byte 1 = 0x15

Number of Bytes Field

This is a two byte field specifying the number of bytes in the data packet that follows. The bytes are received in little endian form with the lower byte sent first followed by the upper byte.

The protocol can support packet payloads up to 4101 bytes in size. This is to help optimize the performance of the programmer by allowing for a full 4096 byte flash page to be programmed in a single operation. The 4101 byte payload allows for the addition of a single byte command and a 4-byte address to be issued along with the data to be programmed.

NOTE: The 4-byte CRC-32 result that is appended to the end of the Data Bytes field does not contribute the number of bytes set in this field. Thus if a data block consists of a single byte command, 4 address bytes, 1024 data bytes and the 4-byte CRC-32 result. The number of bytes field would be set to 1029 bytes.

Data Bytes Field

The protocol supports data byte payloads of 1 up to 4101 bytes. 4101 bytes allows for a single byte command, a 4 byte address and up to 4096 data bytes. The entire data field is protected by a CRC-32 checksum.

The minimum data byte size supported is a single byte. The first byte of the data field is the command to be processed. The bytes that follow the command are determined by the command itself. All data bytes are received in little endian form starting with the single byte command. Refer to Commands.

CRC-32 Checksum Field

The 4-byte CRC-32 checksum field contains the CRC-32 result of the data bytes field.

When the CRC engine and MDMA channel receive the data bytes field they are configured to generate and compare the CRC of the received data. If the CRC passes then the packet data is processed. Otherwise a NACK response is sent back to the host.

Upon receipt of a data bytes field the CRC engine and MDMA channel are configured to generate and compare the CRC of the received data. If the CRC passes then the packet data is processed otherwise a `NACK` response is sent back to the host.

The CRC peripheral is configured to implement bit reversing and byte mirroring when calculating the CRC and as such this needs to be taken into account when calculating the CRC-32 checksum on the host.

CRC-32 Polynomial loaded to CRC peripheral	0xBA0DC66B
CRC-32 Polynomial as used by host software (bit reversed byte mirror)	0xD663B05D
CRC-32 Seed	0x00000000

UART Response Types

The processor supports three response types that consist of a single byte to signal to the host whether the last operation requested was successful:

- 0x06 response for successful acknowledgment of a command (`ACK`)
- 0x07 response to indicate the command was not processed (`NACK`)
- 0x08 response to indicate that the command was not processed due to a security restriction (`Secure NACK`)

If a command is attempted without sufficient security privileges, a `Secure NACK` response is returned.

The responses are sent back only after the complete operation has either passed or failed and not as soon as the processor receives the full data transport packet.

NOTE: The erase and mass erase times of the flash device are specified as 130 ms maximum. It is recommended that any host utility use this as a basis as a minimum timeout for a response to any given data transport packet

All commands from the full command set when privileges are granted return either an ACK or a NACK response.

Commands

The protocol support a number of different commands to instruct the boot software to perform different operations. The command is issued in the data field of the data transport packet. Each command has its own requirements on the data that must follow the command.

Command Set

Table 44-3: Commands

Command	Command Code	Description
Erase All	0x41 ("A")	Erases the entire flash device
Erase User	0x55 ("U")	Erases the complete user area of the flash device while preserving the flash info block
Erase Block	0x45 ("E")	Erases the addressed flash block
Erase Info Block	0x49 ("I")	Erases the addressed flash info block
Debug Key	0x44 ("D")	Sends the debug key to allow for flash and SRAM programming when the device is in a secured state
Write	0x57 ("W")	Writes data to the flash user array or SRAM
Write Info Block	0x50 ("P")	Writes data to the address flash info block
Remote Run	0x52 ("R")	Executes a loaded application from flash or SRAM while keeping the device in the existing boot mode.
System Reset	0x71	Executes a system reset via the RCU

Erase All

Erases the entire flash memory user array and flash info blocks.

The data block of this command consists of the single 0x41 command byte command followed by a 32-bit confirmation key that is received in little endian format.

The Erase All command results in a complete flash device erase operation being performed if the confirmation key following the command is 0xAD47 3128.

The CRC-32 result for the 5 byte data block follows the confirmation key.

The following is an example data sequence issue by the host for the complete data transport packet:

```
0x07, 0x15, 0x05, 0x00, 0x41, 0x28, 0x31, 0x47, 0xAD, 0x1D, 0x33, 0x6F, 0x1B
```

Erase User Area

Erases the entire user area space of the flash memory. The info block is preserved and secure debug access remains intact with the provisioned keys.

This is a single byte command with no further associated payload followed by the CRC-32 checksum.

The entire user flash is erased by issuing a mass erase operation to the flash controller. The FLC_ADDR.IRFSEL and FLC_CTL.IRFWEN bits are cleared to preserve the contents of the flash information blocks.

Erase Block

Erases the addressed block of the user area of flash. The info block cannot be erased using this command. Secure debug access is preserved using the provisioned keys.

This is a 5 data byte command sequence consisting of the 0x45 command byte followed by 4 address bytes indicating the address of the block to erase. The 5 bytes are required to be followed by the CRC-32 checksum.

This command results in a block erase operation being issued to the flash controller.

Erase Info Block

Erases the addressed info block of the flash memory.

This is a single byte command followed by a 32-bit address that must address the info block to be erased. The address is then followed by the CRC-32 result of the previous 5 bytes.

The info block is erased by issuing a block erase operation to the flash controller. This erase command preserves the contents of all blocks in the main user area of the flash.

Debug Key

Issues the debug key to open up the full command set when security is enabled.

This is a single byte command followed by the 16 bytes containing the 128-bit security key. The key must be transmitted least significant byte first. The key received is then compared to the key in the flash info block. A match results in the full command set then being enabled for subsequent command sequences.

The CRC-32 result for the 17-byte data byte payload is required to follow the last byte of the security key.

Write

Writes the required data to the flash array or SRAM.

This is a single byte command followed by a 32-bit address and the data to be written at the supplied address. The CRC-32 result is required to follow the last byte of data to be programmed.

ATTENTION: In order to meet alignment requirements of the flash programmer which supports ECC protection, data targeted for the flash should be supplied in a multiple of 8 bytes and issued on an 8-byte aligned address. Data targeted towards SRAM has no byte or address alignment requirements.

Write Info

Programs data to the addressed info block.

This is a single byte command followed by a 32-bit address specifying the address in the info block that the program operation is targeted for. The data to be programmed follows the address.

When processing this command, there are no restrictions to what area of the info block can be programmed or size limitations as long as the size does not exceed the limits of the info block itself.

This command comes with unique security restrictions for two locations in the info block.

When the device is locked the command is not processed.

If the flash performed a mass erase operation, the contrast and user key are in a blank erase state. In this state the boot software detects that the flash is uninitialized and only allows the programming of the 128-bit contrast key to the two info blocks, attempting to program any other data to any other region is rejected. The contrast key is required to be written to both flash controller info blocks in order to be provisioned correctly.

Once the contrast keys are programmed and a system reset is performed the security state of the device changes from uninitialized to blank. This state indicates that the contrast key is valid but the user key is blank. Writes to any location in the info block are supported in this state allowing for provisioning of the user key.

The CRC-32 result must always follow the last byte of the data block to be programmed.

Remote Run

Instructs the boot code to vector to the user supplied address to start executing code from that location

This is a single byte command followed by a 32-bit address and the CRC-32 result. The address instructs the boot kernel where to vector to. This command can be used to execute a second stage loader from SRAM for example.

The processor branches directly to the address provided so the host must ensure that the address supplied is compliant with the core architecture in regards to instruction execution mode.

System Reset

Instructs the boot code to perform a system reset

This is a single byte command with no further payload expected followed by the CRC-32 result. Upon receiving this command the boot software executes a system reset event by setting the `RCU_CTL.SYSRST` bit.

The response from the processor back to the host is issued before the system reset event is generated and a time delay implemented between the issuing of the response and the system reset event to ensure the response is transmitted and can be received by the host before the reset event occurs. This gives the host indication that the reset request was received correctly.

After a system reset occurs, unless the `SYS_BMODE0` pin changes state, then the UART Recovery mode is re-entered. This requires the host to perform the autobaud sequence in order to re-establish communication with the processor.

The purpose of this command is primarily for use when provisioning items such as the contrast keys where a reset is required in order for the new security state to become active.

Security

In particular security states, commands are blocked, this section describes the security states and allowed commands in each of those security states.

In order to preserve the security of the device, certain commands are only allowed in certain security states. The security states supported by the boot implementation are categorized as follows:

1. Uninitialized — contrast and key fields are empty.
2. Blank — contrast field is valid, key fields are empty.
3. Locked — contrast and key fields are valid and no user key has been supplied via the Debug Key command.
4. Opened — contrast and user key fields are valid and a key has been provided over the UART via the Debug Key command matching the key in flash

NOTE: The locked and opened states listed above are not actual physical firewall states. It is how the boot rom perceives the state to then allow for the restricted or full command set to be executed. The key that is supplied with the Debug Key command only enables the processing of the full command set of the boot mode. The key cannot be used to enable debug access for debug tools which are authenticated separately.

The following table describes which commands are available in the various states, X indicates the command is allowed.

NOTE: The Write Info Block command supports unique functionality for provisioning of contrast and user keys. The address supplied with the command dictates the type of operation to be performed. Refer to the documentation for the Write Info command for full details.

Command	Uninitialized	Blank	Locked	Open
Write Info Block (Contrast Key address)	X	X		X
Write Info Block (User Key address)		X		X
Write Info Block		X		X
Write		X		X
System Reset	X	X	X	X
Debug Key			X	X
Erase Block		X		X
Erase Info Block		X		X
Erase All	X	X	X	X
Remote Run		X		X

Error Codes

The boot process is capable of detecting a number of error conditions. Some error conditions may be detected during execution of the boot software.

When an error is detected during execution of the boot software in the Array Boot ROM, an error code is written to the SYSBLK_LROM_STAT.FLT_BROM bits. The error descriptions for this field are presented below.

NOTE: The error code field is only applicable in the event of a boot error occurring. The default reset state of this field is indicative of an error and is thus not applicable for successful boot operations.

Error	Value	Description
ROM_BOOT_FAILURE	0	General Error
RESERVED	1	Reserved
ROM_BOOT_IMG_VER_FAILURE	2	The Application Image was not able to be verified, ensure the special code is inserted into the vector table.
ROM_BOOT_DMA_FAILURE	3	DMA Failed
ROM_BOOT_CRC_FAILURE	4	CRC-32 on the application failed. Ensure the image complies with the CRC-32 requirements.
ROM_BOOT_MEMINIT_FAILURE	5	Memory initialization failed
ROM_BOOT_MBOXINIT_FAILURE	6	MBOX memory initialization failed
ROM_BOOT_FLASH_FAILURE	7	Flash operation failed
ROM_BOOT_FLASH_INVALID_ADDRESS	8	An attempt to access an invalid flash address was made
ROM_BOOT_FLASH_FAILED_WRITE	9	Failed to write to the flash memory.
ROM_BOOT_DMA_ACTIVE	10	DMA was active when attempting to perform a new DMA operation
ROM_BOOT_MDMA_ID_ERR	11	Invalid MDMA channel ID
ROM_BOOT_MDMA_OPERATION_ERR	12	MDMA Operation error
ROM_BOOT_MDMA_SRC_ERR	13	MDMA Source channel error
ROM_BOOT_MDMA_DST_ERR	14	MDMA Destination channel error
ROM_BOOT_CRC_CONFIG_ERR	15	CRC Peripheral Configuration error

Callable Kernel API

This section describes the kernel APIavailable at run-time.

The boot code stored in ROM exposes several functions that can be used during run-time or within Initcode or callback routines. All functions meet the C run-time calling conventions. Addresses provided for the documented API routines have the least significant bit set to meet the Thumb execution mode calling requirements.

adi_rom_getID()

This API can be used to read the manufacture ID of the chip. Note that this ID is fixed and does not vary processor family products.

Name	adi_rom_getID	
PP Define	FUNC_ROM_PREBOOT_GETID	
ROM Address	0x000100A9	
Prototype	uint32_t adi_rom_getID (void)	
Argument	none	
Return Value	uint32_t	Manufacture Id of the chip (i.e 0x65).

adi_rom_Crc32Poly()

Initializes the CRC Look-up table using the supplied CRC polynomial. The routine loads the supplied CRC polynomial to the CRC peripheral and starts a LUT generation operation. The CRC peripheral is configured for Memory Scan Compute and Compare mode during the process.

Name	adi_rom_Crc32Poly()	-
PP Define	FUNC_ROM_CRCLUT	-
ROM Address	0x00010081	-
Prototype		-
Argument	CrcPoly	CRC32 Polynomial
Argument	pDma	
Return Value	ADI_ROM_BOOT_RESULT	#ADI_ROM_BOOT_SUCCESS If the operation is successful. #ADI_ROM_BOOT_CRC_SUPPORTED_ERR If CRC is not supported by the MDMA Stream.

adi_rom_MemCompare()

This API Compares a specified region of memory against a provide 32-bit reference value. The CRC engine is placed in Data Verify Memory Scan mode. The destination buffer is not used. In Non-Blocking mode, the application must check the Compare Error bit of CRC to determine whether the CRC Compare Error has occured or not.

Name	adi_rom_MemCompare	-
PP Define	FUNC_ROM_MEMCOMPARE	-
ROM Address	0x00010091	-
Prototype	ADI_ROM_BOOT_RESULT adi_rom_MemCompare (ADI_ROM_DMA_MDMA_CONFIG * pDmaCfg, ADI_ROM_BOOT_MDMA_REGS const *const pDma)	-
Argument	pDmaCfg	Pointer to the user configurable structure for controlling the MDMA operation.

Argument	pDma	Pointer to the MDMA Channel Registers.
Return Value	ADI_ROM_BOOT_RESULT	#ADI_ROM_BOOT_SUCCESS If the operation is successful. #ADI_ROM_BOOT_CRC_SUPPORTED_ERR If CRC is not supported by the MDMA Stream. #ADI_ROM_BOOT_DMA_FAILURE If the DMA configuration error (if done detect method is ADI_ROM_DMA_DONE_NON_BLOCKING). #ADI_ROM_BOOT_CRC_FAILURE If the CRC compare error is raised (if done detect method is ADI_ROM_DMA_DONE_POLL_IRQDONE). In Non-Blocking mode the application has to check the CRC_STAT_CMPERR bit to determine whether CRC compare error is raised or not.

adi_rom_MemCopy()

Name	adi_rom_MemCopy	-
PP Define	FUNC_ROM_MEMCOPY	-
ROM Address	0x00010071	-
Prototype	ADI_ROM_BOOT_RESULT adi_rom_MemCopy (ADI_ROM_DMA_MDMA_CONFIG * pDmaCfg, ADI_ROM_BOOT_MDMA_REGS const *const pDma)	-
Argument	PDmaCfg	Pointer to the user configurable structure for controlling the MDMA operation.
Argument	pDma	Pointer to the MDMA Channel Registers.
Return Value	ADI_ROM_BOOT_RESULT	#ADI_ROM_BOOT_SUCCESS If the operation is successfull. #ADI_ROM_BOOT_DMA_FAILURE If the DMA configuration error (if done detect method is ADI_ROM_DMA_DONE_NON_BLOCKING). ADI_ROM_BOOT_MDMA_SRC_ERR If the source MDMA configuration error (if done detect method is ADI_ROM_DMA_DONE_POLL_IRQDONE). ADI_ROM_BOOT_MDMA_SRC_ERR If the destination MDMA configuration error.

adi_rom_MemCrc()

This API can be used to compute the CRC of the given source buffer(Note that the destination buffer is not used) and also compare the result with the given value. This API configures the CRC engine in Memory Scan Compute/Compare mode. The computed value is compared with the value passed in CrcCompare field of the MDMA configuration structure.

Name	adi_rom_MemCrc	-
PP Define	FUNC_ROM_MEMCRC	-
ROM Address	0x00010079	-
Prototype	ADI_ROM_BOOT_RESULT adi_rom_MemCrc (ADI_ROM_DMA_MDMA_CONFIG * pDmaCfg, ADI_ROM_BOOT_MDMA_REGS const *const pDma)	-
Argument	pDmaCfg	Pointer to the user configurable structure for controlling the MDMA operation.
Argument	pDma	Pointer to the MDMA Channel Registers.
Return Value	ADI_ROM_BOOT_RESULT	#ADI_ROM_BOOT_SUCCESS If the operation is successful. #ADI_ROM_BOOT_CRC_SUPPORTED_ERR If CRC is not supported by the MDMA Stream. #ADI_ROM_BOOT_DMA_FAILURE If the DMA configuration error (if done detect method is ADI_ROM_DMA_DONE_NON_BLOCKING). #ADI_ROM_BOOT_CRC_FAILURE If the CRC compare error is raised (if done detect method is ADI_ROM_DMA_DONE_POLL_IRQDONE). In Non-Blocking mode the application has to check the CRC_STAT_CMPERR bit to determine whether CRC compare error is raised or not.

adi_rom_MemFill()

This API can be used to fill a specified region of memory with a 32-bit value provided. The buffer to be filled is passed as the destination buffer. The source buffer is not used in this API.

Name	adi_rom_MemFill	-
PP Define	FUNC_ROM_MEMFILL	-
ROM Address	0x00010089	-
Prototype	ADI_ROM_BOOT_RESULT adi_rom_MemFill (ADI_ROM_DMA_MDMA_CONFIG * pDmaCfg, ADI_ROM_BOOT_MDMA_REGS const *const pDma)	-
Argument	pDmaCfg	Pointer to the user configurable structure for controlling the MDMA operation.
Argument	pDma	Pointer to the MDMA Channel Registers.

Return Value	ADI_ROM_BOOT_RESULT	#ADI_ROM_BOOT_SUCCESS If the operation is successfull. #ADI_ROM_BOOT_CRC_SUPPORTED_ERR If CRC is not supported by the MDMA Stream. #ADI_ROM_BOOT_DMA_FAILURE If the DMA configuration error

adi_rom_memoryDma()

This API can be used to perform various Memory DMA/CRC operations specified by the Operation field of the MDMA configuration structure.

Name	adi_rom_MemDma	-
PP Define	FUNC_ROM_MEMDMA	-
ROM Address	0x00010069	-
Prototype	ADI_ROM_BOOT_RESULT adi_rom_memoryDma (ADI_ROM_DMA_MDMA_CONFIG * pDmaCfg	-
Argument	pDmaCfg	Pointer to the user configurable structure for controlling the MDMA operation.
Return Value	ADI_ROM_BOOT_RESULT	#ADI_ROM_BOOT_SUCCESS If the operation is successful. #ADI_ROM_BOOT_CRC_SUPPORTED_ERR If CRC is not supported by the MDMA Stream. #ADI_ROM_BOOT_DMA_FAILURE If the DMA configuration error(if done detect method is ADI_ROM_DMA_DONE_NON_BLOCKING). #ADI_ROM_BOOT_CRC_FAILURE If the CRC compare error is raised(if done detect method is ADI_ROM_DMA_DONE_POLL_IRQDONE). In Non-Blocking mode the application has to check the CRC_STAT_CMPERR bit to determine whether CRC compare error is raised or not. ADI_ROM_BOOT_MDMA_ID_ERR If the memory DMA Id is invalid. ADI_ROM_BOOT_MDMA_OPERATION_ERR If the Memory DMA operation is invalid

Data Structures

ADI_ROM_DMA_MDMA_OPERATION

Enumeration of different MDMA operations used by Boot ROM API's.

ADI_ROM_DMA_MEM_COPY	Standard MDMA transfer from a source to a destination
ADI_ROM_DMA_MEM_CRC	Performs a CRC32 MDMA read operation and compares the result with an expected result
ADI_ROM_DMA_MEM_FILL	Uses the CRC peripheral to perform a fill operation with a 32-bit value
ADI_ROM_DMA_MEM_COMPARE	Uses the CRC peripheral to compare data with a constant 32-bit value

ADI_ROM_DMA_MDMA_ID

Enumeration of different MDMA channel ID's used by Boot ROM API's.

ADI_ROM_DMA_MDMA0	Memory DMA Stream 0
ADI_ROM_DMA_MEMDMA_END_COUNT	Number of Memory DMA Streams

ADI_ROM_BOOT_DMA_INSTANCE

Specifies the base MMR address of the DMA channel as well as trigger and interrupt IDs

```
typedef struct
   {
   volatile ADI_DMA_TypeDef *const pReg;
   DMA_CHANn_TypeDef eDmaChannelId;
   uint8_t TriggerId;
   uint8_t InterruptId;
   } ADI_ROM_BOOT_DMA_INSTANCE;
```

ADI_ROM_BOOT_MDMA_CRC_SUPPORT

Enumeration of whether CRC is supported by MDMA or not.

ADI_ROM_BOOT_DMA_CRC_SUPPORTED	DMA CRC supported
ADI_ROM_BOOT_DMA_CRC_NOT_SUPPORTED	DMA CRC not supported

ADI_ROM_DMA_DONE_DETECT_METHOD

Enumeration of different DMA detection methods used by Boot ROM API's.

ADI_ROM_DMA_DONE_NON_BLOCKING	Return without waiting for the DMA to complete
ADI_ROM_DMA_DONE_POLL_IRQDONE	Poll on the IRQDONE bit in the DMA Status register

ADI_ROM_DMA_MDMA_CONFIG

The user configurable structure for controlling the MDMA operation

```
typedef struct
    {
    ADI_ROM_DMA_MDMA_OPERATION eOperation;
    ADI_ROM_DMA_MDMA_ID eId;
    const void * pSource;
    void * pDestination;
    uint32_t ByteCount;
    ADI_ROM_DMA_DONE_DETECT_METHOD eDoneDetect;
    uint32_t CrcCtl;
    uint32_t FillVal;
    uint32_t CrcPoly;
    uint32_t CrcCompare;
    } ADI_ROM_DMA_MDMA_CONFIG;
```

ADI_ROM_BOOT_MDMA_REGS

Contains the Source and Destination MDMA channel instances for access to the MMRs and interrupt and trigger information. Information is also provided on the CRC support of the MDMA channel and access is provided to the corresponding CRC peripheral.

```
typedef struct
    {
    ADI_ROM_BOOT_DMA_INSTANCE Src;
    ADI_ROM_BOOT_DMA_INSTANCE Dst;
    volatile ADI_CRC_TypeDef *const pCrc;
    ADI_ROM_BOOT_MDMA_CRC_SUPPORT eCrcSupport;
    } ADI_ROM_BOOT_MDMA_REGS;
```

Control Flags for Booting

A number of flags are provided through the TAPC_RCMSG register at location 0x11800FD4 in the on-chip flash can be used to provide the flags by the user to control the boot process.

The 32-bit location prior to this value at location 0x11800FD0 must contain the key 0x544D5347. Only if this key is valid will the RCMSG at location 0x11800FD4 be valid for the boot ROM. The boot ROM OR's the value of the RCMSG at 0x11800FD4 in the flash and the TAPC_RCMSG register to be used as the flags to control the boot process. This allows users to enable selective processing of various preboot features depending on application requirements while also allowing debug tools when connected to manipulate the preboot process.

The bit fields of the TAPC_RCMSG register are explained in the CM41X_M4 TAPC Register Descriptions section.

45 System Watchpoint Unit (SWU)

The system watchpoint unit (SWU) is a single module used for transaction monitoring. The SWU is attached to each system slave through the system crossbar interface and provides ports for all address channel signals for the system crossbar. The SWU does not have ports for the read/write data channel signals or the low-power interface signals.

Each SWU contains four match groups of registers with associated hardware. These four SWU match groups operate independently, but share common event (interrupt and trigger) outputs. Each match group can monitor either the write or read address channel and can operate in either watchpoint mode or bandwidth mode.

SWU Features

The system watchpoint unit has the following features.

- Four independent match groups for each SWU
- Each match group can operate in either bandwidth mode or watchpoint mode

SWU Functional Description

This section describes the function of the SWU match block, interface block, and MMR block.

The ADSP-CM41x bus interconnect fabric provides the following functional safety features for the MMR access watchdogs, provided by the SWUs or the Cortex internal DWT unit.

Table 45-1: Bus Fabric Access Watchdogs (SWUs)

To (Slave)	From M4	From M0	From MDMA	From Ctrlr DMAs	From Spvsr DMAs
M4 ROMs	DWT	N/A	N/A	N/A	N/A
M4 SRAM	DWT	SWU6	SWU6	SWU6	SWU6
M4 Flash	DWT	SWU6	SWU6	SWU6	SWU6
M4 local peripherals (MATH, FLC, M4P)	DWT	N/A	N/A	N/A	N/A
SPU1 controls	SWU2	SWU0	SWU5	N/A	N/A

Table 45-1: Bus Fabric Access Watchdogs (SWUs) (Continued)

To (Slave)	From M4	From M0	From MDMA	From Ctrlr DMAs	From Spvsr DMAs
Ctrlr System MMRs	SWU2	SWU0	SWU5	N/A	N/A
FFTB memory	SWU3	SWU3	SWU3	SWU3	SWU3
SMC external memory	SWU4	SWU4	SWU4	SWU4	SWU4
MBOX Port1 and MMRs	SWU2	SWU0	SWU5	N/A	N/A
M0 SRAM	SWU1	N/A	SWU1	SWU1	SWU1
SPU0 controls	SWU2	SWU0	SWU5	N/A	N/A
Spvsr System MMRs	SWU2	SWU0	SWU5	N/A	N/A
MBOX Port 0 and MMRs	N/A	SWU0	N/A	N/A	N/A

CM41X_M4 SWU Register List

The System Watchpoint Unit (SWU) provides debug and development support through flexible transaction level and bandwidth monitoring and associated event triggering. The SWU can generate events based on monitoring transactions at the system slaves through watchpoint-match groups. The SWU also provides watchpoint event status reporting, a global lock, and processor reset capability. A set of registers governs SWU operations. For more information on SWU functionality, see the SWU register descriptions.

Table 45-2: CM41X_M4 SWU Register List

Name	Description
SWU_CNT[n]	Count Register n
SWU_CTL[n]	Control Register n
SWU_CUR[n]	Current Register n
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_HIST[n]	Bandwidth History Register n
SWU_ID[n]	ID Register n
SWU_LA[n]	Lower Address Register n
SWU_TARG[n]	Target Register n
SWU_UA[n]	Upper Address Register n

CM41X_M0 SWU Interrupt List

Table 45-3: CM41X_M0 SWU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
11	SWU0_EVT	SWU0 Event	None	
11	SWU1_EVT	SWU1 Event	None	

CM41X_M4 SWU Interrupt List

Table 45-4: CM41X_M4 SWU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
47	SWU2_EVT	SWU2 Event	None	
48	SWU3_EVT	SWU3 Event	None	
49	SWU4_EVT	SWU4 Event	None	
50	SWU5_EVT	SWU5 Event	None	
51	SWU6_EVT	SWU6 Event	None	
52	SWU0_EVT	SWU0 Event	None	
53	SWU1_EVT	SWU1 Event	None	

CM41X_M0 SWU Trigger List

Table 45-5: CM41X_M0 SWU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
23	SWU0_EVT	SWU0 Event	None
24	SWU1_EVT	SWU1 Event	None

Table 45-6: CM41X_M0 SWU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
30	SWU0_EN	SWU0 Enable	Pulse
31	SWU1_EN	SWU1 Enable	Pulse

CM41X_M4 SWU Trigger List

Table 45-7: CM41X_M4 SWU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
61	SWU0_EVT	SWU0 Event	None

Table 45-7: CM41X_M4 SWU Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
62	SWU1_EVT	SWU1 Event	None
63	SWU2_EVT	SWU2 Event	None
64	SWU3_EVT	SWU3 Event	None
65	SWU4_EVT	SWU4 Event	None
66	SWU5_EVT	SWU5 Event	None
67	SWU6_EVT	SWU6 Event	None

Table 45-8: CM41X_M4 SWU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
53	SWU2_EN	SWU2 Enable	Pulse
54	SWU3_EN	SWU3 Enable	Pulse
55	SWU4_EN	SWU4 Enable	Pulse
56	SWU5_EN	SWU5 Enable	Pulse
57	SWU6_EN	SWU6 Enable	Pulse

SWU Definitions

The following definitions are helpful when using the SWU module.

Watchpoint Mode

Mode in which transactions are recognized on an exact match. Actions can be configured to be taken after a specified number of matches have occurred.

Bandwidth Mode

Mode in which transactions are recognized and counted inside sampling window.

SWU Architectural Concepts

The information in this section provides basic module design concepts.

SWU-to-SCB Interface

The SWU system crossbar interface block latches all transactions on the system crossbar read and write address channels when the SWU_GCTL.EN register enable bit is set.

SWU Block Diagram

The *System Watchpoint Unit Top-Level Block Diagram* figure shows the SWU block diagram.

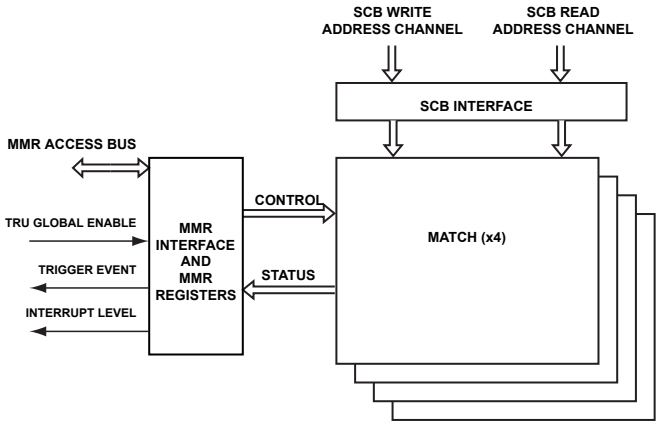


Figure 45-1: System Watchpoint Unit Top-Level Block Diagram

SCB Interface Block

The SWU system crossbar (SCB) latches all transactions on the SCB read and write address channels when the `SWU_GCTL.EN` bit is set.

MMR Interface Block

The SWU MMR block contains the peripheral bus interface and the SWU MMR registers. It also merges all interrupt requests and events from each match block into common outputs.

SWU Operating Modes

There are two operating modes supported by the SWU: bandwidth mode and watchpoint mode.

Bandwidth Mode

In bandwidth mode, the SWU module counts transactions which match the properties specified in the `SWU_CTL[n]` register during a sampling window determined by the respective `SWU_CNT[n]` register. At the end of the sampling window, the SWU stores results in the `SWU_HIST[n]` register. If the sampled bandwidth falls outside a programmed range, then the programmed action occurs.

Watchpoint Mode

In watchpoint mode, if the `SWU_CTL[n].CNTEN` bit is set, the SWU module decrements the `SWU_CUR[n]` register for each match, until it equals zero, at which point any programmed actions occur. The `SWU_CUR[n]` register is then reloaded from the `SWU_CNT[n]` register (if the `SWU_CTL[n].CNTRPTEN` bit is set), and the cycle repeats. If the `SWU_CTL[n].CNTRPTEN` bit is not set, any programmed actions happen on every match.

Match Block

There are four match blocks for each SWU. Each SWU match block can monitor either the read or write address channel, selected by the `SWU_CTL[n].DIR` bit. The SWU match block can operate in either watchpoint or bandwidth mode, as selected by the `SWU_CTL[n].BWEN` bit.

In either mode, the SWU match block can be programmed to match based on address (exact, inclusive or exclusive range), ID (with masking), security, and lock type. All enabled matches are AND'ed together to determine a match.

SWU Event Control

The SWU can generate the following events when a match occurs and when the event is enabled by configuring the proper bits in the control register.

1. Trace Message
2. Trigger
3. Interrupt request

SWU Interrupts

All interrupt requests and events from each match block are merged into common outputs.

SWU Status and Errors

SWU status and errors are reported in the SWU_GSTAT register. The SWU records an address error when a write or read attempt is made to the MMR address space of the SWU and the register does not exist. This error is the only one the SWU records. The register contains bits that perform the following functions.

- Indicate whether a particular match group sampled a transaction that is below a minimum target or above a maximum target in bandwidth mode.
- Indicate whether a watchpoint match occurred for each match group.
- Indicate whether an interrupt request was triggered due to a match event from one of the match groups.

Triggers

The SWU can be either a trigger master or a trigger slave depending on the trigger routing unit (TRU) configuration. As a trigger master, programs must set the SWU_CTL[n].TRGEN bit so that when a match condition is met, a trigger event is generated. Each SWU in the system can also be a trigger slave when mapped as one in the TRU.

When the SWU is a slave, a trigger event activates the SWU by automatically setting the SWU_GCTL.EN bit. Since the SWU can be automatically enabled through a trigger event, programs must pre-configure the SWU before enabling the TRU. Furthermore, although a trigger event can enable the SWU as a slave, to disable the SWU, programs must manually clear the SWU_GCTL.EN bit.

SWU Programming Model

Program the appropriate registers to use the SWU. Each control register configures aspects such as:

- The direction of monitoring (reads or writes)
- Whether SWU uses bandwidth mode or watchpoint mode

- The setup of events that are triggered when a condition is met while monitoring using the SWU

Configure supplemental registers such as the lower (SWU_LA[n]) and upper (SWU_UA[n]) address boundaries before enabling the SWU.

Once the SWU has been enabled and the monitoring conditions are met, events are generated when configured.

The global status register (SWU_GSTAT) can be read to observe the status of the units.

The ***SWU Logical Flow*** diagram shows the logical program flow of the SWU.

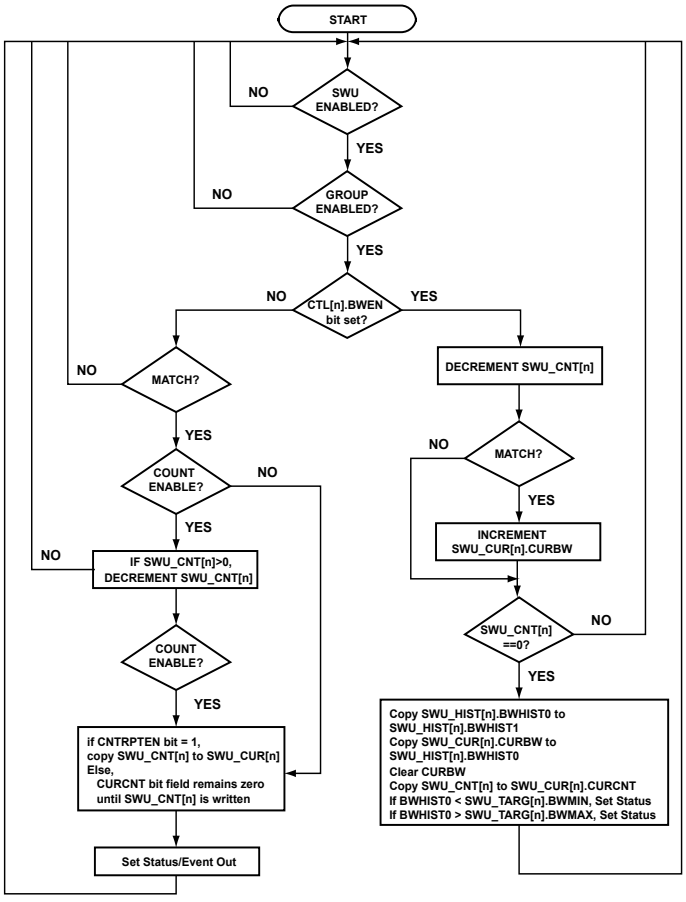


Figure 45-2: SWU Logical Flow

SWU Mode Configuration

The following sections show the steps for configuring SWU bandwidth mode and watchpoint mode.

Configuring the SWU for Bandwidth Mode

In bandwidth mode, the SWU counts transactions which match during a sampling window. At the end of the sampling window, the SWU stores the results. An action can be taken if the sampled bandwidth goes above or falls below a programmed range.

1. Configure the SWU_CTL[n].DIR bit to test the match on writes or reads.
2. Configure the SWU_CTL[n].ACMPM bits to address comparisons, exact match, matches inside a range or matches outside a range.
3. If ID comparison is desired, set the SWU_CTL[n].IDCMPEN bit.
4. Set the SWU_CTL[n].BLENINC bit to increment by burst length or clear it to increment by 1.
5. Configure the SWU_CTL[n].MAXACT and SWU_CTL[n].MINACT bits to enable actions taken when the bandwidth goes above the maximum, or falls below the minimum, respectively.
6. Set the SWU_CTL[n].BWEN=1 to enable bandwidth mode.
7. Program the lower address register, SWU_LA[n], and upper address register, SWU_UA[n], to define the memory range for comparison.
8. If ID comparison is enabled, program the ID register, SWU_ID[n].
9. Program the count register, SWU_CNT[n], with the number of clock cycles for which the SWU counts the number of matches.
10. If the SWU is set to respond when the bandwidth measurement underflows or overflows, program the min and max values into the SWU_TARG[n] register.
11. Enable the SWU

The SWU counts the number of matches in a pre-defined number of clock cycles as programmed. As an option, it can define lower and upper limits. If the matches fall outside the limits, an action can be taken.

Configuring the SWU for Watchpoint Mode

In watchpoint mode, the SWU can trigger a programmed action after every match or after a number of matches. This sequence can be automatically reset.

1. Set the SWU_CTL[n].DIR bit to test the match on writes or reads.
2. Configure the SWU_CTL[n].ACMPM bits for address comparisons, exact match, matches inside a range or matches outside a range.
3. If ID comparison is desired, set the SWU_CTL[n].IDCMPEN.
4. Set the SWU_CTL[n].CNTEN bit to enable the events to be triggered when the count decrements to zero.
5. If needed, set the SWU_CTL[n].CNTRPTEN bit to automatically reload the counter after it has decremented to zero to start another match sequence.
6. Clear the SWU_CTL[n].BWEN = 0 to configure watchpoint mode.
7. Configure the lower address register, SWU_LA[n], and upper address register, SWU_UA[n], to define the memory range for comparison.

8. If ID comparison is enabled, configure the ID register, SWU_ID[n].
9. Configure the count register, SWU_CNT[n], to determine how many matches occur before the watchpoint group responds.
10. Enable the SWU.

The SWU detects and counts down the number of match occurrences. When the counter expires, an action is taken.

CM41X_M4 SWU Register Descriptions

System Watchpoint Unit (SWU) contains the following registers.

Table 45-9: CM41X_M4 SWU Register List

Name	Description
SWU_CNT[n]	Count Register n
SWU_CTL[n]	Control Register n
SWU_CUR[n]	Current Register n
SWU_GCTL	Global Control Register
SWU_GSTAT	Global Status Register
SWU_HIST[n]	Bandwidth History Register n
SWU_ID[n]	ID Register n
SWU_LA[n]	Lower Address Register n
SWU_TARG[n]	Target Register n
SWU_UA[n]	Upper Address Register n

Count Register n

The SWU count registers (SWU_CNT[n]) contain a 16-bit count field (SWU_CNT[n].COUNT) whose usage differs depending on the mode of the watchpoint group. In bandwidth mode, the SWU_CNT[n].COUNT field value defines the number of clock cycles in a bandwidth period. In watchpoint mode, when the cycle count is enabled, the SWU_CNT[n].COUNT field value determines how many matches occur before the watchpoint group takes action.

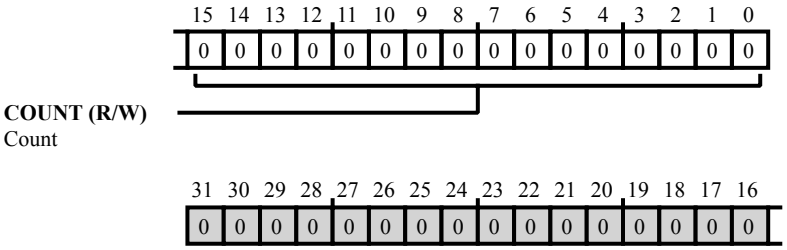


Figure 45-3: SWU_CNT[n] Register Diagram

Table 45-10: SWU_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	COUNT	Count. The SWU_CNT[n].COUNT field value defines the number of clock cycles in a bandwidth period. In watchpoint mode, when the cycle count is enabled, the SWU_CNT[n].COUNT field value determines how many matches occur before the watchpoint group takes action.

Control Register n

The SWU control registers (SWU_CTL[n]) contain watchpoint attribute controls for all four watchpoint groups. These controls include enabling watchpoints, selecting the transaction direction for match, selecting address comparison mode, enabling ID comparison, enabling security comparison, enabling locked comparison, enabling cycle count, enabling count repeat, enabling debug events, enabling interrupts, enabling triggers, enabling trace messages, enabling bandwidth mode, selecting the burst length increment, and enabling bandwidth underflow and overflow detection.

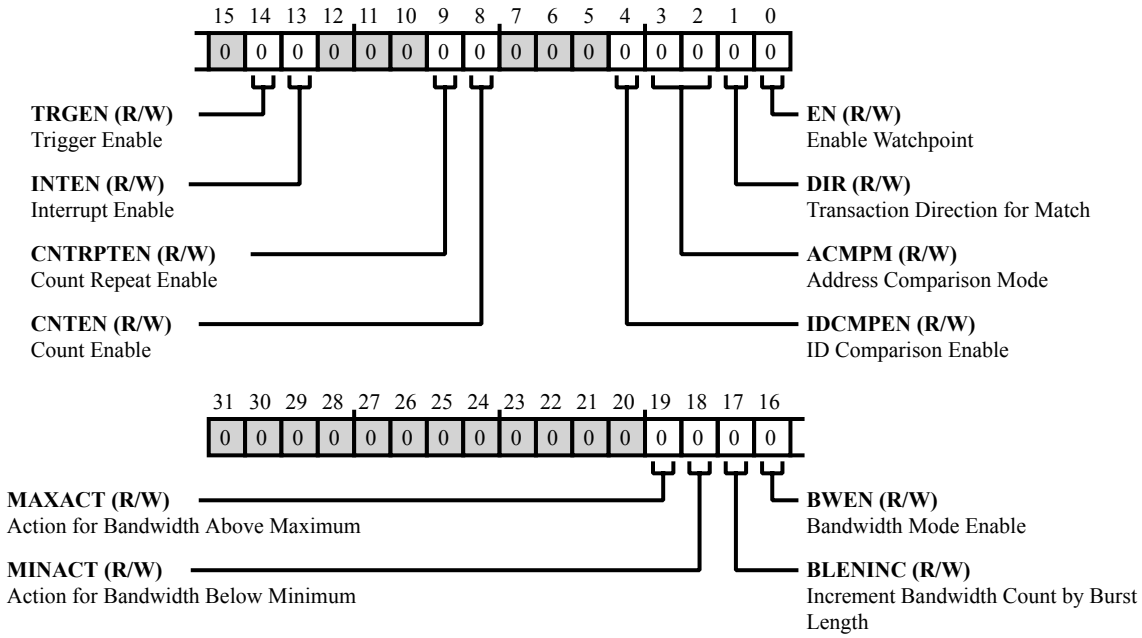


Figure 45-4: SWU_CTL[n] Register Diagram

Table 45-11: SWU_CTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	MAXACT	Action for Bandwidth Above Maximum. Each SWU_CTL[n].MAXACT bit determines whether a watchpoint group takes action on bandwidth overflow. This feature is only valid in bandwidth mode.	
		0	No Action
		1	Take Action
18 (R/W)	MINACT	Action for Bandwidth Below Minimum. Each SWU_CTL[n].MINACT bit determines whether a watchpoint group takes action on bandwidth underflow. This feature is only valid in bandwidth mode.	
		0	No Action
		1	Take Action

Table 45-11: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	BLENINC	Increment Bandwidth Count by Burst Length. Each SWU_CTL[n].BLENINC bit controls how a watchpoint group's bandwidth count is incremented in the SWU_CUR[n] register's SWU_CUR[n].CURBW field. If the SWU_CTL[n].BLENINC bit is cleared (= 0), the SWU increments the bandwidth count by 1 for each matching transaction. If the SWU_CTL[n].BLENINC bit is set (=1), the SWU increments the bandwidth count by the burst length of the transaction for each matching transaction. This feature is only valid for bandwidth mode (SWU_CTL[n].BWEN bit == 1). Note that if the address range match is enabled (SWU_CTL[n].ACMPM bits) and if any address of a burst falls within the address range, the SWU_CUR[n].CURBW field is incremented by the burst length even if some of the burst address fall outside of the range. Also, note that the burst size of the transaction is not included in the increment, only the burst length of the transaction. This increment operation provides an approximate (not exact) number of bus cycles consumed during the bandwidth.	
		0	Increment by 1
		1	Burst Length Increment for Bandwidth Count
16 (R/W)	BWEN	Bandwidth Mode Enable. Each SWU_CTL[n].BWEN bit controls whether a watchpoint group operates in watchpoint mode or bandwidth mode. In watchpoint mode, the SWU_CTL[n].CNTEN and (optionally) SWU_CTL[n].CNTRPTEN registers control usage of the cycle count for watchpoint group operations. In bandwidth mode, the SWU_CTL[n].BLENINC, SWU_TARG[n], and SWU_HIST[n] registers control usage of watchpoint matches for watchpoint group operations.	
		0	Watchpoint Mode
		1	Bandwidth Mode
14 (R/W)	TRGEN	Trigger Enable. Each SWU_CTL[n].TRGEN bit controls whether a match for a watchpoint group generates a trigger event. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable
13 (R/W)	INTEN	Interrupt Enable. Each SWU_CTL[n].INTEN bit controls whether a match for a watchpoint group generates an interrupt. This feature is valid in both bandwidth and watchpoint modes.	
		0	Disable
		1	Enable

Table 45-11: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	CNTRPTEN	Count Repeat Enable. Each SWU_CTL[n].CNTRPTEN bit controls whether the watchpoint group's cycle count is reloaded and repeated after cycle countdown. If the SWU_CTL[n] register's SWU_CTL[n].CNTRPTEN bit is set, the SWU_CUR[n] register's SWU_CUR[n].CURCNT field is reloaded from SWU_CNT[n] register's SWU_CNT[n].COUNT field, and the countdown starts again. If SWU_CTL[n].CNTRPTEN bit is cleared, the expired count remains zero, and no further events are signalled. (See the SWU_CTL[n].CNTEN bit description for information regarding the countdown setup.)	
		0	Disable
		1	Enable
8 (R/W)	CNTEN	Count Enable. Each SWU_CTL[n].CNTEN bit controls whether the cycle count in the watchpoint group's SWU_CNT[n] register is decremented each cycle until it reaches zero. This feature is only valid in watchpoint mode (SWU_CTL[n].BWEN bit == 0).When the count reaches zero, any enabled watchpoint events are triggered. (See the SWU_CTL[n].CNTRPTEN bit description for optional actions at that may occur at the end of the countdown.)	
		0	Disable
		1	Enable
4 (R/W)	IDCMPEN	ID Comparison Enable. Each SWU_CTL[n].IDCMPEN bit controls the ID comparison operation of an SWU watchpoint group. The ID match is based on comparison with the value in the SWU_ID[n] register.	
3:2 (R/W)	ACMPM	Address Comparison Mode. Each set of SWU_CTL[n].ACMPM bits control the address comparison operation of an SWU watchpoint group. The address within range for comparison is defined as (SWU_LA[n] register <= address < SWU_UA[n] register). The address outside range for comparison is defined as (address < SWU_LA[n]) or (SWU_UA[n] <= address).	
		0	No address comparison
		1	Exact match on LAn
		2	Match on address within range
		3	Match on address outside range

Table 45-11: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	DIR	Transaction Direction for Match. Each `SWU_CTL[n].DIR` bit determines whether the SWU check reads or writes for watchpoint matches.	
		0	Match on reads only
		1	Match on writes only
0 (R/W)	EN	Enable Watchpoint. Each `SWU_CTL[n].EN` bit controls the operation of one SWU watchpoint group. Clearing the `SWU_CTL[n].EN` bit halts the execution of watchpoint or bandwidth tracking operations in the watchpoint group without resetting status or configuration registers. Setting the `SWU_CTL[n].EN` bit enables the SWU watchpoint group to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

Current Register n

The SWU current register (SWU_CUR[n]) operation varies depending whether the watchpoint group is in bandwidth mode or watchpoint mode. In both modes, the watchpoint count begins when the SWU loads the register's SWU_CUR[n].CURCNT field from the SWU_CNT[n] register's SWU_CNT[n].COUNT field when the watchpoint count is enabled (SWU_CTL[n] register, SWU_CTL[n].CNTEN bit =1).

In bandwidth mode, the current count field (SWU_CUR[n].CURCNT) contains the cycle count remaining within the current watchpoint period. The SWU decrements this value every cycle until the count reaches zero. At that point, the SWU reloads the SWU_CUR[n].CURCNT field from SWU_CNT[n] register's SWU_CNT[n].COUNT field. In bandwidth mode, the current bandwidth field (SWU_CUR[n].CURBW) contains the count of watchpoint matches (bandwidth) accumulated in the current watchpoint period.

In watchpoint mode, the current count field (SWU_CUR[n].CURCNT) contains the watchpoint match count remaining within the current watchpoint period. The SWU decrements this value with every watchpoint match until the count reaches zero. At that point, the SWU reloads the SWU_CUR[n].CURCNT field from SWU_CNT[n] register's SWU_CNT[n].COUNT field if the SWU_CTL[n] register's SWU_CTL[n].CNTRPTEN bit is set (=1). In watchpoint mode, the current bandwidth field (SWU_CUR[n].CURBW) is undefined.

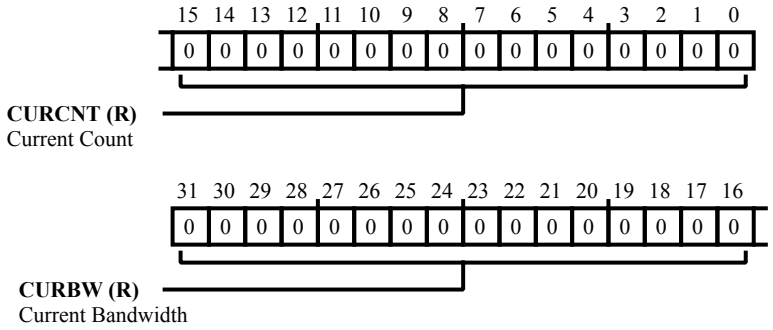


Figure 45-5: SWU_CUR[n] Register Diagram

Table 45-12: SWU_CUR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	CURBW	Current Bandwidth.
15:0 (R/NW)	CURCNT	Current Count.

Global Control Register

The SWU global control register (SWU_GCTL) provides SWU reset and enable.

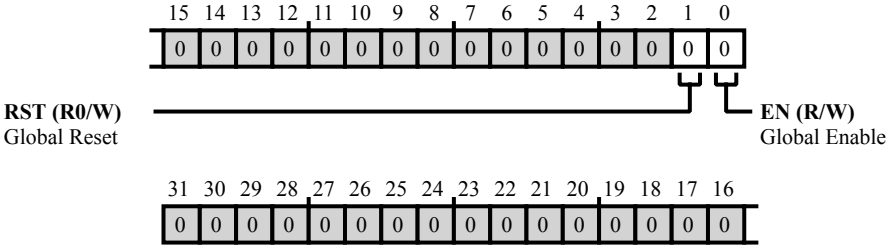


Figure 45-6: SWU_GCTL Register Diagram

Table 45-13: SWU_GCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R0/W)	RST	Global Reset. The SWU_GCTL.RST) is write-1-action/read zero and controls the SWU operational state. Setting SWU_GCTL.RST resets all SWU registers to their default values and halts all SWU operations.	
		0	No Action
		1	Reset
0 (R/W)	EN	Global Enable. The SWU_GCTL.EN controls the SWU operational state. Clearing SWU_GCTL.EN halts the execution of all watchpoint and bandwidth tracking operations without resetting status registers or associated signals. Setting SWU_GCTL.EN enables the SWU to begin/resume operation with the current configuration and status.	
		0	Disable
		1	Enable

Global Status Register

The SWU global status register (SWU_GSTAT) contains status bits for all four watchpoint groups.

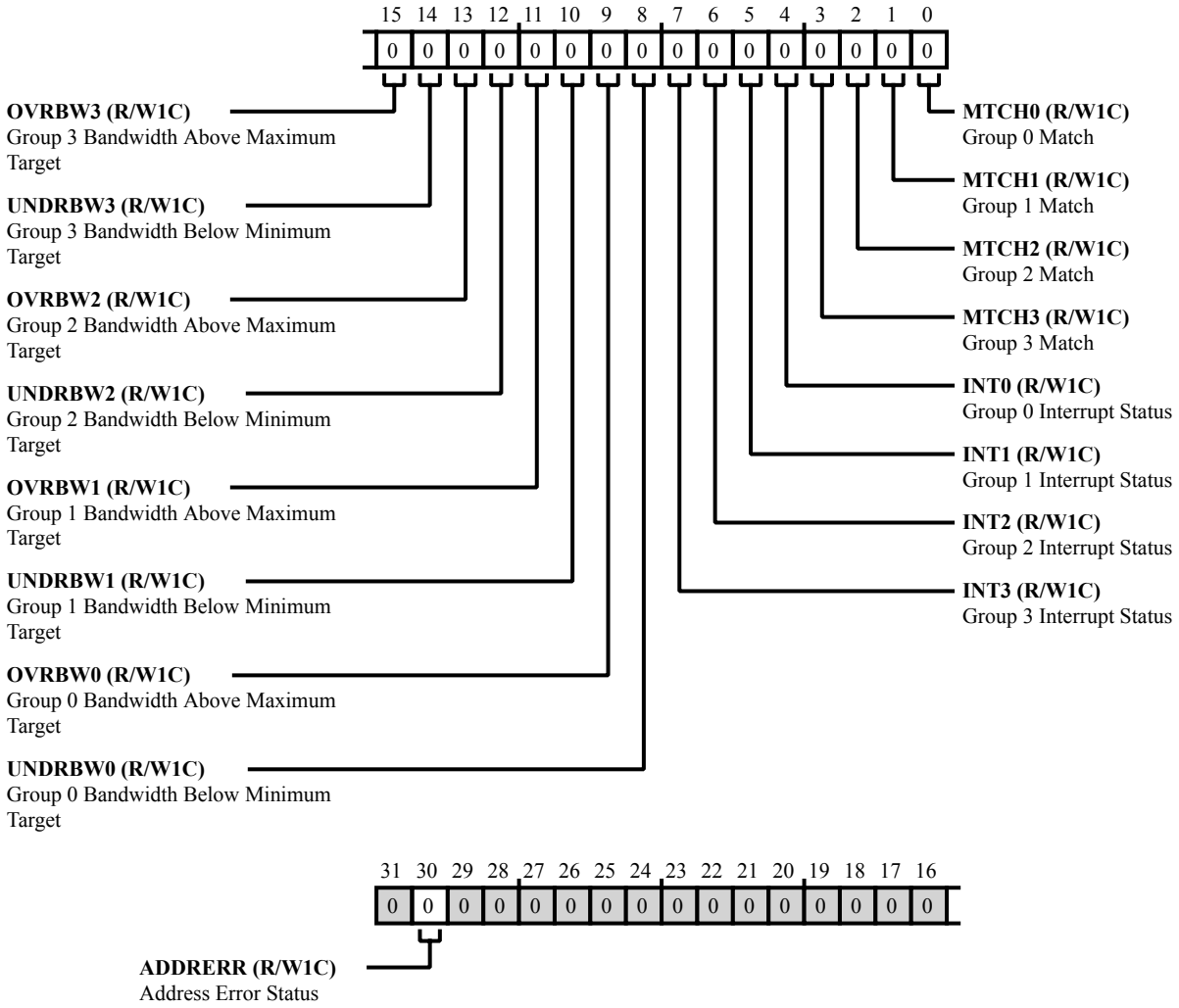


Figure 45-7: SWU_GSTAT Register Diagram

Table 45-14: SWU_GSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W1C)	ADDRERR	Address Error Status. The SWU_GSTAT.ADDRERR indicates that the SWU generated an address error. This status bit is sticky; write-1-to-clear it.	
		0	Inactive
		1	Active

Table 45-14: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	OVRBW3	Group 3 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.	
		0	Group 3 was not above maximum bandwidth
		1	Group 3 was above maximum bandwidth
14 (R/W1C)	UNDRBW3	Group 3 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		0	Group 3 was not below minimum bandwitdth
		1	Group 3 was below minimum bandwitdth
13 (R/W1C)	OVRBW2	Group 2 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.	
		0	Group 2 was not above maximum bandwidth
		1	Group 2 was above maximum bandwidth
12 (R/W1C)	UNDRBW2	Group 2 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		0	Group 2 was not below minimum bandwidth
		1	Group 2 was below minimum bandwidth
11 (R/W1C)	OVRBW1	Group 1 Bandwidth Above Maximum Target. See SWU_GSTAT.OVRBW0 description.	
		0	Group 1 was not above maximum bandwidth
		1	Group 1 was above maximum bandwidth
10 (R/W1C)	UNDRBW1	Group 1 Bandwidth Below Minimum Target. See SWU_GSTAT.UNDRBW0 description.	
		0	Group 1 was not below minimum bandwidth
		1	Group 1 was below minimum bandwidth
9 (R/W1C)	OVRBW0	Group 0 Bandwidth Above Maximum Target. The SWU_GSTAT.OVRBW0 - SWU_GSTAT.OVRBW3 -- Group 0 through 3 watch-point bandwidth over maximum target bits. Each maximum bandwidth bit indicate (for each group)s that the measured bandwidth over the period defined by the SWU_CNT[n] register was over the maximum target. This status bit is sticky; write-1-to-clear it.	
		0	Group 0 was not above maximum bandwidth
		1	Group 0 was above maximum bandwidth

Table 45-14: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	UNDRBW0	Group 0 Bandwidth Below Minimum Target. The SWU_GSTAT.UNDRBW0 - SWU_GSTAT.UNDRBW3 -- Group 0 through 3 watchpoint bandwidth below minimum target bits. Each minimum bandwidth bit indicates (for each group) that the measured bandwidth over the period defined by the SWU_CNT[n] register was below the minimum target. This status bit is sticky; write-1-to-clear it.	
		0	Group 0 was not below minimum bandwidth
		1	Group 0 was below minimum bandwidth
7 (R/W1C)	INT3	Group 3 Interrupt Status. See SWU_GSTAT.INT0 description.	
		0	No Interrupt
		1	Interrupt Occurred
6 (R/W1C)	INT2	Group 2 Interrupt Status. See SWU_GSTAT.INT0 description.	
		0	No Interrupt
		1	Interrupt Occurred
5 (R/W1C)	INT1	Group 1 Interrupt Status. See SWU_GSTAT.INT0 description.	
		0	No Interrupt
		1	Interrupt Occurred
4 (R/W1C)	INT0	Group 0 Interrupt Status. The SWU_GSTAT.INT0 - SWU_GSTAT.INT3 -- Group 0 through 3 interrupt bits. Each interrupt bit indicates (for each group) whether a watchpoint group is contributing to the SWU's interrupt output. This status bit is sticky; write-1-to-clear it.	
		0	No interrupt
		1	Interrupt Occurred
3 (R/W1C)	MTCH3	Group 3 Match. See SWU_GSTAT.MTCH0 description.	
		0	No Match
		1	Group 3 Watchpoint Match

Table 45-14: SWU_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	MTCH2	Group 2 Match. See SWU_GSTAT.MTCH0 description.	
		0	No match
		1	Group 2 Watchpoint Match
1 (R/W1C)	MTCH1	Group 1 Match. See SWU_GSTAT.MTCH0 description.	
		0	No match
		1	Group 1 Watchpoint Match
0 (R/W1C)	MTCH0	Group 0 Match. The SWU_GSTAT.MTCH0 - SWU_GSTAT.MTCH3 -- Group 0 through 3 match bits. Each match bit indicates (for each group) whether a watchpoint match has occurred in a SWU watchpoint group, as controlled by the group's related watchpoint control register (SWU_CTL[n]). This status bit is sticky; write-1-to-clear it.	
		0	No match
		1	Group 0 Watchpoint Match

Bandwidth History Register n

The SWU bandwidth history registers (SWU_HIST[n]) contain data copied from a watchpoint group's current bandwidth value (SWU_CUR[n] register, SWU_CUR[n].CURBW bits) at the end of the last two watchpoint periods. At the end of each watchpoint period, the SWU copies the previous bandwidth value from the SWU_HIST[n].BWHIST0 field to the SWU_HIST[n].BWHIST1 field and copies the new bandwidth value from the SWU_CUR[n].CURBW field to the SWU_HIST[n].BWHIST0 field.

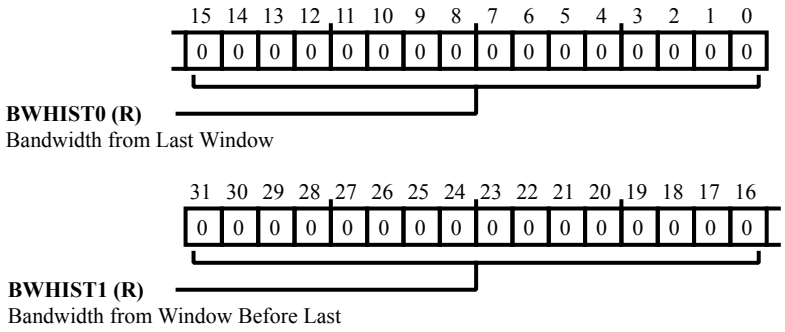


Figure 45-8: SWU_HIST[n] Register Diagram

Table 45-15: SWU_HIST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/NW)	BWHIST1	Bandwidth from Window Before Last.
15:0 (R/NW)	BWHIST0	Bandwidth from Last Window.

ID Register n

The SWU ID registers (SWU_ID[n]) contain a 16-bit ID field (SWU_ID[n].ID) and a 16-bit ID mask field (SWU_ID[n].IDMASK) that watchpoint groups use for ID comparison. The ID on the bus is AND'ed with the SWU_ID[n].IDMASK field, then the watchpoint group compares the result against the SWU_ID[n].ID field.

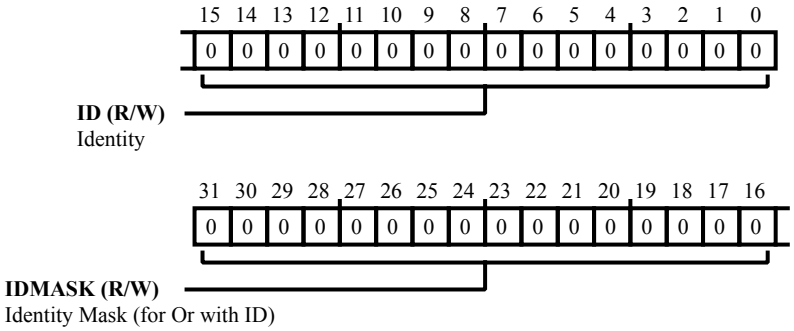


Figure 45-9: SWU_ID[n] Register Diagram

Table 45-16: SWU_ID[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	IDMASK	Identity Mask (for Or with ID).
15:0 (R/W)	ID	Identity.

Lower Address Register n

The SWU lower address registers (SWU_LA[n]) contain each watchpoint group's lower address for address match comparison. In exact match on SWU_LA[n] address mode (SWU_CTL[n].ACMPM bits =01), the watchpoint group uses only this address for match comparison.

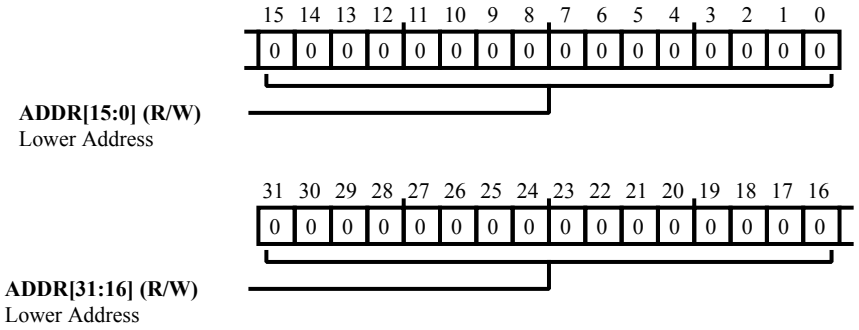


Figure 45-10: SWU_LA[n] Register Diagram

Table 45-17: SWU_LA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Lower Address.

Target Register n

The SWU target registers (SWU_TARG[n]) contain a minimum value field (SWU_TARG[n].BWMIN) and maximum value field (SWU_TARG[n].BWMAX) of bandwidth targets used by watchpoint groups in bandwidth mode. When the bandwidth period expires, if the current bandwidth value (SWU_CUR[n] register, SWU_CUR[n].CURBW bits) is below the minimum target or above the maximum target, the watchpoint group takes action as enabled by the SWU_CTL[n] register's SWU_CTL[n].MINACT or SWU_CTL[n].MAXACT bits.

In bandwidth mode, note that the watchpoint group increments its count of either data bus transactions or address bus transactions (bursts) as selected by the SWU_CTL[n].BLENINC bit. Keep this mode selection in mind when programming the bandwidth target values.

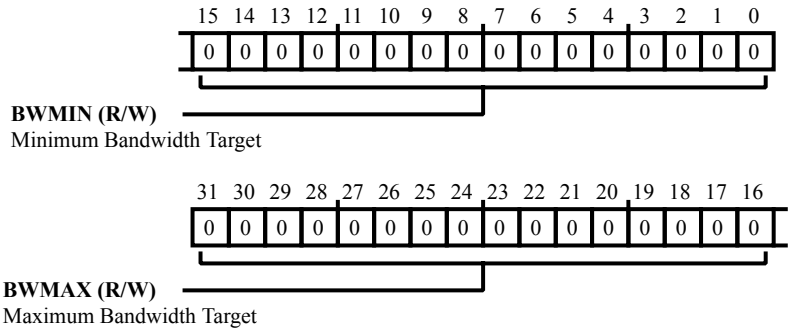


Figure 45-11: SWU_TARG[n] Register Diagram

Table 45-18: SWU_TARG[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	BWMAX	Maximum Bandwidth Target.
15:0 (R/W)	BWMIN	Minimum Bandwidth Target.

Upper Address Register n

The SWU upper address registers (SWU_UA[n]) contain each watchpoint group's upper address for address match comparison. In exact match on SWU_LA[n] address mode (SWU_CTL[n].ACMPM bits =01), the SWU_UA[n] is not used for match comparison.

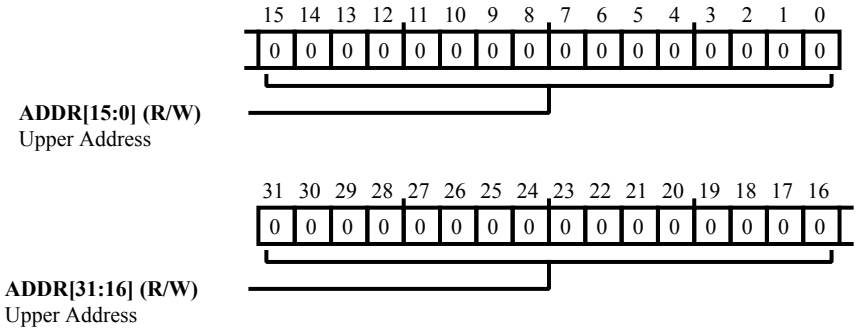


Figure 45-12: SWU_UA[n] Register Diagram

Table 45-19: SWU_UA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Upper Address.

46 JTAG debug and Serial Wire Debug Port (SWJ-DP)

SWJ-DP is a combined JTAG-DP and SW-DP that enables either a Serial Wire Debug (SWD) or JTAG probe to be connected to a target. SWD signals share pins as JTAG. There is an autodetect mechanism that switches between JTAGDP and SW-DP. Depending on which special data sequence the SWJ-DP uses, the emulator pod transmits to the JTAG pins. The SWJ-DP behaves as a JTAG target if normal JTAG sequences are sent to it and as a single wire target if the SW_DP sequence is transmitted.

Embedded Trace Macrocell (ETM) and Instrumentation Trace Macrocell (ITM)

The processor supports both Embedded Trace Macrocell (ETM) and Instrumentation Trace Macrocell (ITM). These features both offer an optional debug component that enables logging of real-time instruction and data flow within the CPU core. This data is stored and read through special debugger pods that have the trace feature capability. ITM is a single-data pin feature and the ETM is a 4-data pin feature.

Debug Access

The ADSP-CM41x processor support standard debugging options, JTAG and SWD for Cortex ARM cores. There are standard debug choices available, based on what is selected in the end user tools: Single core ARM Cortex-M4 debug, single core ARM Cortex-M0 debug and dual core M4 + M0 debug. Both the ARM Cortex-M4 and ARM Cortex-M0 core DAPs are visible in the single JTAG chain when configured. To the external world, debug requires just a single ARM Coresight debug interface (connector). The pin count will vary depending on the system debug choice.

Core Debug

Both the ARM Cortex-M4 and ARM Cortex-M0 processors are configured with the maximum available internal components. See the ARM documents for usage details.

The ARM Cortex-M0 core has hardware for four breakpoints and two watchpoints.

The ARM Cortex-M4 core has hardware for six breakpoints and four watchpoints and a full flash-patch block.

Trace

The ARM Cortex-M4 processor supports both Embedded Trace Macrocell (ETM) and Instrumentation Trace Macrocell (ITM). These features both offer an optional debug component that enables logging of real-time instruction and data flow within the CPU core. This data is stored and read through special debuggers that have the trace feature capability. The ARM Cortex-M0 core does not support any trace output.

ITM is a single-data pin feature and the ETM is a 4-data pin feature. Both the ITM and ETM are byte data from the ARM Cortex-M4 core. There are two distinct modes of trace capture. The first is the standard ARM Cortex-M4 TPIU-Lite to generate trace output data. The TPIU-Lite combines TPIU and SWO capabilities. It natively supports ETM and ITM input channels. Second is a ram based trace capture. ITM and ETM data are funneled into a 4 KB trace ram. Trace data can be stored and processed without need of dedicated pins allocated to trace.

The trace data and trace clock are generated internally with SYSCLK -based timing. TRACECLK is always SYSCLK, up to 100 MHz. TRACEDATA is transferred on both edges of TRACECLK, at frequencies up to 200 MHz. To optimize signal integrity on the trace bus and pod cable, the trace header should be located physically as close as possible to the CM41x, with high-quality PCB traces. Further, the GPIO drive strength for the Trace pads (PC00-PC04) should be programmed to the setting corresponding to ~50Ω for good impedance match to the PCB traces. High speed data rate can cause significant over/under shoot, simulate and add termination resistors as necessary. In this system,there are a maximum of 4 GPIO pins available for trace, operation with 1 or 2 is possible but provides less output bandwidth for real-time trace.

Coresight Cross Triggering

The ARM Cortex-M4 system has a single 4-way cross trigger unit. Of the 4 trigger sub-units, there is one dedicated to control of the M4 core and one dedicated to control of the ARM Cortex-M0. CTI0 for ARM Cortex-M4, CTI1 for ARM Cortex-M0. CTI2 works with the ETF/TMC and TPIU to control trace output/recording. CTI3 has connections to/from both the ARM Cortex-M4 and ARM Cortex-M0 system trigger units (TRU0, 1) as well as controlling the two system slave halt signals.

Table 46-1: CTI0 (M4 debug connections)

Trigger Input Port	Source	ACK Used?	Trigger Output Port	Destination	ACK Used?
CTITRIGIN[7]	M4 ETM TRIG_OUT	N	CTITRIGOUT[7]	M4 DBGSTART	N
CTITRIGIN[6]	M4 ETM TRIG[2]	N	CTITRIGOUT[6]	M4 DBGEND	N
CTITRIGIN[5]	M4 ETM TRIG[1]	N	CTITRIGOUT[5]	M4 ETM EXTIN[1]	N
CTITRIGIN[4]	M4 ETM TRIG[0]	N	CTITRIGOUT[4]	M4 ETM EXTIN[0]	N
CTITRIGIN[3]	Unused	N	CTITRIGOUT[3]	SEC_M4_CTI0_EVT2	Y
CTITRIGIN[2]	Unused	N	CTITRIGOUT[2]	SEC_M4_CTI0_EVT1	Y
CTITRIGIN[1]	Unused	N	CTITRIGOUT[1]	SEC_M4_CTI0_EVT0	Y
CTITRIGIN[0]	M4 HALTED	Y	CTITRIGOUT[0]	M4 EDBGRQ	Y

Table 46-2: CTI1 (M0 debug connections)

Trigger Input Port	Source	ACK Used?	Trigger Output Port	Destination	ACK Used?
CTITRIGIN[7]	Unused	N	CTITRIGOUT[7]	M0 DBGSTART	N
CTITRIGIN[6]	Unused	N	CTITRIGOUT[6]	M0_DBGEND	N
CTITRIGIN[5]	Unused	N	CTITRIGOUT[5]	Unused	N
CTITRIGIN[4]	Unused	N	CTITRIGOUT[4]	Unused	N
CTITRIGIN[3]	Unused	N	CTITRIGOUT[3]	Unused	N
CTITRIGIN[2]	Unused	N	CTITRIGOUT[2]	Unused	N
CTITRIGIN[1]	Unused	N	CTITRIGOUT[1]	SEC_M0_CTI1_EVT0	N
CTITRIGIN[0]	M0 HALTED	Y	CTITRIGOUT[0]	M0 EDBGRQ	Y

Table 46-3: CTI2 (ETF/TPIU connections)

Trigger Input Port	Source	ACK Used?	Trigger Output Port	Destination	ACK Used?
CTITRIGIN[7]	Unused	N	CTITRIGOUT[7]	Unused	N
CTITRIGIN[6]	Unused	N	CTITRIGOUT[6]	Unused	N
CTITRIGIN[5]	Unused	N	CTITRIGOUT[5]	Unused	N
CTITRIGIN[4]	Unused	N	CTITRIGOUT[4]	Unused	N
CTITRIGIN[3]	Unused	N	CTITRIGOUT[3]	Unused	Y
CTITRIGIN[2]	Unused	N	CTITRIGOUT[2]	Unused	Y
CTITRIGIN[1]	ETF_FULL	N	CTITRIGOUT[1]	ETF FLUSHIN	N
CTITRIGIN[0]	ACQCOMP FULL	N	CTITRIGOUT[0]	ETF TRIGIN	N

Table 46-4: CTI3 (System Trigger Unit connections)

Trigger Input Port	Source	ACK Used?	Trigger Output Port	Destination	ACK Used?
CTITRIGIN[7]	TRGS_M0_CTI3_SLV7	N	CTITRIGOUT[7]	ctitrigoutack for halt_slv[1:0]	N
CTITRIGIN[6]	TRGS_M0_CTI3_SLV6	N	CTITRIGOUT[6]	TRGS_M0_CTI3_MST6	N
CTITRIGIN[5]	TRGS_M4_CTI3_SLV5	N	CTITRIGOUT[5]	TRGS_M0_CTI3_MST5	N
CTITRIGIN[4]	TRGS_M4_CTI3_SLV4	N	CTITRIGOUT[4]	TRGS_M4_CTI3_MST4	N
CTITRIGIN[3]	TRGS_M4_CTI3_SLV3	N	CTITRIGOUT[3]	TRGS_M4_CTI3_MST3	N
CTITRIGIN[2]	TRGS_M4_CTI3_SLV2	N	CTITRIGOUT[2]	TRGS_M4_CTI3_MST2	N
CTITRIGIN[1]	TRGS_M4_CTI3_SLV1	N	CTITRIGOUT[1]	halt_slv[1]	Y

Table 46-4: CTI3 (System Trigger Unit connections) (Continued)

Trigger Input Port	Source	ACK Used?	Trigger Output Port	Destination	ACK Used?
CTITRIGIN[0]	TRGS_M4_CTI3_SLV0	N	CTITRIGOUT[0]	halt_slv[0]	Y

Synchronous HALT

During a debug halt it may be necessary/advantageous to halt various system components of the chip for debug of some free-running sub-systems. There are two halt outputs of CTI3. Halt_slv[1] is dedicated to debug halt of the 3 PWM units. Halt_slv[0] halts the bus fabric masters, the timers, capture timers, and the wdog timers. These advanced features will not be natively supported by generic m4 debuggers. Additional debugger software support is necessary for implementing halt operation.

- Halting the timers during debug allows for them to decrement like executing in real-time.
- Halting the watch-dog prevents them from asserting during debug stop.
- Halting bus fabric masters will stall DMA master requests during debug. Fabric operations “in-flight” will be allowed to complete.

The PWM units have an independent halt_slv, stopping an actual motor/inverter/etc. at an indeterminate state for indeterminate time for debug may be problematic for particular external components. See the PWM block documentation for details. At the system level, this works in conjunction with the pin_safe_state logic, which can be configured to go “safe” on a debug halt. This allows for multiple PWM configuration choices while in debug halt for different system requirements.

TAPC Controller

A debug TAP Controller sits in the Debug Controller to aid in additional options implemented for various mechanisms related to Security and Boot.

The main TAP controller is the only item present in the jtag scan chain at power-up. It must be configured for the ARM Cortex-M4/M0 chains to be visible for JTAG based debug.

Refer to the corresponding HW chapter for more details on use the registers related to TAPC.

CM41X_M4 TAPC Register Descriptions

TAPC (TAPC) contains the following registers.

Table 46-5: CM41X_M4 TAPC Register List

Name	Description
TAPC_DBGCTL	Debug Control Register

Table 46-5: CM41X_M4 TAPC Register List (Continued)

Name	Description
TAPC_DBGSTAT	Debug Status Register
TAPC_IDCODE	IDCODE Register
TAPC_RCMSG	Run Control Message Register
TAPC_RCMSG_CLR	Run Control Message Clear Register
TAPC_RCMSG_SET	Run Control Message Set Register
TAPC_RCMSG_TGL	Run Control Message Toggle Register
TAPC_SCMSG	System Run Control Message Register
TAPC_SCMSG_CLR	System Run Control Message Clear Register
TAPC_SCMSG_SET	System Run Control Message Set Register
TAPC_SCMSG_TGL	System Run Control Message Toggle Register
TAPC_SDBGKEY0	Secure Debug Key 0 Register
TAPC_SDBGKEY1	Secure Debug Key 1 Register
TAPC_SDBGKEY2	Secure Debug Key 2 Register
TAPC_SDBGKEY3	Secure Debug Key 3 Register
TAPC_SDBGKEYCMP0	Secure Debug Key 0 Compare Register
TAPC_SDBGKEYCMP1	Secure Debug Key 1 Compare Register
TAPC_SDBGKEYCMP2	Secure Debug Key 2 Compare Register
TAPC_SDBGKEYCMP3	Secure Debug Key 3 Compare Register
TAPC_SDBGKEYID0	Secure Debug Key 0 Identification Register
TAPC_SDBGKEYID1	Secure Debug Key 1 Key Identification Register
TAPC_SDBGKEYID2	Secure Debug Key 2 Key Identification Register
TAPC_SDBGKEYID3	Secure Debug Key 3 Key Identification Register
TAPC_SDBGKEY_CTL	Secure Debug Key Control Register
TAPC_SDBGKEY_STAT	Secure Debug Key Status Register
TAPC_USERCODE	USERCODE Register

Debug Control Register

The TAPC_DBGCTL register creates authentication signals to all CoreSight components in the system. This is a read write register accessible on the JTAG APB bus (not directly accessed by TAP).

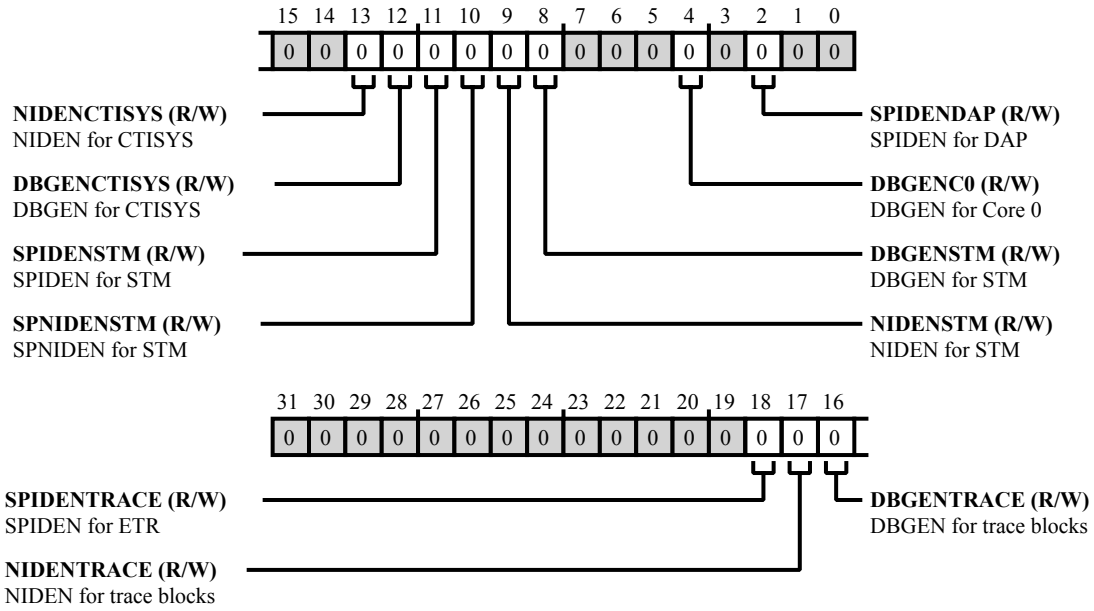


Figure 46-1: TAPC_DBGCTL Register Diagram

Table 46-6: TAPC_DBGCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	SPIDENTRACE	SPIDEN for ETR. Secure Slave Enable Core0
17 (R/W)	NIDENTRACE	NIDEN for trace blocks. Non-Invasive Debug Enable for Trace Blocks
16 (R/W)	DBGENTRACE	DBGEN for trace blocks. DBGEN for Trace Blocks
13 (R/W)	NIDENCTISYS	NIDEN for CTISYS. NIDEN for System CTI
12 (R/W)	DBGENCTISYS	DBGEN for CTISYS. DBGEN for System CTI
11 (R/W)	SPIDENSTM	SPIDEN for STM. SPIDEN for STM

Table 46-6: TAPC_DBGCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	SPNIDENSTM	SPNIDEN for STM. SPINIDEN for STM
9 (R/W)	NIDENSTM	NIDEN for STM. NIDEN for STM
8 (R/W)	DBGENSTM	DBGEN for STM. DBGEN for STM
4 (R/W)	DBGENC0	DBGEN for Core 0. DBGEN for Core0
2 (R/W)	SPIDENDAP	SPIDEN for DAP. DBGEN for DAP

Debug Status Register

The TAPC_DBGSTAT register indicates debug status.

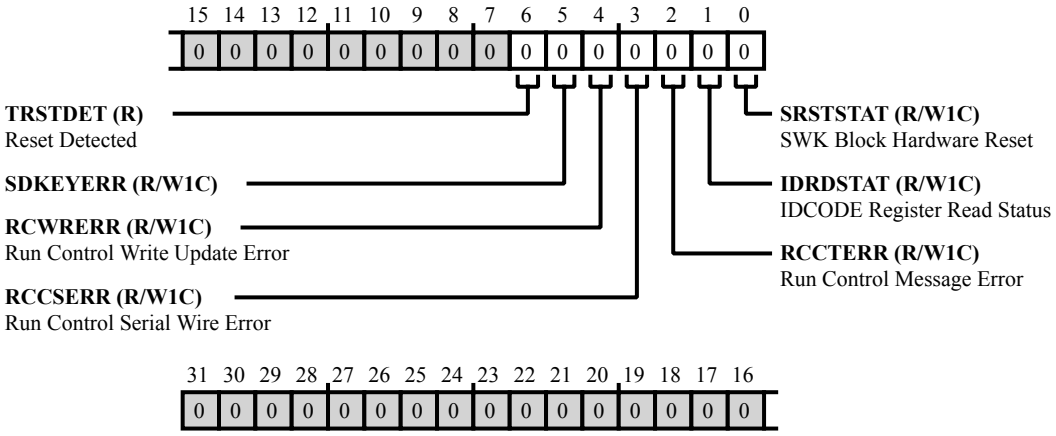


Figure 46-2: TAPC_DBGSTAT Register Diagram

Table 46-7: TAPC_DBGSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/NW)	TRSTDET	Reset Detected. The TAPC_DBGSTAT.TRSTDET bit indicates (if set =1) TAPC has had a valid reset applied, and is in a known state.
5 (R/W1C)	SDKEYERR	. The TAPC_DBGSTAT.SDKEYERR bit indicates (if set =1) concurrent transfers to t_{CLK} domain.
4 (R/W1C)	RCWRERR	Run Control Write Update Error. The TAPC_DBGSTAT.RCWRERR bit indicates (if set =1) the TAPC_RCMSG register had multiple updates requested on same system cycle.
3 (R/W1C)	RCCSERR	Run Control Serial Wire Error. The TAPC_DBGSTAT.RCCSERR bit indicates (if set =1) the serial wire connection had multiple active messages to system domain.
2 (R/W1C)	RCCTERR	Run Control Message Error. The TAPC_DBGSTAT.RCCTERR bit indicates (if set =1) that the JTAG connection has multiple active messages to system domain.
1 (R/W1C)	IDRDSTAT	IDCODE Register Read Status. The TAPC_DBGSTAT.IDRDSTAT bit indicates (if set =1) that the JTAG tap controller performed a capture of the TAPC_IDCODE register.

Table 46-7: TAPC_DBGSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W1C)	SRSTSTAT	SWK Block Hardware Reset. The `TAPC_DBGSTAT.SRSTSTAT` bit indicates (if set =1) that the TAPC Serial Register Interface block initiated a hardware reset.

IDCODE Register

The TAPC_IDCODE register holds the IDCODE. The bit field is defined as follows.

IDCODE[31:28] = 0x1* – REVID

IDCODE[27:12] = 0x280B – JTAG ID

IDCODE[11:1] = 0x65 – Manufacturer ID

IDCODE[0] = 0x1

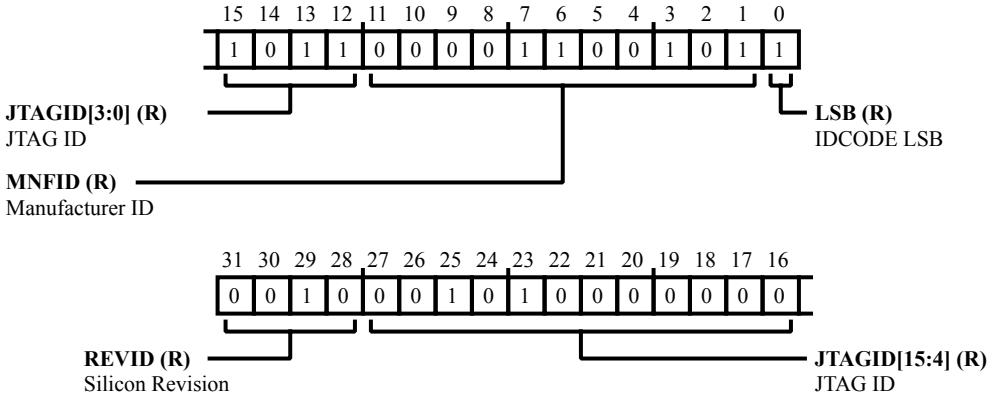


Figure 46-3: TAPC_IDCODE Register Diagram

Table 46-8: TAPC_IDCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/NW)	REVID	Silicon Revision. The TAPC_IDCODE.REVID bit field holds the silicon revision. See the processor anomaly list for details.
27:12 (R/NW)	JTAGID	JTAG ID.
11:1 (R/NW)	MNFID	Manufacturer ID.
0 (R/NW)	LSB	IDCODE LSB.

Run Control Message Register

The TAPC_RCMSG register lets the debugger control boot options through a register that uses system hardware reset (SYS_HWRSTb pin) rather than a system reset (via the TRU or the system fault unit in the SEC).

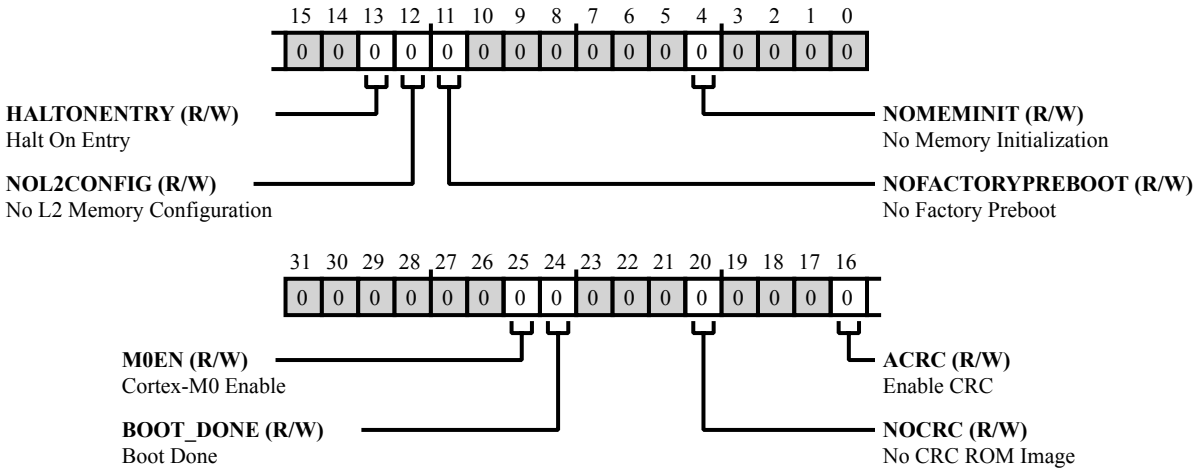


Figure 46-4: TAPC_RCMSG Register Diagram

Table 46-9: TAPC_RCMSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W)	M0EN	Cortex-M0 Enable. The TAPC_RCMSG.M0EN bit (when set =1) takes the Cortex-M0 out of reset and places it into an idle (wait for interrupt) state.
24 (R/W)	BOOT_DONE	Boot Done. The TAPC_RCMSG.BOOT_DONE bit (when set =1) indicates that initial booting and initializations are complete.
20 (R/W)	NOCRC	No CRC ROM Image. The TAPC_RCMSG.NOCRC bit (when set =1) instructs the boot process to disable CRC checking of ROM Image.
16 (R/W)	ACRC	Enable CRC. The TAPC_RCMSG.ACRC bit (when set =1) instructs the boot process to enable CRC of the application (first 4K of flash).
13 (R/W)	HALTONENTRY	Halt On Entry. The TAPC_RCMSG.HALTONENTRY bit (when set =1) instructs the boot process to enter idle, or Wait For Interrupt, on entry.
12 (R/W)	NOL2CONFIG	No L2 Memory Configuration. The TAPC_RCMSG.NOL2CONFIG bit (when set =1) instructs the boot process to not configure the L2 memory.

Table 46-9: TAPC_RCMSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
11 (R/W)	NOFACTORYPREBOOT	No Factory Preboot. The `TAPC_RCMSG.NOFACTORYPREBOOT` bit (when set =1) instructs the boot process to not perform the factory preboot.
4 (R/W)	NOMEMINIT	No Memory Initialization. The `TAPC_RCMSG.NOMEMINIT` bit (when set =1) instructs the boot process to not initialize memory (Main M4 SRAM and MBOX memories).

Run Control Message Clear Register

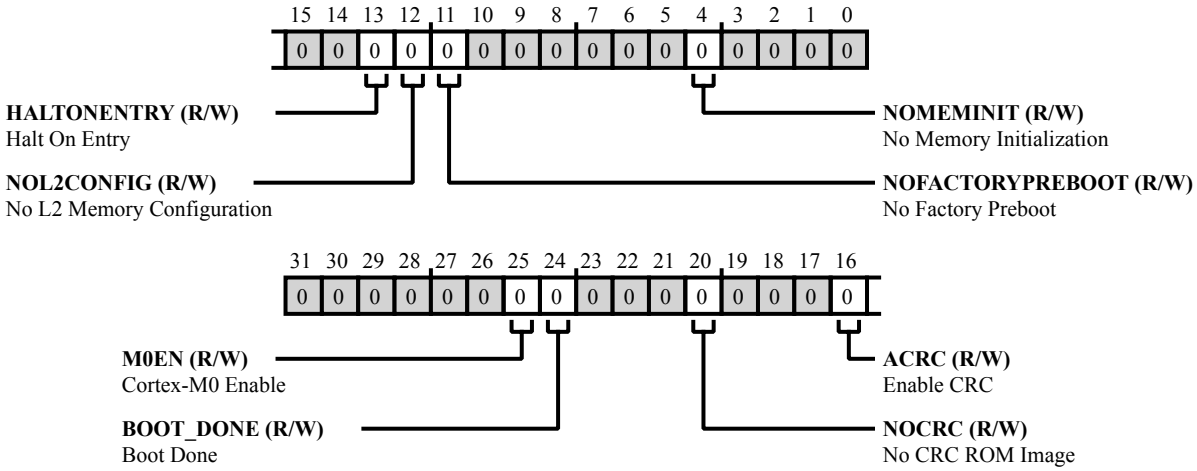


Figure 46-5: TAPC_RCMSG_CLR Register Diagram

Table 46-10: TAPC_RCMSG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W1C)	M0EN	Enable M0 out of reset into idle (wfi) state.
24 (R/W1C)	BOOT_DONE	1 indicates that initial Booting and initializations are complete.
20 (R/W1C)	NOCRC	Disable CRC checking of ROM Image.
16 (R/W1C)	ACRC	Enable CRC of the application (first 4K of flash).
13 (R/W1C)	HALTONENTRY	Enter idle, or wfi, on entry.
12 (R/W1C)	NOL2CONFIG	Do not configure L2.
11 (R/W1C)	NOFACTORYPREBOOT	Do not perform factory preboot.
4 (R/W1C)	NOMEMINIT	Do not initialize memory (Main M4 SRAM and MBOX memories).

Run Control Message Set Register

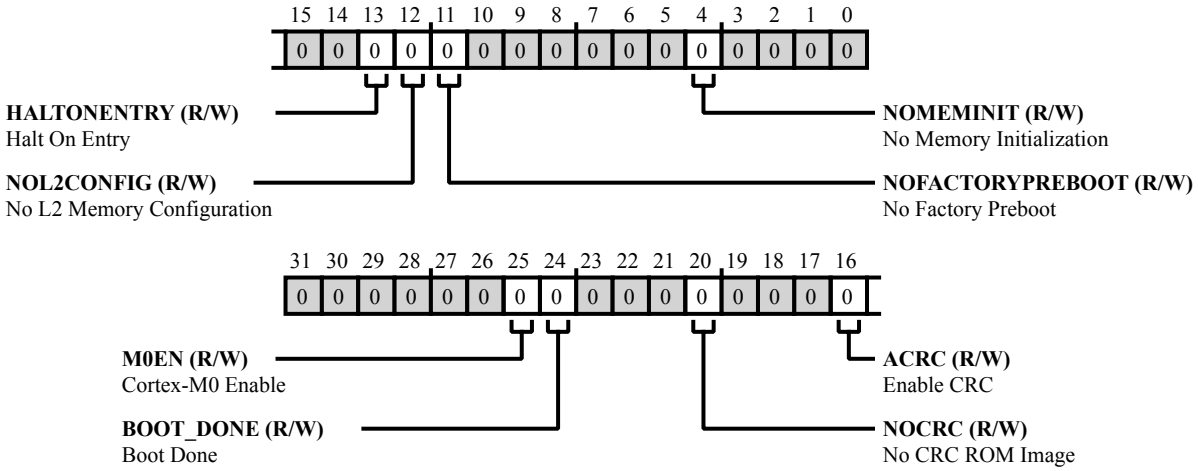


Figure 46-6: TAPC_RCMSG_SET Register Diagram

Table 46-11: TAPC_RCMSG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W1S)	M0EN	Enable M0 out of reset into idle (wfi) state.
24 (R/W1S)	BOOT_DONE	1 indicates that initial Booting and initializations are complete.
20 (R/W1S)	NOCRC	Disable CRC checking of ROM Image.
16 (R/W1S)	ACRC	Enable CRC of the application (first 4K of flash).
13 (R/W1S)	HALTONENTRY	Enter idle, or wfi, on entry.
12 (R/W1S)	NOL2CONFIG	Do not configure L2.
11 (R/W1S)	NOFACTORYPREBOOT	Do not perform factory preboot.
4 (R/W1S)	NOMEMINIT	Do not initialize memory (Main M4 SRAM and MBOX memories).

Run Control Message Toggle Register

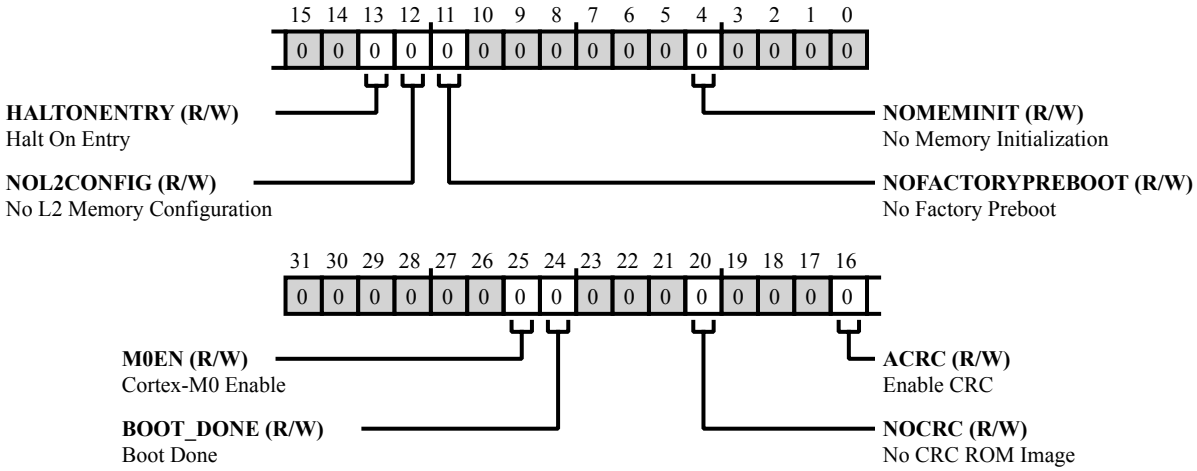


Figure 46-7: TAPC_RCMSG_TGL Register Diagram

Table 46-12: TAPC_RCMSG_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25 (R/W1T)	M0EN	Enable M0 out of reset into idle (wfi) state.
24 (R/W1T)	BOOT_DONE	1 indicates that initial Booting and initializations are complete.
20 (R/W1T)	NOCRC	Disable CRC checking of ROM Image.
16 (R/W1T)	ACRC	Enable CRC of the application (first 4K of flash).
13 (R/W1T)	HALTONENTRY	Enter idle, or wfi, on entry.
12 (R/W1T)	NOL2CONFIG	Do not configure L2.
11 (R/W1T)	NOFACTORYPREBOOT	Do not perform factory preboot.
4 (R/W1T)	NOMEMINIT	Do not initialize memory (Main M4 SRAM and MBOX memories).

System Run Control Message Register

The TAPC_SCMSG register controls boot options through a register that uses the system hardware reset rather than a system reset.

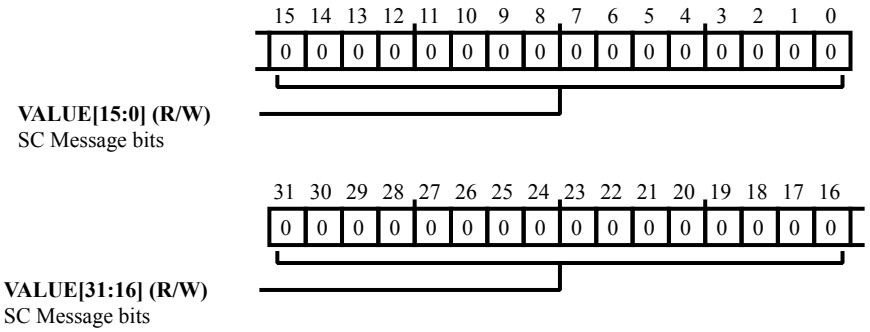


Figure 46-8: TAPC_SCMSG Register Diagram

Table 46-13: TAPC_SCMSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SC Message bits.

System Run Control Message Clear Register

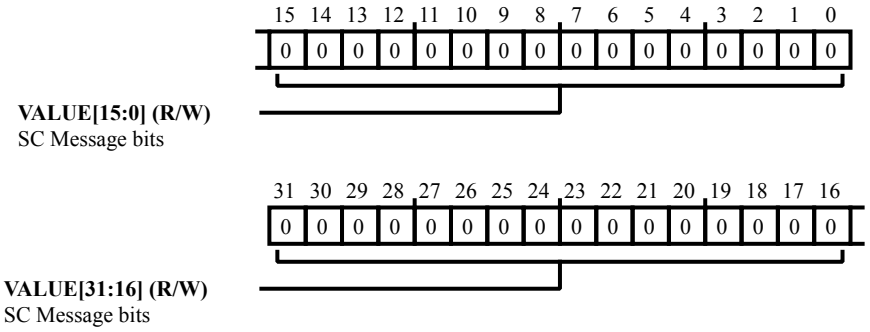


Figure 46-9: TAPC_SCMSG_CLR Register Diagram

Table 46-14: TAPC_SCMSG_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1S)	VALUE	SC Message bits.

System Run Control Message Set Register

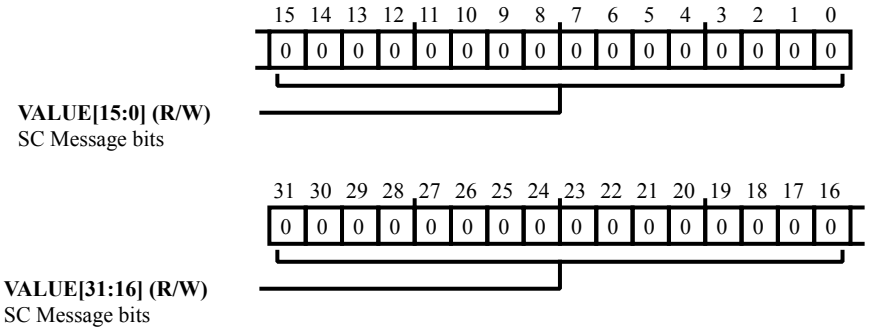


Figure 46-10: TAPC_SCMSG_SET Register Diagram

Table 46-15: TAPC_SCMSG_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1C)	VALUE	SC Message bits.

System Run Control Message Toggle Register

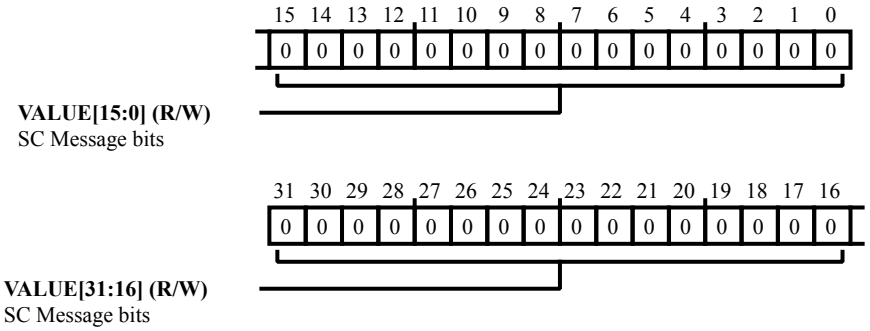


Figure 46-11: TAPC_SCMSG_TGL Register Diagram

Table 46-16: TAPC_SCMSG_TGL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W1T)	VALUE	SC Message bits.

Secure Debug Key 0 Register

The TAPC_SDBGKEY0 register allows a locked part to unlock debug access through the JTAG or SWD interfaces. A debug key of 128 bits needs to be written into the Secure Debug Key registers (TAPC_SDBGKEY0, TAPC_SDBGKEY1, TAPC_SDBGKEY2, TAPC_SDBGKEY3) in the TAPC through the peripheral bus interface.

These registers hold the value of the key against which a matching key provided by the debug user is compared to enable a debug session. The task of writing these registers is performed (initially) by boot ROM code which copies a customer-selected key from the Flash memory info block to these registers.

An SDBGKEY value of all 0's is always an invalid key, a value of all 1's match the default value of the Secure Debug Key Compare registers and requires no entry in these registers. It is recommended programs have a significant number of 0's and 1's in a pseudo-random pattern throughout the 128-bit code for maximum protection.

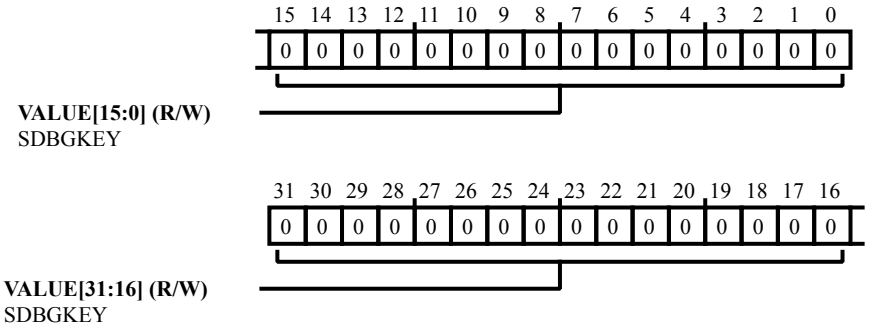


Figure 46-12: TAPC_SDBGKEY0 Register Diagram

Table 46-17: TAPC_SDBGKEY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The TAPC_SDBGKEY0.VALUE bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 1 Register

The TAPC_SDBGKEY1 register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the TAPC_SDBGKEY0 register description for more information.

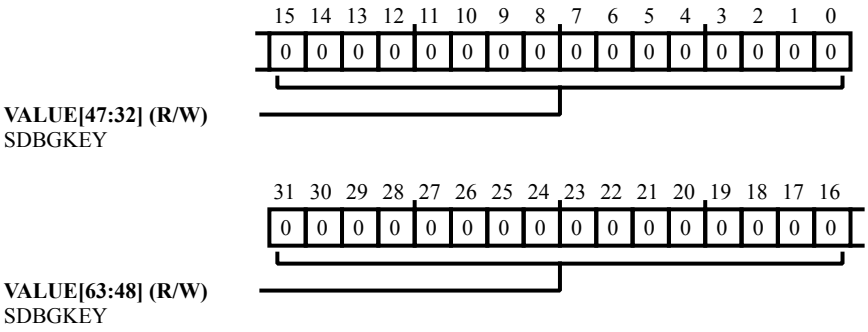


Figure 46-13: TAPC_SDBGKEY1 Register Diagram

Table 46-18: TAPC_SDBGKEY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The TAPC_SDBGKEY1.VALUE bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 2 Register

The TAPC_SDBGKEY2 register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the TAPC_SDBGKEY0 register description for more information.

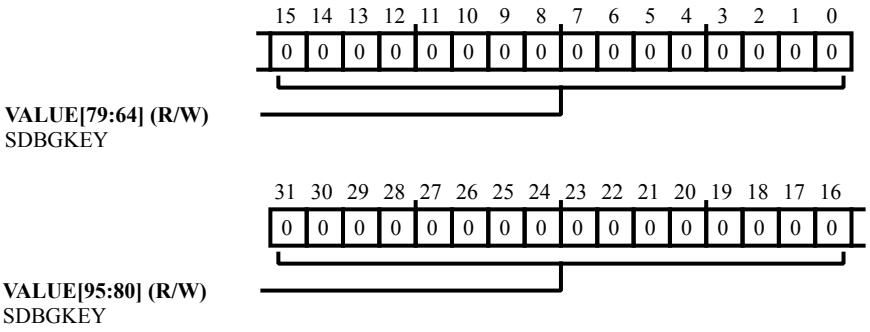


Figure 46-14: TAPC_SDBGKEY2 Register Diagram

Table 46-19: TAPC_SDBGKEY2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The TAPC_SDBGKEY2.VALUE bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 3 Register

The TAPC_SDBGKEY3 register allows a locked part to unlock debug access through the JTAG or SWD interfaces. See the TAPC_SDBGKEY0 register description for more information.

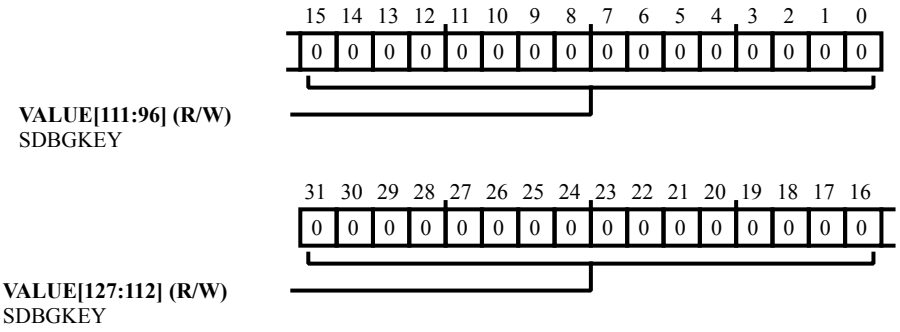


Figure 46-15: TAPC_SDBGKEY3 Register Diagram

Table 46-20: TAPC_SDBGKEY3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY. The TAPC_SDBGKEY3.VALUE bit field holds the value of the key against which a matching key provided by the debug user is compared to enable a debug session.

Secure Debug Key 0 Compare Register

The TAPC_SDBGKEYCMP0 register holds the secure debug comparison key word 0.

To unlock debug access to the system, a Debug key of 128 bits needs to be written into these registers (TAPC_SDBGKEYCMP0, TAPC_SDBGKEYCMP1, TAPC_SDBGKEYCMP2, TAPC_SDBGKEYCMP3). The Compare register access can be performed through JTAG scan or through APB MMR access to facilitate SWD debug options.

For proper secure key operation, the DBGKEY value must be entered after the SDBGKEYCMP value or a sticky fail occurs. The DBGKEY matching the default SDBGKEYCMP (all 1's) may be entered without need to load the SDBGKEYCMP if debugger access is to be granted without and security code requirements.

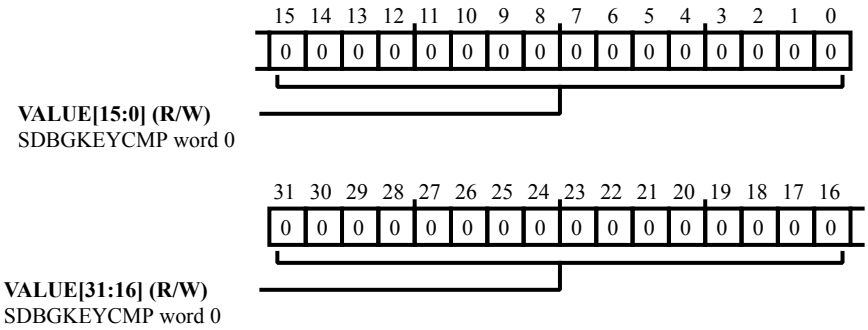


Figure 46-16: TAPC_SDBGKEYCMP0 Register Diagram

Table 46-21: TAPC_SDBGKEYCMP0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEYCMP word 0.

Secure Debug Key 1 Compare Register

The TAPC_SDBGKEYCMP1 register holds the secure debug comparison key word 1. See the TAPC_SDBGKEYCMP0 register description for more information.

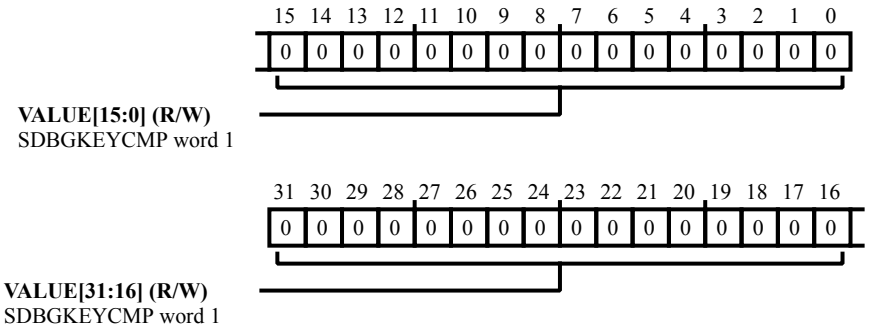


Figure 46-17: TAPC_SDBGKEYCMP1 Register Diagram

Table 46-22: TAPC_SDBGKEYCMP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEYCMP word 1.

Secure Debug Key 2 Compare Register

The TAPC_SDBGKEYCMP2 register holds the secure debug comparison key word 2. See the TAPC_SDBGKEYCMP0 register description for more information.

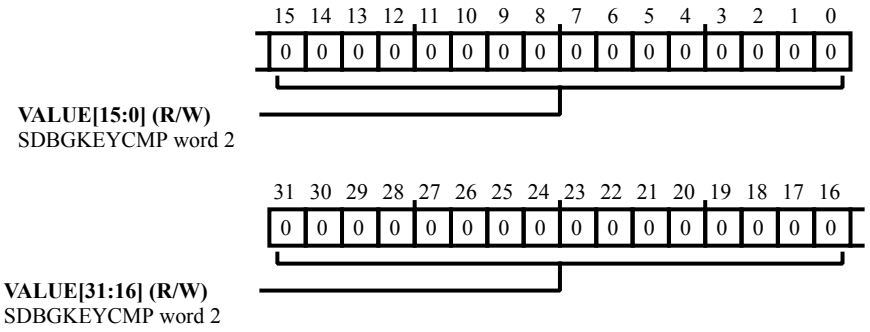


Figure 46-18: TAPC_SDBGKEYCMP2 Register Diagram

Table 46-23: TAPC_SDBGKEYCMP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEYCMP word 2.

Secure Debug Key 3 Compare Register

The TAPC_SDBGKEYCMP3 register holds the secure debug comparison key word 3. See the TAPC_SDBGKEYCMP0 register description for more information.

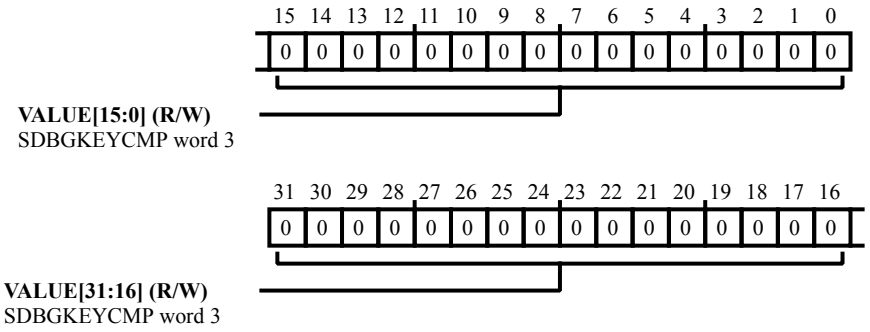


Figure 46-19: TAPC_SDBGKEYCMP3 Register Diagram

Table 46-24: TAPC_SDBGKEYCMP3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEYCMP word 3.

Secure Debug Key 0 Identification Register

The TAPC_SDBGKEYID0 register allows programs to place a unique code from a non-volatile memory that a debugger can use to identify an individual die. The JTAG scan register is read only. An update to this register does not alter the TAPC value.

System writes to this register are transferred to the JTAG scan register when the APB MMR writes to the TAPC_SDBGKEYID3 register. Due to the JTAG having a completely asynchronous clock to the system, it is possible for the system to issue a transfer request to the JTAG system that can effectively take forever to complete. A status bit in the TAPC_DBGSTAT register indicates if an APB MMR write attempts to transfer to the JTAG scan register before a previous transfer attempt completes.

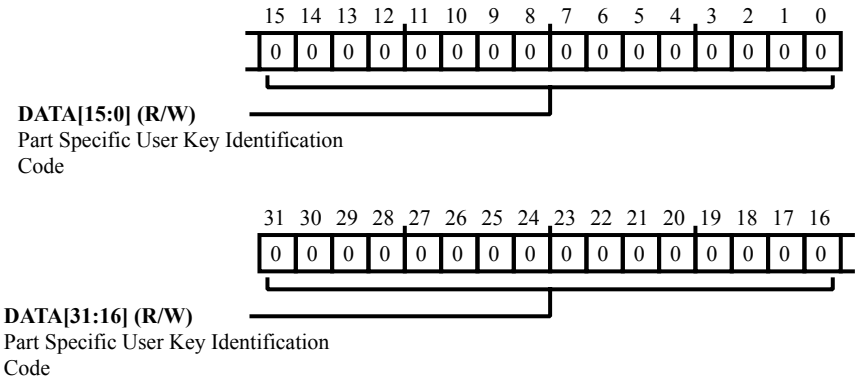


Figure 46-20: TAPC_SDBGKEYID0 Register Diagram

Table 46-25: TAPC_SDBGKEYID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Part Specific User Key Identification Code.

Secure Debug Key 1 Key Identification Register

The TAPC_SDBGKEYID1 register allows programs to place a unique code from a non-volatile memory that a debugger can use to identify an individual die. See the TAPC_SDBGKEYID0 register description for more information.

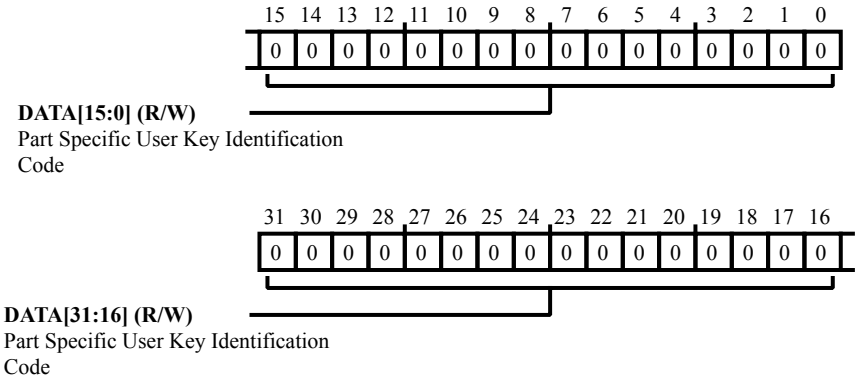


Figure 46-21: TAPC_SDBGKEYID1 Register Diagram

Table 46-26: TAPC_SDBGKEYID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Part Specific User Key Identification Code.

Secure Debug Key 2 Key Identification Register

The TAPC_SDBGKEYID2 register allows programs to place a unique code from a non-volatile memory that a debugger can use to identify an individual die. See the TAPC_SDBGKEYID0 register description for more information.

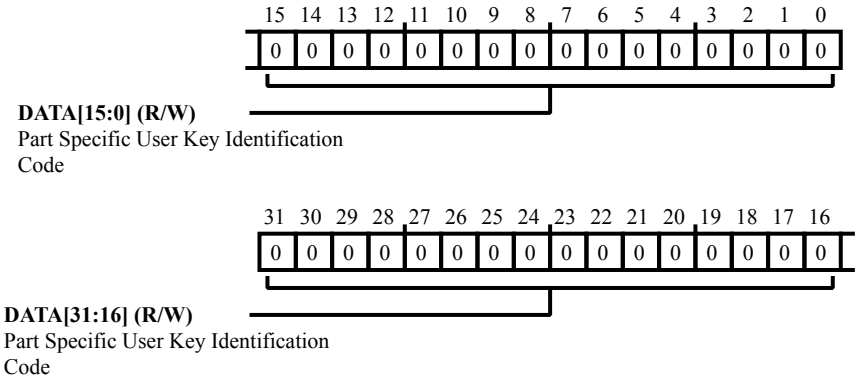


Figure 46-22: TAPC_SDBGKEYID2 Register Diagram

Table 46-27: TAPC_SDBGKEYID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Part Specific User Key Identification Code.

Secure Debug Key 3 Key Identification Register

The TAPC_SDBGKEYID3 register allows programs to place a unique code from a non-volatile memory that a debugger can use to identify an individual die. See the TAPC_SDBGKEYID0 register description for more information.

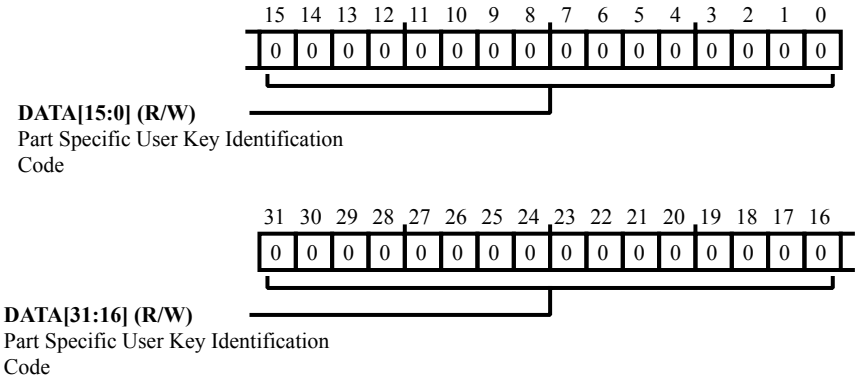


Figure 46-23: TAPC_SDBGKEYID3 Register Diagram

Table 46-28: TAPC_SDBGKEYID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Part Specific User Key Identification Code.

Secure Debug Key Control Register

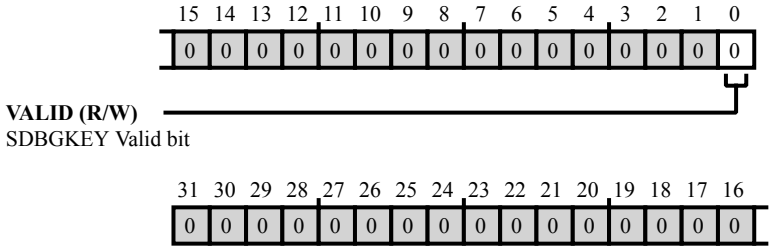


Figure 46-24: TAPC_SDBGKEY_CTL Register Diagram

Table 46-29: TAPC_SDBGKEY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALID	SDBGKEY Valid bit.

Secure Debug Key Status Register

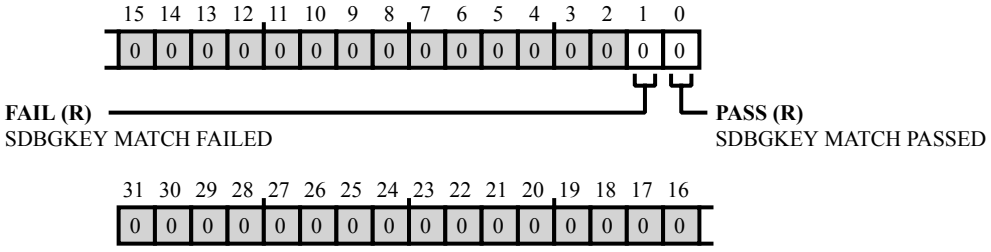


Figure 46-25: TAPC_SDBGKEY_STAT Register Diagram

Table 46-30: TAPC_SDBGKEY_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	FAIL	SDBGKEY MATCH FAILED.
0 (R/NW)	PASS	SDBGKEY MATCH PASSED.

USERCODE Register

The TAPC_USERCODE register

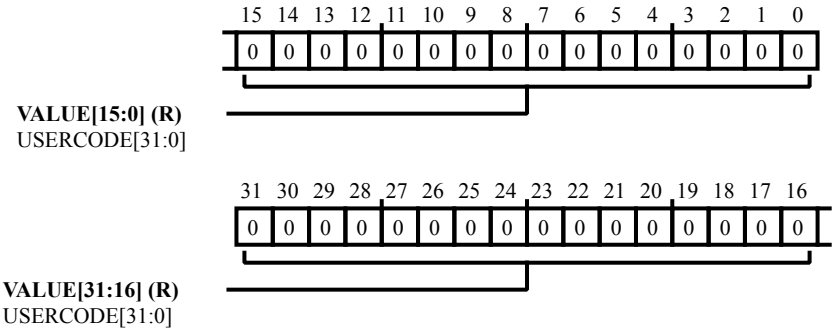


Figure 46-26: TAPC_USERCODE Register Diagram

Table 46-31: TAPC_USERCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	USERCODE[31:0].

CM41X_M4 CTI Register Descriptions

CSCTI_Register_Definitions (CTI) contains the following registers.

Table 46-32: CM41X_M4 CTI Register List

Name	Description
CTI_ASICCTL	External Multiplexor Control Register
CTI_AUTHSTATUS	Authentication Status
CTI_CLAIMCLR	Claim Tag Clear Register
CTI_CLAIMSET	Claim Tag Set Register
CTI_COMPID0	Component ID0
CTI_COMPID1	Component ID1
CTI_COMPID2	Component ID2
CTI_COMPID3	Component ID3
CTI_CTIAPPCLEAR	CTI Application Trigger Clear Register
CTI_CTIAPPPULSE	CTI Application Pulse Register
CTI_CTIAPPSET	CTI Application Trigger Set Register

Table 46-32: CM41X_M4 CTI Register List (Continued)

Name	Description
CTI_CTICHINSTATUS	CTI Channel In Status Register
CTI_CTICHOUTSTATUS	CTI Channel Out Status Register
CTI_CTICONTROL	CTI Control Register
CTI_CTIGATE	Enable CTI Channel Gate Register
CTI_CTIINEN0	CTI Trigger 0 to Channel Enable Register
CTI_CTIINEN1	CTI Trigger 1 to Channel Enable Register
CTI_CTIINEN2	CTI Trigger 2 to Channel Enable Register
CTI_CTIINEN3	CTI Trigger 3 to Channel Enable Register
CTI_CTIINEN4	CTI Trigger 4 to Channel Enable Register
CTI_CTIINEN5	CTI Trigger 5 to Channel Enable Register
CTI_CTIINEN6	CTI Trigger 6 to Channel Enable Register
CTI_CTIINEN7	CTI Trigger 7 to Channel Enable Register
CTI_CTIINTACK	CTI Interrupt Acknowledge Register
CTI_CTIOUTEN0	CTI Channel to Trigger 0 Enable Register
CTI_CTIOUTEN1	CTI Channel to Trigger 1 Enable Register
CTI_CTIOUTEN2	CTI Channel to Trigger 2 Enable Register
CTI_CTIOUTEN3	CTI Channel to Trigger 3 Enable Register
CTI_CTIOUTEN4	CTI Channel to Trigger 4 Enable Register
CTI_CTIOUTEN5	CTI Channel to Trigger 5 Enable Register
CTI_CTIOUTEN6	CTI Channel to Trigger 6 Enable Register
CTI_CTIOUTEN7	CTI Channel to Trigger 7 Enable Register
CTI_CTITRIGINSTATUS	CTI Trigger In Status Register
CTI_CTITRIGOUTSTATUS	CTI Trigger Out Status Register
CTI_DEVID	Device ID
CTI_DEVTYPE	Device Type
CTI_ITCHIN	ITCHIN
CTI_ITCHINACK	ITCHINACK
CTI_ITCHOUT	ITCHOUT
CTI_ITCHOUTACK	ITCHOUTACK
CTI_ITCTRL	Integration Mode Control Register
CTI_ITTRIGIN	ITTRIGIN

Table 46-32: CM41X_M4 CTI Register List (Continued)

Name	Description
CTI_ITTRIGINACK	ITTRIGINACK
CTI_ITTRIGOUT	ITTRIGOUT
CTI_ITTRIGOUTACK	ITTRIGOUTACK
CTI_LAR	Lock Access Register
CTI_LSR	Lock Status Register
CTI_PERIPHID0	Peripheral ID0
CTI_PERIPHID1	Peripheral ID1
CTI_PERIPHID2	Peripheral ID2
CTI_PERIPHID3	Peripheral ID3
CTI_PERIPHID4	Peripheral ID4
CTI_PERIPHID5	Peripheral ID5
CTI_PERIPHID6	Peripheral ID6
CTI_PERIPHID7	Peripheral ID7

External Multiplexor Control Register

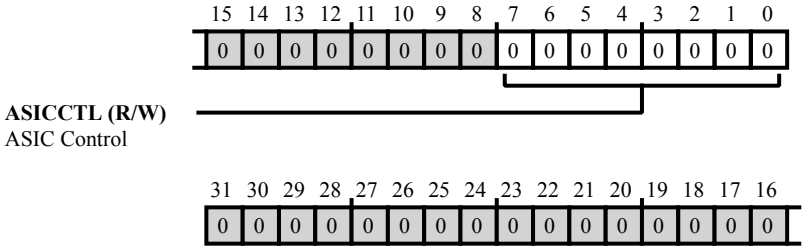


Figure 46-27: CTI_ASICCTL Register Diagram

Table 46-33: CTI_ASICCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	ASICCTL	ASIC Control.

Authentication Status

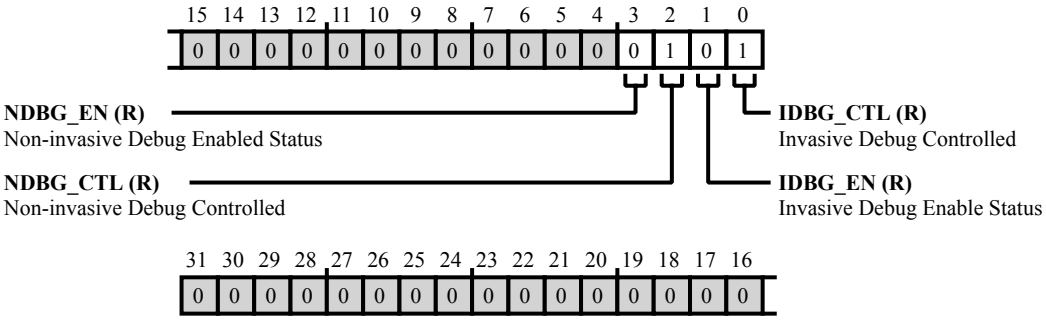


Figure 46-28: CTI_AUTHSTATUS Register Diagram

Table 46-34: CTI_AUTHSTATUS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/NW)	NDBG_EN	Non-invasive Debug Enabled Status.
2 (R/NW)	NDBG_CTL	Non-invasive Debug Controlled.
1 (R/NW)	IDBG_EN	Invasive Debug Enable Status.
0 (R/NW)	IDBG_CTL	Invasive Debug Controlled.

Claim Tag Clear Register

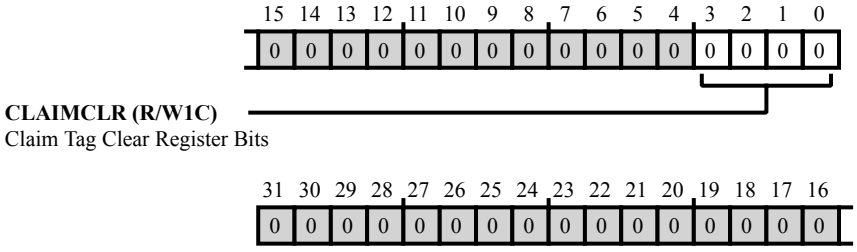


Figure 46-29: CTI_CLAIMCLR Register Diagram

Table 46-35: CTI_CLAIMCLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W1C)	CLAIMCLR	Claim Tag Clear Register Bits.

Claim Tag Set Register

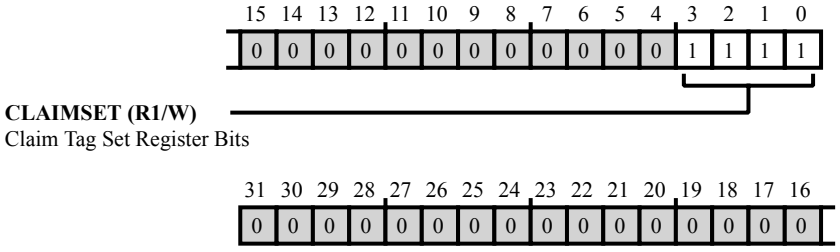


Figure 46-30: CTI_CLAIMSET Register Diagram

Table 46-36: CTI_CLAIMSET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R1/W)	CLAIMSET	Claim Tag Set Register Bits.

Component ID0

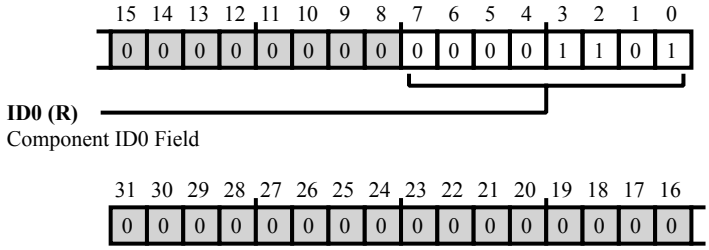


Figure 46-31: CTI_COMPID0 Register Diagram

Table 46-37: CTI_COMPID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	ID0	Component ID0 Field.

Component ID1

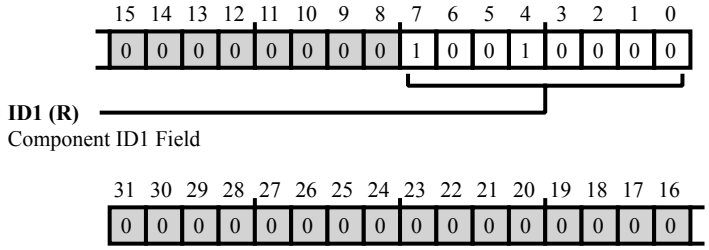


Figure 46-32: CTI_COMPID1 Register Diagram

Table 46-38: CTI_COMPID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	ID1	Component ID1 Field.

Component ID2

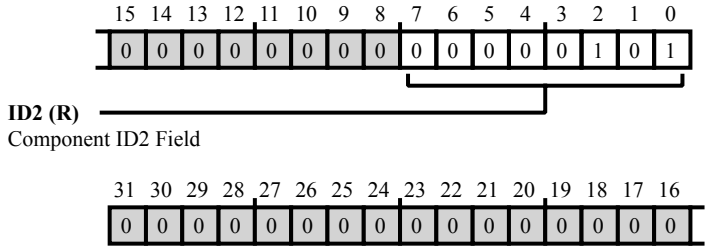


Figure 46-33: CTI_COMPID2 Register Diagram

Table 46-39: CTI_COMPID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	ID2	Component ID2 Field.

Component ID3

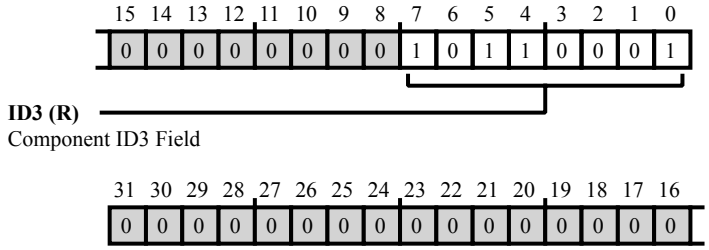


Figure 46-34: CTI_COMPID3 Register Diagram

Table 46-40: CTI_COMPID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	ID3	Component ID3 Field.

CTI Application Trigger Clear Register

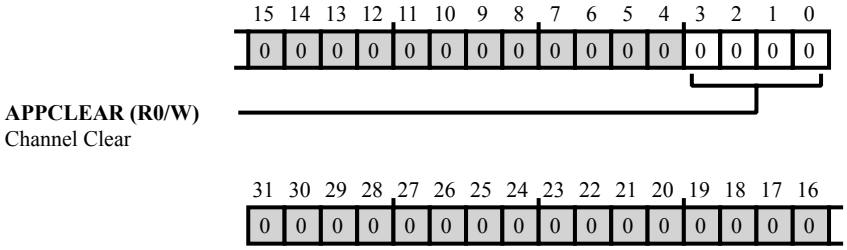


Figure 46-35: CTI_CTIAPPCLEAR Register Diagram

Table 46-41: CTI_CTIAPPCLEAR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R0/W)	APPCLEAR	Channel Clear.

CTI Application Pulse Register

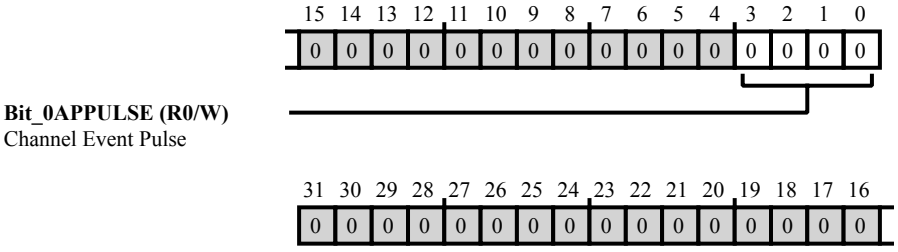


Figure 46-36: CTI_CTIAPPPULSE Register Diagram

Table 46-42: CTI_CTIAPPPULSE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R0/W)	BIT_0APPULSE	Channel Event Pulse.

CTI Application Trigger Set Register

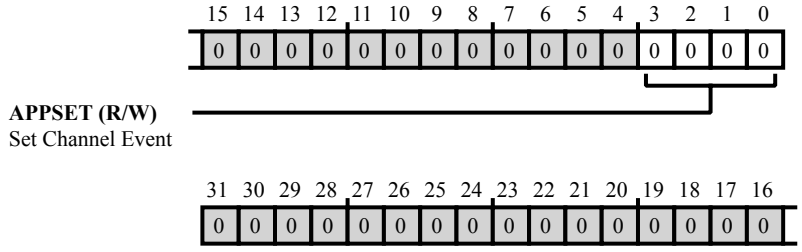


Figure 46-37: CTI_CTIAPPSET Register Diagram

Table 46-43: CTI_CTIAPPSET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	APPSET	Set Channel Event.

CTI Channel In Status Register

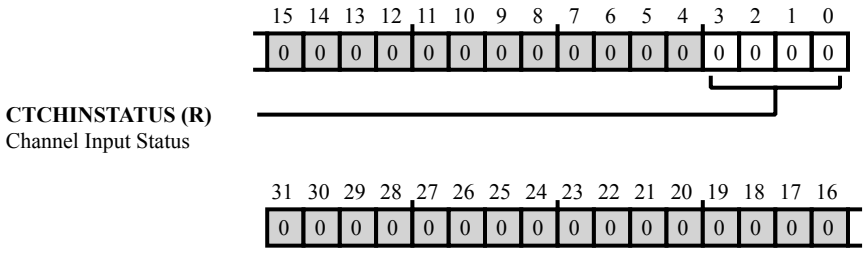


Figure 46-38: CTI_CTICHINSTATUS Register Diagram

Table 46-44: CTI_CTICHINSTATUS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/NW)	CTCHINSTATUS	Channel Input Status.

CTI Channel Out Status Register

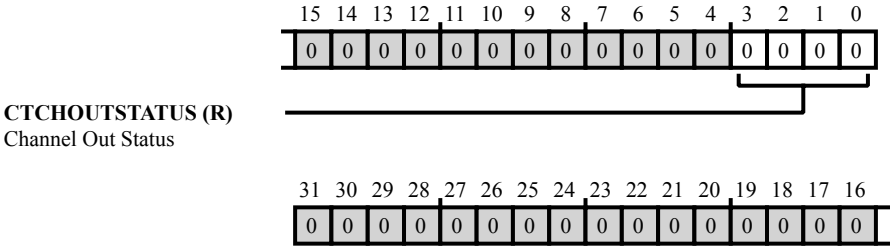


Figure 46-39: CTI_CTICHOUTSTATUS Register Diagram

Table 46-45: CTI_CTICHOUTSTATUS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/NW)	CTCHOUTSTATUS	Channel Out Status.

CTI Control Register

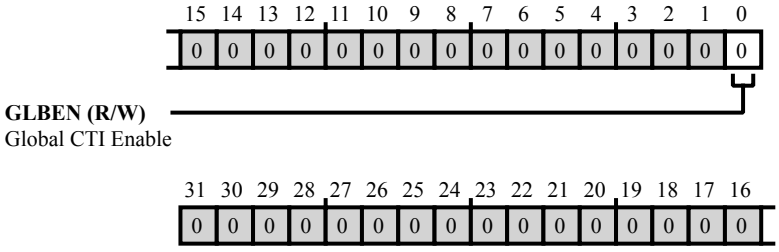


Figure 46-40: CTI_CTICONTROL Register Diagram

Table 46-46: CTI_CTICONTROL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	GLBEN	Global CTI Enable.	
		0	Disabled
		1	Enabled

Enable CTI Channel Gate Register

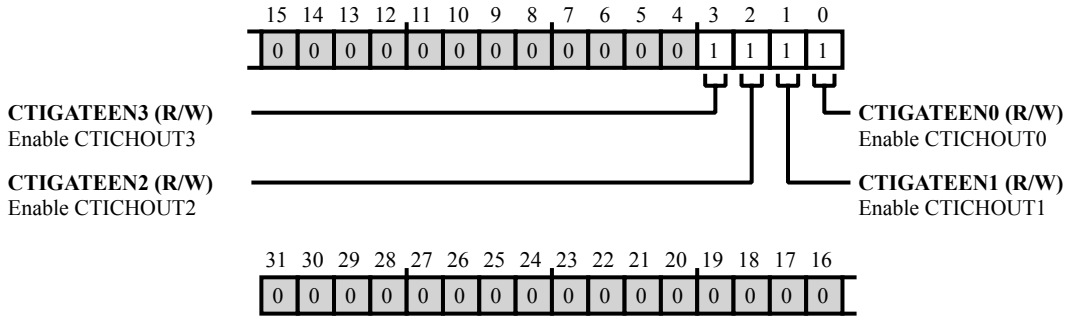


Figure 46-41: CTI_CTIGATE Register Diagram

Table 46-47: CTI_CTIGATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	CTIGATEEN3	Enable CTICHOUT3.
2 (R/W)	CTIGATEEN2	Enable CTICHOUT2.
1 (R/W)	CTIGATEEN1	Enable CTICHOUT1.
0 (R/W)	CTIGATEEN0	Enable CTICHOUT0.

CTI Trigger 0 to Channel Enable Register

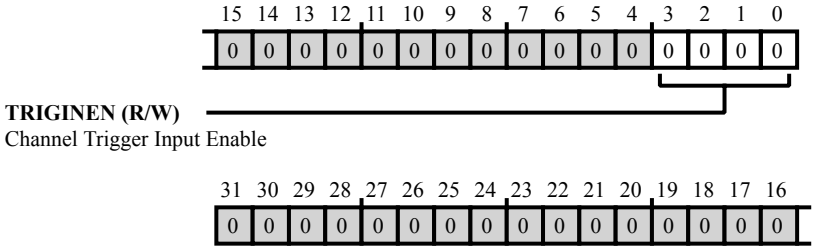


Figure 46-42: CTI_CTIINEN0 Register Diagram

Table 46-48: CTI_CTIINEN0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGINEN	Channel Trigger Input Enable.

CTI Trigger 1 to Channel Enable Register

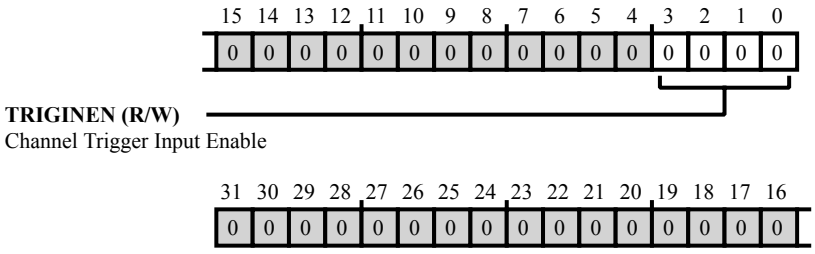


Figure 46-43: CTI_CTIINEN1 Register Diagram

Table 46-49: CTI_CTIINEN1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGINEN	Channel Trigger Input Enable.

CTI Trigger 2 to Channel Enable Register

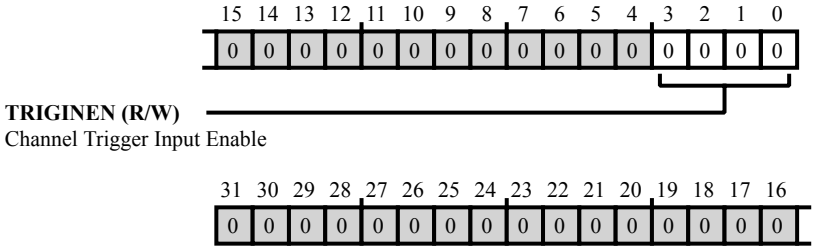


Figure 46-44: CTI_CTIINEN2 Register Diagram

Table 46-50: CTI_CTIINEN2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGINEN	Channel Trigger Input Enable.

CTI Trigger 3 to Channel Enable Register

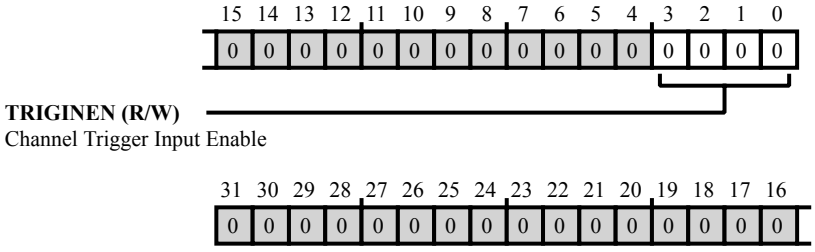


Figure 46-45: CTI_CTIINEN3 Register Diagram

Table 46-51: CTI_CTIINEN3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGINEN	Channel Trigger Input Enable.

CTI Trigger 4 to Channel Enable Register

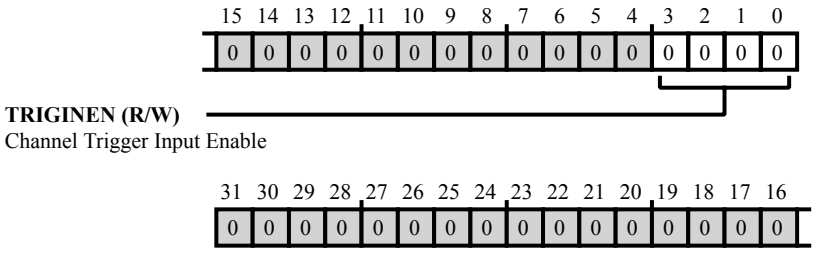


Figure 46-46: CTI_CTIINEN4 Register Diagram

Table 46-52: CTI_CTIINEN4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGINEN	Channel Trigger Input Enable.

CTI Trigger 5 to Channel Enable Register

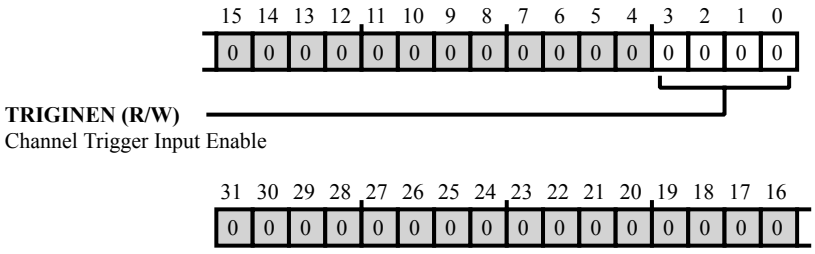


Figure 46-47: CTI_CTIINEN5 Register Diagram

Table 46-53: CTI_CTIINEN5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGINEN	Channel Trigger Input Enable.

CTI Trigger 6 to Channel Enable Register

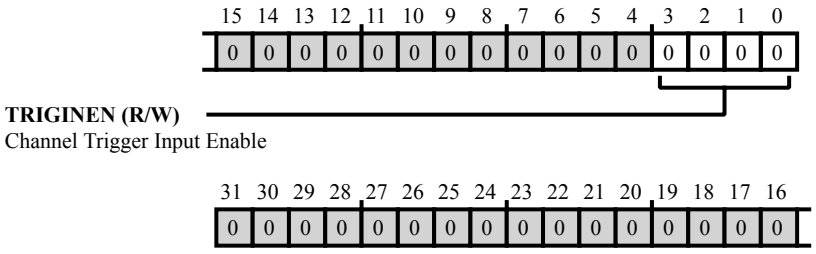


Figure 46-48: CTI_CTIINEN6 Register Diagram

Table 46-54: CTI_CTIINEN6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGINEN	Channel Trigger Input Enable.

CTI Trigger 7 to Channel Enable Register

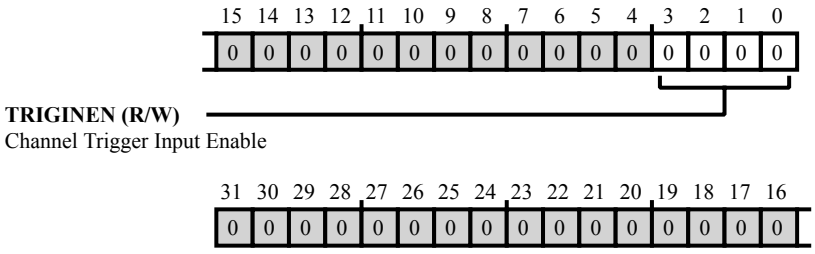


Figure 46-49: CTI_CTIINEN7 Register Diagram

Table 46-55: CTI_CTIINEN7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGINEN	Channel Trigger Input Enable.

CTI Interrupt Acknowledge Register

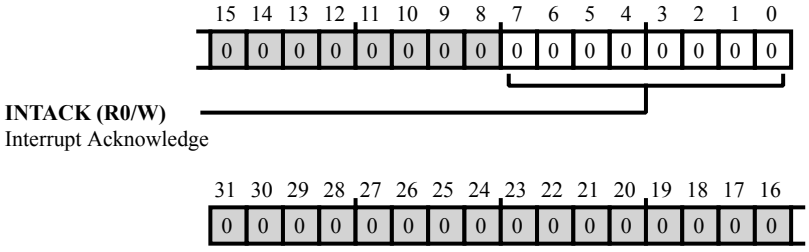


Figure 46-50: CTI_CTIINTACK Register Diagram

Table 46-56: CTI_CTIINTACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	INTACK	Interrupt Acknowledge.

CTI Channel to Trigger 0 Enable Register

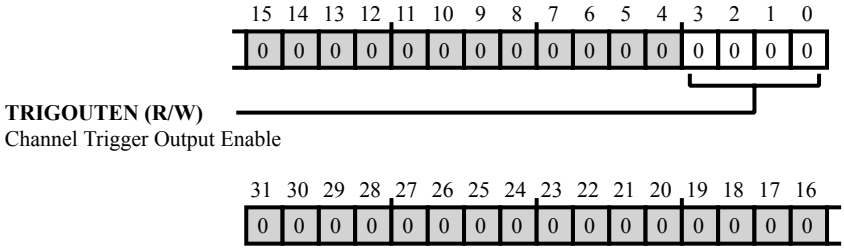


Figure 46-51: CTI_CTIOUTEN0 Register Diagram

Table 46-57: CTI_CTIOUTEN0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGOUTEN	Channel Trigger Output Enable.

CTI Channel to Trigger 1 Enable Register

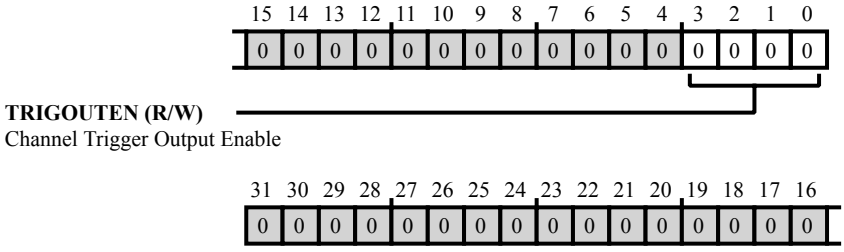


Figure 46-52: CTI_CTIOUTEN1 Register Diagram

Table 46-58: CTI_CTIOUTEN1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGOUTEN	Channel Trigger Output Enable.

CTI Channel to Trigger 2 Enable Register

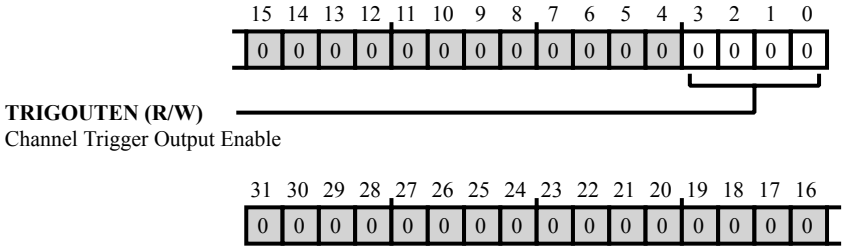


Figure 46-53: CTI_CTIOUTEN2 Register Diagram

Table 46-59: CTI_CTIOUTEN2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGOUTEN	Channel Trigger Output Enable.

CTI Channel to Trigger 3 Enable Register

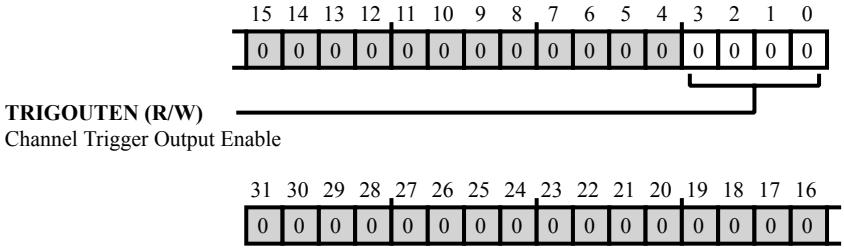


Figure 46-54: CTI_CTIOUTEN3 Register Diagram

Table 46-60: CTI_CTIOUTEN3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGOUTEN	Channel Trigger Output Enable.

CTI Channel to Trigger 4 Enable Register

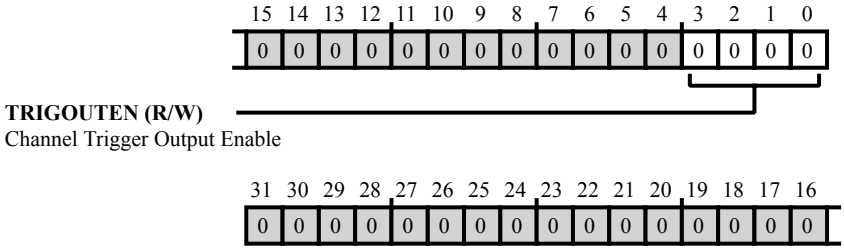


Figure 46-55: CTI_CTIOUTEN4 Register Diagram

Table 46-61: CTI_CTIOUTEN4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGOUTEN	Channel Trigger Output Enable.

CTI Channel to Trigger 5 Enable Register

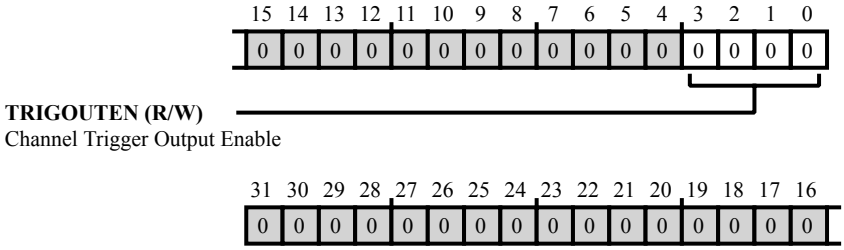


Figure 46-56: CTI_CTIOUTEN5 Register Diagram

Table 46-62: CTI_CTIOUTEN5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGOUTEN	Channel Trigger Output Enable.

CTI Channel to Trigger 6 Enable Register

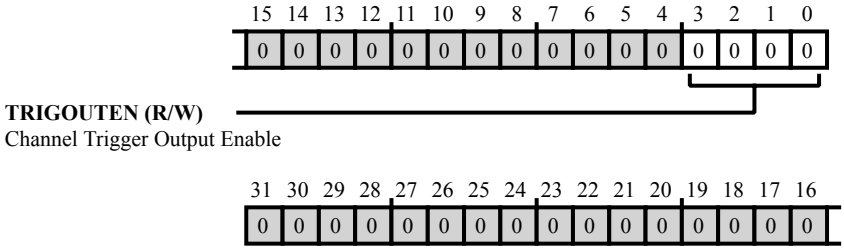


Figure 46-57: CTI_CTIOUTEN6 Register Diagram

Table 46-63: CTI_CTIOUTEN6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGOUTEN	Channel Trigger Output Enable.

CTI Channel to Trigger 7 Enable Register

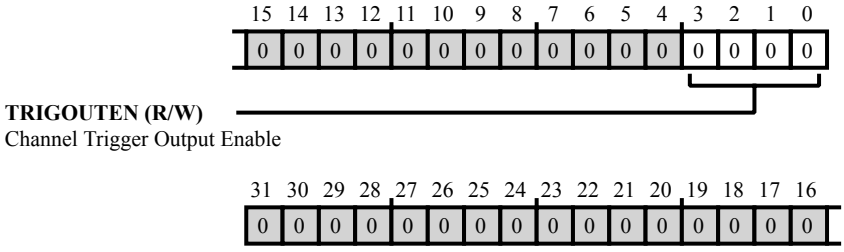


Figure 46-58: CTI_CTIOUTEN7 Register Diagram

Table 46-64: CTI_CTIOUTEN7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	TRIGOUTEN	Channel Trigger Output Enable.

CTI Trigger In Status Register

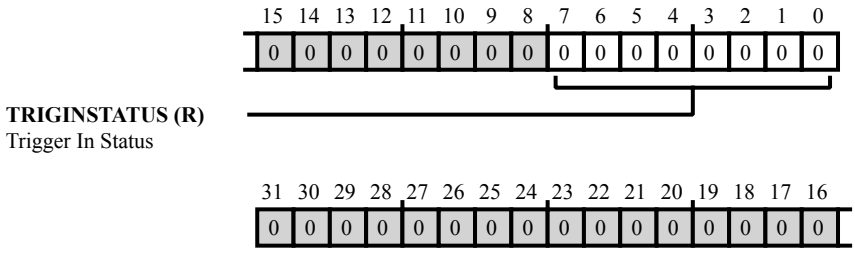


Figure 46-59: CTI_CTITRIGINSTATUS Register Diagram

Table 46-65: CTI_CTITRIGINSTATUS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	TRIGINSTATUS	Trigger In Status.

CTI Trigger Out Status Register

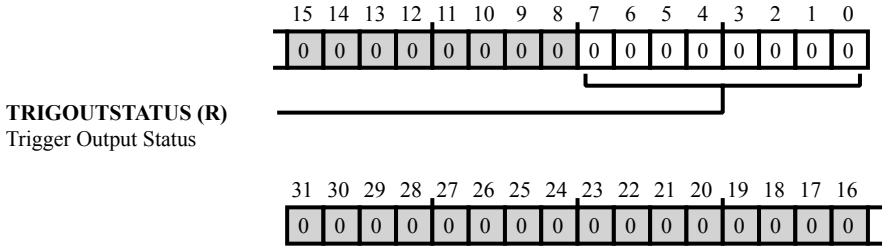


Figure 46-60: CTI_CTITRIGOUTSTATUS Register Diagram

Table 46-66: CTI_CTITRIGOUTSTATUS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	TRIGOUTSTATUS	Trigger Output Status.

Device ID

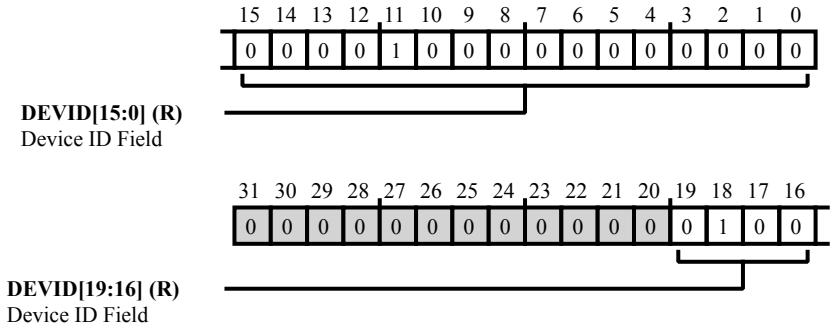


Figure 46-61: CTI_DEVID Register Diagram

Table 46-67: CTI_DEVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:0 (R/NW)	DEVID	Device ID Field.

Device Type

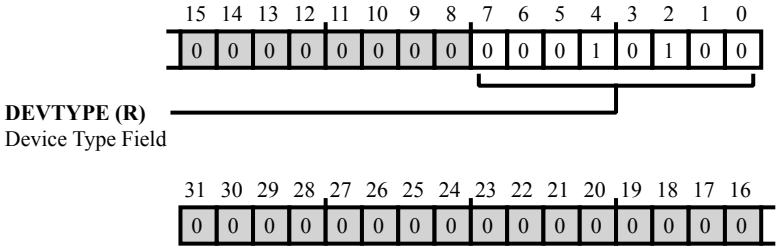


Figure 46-62: CTI_DEVTYPE Register Diagram

Table 46-68: CTI_DEVTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	DEVTYPE	Device Type Field.

ITCHIN

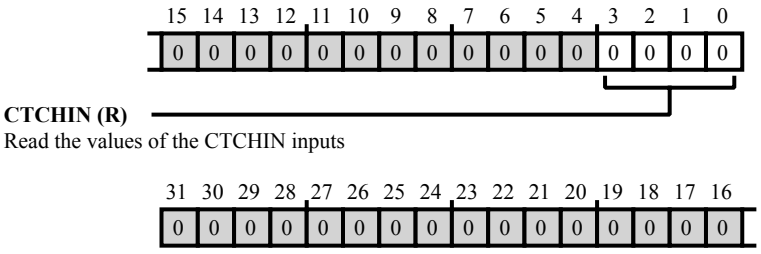


Figure 46-63: CTI_ITCHIN Register Diagram

Table 46-69: CTI_ITCHIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/NW)	CTCHIN	Read the values of the CTCHIN inputs.

ITCHINACK

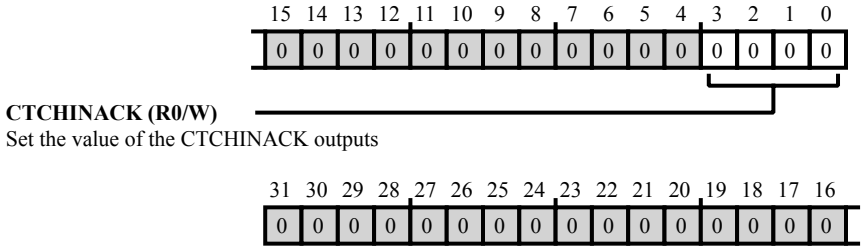


Figure 46-64: CTI_ITCHINACK Register Diagram

Table 46-70: CTI_ITCHINACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R0/W)	CTCHINACK	Set the value of the CTCHINACK outputs.

ITCHOUT

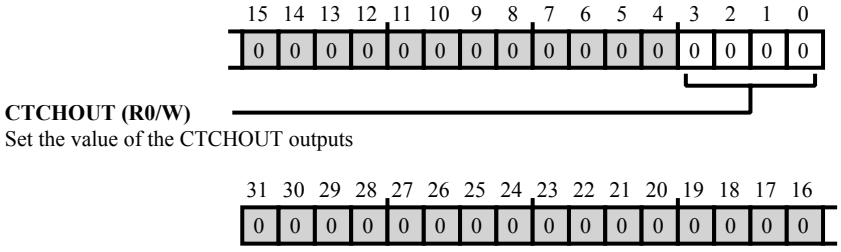


Figure 46-65: CTI_ITCHOUT Register Diagram

Table 46-71: CTI_ITCHOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R0/W)	CTCHOUT	Set the value of the CTCHOUT outputs.

ITCHOUTACK

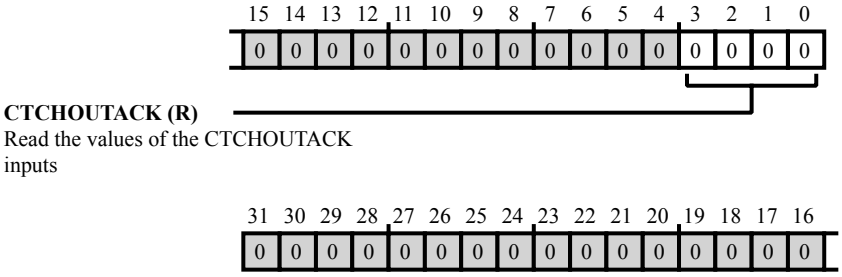


Figure 46-66: CTI_ITCHOUTACK Register Diagram

Table 46-72: CTI_ITCHOUTACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/NW)	CTCHOUTACK	Read the values of the CTCHOUTACK inputs.

Integration Mode Control Register

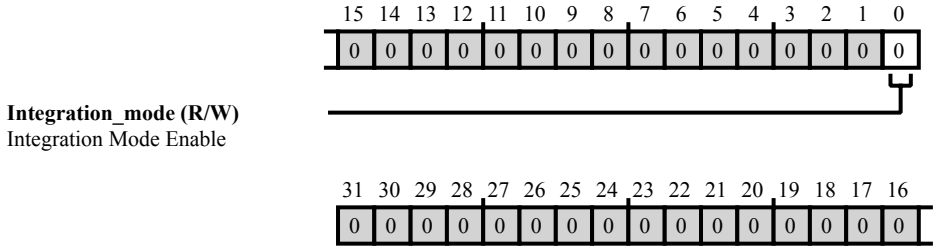


Figure 46-67: CTI_ITCTRL Register Diagram

Table 46-73: CTI_ITCTRL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	INTEGRATION_MODE	Integration Mode Enable.

ITTRIGIN

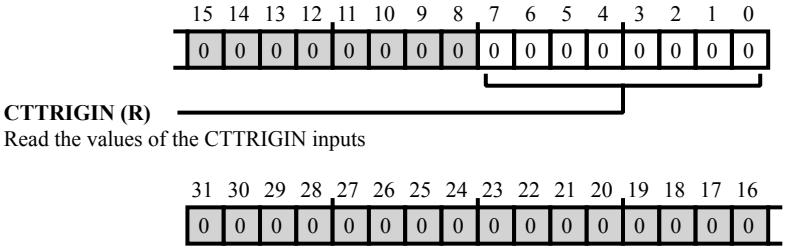


Figure 46-68: CTI_ITTRIGIN Register Diagram

Table 46-74: CTI_ITTRIGIN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	CTTRIGIN	Read the values of the CTTRIGIN inputs.

ITTRIGINACK

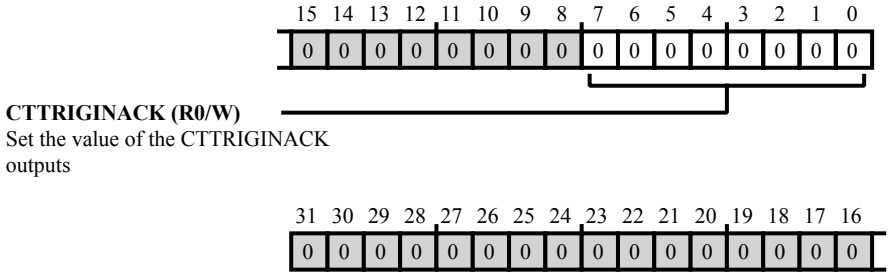


Figure 46-69: CTI_ITTRIGINACK Register Diagram

Table 46-75: CTI_ITTRIGINACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	CTTRIGINACK	Set the value of the CTTRIGINACK outputs.

ITTRIGOUT

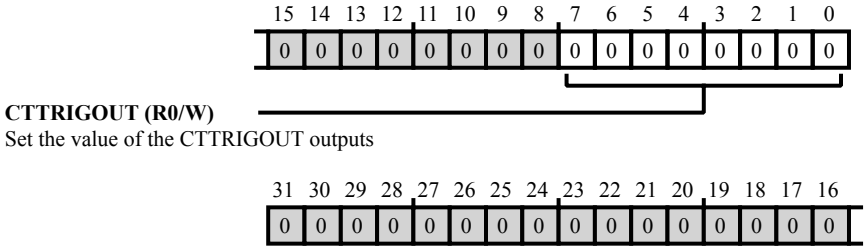


Figure 46-70: CTI_ITTRIGOUT Register Diagram

Table 46-76: CTI_ITTRIGOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R0/W)	CTTRIGOUT	Set the value of the CTTRIGOUT outputs.

ITTRIGOUTACK

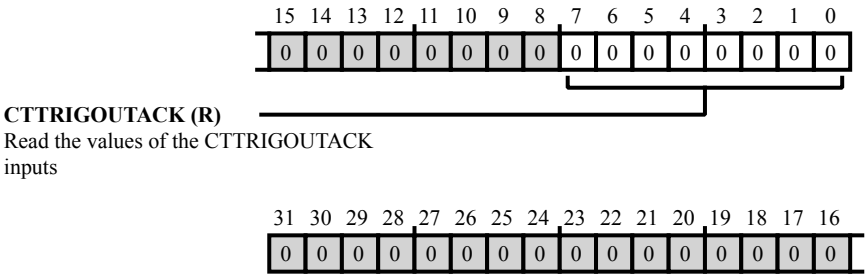


Figure 46-71: CTI_ITTRIGOUTACK Register Diagram

Table 46-77: CTI_ITTRIGOUTACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	CTTRIGOUTACK	Read the values of the CTTRIGOUTACK inputs.

Lock Access Register

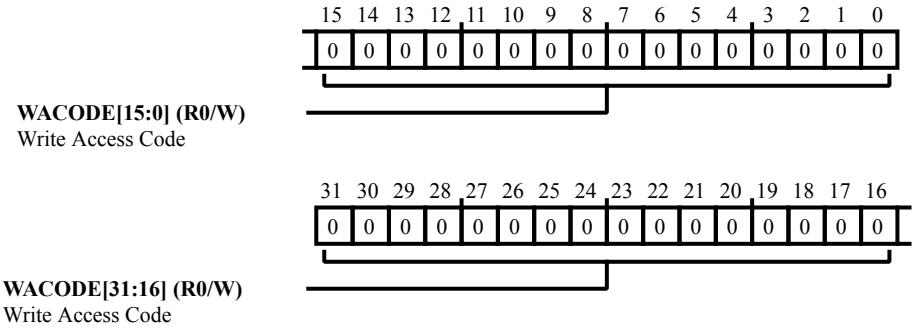


Figure 46-72: CTI_LAR Register Diagram

Table 46-78: CTI_LAR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R0/W)	WACODE	Write Access Code.

Lock Status Register

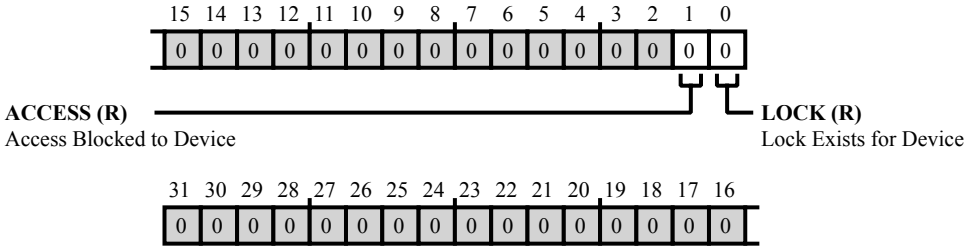


Figure 46-73: CTI_LSR Register Diagram

Table 46-79: CTI_LSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	ACCESS	Access Blocked to Device.
0 (R/NW)	LOCK	Lock Exists for Device.

Peripheral ID0

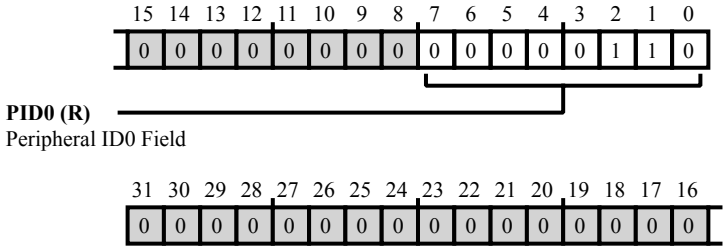


Figure 46-74: CTI_PERIPHID0 Register Diagram

Table 46-80: CTI_PERIPHID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PID0	Peripheral ID0 Field.

Peripheral ID1

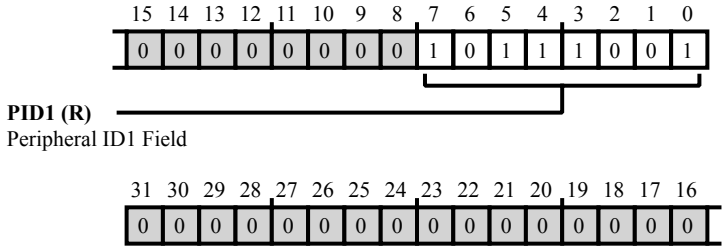


Figure 46-75: CTI_PERIPHID1 Register Diagram

Table 46-81: CTI_PERIPHID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PID1	Peripheral ID1 Field.

Peripheral ID2

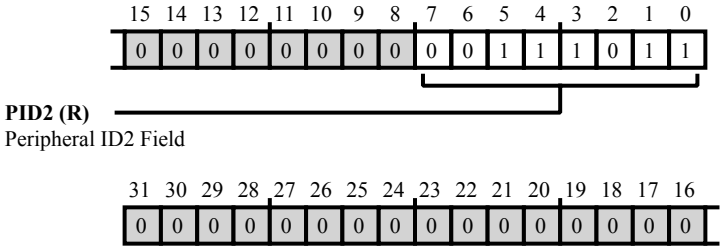


Figure 46-76: CTI_PERIPHID2 Register Diagram

Table 46-82: CTI_PERIPHID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PID2	Peripheral ID2 Field.

Peripheral ID3

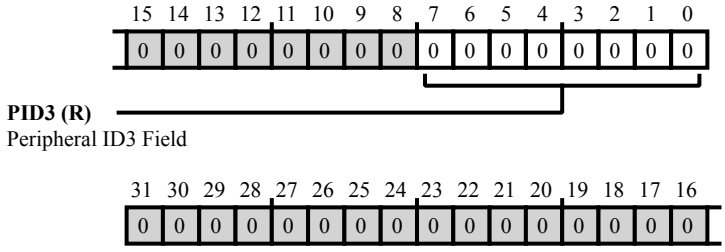


Figure 46-77: CTI_PERIPHID3 Register Diagram

Table 46-83: CTI_PERIPHID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PID3	Peripheral ID3 Field.

Peripheral ID4

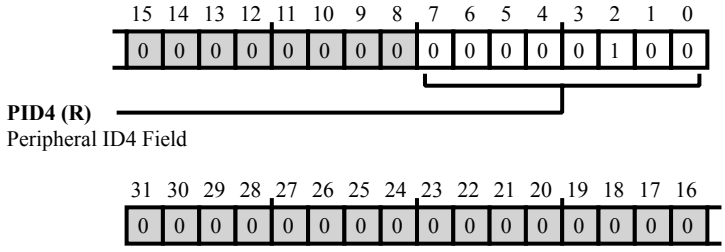


Figure 46-78: CTI_PERIPHID4 Register Diagram

Table 46-84: CTI_PERIPHID4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PID4	Peripheral ID4 Field.

Peripheral ID5

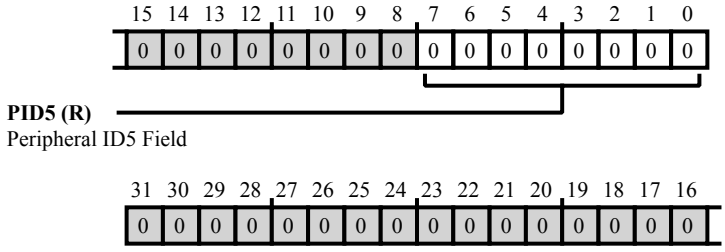


Figure 46-79: CTI_PERIPHID5 Register Diagram

Table 46-85: CTI_PERIPHID5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PID5	Peripheral ID5 Field.

Peripheral ID6

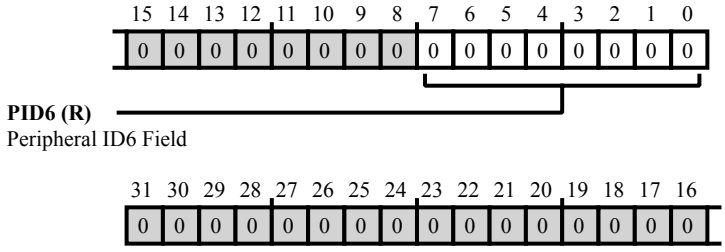


Figure 46-80: CTI_PERIPHID6 Register Diagram

Table 46-86: CTI_PERIPHID6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PID6	Peripheral ID6 Field.

Peripheral ID7

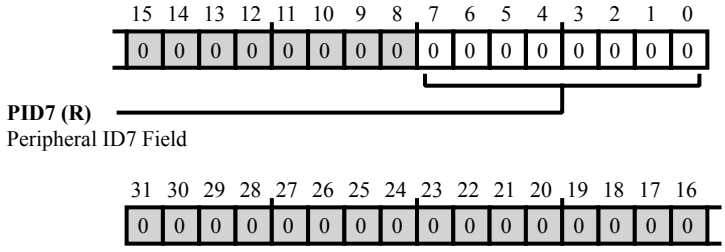


Figure 46-81: CTI_PERIPHID7 Register Diagram

Table 46-87: CTI_PERIPHID7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PID7	Peripheral ID7 Field.

47 CM41X_M0 Register List

This appendix lists Memory-Mapped Register address and register names. The modules are presented in alphabetical order.

Table 47-1: CM41X_M0 ADCC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B000	ADCC0_CTL	ADCC0 Control Register	0x00000000
0x4100B004	ADCC0_ERRSTAT	ADCC0 Error Status Register	0x00000000
0x4100B008	ADCC0_ERRMSK	ADCC0 Error Mask Register	0x00000000
0x4100B00C	ADCC0_ERRMSK_SET	ADCC0 Error Mask Set Register	0x00000000
0x4100B010	ADCC0_ERRMSK_CLR	ADCC0 Error Mask Clear Register	0x00000000
0x4100B014	ADCC0_EISTAT	ADCC0 Event Interrupt Status Register	0x00000000
0x4100B018	ADCC0_EIMSK	ADCC0 Event Interrupt Mask Register	0x00000000
0x4100B01C	ADCC0_EIMSK_SET	ADCC0 Event Interrupt Mask Set Register	0x00000000
0x4100B020	ADCC0_EIMSK_CLR	ADCC0 Event Interrupt Mask Clear Register	0x00000000
0x4100B024	ADCC0_FISTAT	ADCC0 Frame Interrupt Status Register	0x00000000
0x4100B028	ADCC0_FIMSK	ADCC0 Frame Interrupt Mask Register	0x00000000
0x4100B02C	ADCC0_FIMSK_SET	ADCC0 Frame Interrupt Mask Set Register	0x00000000
0x4100B030	ADCC0_FIMSK_CLR	ADCC0 Frame Interrupt Mask Clear Register	0x00000000
0x4100B034	ADCC0_EVTEN	ADCC0 Event Enable Register	0x00000000
0x4100B038	ADCC0_EVTEN_SET	ADCC0 Event Enable Set Register	0x00000000
0x4100B03C	ADCC0_EVTEN_CLR	ADCC0 Event Enable Clear Register	0x00000000
0x4100B040	ADCC0_ECOL	ADCC0 Event Collision Status Register	0x00000000
0x4100B044	ADCC0_EMISS	ADCC0 Event Miss Status Register	0x00000000
0x4100B048	ADCC0_BPTR0	ADCC0 Base Pointer 0 Register	0x00000000
0x4100B04C	ADCC0_FRINC0	ADCC0 Frame Increment 0 Register	0x00000000

Table 47-1: CM41X_M0 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B050	ADCC0_CBSIZ0	ADCC0 Circular Buffer Size 0 Register	0x00000000
0x4100B054	ADCC0_TCA0	ADCC0 Timing Control A (ADC0) Register	0x00000000
0x4100B058	ADCC0_TCB0	ADCC0 Timing Control B (ADC0) Register	0x00000000
0x4100B05C	ADCC0_BWMON0	ADCC0 Bandwidth Monitor 0 Register	0x00000000
0x4100B064	ADCC0_BPTR1	ADCC0 DMA Base Pointer 1 Register	0x00000000
0x4100B068	ADCC0_FRINC1	ADCC0 Frame Increment 1 Register	0x00000000
0x4100B06C	ADCC0_CBSIZ1	ADCC0 Circular Buffer Size 1 Register	0x00000000
0x4100B070	ADCC0_TCA1	ADCC0 Timing Control A (ADC1) Register	0x00000000
0x4100B074	ADCC0_TCB1	ADCC0 Timing Control B (ADC1) Register	0x00000000
0x4100B078	ADCC0_BWMON1	ADCC0 Bandwidth Monitor 1 Register	0x00000000
0x4100B07C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B080	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B084	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B088	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B08C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B090	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B094	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B098	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B09C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0A0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0A4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0A8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0AC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0B0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0B4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0B8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0BC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0C0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0C4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0C8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0CC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000

Table 47-1: CM41X_M0 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B0D0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0D4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0D8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0DC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0E0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0E4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0E8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0EC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0F0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0F4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0F8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0FC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B100	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B104	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B108	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B10C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B110	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B114	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B118	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B11C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B120	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B124	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B128	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B12C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B130	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B134	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B138	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B13C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B140	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B144	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B148	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000

Table 47-1: CM41X_M0 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B14C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B150	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B154	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B158	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B15C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B160	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B164	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B168	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B16C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B170	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B174	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B178	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B17C	ADCC0_NUMFRAM0	ADCC0 Timer0 Frame Limit Count Register	0x00000000
0x4100B180	ADCC0_NUMFRAM1	ADCC0 Timer1 Frame Limit Count Register	0x00000000
0x4100B184	ADCC0_DATOVF	ADCC0 Data Overflow Indication Register	0x00000000
0x4100B200	ADCC0_EPND	ADCC0 Pending Events Status Register	0x00000000
0x4100B204	ADCC0_T0STAT	ADCC0 Timer 0 Status Register	0x00000000
0x4100B208	ADCC0_TMR0	ADCC0 Timer 0 Current Count Register	0x00000000
0x4100B20C	ADCC0_T1STAT	ADCC0 Timer 1 Status Register	0x00000000
0x4100B210	ADCC0_TMR1	ADCC0 Timer 1 Current Count Register	0x00000000
0x4100B214	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B218	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B21C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B220	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B224	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B228	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B22C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B230	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B234	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B238	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B23C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000

Table 47-1: CM41X_M0 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B240	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B244	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B248	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B24C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B250	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B254	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B258	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B25C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B260	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B264	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B268	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B26C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B270	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B274	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B278	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B27C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B280	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B284	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B288	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B28C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B290	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B294	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B298	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B29C	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2A0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2A4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2A8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2AC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2B0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2B4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2B8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000

Table 47-1: CM41X_M0 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B2BC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2C0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2C4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2C8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2CC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2D0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2D4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2D8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2DC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2E0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2E4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2E8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2EC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2F0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2F4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2F8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2FC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B300	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B304	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B308	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B30C	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B310	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B314	ADCC0_CBNUM0	ADCC0 Timer0 Circular Buffer DMA Wrap Number Register	0x00000000
0x4100B318	ADCC0_CBNUM1	ADCC0 Timer1 Circular Buffer DMA Wrap Number Register	0x00000000
0x4100B31C	ADCC0_TRGCNT0	ADCC0 Trigger Count TIMER0 Register	0x00000000
0x4100B320	ADCC0_TRGCNT1	ADCC0 Trigger Count TIMER1 Register	0x00000000
0x4100B404	ADCC0_ADCRW0	ADCC0 ADC2 Interface RW Access Register	0x00000000
0x4100B408	ADCC0_ADCRW1	ADCC0 ADC2 Interface RW Access Register	0x00000000

Table 47-2: CM41X_M0 ADCC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A000	ADCC1_CTL	ADCC1 Control Register	0x00000000
0x4004A004	ADCC1_ERRSTAT	ADCC1 Error Status Register	0x00000000
0x4004A008	ADCC1_ERRMSK	ADCC1 Error Mask Register	0x00000000
0x4004A00C	ADCC1_ERRMSK_SET	ADCC1 Error Mask Set Register	0x00000000
0x4004A010	ADCC1_ERRMSK_CLR	ADCC1 Error Mask Clear Register	0x00000000
0x4004A014	ADCC1_EISTAT	ADCC1 Event Interrupt Status Register	0x00000000
0x4004A018	ADCC1_EIMSK	ADCC1 Event Interrupt Mask Register	0x00000000
0x4004A01C	ADCC1_EIMSK_SET	ADCC1 Event Interrupt Mask Set Register	0x00000000
0x4004A020	ADCC1_EIMSK_CLR	ADCC1 Event Interrupt Mask Clear Register	0x00000000
0x4004A024	ADCC1_FISTAT	ADCC1 Frame Interrupt Status Register	0x00000000
0x4004A028	ADCC1_FIMSK	ADCC1 Frame Interrupt Mask Register	0x00000000
0x4004A02C	ADCC1_FIMSK_SET	ADCC1 Frame Interrupt Mask Set Register	0x00000000
0x4004A030	ADCC1_FIMSK_CLR	ADCC1 Frame Interrupt Mask Clear Register	0x00000000
0x4004A034	ADCC1_EVTEN	ADCC1 Event Enable Register	0x00000000
0x4004A038	ADCC1_EVTEN_SET	ADCC1 Event Enable Set Register	0x00000000
0x4004A03C	ADCC1_EVTEN_CLR	ADCC1 Event Enable Clear Register	0x00000000
0x4004A040	ADCC1_ECOL	ADCC1 Event Collision Status Register	0x00000000
0x4004A044	ADCC1_EMISS	ADCC1 Event Miss Status Register	0x00000000
0x4004A048	ADCC1_BPTR0	ADCC1 Base Pointer 0 Register	0x00000000
0x4004A04C	ADCC1_FRINC0	ADCC1 Frame Increment 0 Register	0x00000000
0x4004A050	ADCC1_CBSIZ0	ADCC1 Circular Buffer Size 0 Register	0x00000000
0x4004A054	ADCC1_TCA0	ADCC1 Timing Control A (ADC0) Register	0x00000000
0x4004A058	ADCC1_TCB0	ADCC1 Timing Control B (ADC0) Register	0x00000000
0x4004A05C	ADCC1_BWMON0	ADCC1 Bandwidth Monitor 0 Register	0x00000000
0x4004A064	ADCC1_BPTR1	ADCC1 DMA Base Pointer 1 Register	0x00000000
0x4004A068	ADCC1_FRINC1	ADCC1 Frame Increment 1 Register	0x00000000
0x4004A06C	ADCC1_CBSIZ1	ADCC1 Circular Buffer Size 1 Register	0x00000000
0x4004A070	ADCC1_TCA1	ADCC1 Timing Control A (ADC1) Register	0x00000000
0x4004A074	ADCC1_TCB1	ADCC1 Timing Control B (ADC1) Register	0x00000000
0x4004A078	ADCC1_BWMON1	ADCC1 Bandwidth Monitor 1 Register	0x00000000
0x4004A07C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000

Table 47-2: CM41X_M0 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A080	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A084	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A088	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A08C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A090	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A094	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A098	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A09C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0A0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0A4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0A8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0AC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0B0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0B4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0B8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0BC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0C0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0C4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0C8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0CC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0D0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0D4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0D8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0DC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0E0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0E4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0E8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0EC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0F0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0F4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0F8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000

Table 47-2: CM41X_M0 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A0FC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A100	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A104	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A108	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A10C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A110	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A114	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A118	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A11C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A120	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A124	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A128	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A12C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A130	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A134	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A138	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A13C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A140	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A144	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A148	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A14C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A150	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A154	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A158	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A15C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A160	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A164	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A168	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A16C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A170	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A174	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000

Table 47-2: CM41X_M0 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A178	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A17C	ADCC1_NUMFRAM0	ADCC1 Timer0 Frame Limit Count Register	0x00000000
0x4004A180	ADCC1_NUMFRAM1	ADCC1 Timer1 Frame Limit Count Register	0x00000000
0x4004A184	ADCC1_DATOVF	ADCC1 Data Overflow Indication Register	0x00000000
0x4004A200	ADCC1_EPND	ADCC1 Pending Events Status Register	0x00000000
0x4004A204	ADCC1_T0STAT	ADCC1 Timer 0 Status Register	0x00000000
0x4004A208	ADCC1_TMR0	ADCC1 Timer 0 Current Count Register	0x00000000
0x4004A20C	ADCC1_T1STAT	ADCC1 Timer 1 Status Register	0x00000000
0x4004A210	ADCC1_TMR1	ADCC1 Timer 1 Current Count Register	0x00000000
0x4004A214	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A218	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A21C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A220	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A224	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A228	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A22C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A230	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A234	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A238	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A23C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A240	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A244	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A248	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A24C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A250	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A254	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A258	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A25C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A260	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A264	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A268	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000

Table 47-2: CM41X_M0 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A26C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A270	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A274	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A278	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A27C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A280	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A284	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A288	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A28C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A290	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A294	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A298	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A29C	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2A0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2A4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2A8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2AC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2B0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2B4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2B8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2BC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2C0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2C4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2C8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2CC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2D0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2D4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2D8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2DC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2E0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2E4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000

Table 47-2: CM41X_M0 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A2E8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2EC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2F0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2F4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2F8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2FC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A300	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A304	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A308	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A30C	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A310	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A314	ADCC1_CBNUM0	ADCC1 Timer0 Circular Buffer DMA Wrap Number Register	0x00000000
0x4004A318	ADCC1_CBNUM1	ADCC1 Timer1 Circular Buffer DMA Wrap Number Register	0x00000000
0x4004A31C	ADCC1_TRGCNT0	ADCC1 Trigger Count TIMER0 Register	0x00000000
0x4004A320	ADCC1_TRGCNT1	ADCC1 Trigger Count TIMER1 Register	0x00000000
0x4004A404	ADCC1_ADCRW0	ADCC1 ADC2 Interface RW Access Register	0x00000000
0x4004A408	ADCC1_ADCRW1	ADCC1 ADC2 Interface RW Access Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41008200	CAN0_MC1	CAN0 Mailbox Configuration 1 Register	0x00000000
0x41008204	CAN0_MD1	CAN0 Mailbox Direction 1 Register	0x000000FF
0x41008208	CAN0_TRS1	CAN0 Transmission Request Set 1 Register	0x00000000
0x4100820C	CAN0_TRR1	CAN0 Transmission Request Reset 1 Register	0x00000000
0x41008210	CAN0_TA1	CAN0 Transmission Acknowledge 1 Register	0x00000000
0x41008214	CAN0_AA1	CAN0 Abort Acknowledge 1 Register	0x00000000
0x41008218	CAN0_RMP1	CAN0 Receive Message Pending 1 Register	0x00000000
0x4100821C	CAN0_RML1	CAN0 Receive Message Lost 1 Register	0x00000000
0x41008220	CAN0_MBTIF1	CAN0 Mailbox Transmit Interrupt Flag 1 Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008224	CAN0_MBRIF1	CAN0 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x41008228	CAN0_MBIM1	CAN0 Mailbox Interrupt Mask 1 Register	0x00000000
0x4100822C	CAN0_RFH1	CAN0 Remote Frame Handling 1 Register	0x00000000
0x41008230	CAN0_OPSS1	CAN0 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x41008240	CAN0_MC2	CAN0 Mailbox Configuration 2 Register	0x00000000
0x41008244	CAN0_MD2	CAN0 Mailbox Direction 2 Register	0x00000000
0x41008248	CAN0_TRS2	CAN0 Transmission Request Set 2 Register	0x00000000
0x4100824C	CAN0_TRR2	CAN0 Transmission Request Reset 2 Register	0x00000000
0x41008250	CAN0_TA2	CAN0 Transmission Acknowledge 2 Register	0x00000000
0x41008254	CAN0_AA2	CAN0 Abort Acknowledge 2 Register	0x00000000
0x41008258	CAN0_RMP2	CAN0 Receive Message Pending 2 Register	0x00000000
0x4100825C	CAN0_RML2	CAN0 Receive Message Lost 2 Register	0x00000000
0x41008260	CAN0_MBTIF2	CAN0 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x41008264	CAN0_MBRIF2	CAN0 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x41008268	CAN0_MBIM2	CAN0 Mailbox Interrupt Mask 2 Register	0x00000000
0x4100826C	CAN0_RFH2	CAN0 Remote Frame Handling 2 Register	0x00000000
0x41008270	CAN0_OPSS2	CAN0 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x41008280	CAN0_CLK	CAN0 Clock Register	0x00000000
0x41008284	CAN0_TIMING	CAN0 Timing Register	0x00000000
0x41008288	CAN0_DBG	CAN0 Debug Register	0x00000008
0x4100828C	CAN0_STAT	CAN0 Status Register	0x00000080
0x41008290	CAN0_CEC	CAN0 Error Counter Register	0x00000000
0x41008294	CAN0_GIS	CAN0 Global CAN Interrupt Status Register	0x00000000
0x41008298	CAN0_GIM	CAN0 Global CAN Interrupt Mask Register	0x00000000
0x4100829C	CAN0_GIF	CAN0 Global CAN Interrupt Flag Register	0x00000000
0x410082A0	CAN0_CTL	CAN0 CAN Master Control Register	0x00000080
0x410082A4	CAN0_INT	CAN0 Interrupt Pending Register	0x00000000
0x410082AC	CAN0_MBTD	CAN0 Temporary Mailbox Disable Register	0x00000000
0x410082B0	CAN0_EWR	CAN0 Error Counter Warning Level Register	0x00006060

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410082B4	CAN0_ESR	CAN0 Error Status Register	0x00000020
0x410082C4	CAN0_UCCNT	CAN0 Universal Counter Register	0x00000000
0x410082C8	CAN0_UCRC	CAN0 Universal Counter Reload/Capture Register	0x00000000
0x410082CC	CAN0_UCCNF	CAN0 Universal Counter Configuration Mode Register	0x00000000
0x41008300	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008304	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008308	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100830C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008310	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008314	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008318	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100831C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008320	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008324	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008328	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100832C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008330	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008334	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008338	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100833C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008340	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008344	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008348	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100834C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008350	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008354	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008358	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100835C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008360	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008364	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008368	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100836C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008370	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008374	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008378	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100837C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008380	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008384	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008388	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100838C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008390	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008394	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008398	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100839C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083A0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083A4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083A8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083AC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083B0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083B4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083B8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083BC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083C0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083C4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083C8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083CC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083D0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083D4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083D8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083DC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083E0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083E4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410083E8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083EC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083F0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083F4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083F8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083FC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008400	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008404	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008408	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100840C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008410	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008414	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008418	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100841C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008420	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008424	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008428	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100842C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008430	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008434	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008438	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100843C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008440	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008444	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008448	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100844C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008450	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008454	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008458	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100845C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008460	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008464	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008468	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100846C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008470	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008474	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008478	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100847C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008480	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008484	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008488	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100848C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008490	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008494	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008498	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100849C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410084A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410084A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410084A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410084AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410084B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410084B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410084B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410084BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410084C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410084C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410084C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410084CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410084D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410084D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410084D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410084DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410084E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410084E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410084E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410084EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410084F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410084F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410084F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410084FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008500	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008504	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008508	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100850C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008510	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008514	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008518	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100851C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008520	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008524	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008528	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100852C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008530	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008534	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008538	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100853C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008540	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008544	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008548	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100854C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008550	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008554	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008558	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100855C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008560	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008564	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008568	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100856C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008570	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008574	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008578	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100857C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008580	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008584	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008588	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100858C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008590	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008594	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008598	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100859C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410085A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410085A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410085A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410085AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410085B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410085B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410085B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410085BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410085C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410085C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410085C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410085CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410085D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410085D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410085D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410085DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410085E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410085E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410085E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410085EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410085F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410085F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410085F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410085FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008600	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008604	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008608	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100860C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008610	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008614	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008618	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100861C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008620	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008624	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008628	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100862C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008630	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008634	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008638	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100863C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008640	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008644	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008648	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100864C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008650	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008654	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008658	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100865C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008660	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008664	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008668	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100866C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008670	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008674	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008678	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100867C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008680	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008684	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008688	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100868C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008690	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008694	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008698	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100869C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410086A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410086A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410086A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410086AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410086B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410086B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410086B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410086BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410086C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410086C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410086C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410086CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410086D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410086D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410086D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410086DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410086E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410086E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410086E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410086EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410086F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410086F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410086F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410086FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008700	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008704	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008708	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100870C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008710	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008714	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008718	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100871C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008720	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008724	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008728	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100872C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008730	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008734	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008738	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100873C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008740	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008744	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008748	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100874C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008750	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008754	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008758	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100875C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008760	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008764	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008768	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100876C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008770	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008774	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008778	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100877C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008780	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008784	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008788	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100878C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008790	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008794	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008798	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100879C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410087A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410087A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410087A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410087AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410087B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410087B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410087B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410087BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410087C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410087C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410087C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410087CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410087D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410087D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410087D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410087DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410087E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table 47-3: CM41X_M0 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410087E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410087E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410087EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410087F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410087F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410087F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410087FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40024200	CAN1_MC1	CAN1 Mailbox Configuration 1 Register	0x00000000
0x40024204	CAN1_MD1	CAN1 Mailbox Direction 1 Register	0x000000FF
0x40024208	CAN1_TRS1	CAN1 Transmission Request Set 1 Register	0x00000000
0x4002420C	CAN1_TRR1	CAN1 Transmission Request Reset 1 Register	0x00000000
0x40024210	CAN1_TA1	CAN1 Transmission Acknowledge 1 Register	0x00000000
0x40024214	CAN1_AA1	CAN1 Abort Acknowledge 1 Register	0x00000000
0x40024218	CAN1_RMP1	CAN1 Receive Message Pending 1 Register	0x00000000
0x4002421C	CAN1_RML1	CAN1 Receive Message Lost 1 Register	0x00000000
0x40024220	CAN1_MBTIF1	CAN1 Mailbox Transmit Interrupt Flag 1 Register	0x00000000
0x40024224	CAN1_MBRIF1	CAN1 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x40024228	CAN1_MBIM1	CAN1 Mailbox Interrupt Mask 1 Register	0x00000000
0x4002422C	CAN1_RFH1	CAN1 Remote Frame Handling 1 Register	0x00000000
0x40024230	CAN1_OPSS1	CAN1 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x40024240	CAN1_MC2	CAN1 Mailbox Configuration 2 Register	0x00000000
0x40024244	CAN1_MD2	CAN1 Mailbox Direction 2 Register	0x00000000
0x40024248	CAN1_TRS2	CAN1 Transmission Request Set 2 Register	0x00000000
0x4002424C	CAN1_TRR2	CAN1 Transmission Request Reset 2 Register	0x00000000
0x40024250	CAN1_TA2	CAN1 Transmission Acknowledge 2 Register	0x00000000
0x40024254	CAN1_AA2	CAN1 Abort Acknowledge 2 Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024258	CAN1_RMP2	CAN1 Receive Message Pending 2 Register	0x00000000
0x4002425C	CAN1_RML2	CAN1 Receive Message Lost 2 Register	0x00000000
0x40024260	CAN1_MBTIF2	CAN1 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x40024264	CAN1_MBRIF2	CAN1 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x40024268	CAN1_MBIM2	CAN1 Mailbox Interrupt Mask 2 Register	0x00000000
0x4002426C	CAN1_RFH2	CAN1 Remote Frame Handling 2 Register	0x00000000
0x40024270	CAN1_OPSS2	CAN1 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x40024280	CAN1_CLK	CAN1 Clock Register	0x00000000
0x40024284	CAN1_TIMING	CAN1 Timing Register	0x00000000
0x40024288	CAN1_DBG	CAN1 Debug Register	0x00000008
0x4002428C	CAN1_STAT	CAN1 Status Register	0x00000080
0x40024290	CAN1_CEC	CAN1 Error Counter Register	0x00000000
0x40024294	CAN1_GIS	CAN1 Global CAN Interrupt Status Register	0x00000000
0x40024298	CAN1_GIM	CAN1 Global CAN Interrupt Mask Register	0x00000000
0x4002429C	CAN1_GIF	CAN1 Global CAN Interrupt Flag Register	0x00000000
0x400242A0	CAN1_CTL	CAN1 CAN Master Control Register	0x00000080
0x400242A4	CAN1_INT	CAN1 Interrupt Pending Register	0x00000000
0x400242AC	CAN1_MBTD	CAN1 Temporary Mailbox Disable Register	0x00000000
0x400242B0	CAN1_EWR	CAN1 Error Counter Warning Level Register	0x00006060
0x400242B4	CAN1_ESR	CAN1 Error Status Register	0x00000020
0x400242C4	CAN1_UCCNT	CAN1 Universal Counter Register	0x00000000
0x400242C8	CAN1_UCRC	CAN1 Universal Counter Reload/Capture Register	0x00000000
0x400242CC	CAN1_UCCNF	CAN1 Universal Counter Configuration Mode Register	0x00000000
0x40024300	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024304	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024308	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002430C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024310	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024314	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024318	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002431C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024320	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024324	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024328	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002432C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024330	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024334	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024338	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002433C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024340	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024344	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024348	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002434C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024350	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024354	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024358	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002435C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024360	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024364	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024368	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002436C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024370	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024374	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024378	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002437C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024380	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024384	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024388	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002438C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024390	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024394	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024398	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002439C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243A0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243A4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243A8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243AC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243B0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243B4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243B8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243BC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243C0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243C4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243C8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243CC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243D0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243D4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243D8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243DC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243E0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243E4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243E8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243EC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243F0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243F4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243F8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243FC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024400	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024404	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024408	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002440C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024410	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024414	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024418	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002441C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024420	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024424	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024428	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002442C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024430	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024434	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024438	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002443C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024440	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024444	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024448	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002444C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024450	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024454	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024458	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002445C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024460	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024464	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024468	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002446C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024470	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024474	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024478	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002447C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024480	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024484	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024488	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002448C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024490	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024494	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024498	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002449C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400244A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400244A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400244A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400244AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400244B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400244B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400244B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400244BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400244C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400244C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400244C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400244CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400244D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400244D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400244D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400244DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400244E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400244E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400244E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400244EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400244F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400244F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400244F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400244FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024500	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024504	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024508	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002450C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024510	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024514	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024518	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002451C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024520	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024524	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024528	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002452C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024530	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024534	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024538	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002453C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024540	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024544	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024548	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002454C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024550	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024554	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024558	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002455C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024560	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024564	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024568	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002456C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024570	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024574	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024578	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002457C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024580	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024584	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024588	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002458C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024590	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024594	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024598	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002459C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400245A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400245A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400245A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400245AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400245B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400245B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400245B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400245BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400245C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400245C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400245C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400245CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400245D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400245D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400245D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400245DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400245E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400245E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400245E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400245EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400245F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400245F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400245F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400245FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024600	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024604	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024608	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002460C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024610	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024614	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024618	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002461C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024620	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024624	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024628	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002462C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024630	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024634	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024638	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002463C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024640	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024644	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024648	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002464C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024650	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024654	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024658	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002465C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024660	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024664	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024668	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002466C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024670	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024674	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024678	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002467C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024680	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024684	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024688	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002468C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024690	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024694	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024698	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002469C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400246A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400246A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400246A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400246AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400246B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400246B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400246B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400246BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400246C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400246C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400246C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400246CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400246D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400246D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400246D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400246DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400246E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400246E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400246E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400246EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400246F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400246F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400246F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400246FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024700	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024704	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024708	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002470C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024710	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024714	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024718	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002471C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024720	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024724	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024728	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002472C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024730	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024734	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024738	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002473C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024740	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024744	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024748	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002474C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024750	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024754	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024758	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002475C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024760	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024764	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024768	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002476C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024770	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024774	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024778	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002477C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024780	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024784	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024788	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002478C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024790	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 47-4: CM41X_M0 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024794	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024798	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002479C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400247A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400247A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400247A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400247AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400247B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400247B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400247B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400247BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400247C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400247C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400247C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400247CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400247D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400247D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400247D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400247DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400247E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400247E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400247E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400247EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400247F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400247F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400247F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400247FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000

Table 47-5: CM41X_M0 CGU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40014000	CGU0_CTL	CGU0 Control Register	0x00001200
0x40014004	CGU0_PLLCTL	CGU0 PLL Control Register	0x00000000
0x40014008	CGU0_STAT	CGU0 Status Register	0x0000000F
0x4001400C	CGU0_DIV	CGU0 Clocks Divisor Register	0x168F2926
0x40014010	CGU0_CLKOUTSEL	CGU0 CLKOUT Select Register	0x00000000
0x40014014	CGU0_OSCWDCTL	CGU0 Oscillator Watchdog Register	0x00007600
0x40014018	CGU0_TSCTL	CGU0 Time Stamp Control Register	0x00000000
0x4001401C	CGU0_TSVALUE0	CGU0 Time Stamp Counter Initial 32 LSB Value Register	0x00000000
0x40014020	CGU0_TSVALUE1	CGU0 Time Stamp Counter Initial MSB Value Register	0x00000000
0x40014024	CGU0_TSCOUNT0	CGU0 Time Stamp Counter 32 LSB Register	0x00000000
0x40014028	CGU0_TSCOUNT1	CGU0 Time Stamp Counter 32 MSB Register	0x00000000
0x4001402C	CGU0_CCBF_DIS	CGU0 Core Clock Buffer Disable Register	0x00000000
0x40014030	CGU0_CCBF_STAT	CGU0 Core Clock Buffer Status Register	0x00000000
0x40014038	CGU0_SCBF_DIS	CGU0 System Clock Buffer Disable Register	0x00000000
0x4001403C	CGU0_SCBF_STAT	CGU0 System Clock Buffer Status Register	0x00000000
0x40014048	CGU0_REVID	CGU0 Revision ID Register	0x00000020

Table 47-6: CM41X_M0 CNT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40022000	CNT0_CFG	CNT0 Configuration Register	0x00000000
0x40022004	CNT0_IMSK	CNT0 Interrupt Mask Register	0x00000000
0x40022008	CNT0_STAT	CNT0 Status Register	0x00000000
0x4002200C	CNT0_CMD	CNT0 Command Register	0x00000000
0x40022010	CNT0_DEBNCE	CNT0 Debounce Register	0x00000000
0x40022014	CNT0_CNTR	CNT0 Counter Register	0x00000000
0x40022018	CNT0_MAX	CNT0 Maximum Count Register	0x00000000
0x4002201C	CNT0_MIN	CNT0 Minimum Count Register	0x00000000
0x40022020	CNT0_MDIV	CNT0 M Value for Divider	0x00000000
0x40022024	CNT0_NDIV	CNT0 N Value for Divider	0x00000000

Table 47-7: CM41X_M0 CPTMR0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40025004	CPTMR0_RUN	CPTMR0 Run Register	0x00000000
0x40025008	CPTMR0_RUN_SET	CPTMR0 Run Set Register	0x00000000
0x4002500C	CPTMR0_RUN_CLR	CPTMR0 Run Clear Register	0x00000000
0x40025010	CPTMR0_DATA_IMSK	CPTMR0 Data Interrupt Mask Register	0x00000007
0x40025014	CPTMR0_DATA_IMSK_SET	CPTMR0 Data Interrupt Mask Set Register	0x00000007
0x40025018	CPTMR0_DATA_IMSK_CLR	CPTMR0 Data Interrupt Mask Clear Register	0x00000007
0x4002501C	CPTMR0_STAT_IMSK	CPTMR0 Status Interrupt Mask Register	0x00000007
0x40025020	CPTMR0_STAT_IMSK_SET	CPTMR0 Status Interrupt Mask Set Register	0x00000007
0x40025024	CPTMR0_STAT_IMSK_CLR	CPTMR0 Status Interrupt Mask Clear Register	0x00000007
0x40025028	CPTMR0_DATA_ILAT	CPTMR0 Data Interrupt Latch Status Register	0x00000000
0x4002502C	CPTMR0_STAT_ILAT	CPTMR0 Interrupt Latch Status Register	0x00000000
0x40025800	CPTMR0_CFG[n]	CPTMR0 Configuration Register	0x00000000
0x40025804	CPTMR0_CNT[n]	CPTMR0 Counter Register	0x00000001
0x40025808	CPTMR0_TON[n]	CPTMR0 On-time Capture Register	0x00000000
0x40025880	CPTMR0_CFG[n]	CPTMR0 Configuration Register	0x00000000
0x40025884	CPTMR0_CNT[n]	CPTMR0 Counter Register	0x00000001
0x40025888	CPTMR0_TON[n]	CPTMR0 On-time Capture Register	0x00000000
0x40025900	CPTMR0_CFG[n]	CPTMR0 Configuration Register	0x00000000
0x40025904	CPTMR0_CNT[n]	CPTMR0 Counter Register	0x00000001
0x40025908	CPTMR0_TON[n]	CPTMR0 On-time Capture Register	0x00000000

Table 47-8: CM41X_M0 CRC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C000	CRC0_CTL	CRC0 Control Register	0x00000000
0x4004C004	CRC0_DCNT	CRC0 Data Word Count Register	0x00000000
0x4004C008	CRC0_DCNTRLD	CRC0 Data Word Count Reload Register	0x00000000
0x4004C014	CRC0_COMP	CRC0 Data Compare Register	0x00000000
0x4004C018	CRC0_FILLVAL	CRC0 Fill Value Register	0x00000000

Table 47-8: CM41X_M0 CRC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C01C	CRC0_DFIFO	CRC0 Data FIFO Register	0x00000000
0x4004C020	CRC0_INEN	CRC0 Interrupt Enable Register	0x00000000
0x4004C024	CRC0_INEN_SET	CRC0 Interrupt Enable Set Register	0x00000000
0x4004C028	CRC0_INEN_CLR	CRC0 Interrupt Enable Clear Register	0x00000000
0x4004C02C	CRC0_POLY	CRC0 Polynomial Register	0x00000000
0x4004C040	CRC0_STAT	CRC0 Status Register	0x00000000
0x4004C044	CRC0_DCNTCAP	CRC0 Data Count Capture Register	0x00000000
0x4004C04C	CRC0_RESULT_FIN	CRC0 CRC Final Result Register	0x00000000
0x4004C050	CRC0_RESULT_CUR	CRC0 CRC Current Result Register	0x00000000

Table 47-9: CM41X_M0 DACC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004B000	DACC0_CTL0	DACC0 Control 0 Register	0x00000000
0x4004B008	DACC0_ERRSTAT	DACC0 Error Status Register	0x00000000
0x4004B00C	DACC0_ERRMSK	DACC0 Error Mask Register	0x00000000
0x4004B010	DACC0_ERRMSK_SET	DACC0 Error Mask Set Register	0x00000000
0x4004B014	DACC0_ERRMSK_CLR	DACC0 Error Mask Clear Register	0x00000000
0x4004B018	DACC0_ISTAT	DACC0 Interrupt Status Register	0x00000000
0x4004B01C	DACC0_IMSK	DACC0 Interrupt Mask Register	0x00000000
0x4004B020	DACC0_IMSK_SET	DACC0 Interrupt Mask Set Register	0x00000000
0x4004B024	DACC0_IMSK_CLR	DACC0 Interrupt Mask Clear Register	0x00000000
0x4004B028	DACC0_TC0	DACC0 Timing Control 0 Register	0x00000000
0x4004B02C	DACC0_BPTR0	DACC0 Base Pointer 0 Register	0x00000000
0x4004B030	DACC0_MOD0	DACC0 Modify 0 Register	0x00000000
0x4004B034	DACC0_CNT0	DACC0 Count 0 Register	0x00000000
0x4004B038	DACC0_DAT0	DACC0 Data FIFO 0 Register	0x00000000
0x4004B050	DACC0_BCST_CTL	DACC0 Broadcast (Write) Control Register	0x00000000
0x4004B100	DACC0_CNTCUR0	DACC0 Current Count 0 Register	0x00000000
0x4004B108	DACC0_STAT	DACC0 Status Register	0x00000000

Table 47-10: CM41X_M0 DMA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100C000	DMA0_DSCPTR_NXT	DMA0 Pointer to Next Initial Descriptor Register	0x00000000
0x4100C004	DMA0_ADDRSTART	DMA0 Start Address of Current Buffer Register	0x00000000
0x4100C008	DMA0_CFG	DMA0 Configuration Register	0x00000000
0x4100C00C	DMA0_XCNT	DMA0 Inner Loop Count Start Value Register	0x00000000
0x4100C010	DMA0_XMOD	DMA0 Inner Loop Address Increment Register	0x00000000
0x4100C014	DMA0_YCNT	DMA0 Outer Loop Count Start Value (2D only) Register	0x00000000
0x4100C018	DMA0_YMOD	DMA0 Outer Loop Address Increment (2D only) Register	0x00000000
0x4100C024	DMA0_DSCPTR_CUR	DMA0 Current Descriptor Pointer Register	0x00000000
0x4100C028	DMA0_DSCPTR_PRV	DMA0 Previous Initial Descriptor Pointer Register	0x00000000
0x4100C02C	DMA0_ADDR_CUR	DMA0 Current Address Register	0x00000000
0x4100C030	DMA0_STAT	DMA0 Status Register	0x00006000
0x4100C034	DMA0_XCNT_CUR	DMA0 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x4100C038	DMA0_YCNT_CUR	DMA0 Current Row Count (2D only) Register	0x00000000

Table 47-11: CM41X_M0 DMA1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100D000	DMA1_DSCPTR_NXT	DMA1 Pointer to Next Initial Descriptor Register	0x00000000
0x4100D004	DMA1_ADDRSTART	DMA1 Start Address of Current Buffer Register	0x00000000
0x4100D008	DMA1_CFG	DMA1 Configuration Register	0x00000000
0x4100D00C	DMA1_XCNT	DMA1 Inner Loop Count Start Value Register	0x00000000
0x4100D010	DMA1_XMOD	DMA1 Inner Loop Address Increment Register	0x00000000
0x4100D014	DMA1_YCNT	DMA1 Outer Loop Count Start Value (2D only) Register	0x00000000
0x4100D018	DMA1_YMOD	DMA1 Outer Loop Address Increment (2D only) Register	0x00000000
0x4100D024	DMA1_DSCPTR_CUR	DMA1 Current Descriptor Pointer Register	0x00000000
0x4100D028	DMA1_DSCPTR_PRV	DMA1 Previous Initial Descriptor Pointer Register	0x00000000
0x4100D02C	DMA1_ADDR_CUR	DMA1 Current Address Register	0x00000000
0x4100D030	DMA1_STAT	DMA1 Status Register	0x00006000
0x4100D034	DMA1_XCNT_CUR	DMA1 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x4100D038	DMA1_YCNT_CUR	DMA1 Current Row Count (2D only) Register	0x00000000

Table 47-12: CM41X_M0 DMA10 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40066000	DMA10_DSCPTR_NXT	DMA10 Pointer to Next Initial Descriptor Register	0x00000000
0x40066004	DMA10_ADDRSTART	DMA10 Start Address of Current Buffer Register	0x00000000
0x40066008	DMA10_CFG	DMA10 Configuration Register	0x00000000
0x4006600C	DMA10_XCNT	DMA10 Inner Loop Count Start Value Register	0x00000000
0x40066010	DMA10_XMOD	DMA10 Inner Loop Address Increment Register	0x00000000
0x40066014	DMA10_YCNT	DMA10 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40066018	DMA10_YMOD	DMA10 Outer Loop Address Increment (2D only) Register	0x00000000
0x40066024	DMA10_DSCPTR_CUR	DMA10 Current Descriptor Pointer Register	0x00000000
0x40066028	DMA10_DSCPTR_PRV	DMA10 Previous Initial Descriptor Pointer Register	0x00000000
0x4006602C	DMA10_ADDR_CUR	DMA10 Current Address Register	0x00000000
0x40066030	DMA10_STAT	DMA10 Status Register	0x00006000
0x40066034	DMA10_XCNT_CUR	DMA10 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40066038	DMA10_YCNT_CUR	DMA10 Current Row Count (2D only) Register	0x00000000

Table 47-13: CM41X_M0 DMA11 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40067000	DMA11_DSCPTR_NXT	DMA11 Pointer to Next Initial Descriptor Register	0x00000000
0x40067004	DMA11_ADDRSTART	DMA11 Start Address of Current Buffer Register	0x00000000
0x40067008	DMA11_CFG	DMA11 Configuration Register	0x00000000
0x4006700C	DMA11_XCNT	DMA11 Inner Loop Count Start Value Register	0x00000000
0x40067010	DMA11_XMOD	DMA11 Inner Loop Address Increment Register	0x00000000
0x40067014	DMA11_YCNT	DMA11 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40067018	DMA11_YMOD	DMA11 Outer Loop Address Increment (2D only) Register	0x00000000
0x40067024	DMA11_DSCPTR_CUR	DMA11 Current Descriptor Pointer Register	0x00000000
0x40067028	DMA11_DSCPTR_PRV	DMA11 Previous Initial Descriptor Pointer Register	0x00000000
0x4006702C	DMA11_ADDR_CUR	DMA11 Current Address Register	0x00000000
0x40067030	DMA11_STAT	DMA11 Status Register	0x00006000

Table 47-13: CM41X_M0 DMA11 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40067034	DMA11_XCNT_CUR	DMA11 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40067038	DMA11_YCNT_CUR	DMA11 Current Row Count (2D only) Register	0x00000000

Table 47-14: CM41X_M0 DMA12 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40068000	DMA12_DSCPTR_NXT	DMA12 Pointer to Next Initial Descriptor Register	0x00000000
0x40068004	DMA12_ADDRSTART	DMA12 Start Address of Current Buffer Register	0x00000000
0x40068008	DMA12_CFG	DMA12 Configuration Register	0x00000000
0x4006800C	DMA12_XCNT	DMA12 Inner Loop Count Start Value Register	0x00000000
0x40068010	DMA12_XMOD	DMA12 Inner Loop Address Increment Register	0x00000000
0x40068014	DMA12_YCNT	DMA12 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40068018	DMA12_YMOD	DMA12 Outer Loop Address Increment (2D only) Register	0x00000000
0x40068024	DMA12_DSCPTR_CUR	DMA12 Current Descriptor Pointer Register	0x00000000
0x40068028	DMA12_DSCPTR_PRV	DMA12 Previous Initial Descriptor Pointer Register	0x00000000
0x4006802C	DMA12_ADDR_CUR	DMA12 Current Address Register	0x00000000
0x40068030	DMA12_STAT	DMA12 Status Register	0x00006000
0x40068034	DMA12_XCNT_CUR	DMA12 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40068038	DMA12_YCNT_CUR	DMA12 Current Row Count (2D only) Register	0x00000000
0x40068040	DMA12_BWLCNT	DMA12 Bandwidth Limit Count Register	0x00000000
0x40068044	DMA12_BWLCNT_CUR	DMA12 Bandwidth Limit Count Current Register	0x00000000
0x40068048	DMA12_BWMCNT	DMA12 Bandwidth Monitor Count Register	0x00000000
0x4006804C	DMA12_BWMCNT_CUR	DMA12 Bandwidth Monitor Count Current Register	0x00000000

Table 47-15: CM41X_M0 DMA13 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40069000	DMA13_DSCPTR_NXT	DMA13 Pointer to Next Initial Descriptor Register	0x00000000
0x40069004	DMA13_ADDRSTART	DMA13 Start Address of Current Buffer Register	0x00000000

Table 47-15: CM41X_M0 DMA13 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40069008	DMA13_CFG	DMA13 Configuration Register	0x00000000
0x4006900C	DMA13_XCNT	DMA13 Inner Loop Count Start Value Register	0x00000000
0x40069010	DMA13_XMOD	DMA13 Inner Loop Address Increment Register	0x00000000
0x40069014	DMA13_YCNT	DMA13 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40069018	DMA13_YMOD	DMA13 Outer Loop Address Increment (2D only) Register	0x00000000
0x40069024	DMA13_DSCPTR_CUR	DMA13 Current Descriptor Pointer Register	0x00000000
0x40069028	DMA13_DSCPTR_PRV	DMA13 Previous Initial Descriptor Pointer Register	0x00000000
0x4006902C	DMA13_ADDR_CUR	DMA13 Current Address Register	0x00000000
0x40069030	DMA13_STAT	DMA13 Status Register	0x00006000
0x40069034	DMA13_XCNT_CUR	DMA13 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40069038	DMA13_YCNT_CUR	DMA13 Current Row Count (2D only) Register	0x00000000
0x40069040	DMA13_BWLCNT	DMA13 Bandwidth Limit Count Register	0x00000000
0x40069044	DMA13_BWLCNT_CUR	DMA13 Bandwidth Limit Count Current Register	0x00000000
0x40069048	DMA13_BWMCNT	DMA13 Bandwidth Monitor Count Register	0x00000000
0x4006904C	DMA13_BWMCNT_CUR	DMA13 Bandwidth Monitor Count Current Register	0x00000000

Table 47-16: CM41X_M0 DMA2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100E000	DMA2_DSCPTR_NXT	DMA2 Pointer to Next Initial Descriptor Register	0x00000000
0x4100E004	DMA2_ADDRSTART	DMA2 Start Address of Current Buffer Register	0x00000000
0x4100E008	DMA2_CFG	DMA2 Configuration Register	0x00000000
0x4100E00C	DMA2_XCNT	DMA2 Inner Loop Count Start Value Register	0x00000000
0x4100E010	DMA2_XMOD	DMA2 Inner Loop Address Increment Register	0x00000000
0x4100E014	DMA2_YCNT	DMA2 Outer Loop Count Start Value (2D only) Register	0x00000000
0x4100E018	DMA2_YMOD	DMA2 Outer Loop Address Increment (2D only) Register	0x00000000
0x4100E024	DMA2_DSCPTR_CUR	DMA2 Current Descriptor Pointer Register	0x00000000
0x4100E028	DMA2_DSCPTR_PRV	DMA2 Previous Initial Descriptor Pointer Register	0x00000000
0x4100E02C	DMA2_ADDR_CUR	DMA2 Current Address Register	0x00000000

Table 47-16: CM41X_M0 DMA2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100E030	DMA2_STAT	DMA2 Status Register	0x00006000
0x4100E034	DMA2_XCNT_CUR	DMA2 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x4100E038	DMA2_YCNT_CUR	DMA2 Current Row Count (2D only) Register	0x00000000

Table 47-17: CM41X_M0 DMA3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100F000	DMA3_DSCPTR_NXT	DMA3 Pointer to Next Initial Descriptor Register	0x00000000
0x4100F004	DMA3_ADDRSTART	DMA3 Start Address of Current Buffer Register	0x00000000
0x4100F008	DMA3_CFG	DMA3 Configuration Register	0x00000000
0x4100F00C	DMA3_XCNT	DMA3 Inner Loop Count Start Value Register	0x00000000
0x4100F010	DMA3_XMOD	DMA3 Inner Loop Address Increment Register	0x00000000
0x4100F014	DMA3_YCNT	DMA3 Outer Loop Count Start Value (2D only) Register	0x00000000
0x4100F018	DMA3_YMOD	DMA3 Outer Loop Address Increment (2D only) Register	0x00000000
0x4100F024	DMA3_DSCPTR_CUR	DMA3 Current Descriptor Pointer Register	0x00000000
0x4100F028	DMA3_DSCPTR_PRV	DMA3 Previous Initial Descriptor Pointer Register	0x00000000
0x4100F02C	DMA3_ADDR_CUR	DMA3 Current Address Register	0x00000000
0x4100F030	DMA3_STAT	DMA3 Status Register	0x00006000
0x4100F034	DMA3_XCNT_CUR	DMA3 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x4100F038	DMA3_YCNT_CUR	DMA3 Current Row Count (2D only) Register	0x00000000

Table 47-18: CM41X_M0 DMA4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40060000	DMA4_DSCPTR_NXT	DMA4 Pointer to Next Initial Descriptor Register	0x00000000
0x40060004	DMA4_ADDRSTART	DMA4 Start Address of Current Buffer Register	0x00000000
0x40060008	DMA4_CFG	DMA4 Configuration Register	0x00000000
0x4006000C	DMA4_XCNT	DMA4 Inner Loop Count Start Value Register	0x00000000
0x40060010	DMA4_XMOD	DMA4 Inner Loop Address Increment Register	0x00000000
0x40060014	DMA4_YCNT	DMA4 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40060018	DMA4_YMOD	DMA4 Outer Loop Address Increment (2D only) Register	0x00000000

Table 47-18: CM41X_M0 DMA4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40060024	DMA4_DSCPTR_CUR	DMA4 Current Descriptor Pointer Register	0x00000000
0x40060028	DMA4_DSCPTR_PRV	DMA4 Previous Initial Descriptor Pointer Register	0x00000000
0x4006002C	DMA4_ADDR_CUR	DMA4 Current Address Register	0x00000000
0x40060030	DMA4_STAT	DMA4 Status Register	0x00006000
0x40060034	DMA4_XCNT_CUR	DMA4 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40060038	DMA4_YCNT_CUR	DMA4 Current Row Count (2D only) Register	0x00000000

Table 47-19: CM41X_M0 DMA5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40061000	DMA5_DSCPTR_NXT	DMA5 Pointer to Next Initial Descriptor Register	0x00000000
0x40061004	DMA5_ADDRSTART	DMA5 Start Address of Current Buffer Register	0x00000000
0x40061008	DMA5_CFG	DMA5 Configuration Register	0x00000000
0x4006100C	DMA5_XCNT	DMA5 Inner Loop Count Start Value Register	0x00000000
0x40061010	DMA5_XMOD	DMA5 Inner Loop Address Increment Register	0x00000000
0x40061014	DMA5_YCNT	DMA5 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40061018	DMA5_YMOD	DMA5 Outer Loop Address Increment (2D only) Register	0x00000000
0x40061024	DMA5_DSCPTR_CUR	DMA5 Current Descriptor Pointer Register	0x00000000
0x40061028	DMA5_DSCPTR_PRV	DMA5 Previous Initial Descriptor Pointer Register	0x00000000
0x4006102C	DMA5_ADDR_CUR	DMA5 Current Address Register	0x00000000
0x40061030	DMA5_STAT	DMA5 Status Register	0x00006000
0x40061034	DMA5_XCNT_CUR	DMA5 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40061038	DMA5_YCNT_CUR	DMA5 Current Row Count (2D only) Register	0x00000000

Table 47-20: CM41X_M0 DMA6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40062000	DMA6_DSCPTR_NXT	DMA6 Pointer to Next Initial Descriptor Register	0x00000000
0x40062004	DMA6_ADDRSTART	DMA6 Start Address of Current Buffer Register	0x00000000
0x40062008	DMA6_CFG	DMA6 Configuration Register	0x00000000
0x4006200C	DMA6_XCNT	DMA6 Inner Loop Count Start Value Register	0x00000000

Table 47-20: CM41X_M0 DMA6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40062010	DMA6_XMOD	DMA6 Inner Loop Address Increment Register	0x00000000
0x40062014	DMA6_YCNT	DMA6 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40062018	DMA6_YMOD	DMA6 Outer Loop Address Increment (2D only) Register	0x00000000
0x40062024	DMA6_DSCPTR_CUR	DMA6 Current Descriptor Pointer Register	0x00000000
0x40062028	DMA6_DSCPTR_PRV	DMA6 Previous Initial Descriptor Pointer Register	0x00000000
0x4006202C	DMA6_ADDR_CUR	DMA6 Current Address Register	0x00000000
0x40062030	DMA6_STAT	DMA6 Status Register	0x00006000
0x40062034	DMA6_XCNT_CUR	DMA6 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40062038	DMA6_YCNT_CUR	DMA6 Current Row Count (2D only) Register	0x00000000

Table 47-21: CM41X_M0 DMA7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40063000	DMA7_DSCPTR_NXT	DMA7 Pointer to Next Initial Descriptor Register	0x00000000
0x40063004	DMA7_ADDRSTART	DMA7 Start Address of Current Buffer Register	0x00000000
0x40063008	DMA7_CFG	DMA7 Configuration Register	0x00000000
0x4006300C	DMA7_XCNT	DMA7 Inner Loop Count Start Value Register	0x00000000
0x40063010	DMA7_XMOD	DMA7 Inner Loop Address Increment Register	0x00000000
0x40063014	DMA7_YCNT	DMA7 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40063018	DMA7_YMOD	DMA7 Outer Loop Address Increment (2D only) Register	0x00000000
0x40063024	DMA7_DSCPTR_CUR	DMA7 Current Descriptor Pointer Register	0x00000000
0x40063028	DMA7_DSCPTR_PRV	DMA7 Previous Initial Descriptor Pointer Register	0x00000000
0x4006302C	DMA7_ADDR_CUR	DMA7 Current Address Register	0x00000000
0x40063030	DMA7_STAT	DMA7 Status Register	0x00006000
0x40063034	DMA7_XCNT_CUR	DMA7 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40063038	DMA7_YCNT_CUR	DMA7 Current Row Count (2D only) Register	0x00000000

Table 47-22: CM41X_M0 DMA8 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40064000	DMA8_DSCPTR_NXT	DMA8 Pointer to Next Initial Descriptor Register	0x00000000

Table 47-22: CM41X_M0 DMA8 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40064004	DMA8_ADDRSTART	DMA8 Start Address of Current Buffer Register	0x00000000
0x40064008	DMA8_CFG	DMA8 Configuration Register	0x00000000
0x4006400C	DMA8_XCNT	DMA8 Inner Loop Count Start Value Register	0x00000000
0x40064010	DMA8_XMOD	DMA8 Inner Loop Address Increment Register	0x00000000
0x40064014	DMA8_YCNT	DMA8 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40064018	DMA8_YMOD	DMA8 Outer Loop Address Increment (2D only) Register	0x00000000
0x40064024	DMA8_DSCPTR_CUR	DMA8 Current Descriptor Pointer Register	0x00000000
0x40064028	DMA8_DSCPTR_PRV	DMA8 Previous Initial Descriptor Pointer Register	0x00000000
0x4006402C	DMA8_ADDR_CUR	DMA8 Current Address Register	0x00000000
0x40064030	DMA8_STAT	DMA8 Status Register	0x00006000
0x40064034	DMA8_XCNT_CUR	DMA8 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40064038	DMA8_YCNT_CUR	DMA8 Current Row Count (2D only) Register	0x00000000

Table 47-23: CM41X_M0 DMA9 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40065000	DMA9_DSCPTR_NXT	DMA9 Pointer to Next Initial Descriptor Register	0x00000000
0x40065004	DMA9_ADDRSTART	DMA9 Start Address of Current Buffer Register	0x00000000
0x40065008	DMA9_CFG	DMA9 Configuration Register	0x00000000
0x4006500C	DMA9_XCNT	DMA9 Inner Loop Count Start Value Register	0x00000000
0x40065010	DMA9_XMOD	DMA9 Inner Loop Address Increment Register	0x00000000
0x40065014	DMA9_YCNT	DMA9 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40065018	DMA9_YMOD	DMA9 Outer Loop Address Increment (2D only) Register	0x00000000
0x40065024	DMA9_DSCPTR_CUR	DMA9 Current Descriptor Pointer Register	0x00000000
0x40065028	DMA9_DSCPTR_PRV	DMA9 Previous Initial Descriptor Pointer Register	0x00000000
0x4006502C	DMA9_ADDR_CUR	DMA9 Current Address Register	0x00000000
0x40065030	DMA9_STAT	DMA9 Status Register	0x00006000
0x40065034	DMA9_XCNT_CUR	DMA9 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40065038	DMA9_YCNT_CUR	DMA9 Current Row Count (2D only) Register	0x00000000

Table 47-24: CM41X_M0 DPM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40013000	DPM0_CTL	DPM0 Control Register	0x00000000
0x40013004	DPM0_STAT	DPM0 Status Register	0x00000001
0x4001301C	DPM0_WAKE_EN	DPM0 Wakeup Enable Register	0x00000000
0x40013020	DPM0_WAKE_POL	DPM0 Wakeup Polarity Register	0x00000000
0x40013024	DPM0_WAKE_STAT	DPM0 Wakeup Status Register	0x00000000
0x40013070	DPM0_PER_DIS0	DPM0 Peripherals Disable Register 0	0x00000000
0x40013084	DPM0_REVID	DPM0 Revision ID	0x00000020

Table 47-25: CM41X_M0 EMUID0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40002020	EMUID0_ADIID	EMUID0 Analog Devices Identification	0x00004144
0x40002024	EMUID0_CHIPID	EMUID0 Chip Identification	0x00000402

Table 47-26: CM41X_M0 FFTB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40048000	FFTB0_CTL	FFTB0 FFTB Control Register	0x00500000
0x40048004	FFTB0_STAT	FFTB0 FFTB Status Register	0x00000000
0x40048008	FFTB0_FMTCTL	FFTB0 Format Converter Control Register	0x0000FFF0
0x4004800C	FFTB0_INOFST	FFTB0 Input Offset Register	0x00000000
0x40048010	FFTB0_COMB	FFTB0 FFTB Input Comb Filter Control Register	0x00000000
0x40048014	FFTB0_DMABASE	FFTB0 DMA Output Base Address Register	0x00000000
0x40048018	FFTB0_DMAWR	FFTB0 DMA Output Write Address Register	0x00000000
0x40048020	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048024	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048028	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x4004802C	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048030	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048034	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048038	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x4004803C	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF

Table 47-26: CM41X_M0 FFTB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40048040	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048070	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048074	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048078	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x4004807C	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048080	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048084	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048088	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x4004808C	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048090	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x400480C0	FFTB0_MAXMAG	FFTB0 Maximum Magnitude Register	0x00000000
0x400480C4	FFTB0_MINMAG	FFTB0 Minimum Magnitude Register	0x00000000

Table 47-27: CM41X_M0 HAE0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400500A0	HAE0_RUN	HAE0 Run Register	0x00000000
0x40050B00	HAE0_CFG0	HAE0 Configuration 0 Register	0x00000000
0x40050B04	HAE0_CFG1	HAE0 Configuration 1 Register	0x00000000
0x40050B08	HAE0_CFG2	HAE0 Configuration 2 Register	0x00000000
0x40050B0C	HAE0_CFG3	HAE0 Configuration 3 Register	0x00000000
0x40050B20	HAE0_STAT	HAE0 Status Register	0x00000000
0x40050B40	HAE0_ISAMPLE	HAE0 I (Current) Sample Register	0x00000000
0x40050B44	HAE0_VSAMPLE	HAE0 V (Voltage) Sample Register	0x00000000
0x40050B80	HAE0_IWAVEFORM	HAE0 I (Current) Waveform Register	0x00000000
0x40050B84	HAE0_VWAVEFORM	HAE0 V (Voltage) Waveform Register	0x00000000
0x40051E00	HAE0_CFG4	HAE0 Configuration 4 Register	0x00000000
0x40051E04	HAE0_DIDT_GAIN	HAE0 DIDT Gain Register	0x00000000
0x40051E08	HAE0_DIDT_COEF	HAE0 DIDT Coefficient Register	0x00000000
0x40051E10	HAE0_VLEVEL	HAE0 Voltage Level Register	0x00000000
0x40051E2C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000

Table 47-27: CM41X_M0 HAE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40051E30	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E34	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E38	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E3C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E40	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E44	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E48	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E4C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E50	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E54	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E58	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000

Table 47-28: CM41X_M0 LBA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4000F004	LBA0_CLK_CTL	LBA0 Logic Block Array Clock Divisor Register	0x00000000
0x4000F008	LBA0_DIR	LBA0 Logic Block Array Pin Direction Register	0x00000000
0x4000F00C	LBA0_DIN	LBA0 Logic Block Array Data Input Register	0x00000000
0x4000F010	LBA0_DOUT	LBA0 Logic Block Array Data Output Register	0x00000000
0x4000F014	LBA0_CFG	LBA0 Logic Block Array Configuration Register	0x00000000
0x4000F018	LBA0_SYNC_CTL	LBA0 Logic Block Array GPIO Input Synchronization Control Register	0x00000000
0x4000F020	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F024	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F028	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F02C	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F030	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F034	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F038	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F03C	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F100	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F104	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000

Table 47-28: CM41X_M0 LBA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4000F108	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F10C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F110	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F114	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F118	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F11C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F120	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F124	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F128	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F12C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F130	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F134	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F138	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F13C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F140	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F144	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F148	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F14C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F150	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F154	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F158	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F15C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F160	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F164	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F168	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F16C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F170	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F174	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F178	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F17C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F180	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000

Table 47-28: CM41X_M0 LBA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4000F184	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F188	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F18C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F190	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F194	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F198	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F19C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F1A0	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F1A4	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F1A8	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F1AC	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F1B0	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F1B4	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F1B8	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F1BC	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F1C0	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F1C4	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F1C8	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F1CC	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F1D0	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F1D4	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F1D8	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F1DC	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F1E0	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F1E4	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F1E8	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F1EC	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F1F0	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F1F4	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F1F8	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F1FC	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000

Table 47-29: CM41X_M0 MBOX0_PORT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400D1000	MBOX0_PORT1_CTL	MBOX0_PORT1 Port 1 Control Register	0x00000000
0x400D1004	MBOX0_PORT1_FORCE-IRQ	MBOX0_PORT1 Port 1 Force IRQ Register	0x00000000
0x400D1008	MBOX0_PORT1_STAT	MBOX0_PORT1 Port 1 Status Register	0x00000000
0x400D100C	MBOX0_PORT1_ESTAT	MBOX0_PORT1 Port 1 ECC Status Register	0x00000000
0x400D1010	MBOX0_PORT1_RFRPER	MBOX0_PORT1 Auto-refresh Period Register	0x00000000
0x400D1014	MBOX0_PORT1_RFRADDR	MBOX0_PORT1 Auto-refresh Address Register	0x00000000
0x400D1018	MBOX0_PORT1_RFRCNT	MBOX0_PORT1 Auto-refresh Counter Register	0x00000001

Table 47-30: CM41X_M0 OCU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40015000	OCU0_CTL	OCU0 Control Register	0x00000000
0x40015004	OCU0_STAT	OCU0 Status Register	0x00000000
0x40015008	OCU0_CLKCNT	OCU0 Frequency Measured Count Register	0x00000000
0x4001500C	OCU0_REFCNT	OCU0 Reference Count Register	0x00000000
0x40015010	OCU0_MINCNT	OCU0 Minimum Limit Register	0x00000000
0x40015014	OCU0_MAXCNT	OCU0 Maximum Count Register	0x00000000
0x40015018	OCU0_LMONCNT	OCU0 LFO Monitor Reference Count Register	0x00000000
0x4001501C	OCU0_CMONCNT	OCU0 CLKIN Monitor Reference Count Register	0x00000000

Table 47-31: CM41X_M0 PINT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41002000	PINT0_MSK_SET	PINT0 PINT Mask Set Register	0x00000000
0x41002004	PINT0_MSK_CLR	PINT0 PINT Mask Clear Register	0x00000000
0x41002008	PINT0_REQ	PINT0 PINT Request Register	0x00000000
0x4100200C	PINT0_ASSIGN	PINT0 PINT Assign Register	0x00000101
0x41002010	PINT0_EDGE_SET	PINT0 PINT Edge Set Register	0x00000000
0x41002014	PINT0_EDGE_CLR	PINT0 PINT Edge Clear Register	0x00000000
0x41002018	PINT0_INV_SET	PINT0 PINT Invert Set Register	0x00000000
0x4100201C	PINT0_INV_CLR	PINT0 PINT Invert Clear Register	0x00000000

Table 47-31: CM41X_M0 PINT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41002020	PINT0_PINSTATE	PINT0 PINT Pin State Register	0x00000000
0x41002024	PINT0_LATCH	PINT0 PINT Latch Register	0x00000000

Table 47-32: CM41X_M0 PINT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40003000	PINT1_MSK_SET	PINT1 PINT Mask Set Register	0x00000000
0x40003004	PINT1_MSK_CLR	PINT1 PINT Mask Clear Register	0x00000000
0x40003008	PINT1_REQ	PINT1 PINT Request Register	0x00000000
0x4000300C	PINT1_ASSIGN	PINT1 PINT Assign Register	0x00000101
0x40003010	PINT1_EDGE_SET	PINT1 PINT Edge Set Register	0x00000000
0x40003014	PINT1_EDGE_CLR	PINT1 PINT Edge Clear Register	0x00000000
0x40003018	PINT1_INV_SET	PINT1 PINT Invert Set Register	0x00000000
0x4000301C	PINT1_INV_CLR	PINT1 PINT Invert Clear Register	0x00000000
0x40003020	PINT1_PINSTATE	PINT1 PINT Pin State Register	0x00000000
0x40003024	PINT1_LATCH	PINT1 PINT Latch Register	0x00000000

Table 47-33: CM41X_M0 PINT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40004000	PINT2_MSK_SET	PINT2 PINT Mask Set Register	0x00000000
0x40004004	PINT2_MSK_CLR	PINT2 PINT Mask Clear Register	0x00000000
0x40004008	PINT2_REQ	PINT2 PINT Request Register	0x00000000
0x4000400C	PINT2_ASSIGN	PINT2 PINT Assign Register	0x00000101
0x40004010	PINT2_EDGE_SET	PINT2 PINT Edge Set Register	0x00000000
0x40004014	PINT2_EDGE_CLR	PINT2 PINT Edge Clear Register	0x00000000
0x40004018	PINT2_INV_SET	PINT2 PINT Invert Set Register	0x00000000
0x4000401C	PINT2_INV_CLR	PINT2 PINT Invert Clear Register	0x00000000
0x40004020	PINT2_PINSTATE	PINT2 PINT Pin State Register	0x00000000
0x40004024	PINT2_LATCH	PINT2 PINT Latch Register	0x00000000

Table 47-34: CM41X_M0 PINT3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40005000	PINT3_MSK_SET	PINT3 PINT Mask Set Register	0x00000000
0x40005004	PINT3_MSK_CLR	PINT3 PINT Mask Clear Register	0x00000000
0x40005008	PINT3_REQ	PINT3 PINT Request Register	0x00000000
0x4000500C	PINT3_ASSIGN	PINT3 PINT Assign Register	0x00000101
0x40005010	PINT3_EDGE_SET	PINT3 PINT Edge Set Register	0x00000000
0x40005014	PINT3_EDGE_CLR	PINT3 PINT Edge Clear Register	0x00000000
0x40005018	PINT3_INV_SET	PINT3 PINT Invert Set Register	0x00000000
0x4000501C	PINT3_INV_CLR	PINT3 PINT Invert Clear Register	0x00000000
0x40005020	PINT3_PINSTATE	PINT3 PINT Pin State Register	0x00000000
0x40005024	PINT3_LATCH	PINT3 PINT Latch Register	0x00000000

Table 47-35: CM41X_M0 PINT4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40006000	PINT4_MSK_SET	PINT4 PINT Mask Set Register	0x00000000
0x40006004	PINT4_MSK_CLR	PINT4 PINT Mask Clear Register	0x00000000
0x40006008	PINT4_REQ	PINT4 PINT Request Register	0x00000000
0x4000600C	PINT4_ASSIGN	PINT4 PINT Assign Register	0x00000101
0x40006010	PINT4_EDGE_SET	PINT4 PINT Edge Set Register	0x00000000
0x40006014	PINT4_EDGE_CLR	PINT4 PINT Edge Clear Register	0x00000000
0x40006018	PINT4_INV_SET	PINT4 PINT Invert Set Register	0x00000000
0x4000601C	PINT4_INV_CLR	PINT4 PINT Invert Clear Register	0x00000000
0x40006020	PINT4_PINSTATE	PINT4 PINT Pin State Register	0x00000000
0x40006024	PINT4_LATCH	PINT4 PINT Latch Register	0x00000000

Table 47-36: CM41X_M0 PINT5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40007000	PINT5_MSK_SET	PINT5 PINT Mask Set Register	0x00000000
0x40007004	PINT5_MSK_CLR	PINT5 PINT Mask Clear Register	0x00000000
0x40007008	PINT5_REQ	PINT5 PINT Request Register	0x00000000
0x4000700C	PINT5_ASSIGN	PINT5 PINT Assign Register	0x00000101

Table 47-36: CM41X_M0 PINT5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40007010	PINT5_EDGE_SET	PINT5 PINT Edge Set Register	0x00000000
0x40007014	PINT5_EDGE_CLR	PINT5 PINT Edge Clear Register	0x00000000
0x40007018	PINT5_INV_SET	PINT5 PINT Invert Set Register	0x00000000
0x4000701C	PINT5_INV_CLR	PINT5 PINT Invert Clear Register	0x00000000
0x40007020	PINT5_PINSTATE	PINT5 PINT Pin State Register	0x00000000
0x40007024	PINT5_LATCH	PINT5 PINT Latch Register	0x00000000

Table 47-37: CM41X_M0 PORTA MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41003000	PORTA_FER	PORTA Port x Function Enable Register	0x00000000
0x41003004	PORTA_FER_SET	PORTA Port x Function Enable Set Register	0x00000000
0x41003008	PORTA_FER_CLR	PORTA Port x Function Enable Clear Register	0x00000000
0x4100300C	PORTA_DATA	PORTA Port x GPIO Data Register	0x00000000
0x41003010	PORTA_DATA_SET	PORTA Port x GPIO Data Set Register	0x00000000
0x41003014	PORTA_DATA_CLR	PORTA Port x GPIO Data Clear Register	0x00000000
0x41003018	PORTA_DIR	PORTA Port x GPIO Direction Register	0x00000000
0x4100301C	PORTA_DIR_SET	PORTA Port x GPIO Direction Set Register	0x00000000
0x41003020	PORTA_DIR_CLR	PORTA Port x GPIO Direction Clear Register	0x00000000
0x41003024	PORTA_INEN	PORTA Port x GPIO Input Enable Register	0x00000000
0x41003028	PORTA_INEN_SET	PORTA Port x GPIO Input Enable Set Register	0x00000000
0x4100302C	PORTA_INEN_CLR	PORTA Port x GPIO Input Enable Clear Register	0x00000000
0x41003030	PORTA_MUX	PORTA Port x Multiplexer Control Register	0x00000000
0x41003034	PORTA_DATA_TGL	PORTA Port x GPIO Output Toggle Register	0x00000000
0x41003038	PORTA_POL	PORTA Port x GPIO Polarity Invert Register	0x00000000
0x4100303C	PORTA_POL_SET	PORTA Port x GPIO Polarity Invert Set Register	0x00000000
0x41003040	PORTA_POL_CLR	PORTA Port x GPIO Polarity Invert Clear Register	0x00000000
0x41003044	PORTA_LOCK	PORTA Port x GPIO Lock Register	0x00000000
0x41003048	PORTA_TRIG_TGL	PORTA Port x GPIO Trigger Toggle Register	0x00000000

Table 47-38: CM41X_M0 PORTB MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40008000	PORTB_FER	PORTB Port x Function Enable Register	0x00000000
0x40008004	PORTB_FER_SET	PORTB Port x Function Enable Set Register	0x00000000
0x40008008	PORTB_FER_CLR	PORTB Port x Function Enable Clear Register	0x00000000
0x4000800C	PORTB_DATA	PORTB Port x GPIO Data Register	0x00000000
0x40008010	PORTB_DATA_SET	PORTB Port x GPIO Data Set Register	0x00000000
0x40008014	PORTB_DATA_CLR	PORTB Port x GPIO Data Clear Register	0x00000000
0x40008018	PORTB_DIR	PORTB Port x GPIO Direction Register	0x00000000
0x4000801C	PORTB_DIR_SET	PORTB Port x GPIO Direction Set Register	0x00000000
0x40008020	PORTB_DIR_CLR	PORTB Port x GPIO Direction Clear Register	0x00000000
0x40008024	PORTB_INEN	PORTB Port x GPIO Input Enable Register	0x00000000
0x40008028	PORTB_INEN_SET	PORTB Port x GPIO Input Enable Set Register	0x00000000
0x4000802C	PORTB_INEN_CLR	PORTB Port x GPIO Input Enable Clear Register	0x00000000
0x40008030	PORTB_MUX	PORTB Port x Multiplexer Control Register	0x00000000
0x40008034	PORTB_DATA_TGL	PORTB Port x GPIO Output Toggle Register	0x00000000
0x40008038	PORTB_POL	PORTB Port x GPIO Polarity Invert Register	0x00000000
0x4000803C	PORTB_POL_SET	PORTB Port x GPIO Polarity Invert Set Register	0x00000000
0x40008040	PORTB_POL_CLR	PORTB Port x GPIO Polarity Invert Clear Register	0x00000000
0x40008044	PORTB_LOCK	PORTB Port x GPIO Lock Register	0x00000000
0x40008048	PORTB_TRIG_TGL	PORTB Port x GPIO Trigger Toggle Register	0x00000000

Table 47-39: CM41X_M0 PORTC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40009000	PORTC_FER	PORTC Port x Function Enable Register	0x00000000
0x40009004	PORTC_FER_SET	PORTC Port x Function Enable Set Register	0x00000000
0x40009008	PORTC_FER_CLR	PORTC Port x Function Enable Clear Register	0x00000000
0x4000900C	PORTC_DATA	PORTC Port x GPIO Data Register	0x00000000
0x40009010	PORTC_DATA_SET	PORTC Port x GPIO Data Set Register	0x00000000
0x40009014	PORTC_DATA_CLR	PORTC Port x GPIO Data Clear Register	0x00000000
0x40009018	PORTC_DIR	PORTC Port x GPIO Direction Register	0x00000000
0x4000901C	PORTC_DIR_SET	PORTC Port x GPIO Direction Set Register	0x00000000

Table 47-39: CM41X_M0 PORTC MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40009020	PORTC_DIR_CLR	PORTC Port x GPIO Direction Clear Register	0x00000000
0x40009024	PORTC_INEN	PORTC Port x GPIO Input Enable Register	0x00000000
0x40009028	PORTC_INEN_SET	PORTC Port x GPIO Input Enable Set Register	0x00000000
0x4000902C	PORTC_INEN_CLR	PORTC Port x GPIO Input Enable Clear Register	0x00000000
0x40009030	PORTC_MUX	PORTC Port x Multiplexer Control Register	0x00000000
0x40009034	PORTC_DATA_TGL	PORTC Port x GPIO Output Toggle Register	0x00000000
0x40009038	PORTC_POL	PORTC Port x GPIO Polarity Invert Register	0x00000000
0x4000903C	PORTC_POL_SET	PORTC Port x GPIO Polarity Invert Set Register	0x00000000
0x40009040	PORTC_POL_CLR	PORTC Port x GPIO Polarity Invert Clear Register	0x00000000
0x40009044	PORTC_LOCK	PORTC Port x GPIO Lock Register	0x00000000
0x40009048	PORTC_TRIG_TGL	PORTC Port x GPIO Trigger Toggle Register	0x00000000

Table 47-40: CM41X_M0 PORTD MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4000A000	PORTD_FER	PORTD Port x Function Enable Register	0x00000000
0x4000A004	PORTD_FER_SET	PORTD Port x Function Enable Set Register	0x00000000
0x4000A008	PORTD_FER_CLR	PORTD Port x Function Enable Clear Register	0x00000000
0x4000A00C	PORTD_DATA	PORTD Port x GPIO Data Register	0x00000000
0x4000A010	PORTD_DATA_SET	PORTD Port x GPIO Data Set Register	0x00000000
0x4000A014	PORTD_DATA_CLR	PORTD Port x GPIO Data Clear Register	0x00000000
0x4000A018	PORTD_DIR	PORTD Port x GPIO Direction Register	0x00000000
0x4000A01C	PORTD_DIR_SET	PORTD Port x GPIO Direction Set Register	0x00000000
0x4000A020	PORTD_DIR_CLR	PORTD Port x GPIO Direction Clear Register	0x00000000
0x4000A024	PORTD_INEN	PORTD Port x GPIO Input Enable Register	0x00000000
0x4000A028	PORTD_INEN_SET	PORTD Port x GPIO Input Enable Set Register	0x00000000
0x4000A02C	PORTD_INEN_CLR	PORTD Port x GPIO Input Enable Clear Register	0x00000000
0x4000A030	PORTD_MUX	PORTD Port x Multiplexer Control Register	0x00000000
0x4000A034	PORTD_DATA_TGL	PORTD Port x GPIO Output Toggle Register	0x00000000
0x4000A038	PORTD_POL	PORTD Port x GPIO Polarity Invert Register	0x00000000
0x4000A03C	PORTD_POL_SET	PORTD Port x GPIO Polarity Invert Set Register	0x00000000

Table 47-40: CM41X_M0 PORTD MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4000A040	PORTD_POL_CLR	PORTD Port x GPIO Polarity Invert Clear Register	0x00000000
0x4000A044	PORTD_LOCK	PORTD Port x GPIO Lock Register	0x00000000
0x4000A048	PORTD_TRIG_TGL	PORTD Port x GPIO Trigger Toggle Register	0x00000000

Table 47-41: CM41X_M0 PORTE MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4000B000	PORTE_FER	PORTE Port x Function Enable Register	0x00000000
0x4000B004	PORTE_FER_SET	PORTE Port x Function Enable Set Register	0x00000000
0x4000B008	PORTE_FER_CLR	PORTE Port x Function Enable Clear Register	0x00000000
0x4000B00C	PORTE_DATA	PORTE Port x GPIO Data Register	0x00000000
0x4000B010	PORTE_DATA_SET	PORTE Port x GPIO Data Set Register	0x00000000
0x4000B014	PORTE_DATA_CLR	PORTE Port x GPIO Data Clear Register	0x00000000
0x4000B018	PORTE_DIR	PORTE Port x GPIO Direction Register	0x00000000
0x4000B01C	PORTE_DIR_SET	PORTE Port x GPIO Direction Set Register	0x00000000
0x4000B020	PORTE_DIR_CLR	PORTE Port x GPIO Direction Clear Register	0x00000000
0x4000B024	PORTE_INEN	PORTE Port x GPIO Input Enable Register	0x00000000
0x4000B028	PORTE_INEN_SET	PORTE Port x GPIO Input Enable Set Register	0x00000000
0x4000B02C	PORTE_INEN_CLR	PORTE Port x GPIO Input Enable Clear Register	0x00000000
0x4000B030	PORTE_MUX	PORTE Port x Multiplexer Control Register	0x00000000
0x4000B034	PORTE_DATA_TGL	PORTE Port x GPIO Output Toggle Register	0x00000000
0x4000B038	PORTE_POL	PORTE Port x GPIO Polarity Invert Register	0x00000000
0x4000B03C	PORTE_POL_SET	PORTE Port x GPIO Polarity Invert Set Register	0x00000000
0x4000B040	PORTE_POL_CLR	PORTE Port x GPIO Polarity Invert Clear Register	0x00000000
0x4000B044	PORTE_LOCK	PORTE Port x GPIO Lock Register	0x00000000
0x4000B048	PORTE_TRIG_TGL	PORTE Port x GPIO Trigger Toggle Register	0x00000000

Table 47-42: CM41X_M0 PORTF MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4000C000	PORTF_FER	PORTF Port x Function Enable Register	0x00000000
0x4000C004	PORTF_FER_SET	PORTF Port x Function Enable Set Register	0x00000000

Table 47-42: CM41X_M0 PORTF MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4000C008	PORTF_FER_CLR	PORTF Port x Function Enable Clear Register	0x00000000
0x4000C00C	PORTF_DATA	PORTF Port x GPIO Data Register	0x00000000
0x4000C010	PORTF_DATA_SET	PORTF Port x GPIO Data Set Register	0x00000000
0x4000C014	PORTF_DATA_CLR	PORTF Port x GPIO Data Clear Register	0x00000000
0x4000C018	PORTF_DIR	PORTF Port x GPIO Direction Register	0x00000000
0x4000C01C	PORTF_DIR_SET	PORTF Port x GPIO Direction Set Register	0x00000000
0x4000C020	PORTF_DIR_CLR	PORTF Port x GPIO Direction Clear Register	0x00000000
0x4000C024	PORTF_INEN	PORTF Port x GPIO Input Enable Register	0x00000000
0x4000C028	PORTF_INEN_SET	PORTF Port x GPIO Input Enable Set Register	0x00000000
0x4000C02C	PORTF_INEN_CLR	PORTF Port x GPIO Input Enable Clear Register	0x00000000
0x4000C030	PORTF_MUX	PORTF Port x Multiplexer Control Register	0x00000000
0x4000C034	PORTF_DATA_TGL	PORTF Port x GPIO Output Toggle Register	0x00000000
0x4000C038	PORTF_POL	PORTF Port x GPIO Polarity Invert Register	0x00000000
0x4000C03C	PORTF_POL_SET	PORTF Port x GPIO Polarity Invert Set Register	0x00000000
0x4000C040	PORTF_POL_CLR	PORTF Port x GPIO Polarity Invert Clear Register	0x00000000
0x4000C044	PORTF_LOCK	PORTF Port x GPIO Lock Register	0x00000000
0x4000C048	PORTF_TRIG_TGL	PORTF Port x GPIO Trigger Toggle Register	0x00000000

Table 47-43: CM41X_M0 PWM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40028000	PWM0_CTL	PWM0 Control Register	0x00020000
0x40028004	PWM0_CHANCFG	PWM0 Channel Configuration Register	0x00000000
0x40028008	PWM0_TRIPCFG	PWM0 Trip Configuration Register	0x00000000
0x4002800C	PWM0_STAT	PWM0 Status Register	0x00000000
0x40028010	PWM0_IMSK	PWM0 Interrupt Mask Register	0x00000000
0x40028014	PWM0_ILAT	PWM0 Interrupt Latch Register	0x00000000
0x40028018	PWM0_CHOPCFG	PWM0 Chop Configuration Register	0x00000000
0x40028020	PWM0_SYNC_WID	PWM0 Sync Pulse Width Register	0x000003FF
0x40028024	PWM0_TM0	PWM0 Timer 0 Period Register	0x00000000
0x40028028	PWM0_TM1	PWM0 Timer 1 Period Register	0x00000000

Table 47-43: CM41X_M0 PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002802C	PWM0_TM2	PWM0 Timer 2 Period Register	0x00000000
0x40028030	PWM0_TM3	PWM0 Timer 3 Period Register	0x00000000
0x40028034	PWM0_TM4	PWM0 Timer 4 Period Register	0x00000000
0x40028038	PWM0_DLYA	PWM0 Channel A Delay Register	0x00000000
0x4002803C	PWM0_DLYB	PWM0 Channel B Delay Register	0x00000000
0x40028040	PWM0_DLYC	PWM0 Channel C Delay Register	0x00000000
0x40028044	PWM0_DLYD	PWM0 Channel D Delay Register	0x00000000
0x40028048	PWM0_ACTL	PWM0 Channel A Control Register	0x00000000
0x4002804C	PWM0_AH0	PWM0 Channel A-High Duty-0 Register	0x00000000
0x40028050	PWM0_AH1	PWM0 Channel A-High Duty-1 Register	0x00000000
0x40028054	PWM0_AH0_HP	PWM0 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x40028058	PWM0_AH1_HP	PWM0 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4002805C	PWM0_AL0	PWM0 Channel A-Low Duty-0 Register	0x00000000
0x40028060	PWM0_AL1	PWM0 Channel A-Low Duty-1 Register	0x00000000
0x40028064	PWM0_AL0_HP	PWM0 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x40028068	PWM0_AL1_HP	PWM0 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002806C	PWM0_BCTL	PWM0 Channel B Control Register	0x00000000
0x40028070	PWM0_BH0	PWM0 Channel B-High Duty-0 Register	0x00000000
0x40028074	PWM0_BH1	PWM0 Channel B-High Duty-1 Register	0x00000000
0x40028078	PWM0_BH0_HP	PWM0 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x4002807C	PWM0_BH1_HP	PWM0 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x40028080	PWM0_BL0	PWM0 Channel B-Low Duty-0 Register	0x00000000
0x40028084	PWM0_BL1	PWM0 Channel B-Low Duty-1 Register	0x00000000
0x40028088	PWM0_BL0_HP	PWM0 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002808C	PWM0_BL1_HP	PWM0 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000

Table 47-43: CM41X_M0 PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40028090	PWM0_CCTL	PWM0 Channel C Control Register	0x00000000
0x40028094	PWM0_CH0	PWM0 Channel C-High Pulse Duty Register 0	0x00000000
0x40028098	PWM0_CH1	PWM0 Channel C-High Pulse Duty Register 1	0x00000000
0x4002809C	PWM0_CH0_HP	PWM0 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400280A0	PWM0_CH1_HP	PWM0 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400280A4	PWM0_CL0	PWM0 Channel C-Low Pulse Duty Register 0	0x00000000
0x400280A8	PWM0_CL1	PWM0 Channel C-Low Duty-1 Register	0x00000000
0x400280AC	PWM0_CL0_HP	PWM0 Channel C-Low Pulse Duty Register 1	0x00000000
0x400280B0	PWM0_CL1_HP	PWM0 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x400280B4	PWM0_DCTL	PWM0 Channel D Control Register	0x00000000
0x400280B8	PWM0_DH0	PWM0 Channel D-High Duty-0 Register	0x00000000
0x400280BC	PWM0_DH1	PWM0 Channel D-High Pulse Duty Register 1	0x00000000
0x400280C0	PWM0_DH0_HP	PWM0 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400280C4	PWM0_DH1_HP	PWM0 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400280C8	PWM0_DL0	PWM0 Channel D-Low Pulse Duty Register 0	0x00000000
0x400280CC	PWM0_DL1	PWM0 Channel D-Low Pulse Duty Register 1	0x00000000
0x400280D0	PWM0_DL0_HP	PWM0 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x400280D4	PWM0_DL1_HP	PWM0 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x400280D8	PWM0_AH_DUTY0	PWM0 Channel A-High Full Duty0 Register	0x00000000
0x400280DC	PWM0_AH_DUTY1	PWM0 Channel A-High Full Duty1 Register	0x00000000
0x400280E0	PWM0_AL_DUTY0	PWM0 Channel A-Low Full Duty0 Register	0x00000000
0x400280E4	PWM0_AL_DUTY1	PWM0 Channel A-Low Full Duty1 Register	0x00000000
0x400280E8	PWM0_BH_DUTY0	PWM0 Channel B-High Full Duty0 Register	0x00000000
0x400280EC	PWM0_BH_DUTY1	PWM0 Channel B-High Full Duty1 Register	0x00000000
0x400280F0	PWM0_BL_DUTY0	PWM0 Channel B-Low Full Duty0 Register	0x00000000
0x400280F4	PWM0_BL_DUTY1	PWM0 Channel B-Low Full Duty1 Register	0x00000000

Table 47-43: CM41X_M0 PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400280F8	PWM0_CH_DUTY0	PWM0 Channel C-High Full Duty0 Register	0x00000000
0x400280FC	PWM0_CH_DUTY1	PWM0 Channel C-High Full Duty1 Register	0x00000000
0x40028100	PWM0_CL_DUTY0	PWM0 Channel C-Low Full Duty0 Register	0x00000000
0x40028104	PWM0_CL_DUTY1	PWM0 Channel C-Low Full Duty1 Register	0x00000000
0x40028108	PWM0_DH_DUTY0	PWM0 Channel D-High Full Duty0 Register	0x00000000
0x4002810C	PWM0_DH_DUTY1	PWM0 Channel D-High Full Duty1 Register	0x00000000
0x40028110	PWM0_DL_DUTY0	PWM0 Channel D-Low Full Duty0 Register	0x00000000
0x40028114	PWM0_DL_DUTY1	PWM0 Channel D-Low Full Duty1 Register	0x00000000
0x40028118	PWM0_CHA_DT	PWM0 Channel A Dead-time Register	0x00000000
0x4002811C	PWM0_CHB_DT	PWM0 Channel B Dead-time Register	0x00000000
0x40028120	PWM0_CHC_DT	PWM0 Channel C Dead-time Register	0x00000000
0x40028124	PWM0_CHD_DT	PWM0 Channel D Dead-time Register	0x00000000
0x40028130	PWM0_SWTRIP	PWM0 Software Trip Register	0x00000000
0x40028134	PWM0_TRIP_POL	PWM0 Trip Polarity Register	0x00000000

Table 47-44: CM41X_M0 PWM1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40029000	PWM1_CTL	PWM1 Control Register	0x00020000
0x40029004	PWM1_CHANCFG	PWM1 Channel Configuration Register	0x00000000
0x40029008	PWM1_TRIPCFG	PWM1 Trip Configuration Register	0x00000000
0x4002900C	PWM1_STAT	PWM1 Status Register	0x00000000
0x40029010	PWM1_IMSK	PWM1 Interrupt Mask Register	0x00000000
0x40029014	PWM1_ILAT	PWM1 Interrupt Latch Register	0x00000000
0x40029018	PWM1_CHOPCFG	PWM1 Chop Configuration Register	0x00000000
0x40029020	PWM1_SYNC_WID	PWM1 Sync Pulse Width Register	0x000003FF
0x40029024	PWM1_TM0	PWM1 Timer 0 Period Register	0x00000000
0x40029028	PWM1_TM1	PWM1 Timer 1 Period Register	0x00000000
0x4002902C	PWM1_TM2	PWM1 Timer 2 Period Register	0x00000000
0x40029030	PWM1_TM3	PWM1 Timer 3 Period Register	0x00000000
0x40029034	PWM1_TM4	PWM1 Timer 4 Period Register	0x00000000

Table 47-44: CM41X_M0 PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40029038	PWM1_DLYA	PWM1 Channel A Delay Register	0x00000000
0x4002903C	PWM1_DLYB	PWM1 Channel B Delay Register	0x00000000
0x40029040	PWM1_DLYC	PWM1 Channel C Delay Register	0x00000000
0x40029044	PWM1_DLYD	PWM1 Channel D Delay Register	0x00000000
0x40029048	PWM1_ACTL	PWM1 Channel A Control Register	0x00000000
0x4002904C	PWM1_AH0	PWM1 Channel A-High Duty-0 Register	0x00000000
0x40029050	PWM1_AH1	PWM1 Channel A-High Duty-1 Register	0x00000000
0x40029054	PWM1_AH0_HP	PWM1 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x40029058	PWM1_AH1_HP	PWM1 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4002905C	PWM1_AL0	PWM1 Channel A-Low Duty-0 Register	0x00000000
0x40029060	PWM1_AL1	PWM1 Channel A-Low Duty-1 Register	0x00000000
0x40029064	PWM1_AL0_HP	PWM1 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x40029068	PWM1_AL1_HP	PWM1 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002906C	PWM1_BCTL	PWM1 Channel B Control Register	0x00000000
0x40029070	PWM1_BH0	PWM1 Channel B-High Duty-0 Register	0x00000000
0x40029074	PWM1_BH1	PWM1 Channel B-High Duty-1 Register	0x00000000
0x40029078	PWM1_BH0_HP	PWM1 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x4002907C	PWM1_BH1_HP	PWM1 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x40029080	PWM1_BL0	PWM1 Channel B-Low Duty-0 Register	0x00000000
0x40029084	PWM1_BL1	PWM1 Channel B-Low Duty-1 Register	0x00000000
0x40029088	PWM1_BL0_HP	PWM1 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002908C	PWM1_BL1_HP	PWM1 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x40029090	PWM1_CCTL	PWM1 Channel C Control Register	0x00000000
0x40029094	PWM1_CH0	PWM1 Channel C-High Pulse Duty Register 0	0x00000000
0x40029098	PWM1_CH1	PWM1 Channel C-High Pulse Duty Register 1	0x00000000

Table 47-44: CM41X_M0 PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002909C	PWM1_CH0_HP	PWM1 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400290A0	PWM1_CH1_HP	PWM1 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400290A4	PWM1_CL0	PWM1 Channel C-Low Pulse Duty Register 0	0x00000000
0x400290A8	PWM1_CL1	PWM1 Channel C-Low Duty-1 Register	0x00000000
0x400290AC	PWM1_CL0_HP	PWM1 Channel C-Low Pulse Duty Register 1	0x00000000
0x400290B0	PWM1_CL1_HP	PWM1 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x400290B4	PWM1_DCTL	PWM1 Channel D Control Register	0x00000000
0x400290B8	PWM1_DH0	PWM1 Channel D-High Duty-0 Register	0x00000000
0x400290BC	PWM1_DH1	PWM1 Channel D-High Pulse Duty Register 1	0x00000000
0x400290C0	PWM1_DH0_HP	PWM1 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400290C4	PWM1_DH1_HP	PWM1 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400290C8	PWM1_DL0	PWM1 Channel D-Low Pulse Duty Register 0	0x00000000
0x400290CC	PWM1_DL1	PWM1 Channel D-Low Pulse Duty Register 1	0x00000000
0x400290D0	PWM1_DL0_HP	PWM1 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x400290D4	PWM1_DL1_HP	PWM1 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x400290D8	PWM1_AH_DUTY0	PWM1 Channel A-High Full Duty0 Register	0x00000000
0x400290DC	PWM1_AH_DUTY1	PWM1 Channel A-High Full Duty1 Register	0x00000000
0x400290E0	PWM1_AL_DUTY0	PWM1 Channel A-Low Full Duty0 Register	0x00000000
0x400290E4	PWM1_AL_DUTY1	PWM1 Channel A-Low Full Duty1 Register	0x00000000
0x400290E8	PWM1_BH_DUTY0	PWM1 Channel B-High Full Duty0 Register	0x00000000
0x400290EC	PWM1_BH_DUTY1	PWM1 Channel B-High Full Duty1 Register	0x00000000
0x400290F0	PWM1_BL_DUTY0	PWM1 Channel B-Low Full Duty0 Register	0x00000000
0x400290F4	PWM1_BL_DUTY1	PWM1 Channel B-Low Full Duty1 Register	0x00000000
0x400290F8	PWM1_CH_DUTY0	PWM1 Channel C-High Full Duty0 Register	0x00000000
0x400290FC	PWM1_CH_DUTY1	PWM1 Channel C-High Full Duty1 Register	0x00000000
0x40029100	PWM1_CL_DUTY0	PWM1 Channel C-Low Full Duty0 Register	0x00000000

Table 47-44: CM41X_M0 PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40029104	PWM1_CL_DUTY1	PWM1 Channel C-Low Full Duty1 Register	0x00000000
0x40029108	PWM1_DH_DUTY0	PWM1 Channel D-High Full Duty0 Register	0x00000000
0x4002910C	PWM1_DH_DUTY1	PWM1 Channel D-High Full Duty1 Register	0x00000000
0x40029110	PWM1_DL_DUTY0	PWM1 Channel D-Low Full Duty0 Register	0x00000000
0x40029114	PWM1_DL_DUTY1	PWM1 Channel D-Low Full Duty1 Register	0x00000000
0x40029118	PWM1_CHA_DT	PWM1 Channel A Dead-time Register	0x00000000
0x4002911C	PWM1_CHB_DT	PWM1 Channel B Dead-time Register	0x00000000
0x40029120	PWM1_CHC_DT	PWM1 Channel C Dead-time Register	0x00000000
0x40029124	PWM1_CHD_DT	PWM1 Channel D Dead-time Register	0x00000000
0x40029130	PWM1_SWTRIP	PWM1 Software Trip Register	0x00000000
0x40029134	PWM1_TRIP_POL	PWM1 Trip Polarity Register	0x00000000

Table 47-45: CM41X_M0 PWM2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4002A000	PWM2_CTL	PWM2 Control Register	0x00020000
0x4002A004	PWM2_CHANCFG	PWM2 Channel Configuration Register	0x00000000
0x4002A008	PWM2_TRIPCFG	PWM2 Trip Configuration Register	0x00000000
0x4002A00C	PWM2_STAT	PWM2 Status Register	0x00000000
0x4002A010	PWM2_IMSK	PWM2 Interrupt Mask Register	0x00000000
0x4002A014	PWM2_ILAT	PWM2 Interrupt Latch Register	0x00000000
0x4002A018	PWM2_CHOPCFG	PWM2 Chop Configuration Register	0x00000000
0x4002A020	PWM2_SYNC_WID	PWM2 Sync Pulse Width Register	0x000003FF
0x4002A024	PWM2_TM0	PWM2 Timer 0 Period Register	0x00000000
0x4002A028	PWM2_TM1	PWM2 Timer 1 Period Register	0x00000000
0x4002A02C	PWM2_TM2	PWM2 Timer 2 Period Register	0x00000000
0x4002A030	PWM2_TM3	PWM2 Timer 3 Period Register	0x00000000
0x4002A034	PWM2_TM4	PWM2 Timer 4 Period Register	0x00000000
0x4002A038	PWM2_DLYA	PWM2 Channel A Delay Register	0x00000000
0x4002A03C	PWM2_DLYB	PWM2 Channel B Delay Register	0x00000000
0x4002A040	PWM2_DLYC	PWM2 Channel C Delay Register	0x00000000

Table 47-45: CM41X_M0 PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002A044	PWM2_DLYD	PWM2 Channel D Delay Register	0x00000000
0x4002A048	PWM2_ACTL	PWM2 Channel A Control Register	0x00000000
0x4002A04C	PWM2_AH0	PWM2 Channel A-High Duty-0 Register	0x00000000
0x4002A050	PWM2_AH1	PWM2 Channel A-High Duty-1 Register	0x00000000
0x4002A054	PWM2_AH0_HP	PWM2 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x4002A058	PWM2_AH1_HP	PWM2 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4002A05C	PWM2_AL0	PWM2 Channel A-Low Duty-0 Register	0x00000000
0x4002A060	PWM2_AL1	PWM2 Channel A-Low Duty-1 Register	0x00000000
0x4002A064	PWM2_AL0_HP	PWM2 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002A068	PWM2_AL1_HP	PWM2 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002A06C	PWM2_BCTL	PWM2 Channel B Control Register	0x00000000
0x4002A070	PWM2_BH0	PWM2 Channel B-High Duty-0 Register	0x00000000
0x4002A074	PWM2_BH1	PWM2 Channel B-High Duty-1 Register	0x00000000
0x4002A078	PWM2_BH0_HP	PWM2 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x4002A07C	PWM2_BH1_HP	PWM2 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x4002A080	PWM2_BL0	PWM2 Channel B-Low Duty-0 Register	0x00000000
0x4002A084	PWM2_BL1	PWM2 Channel B-Low Duty-1 Register	0x00000000
0x4002A088	PWM2_BL0_HP	PWM2 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002A08C	PWM2_BL1_HP	PWM2 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002A090	PWM2_CCTL	PWM2 Channel C Control Register	0x00000000
0x4002A094	PWM2_CH0	PWM2 Channel C-High Pulse Duty Register 0	0x00000000
0x4002A098	PWM2_CH1	PWM2 Channel C-High Pulse Duty Register 1	0x00000000
0x4002A09C	PWM2_CH0_HP	PWM2 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x4002A0A0	PWM2_CH1_HP	PWM2 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000

Table 47-45: CM41X_M0 PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002A0A4	PWM2_CL0	PWM2 Channel C-Low Pulse Duty Register 0	0x00000000
0x4002A0A8	PWM2_CL1	PWM2 Channel C-Low Duty-1 Register	0x00000000
0x4002A0AC	PWM2_CL0_HP	PWM2 Channel C-Low Pulse Duty Register 1	0x00000000
0x4002A0B0	PWM2_CL1_HP	PWM2 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002A0B4	PWM2_DCTL	PWM2 Channel D Control Register	0x00000000
0x4002A0B8	PWM2_DH0	PWM2 Channel D-High Duty-0 Register	0x00000000
0x4002A0BC	PWM2_DH1	PWM2 Channel D-High Pulse Duty Register 1	0x00000000
0x4002A0C0	PWM2_DH0_HP	PWM2 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x4002A0C4	PWM2_DH1_HP	PWM2 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x4002A0C8	PWM2_DL0	PWM2 Channel D-Low Pulse Duty Register 0	0x00000000
0x4002A0CC	PWM2_DL1	PWM2 Channel D-Low Pulse Duty Register 1	0x00000000
0x4002A0D0	PWM2_DL0_HP	PWM2 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002A0D4	PWM2_DL1_HP	PWM2 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002A0D8	PWM2_AH_DUTY0	PWM2 Channel A-High Full Duty0 Register	0x00000000
0x4002A0DC	PWM2_AH_DUTY1	PWM2 Channel A-High Full Duty1 Register	0x00000000
0x4002A0E0	PWM2_AL_DUTY0	PWM2 Channel A-Low Full Duty0 Register	0x00000000
0x4002A0E4	PWM2_AL_DUTY1	PWM2 Channel A-Low Full Duty1 Register	0x00000000
0x4002A0E8	PWM2_BH_DUTY0	PWM2 Channel B-High Full Duty0 Register	0x00000000
0x4002A0EC	PWM2_BH_DUTY1	PWM2 Channel B-High Full Duty1 Register	0x00000000
0x4002A0F0	PWM2_BL_DUTY0	PWM2 Channel B-Low Full Duty0 Register	0x00000000
0x4002A0F4	PWM2_BL_DUTY1	PWM2 Channel B-Low Full Duty1 Register	0x00000000
0x4002A0F8	PWM2_CH_DUTY0	PWM2 Channel C-High Full Duty0 Register	0x00000000
0x4002A0FC	PWM2_CH_DUTY1	PWM2 Channel C-High Full Duty1 Register	0x00000000
0x4002A100	PWM2_CL_DUTY0	PWM2 Channel C-Low Full Duty0 Register	0x00000000
0x4002A104	PWM2_CL_DUTY1	PWM2 Channel C-Low Full Duty1 Register	0x00000000
0x4002A108	PWM2_DH_DUTY0	PWM2 Channel D-High Full Duty0 Register	0x00000000
0x4002A10C	PWM2_DH_DUTY1	PWM2 Channel D-High Full Duty1 Register	0x00000000

Table 47-45: CM41X_M0 PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002A110	PWM2_DL_DUTY0	PWM2 Channel D-Low Full Duty0 Register	0x00000000
0x4002A114	PWM2_DL_DUTY1	PWM2 Channel D-Low Full Duty1 Register	0x00000000
0x4002A118	PWM2_CHA_DT	PWM2 Channel A Dead-time Register	0x00000000
0x4002A11C	PWM2_CHB_DT	PWM2 Channel B Dead-time Register	0x00000000
0x4002A120	PWM2_CHC_DT	PWM2 Channel C Dead-time Register	0x00000000
0x4002A124	PWM2_CHD_DT	PWM2 Channel D Dead-time Register	0x00000000
0x4002A130	PWM2_SWTRIP	PWM2 Software Trip Register	0x00000000
0x4002A134	PWM2_TRIP_POL	PWM2 Trip Polarity Register	0x00000000

Table 47-46: CM41X_M0 RCU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40012000	RCU0_CTL	RCU0 Control Register	0x00000700
0x40012004	RCU0_STAT	RCU0 Status Register	0x00000021
0x40012008	RCU0_CRCTL	RCU0 Core Reset Outputs Control Register	0x00000002
0x4001200C	RCU0_CRSTAT	RCU0 Core Reset Outputs Status Register	0x00000007
0x40012018	RCU0_SRRQSTAT	RCU0 System Reset Request Status Register	0x00000000
0x40012028	RCU0_BCODE	RCU0 Boot Code Register	0x00000000
0x4001206C	RCU0_MSG	RCU0 Message Register	0x00000000
0x40012070	RCU0_MSG_SET	RCU0 Message Set Bits Register	0x00000000
0x40012074	RCU0_MSG_CLR	RCU0 Message Clear Bits Register	0x00000000

Table 47-47: CM41X_M0 SEC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41004000	SEC0_GCTL	SEC0 Global Control Register	0x00000000
0x41004004	SEC0_GSTAT	SEC0 Global Status Register	0x00000000
0x41004008	SEC0_RAISE	SEC0 Global Raise Register	0x00000000
0x41004010	SEC0_FCTL	SEC0 Fault Control Register	0x00000000
0x41004014	SEC0_FSTAT	SEC0 Fault Status Register	0x00000000
0x41004018	SEC0_FSID	SEC0 Fault Source ID Register	0x00000000
0x4100401C	SEC0_FEND	SEC0 Fault End Register	0x00000000

Table 47-47: CM41X_M0 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41004020	SEC0_FDLY	SEC0 Fault Delay Register	0x00000000
0x41004024	SEC0_FDLY_CUR	SEC0 Fault Delay Current Register	0x00000000
0x41004028	SEC0_FSRDLY	SEC0 Fault System Reset Delay Register	0x00000000
0x4100402C	SEC0_FSRDLY_CUR	SEC0 Fault System Reset Delay Current Register	0x00000000
0x41004030	SEC0_FCOPP	SEC0 Fault COP Period Register	0x00000000
0x41004034	SEC0_FCOPP_CUR	SEC0 Fault COP Period Current Register	0x00000000
0x41004800	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004804	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004808	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100480C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004810	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004814	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004818	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100481C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004820	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004824	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004828	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100482C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004830	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004834	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004838	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100483C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004840	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004844	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004848	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100484C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004850	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004854	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004858	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100485C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004860	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table 47-47: CM41X_M0 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41004864	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004868	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100486C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004870	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004874	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004878	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100487C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004880	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004884	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004888	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100488C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004890	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004894	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004898	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100489C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table 47-47: CM41X_M0 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410048E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004900	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004904	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004908	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100490C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004910	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004914	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004918	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100491C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004920	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004924	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004928	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100492C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004930	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004934	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004938	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100493C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004940	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004944	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004948	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100494C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004950	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004954	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004958	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table 47-47: CM41X_M0 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100495C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004960	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004964	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004968	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100496C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004970	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004974	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004978	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100497C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004980	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004984	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004988	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100498C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004990	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004994	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004998	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100499C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table 47-47: CM41X_M0 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410049D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table 47-47: CM41X_M0 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41004A54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40010000	SEC1_GCTL	SEC1 Global Control Register	0x00000000
0x40010004	SEC1_GSTAT	SEC1 Global Status Register	0x00000000
0x40010008	SEC1_RAISE	SEC1 Global Raise Register	0x00000000
0x40010010	SEC1_FCTL	SEC1 Fault Control Register	0x00000000
0x40010014	SEC1_FSTAT	SEC1 Fault Status Register	0x00000000
0x40010018	SEC1_FSID	SEC1 Fault Source ID Register	0x00000000
0x4001001C	SEC1_FEND	SEC1 Fault End Register	0x00000000
0x40010020	SEC1_FDLY	SEC1 Fault Delay Register	0x00000000
0x40010024	SEC1_FDLY_CUR	SEC1 Fault Delay Current Register	0x00000000
0x40010028	SEC1_FSRDLY	SEC1 Fault System Reset Delay Register	0x00000000
0x4001002C	SEC1_FSRDLY_CUR	SEC1 Fault System Reset Delay Current Register	0x00000000
0x40010030	SEC1_FCOPP	SEC1 Fault COP Period Register	0x00000000
0x40010034	SEC1_FCOPP_CUR	SEC1 Fault COP Period Current Register	0x00000000
0x40010800	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010804	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010808	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001080C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010810	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010814	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010818	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001081C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010820	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010824	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010828	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4001082C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010830	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010834	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010838	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001083C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010840	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010844	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010848	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001084C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010850	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010854	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010858	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001085C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010860	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010864	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010868	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001086C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010870	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010874	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010878	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001087C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010880	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010884	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010888	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001088C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010890	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010894	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010898	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001089C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108A0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108A4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400108A8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108AC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108B0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108B4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108B8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108BC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108C0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108C4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108C8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108CC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108D0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108D4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108D8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108DC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108E0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108E4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108E8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108EC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108F0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108F4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108F8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108FC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010900	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010904	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010908	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001090C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010910	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010914	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010918	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001091C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010920	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010924	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010928	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001092C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010930	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010934	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010938	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001093C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010940	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010944	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010948	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001094C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010950	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010954	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010958	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001095C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010960	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010964	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010968	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001096C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010970	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010974	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010978	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001097C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010980	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010984	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010988	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001098C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010990	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010994	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010998	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001099C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400109A0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109A4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109A8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109AC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109B0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109B4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109B8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109BC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109C0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109C4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109C8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109CC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109D0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109D4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109D8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109DC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109E0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109E4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109E8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109EC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109F0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109F4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109F8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109FC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A00	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A04	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A08	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A0C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A10	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A14	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A18	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010A1C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A20	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A24	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A28	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A2C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A30	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A34	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A38	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A3C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A40	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A44	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A48	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A4C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A50	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A54	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A58	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A5C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A60	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A64	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A68	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A6C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A70	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A74	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A78	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A7C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A80	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A84	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A88	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A8C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A90	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A94	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010A98	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A9C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AA0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AA4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AA8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AAC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AB0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AB4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AB8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010ABC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AC0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AC4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AC8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010ACC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AD0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AD4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AD8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010ADC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AE0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AE4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AE8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AEC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AF0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AF4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AF8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AFC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B00	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B04	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B08	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B0C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B10	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010B14	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B18	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B1C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B20	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B24	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B28	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B2C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B30	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B34	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B38	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B3C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B40	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B44	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B48	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B4C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B50	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B54	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B58	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B5C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B60	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B64	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B68	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B6C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B70	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B74	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B78	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B7C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B80	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B84	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B88	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B8C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010B90	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B94	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B98	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B9C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BA0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BA4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BA8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BAC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BB0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BB4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BB8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BBC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BC0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BC4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BC8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BCC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BD0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BD4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BD8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BDC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BE0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BE4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BE8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BEC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BF0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BF4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BF8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BFC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C00	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C04	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C08	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010C0C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C10	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C14	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C18	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C1C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C20	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C24	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C28	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C2C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C30	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C34	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C38	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C3C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C40	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C44	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C48	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C4C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C50	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C54	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C58	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C5C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C60	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C64	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C68	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C6C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C70	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C74	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C78	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C7C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C80	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C84	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010C88	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C8C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C90	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C94	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C98	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C9C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CA0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CA4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CA8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CAC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CB0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CB4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CB8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CBC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CC0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CC4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CC8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CCC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CD0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CD4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CD8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CDC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CE0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CE4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CE8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CEC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CF0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CF4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CF8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CFC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010D00	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 47-48: CM41X_M0 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010D04	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010D08	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010D0C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010D10	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010D14	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010D18	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010D1C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 47-49: CM41X_M0 SINC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40049000	SINC0_CTL	SINC0 Control Register	0x00000000
0x40049004	SINC0_STAT	SINC0 Status Register	0x00000000
0x40049008	SINC0_CLK	SINC0 Clock Control Register	0x10001000
0x40049010	SINC0_RATE0	SINC0 Rate Control for Group 0 Register	0x00000804
0x40049014	SINC0_RATE1	SINC0 Rate Control for Group 1 Register	0x00000804
0x40049018	SINC0_LEVEL0	SINC0 Level Control for Group 0 Register	0x04000000
0x4004901C	SINC0_LEVEL1	SINC0 Level Control for Group 1 Register	0x04000000
0x40049020	SINC0_LIMIT0	SINC0 (Amplitude) Limits for Secondary Filter 0 Register	0x00000000
0x40049024	SINC0_LIMIT1	SINC0 (Amplitude) Limits for Secondary Filter 1 Register	0x00000000
0x40049028	SINC0_LIMIT2	SINC0 (Amplitude) Limits for Secondary Filter 2 Register	0x00000000
0x4004902C	SINC0_LIMIT3	SINC0 (Amplitude) Limits for Secondary Filter 3 Register	0x00000000
0x40049030	SINC0_BIAS0	SINC0 Bias for Group 0 Register	0x00000000
0x40049034	SINC0_BIAS1	SINC0 Bias for Group 1 Register	0x00000000
0x40049038	SINC0_PPTR0	SINC0 Primary (Filters) Pointer for Group 0 Register	0x00000000
0x4004903C	SINC0_PPTR1	SINC0 Primary (Filters) Pointer for Group 1 Register	0x00000000
0x40049040	SINC0_PHEAD0	SINC0 Primary (Filters) Head for Group 0 Register	0x00000000
0x40049044	SINC0_PHEAD1	SINC0 Primary (Filters) Head for Group 1 Register	0x00000000
0x40049048	SINC0_PTAIL0	SINC0 Primary (Filters) Tail for Group 0 Register	0x00000000
0x4004904C	SINC0_PTAIL1	SINC0 Primary (Filters) Tail for Group 1 Register	0x00000000
0x40049050	SINC0_HIS_STAT	SINC0 History Status Register	0x00000000

Table 47-49: CM41X_M0 SINC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40049080	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x40049084	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x40049088	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x4004908C	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x40049090	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x40049094	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x40049098	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x4004909C	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x400490A0	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x400490A4	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x400490A8	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x400490AC	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x400490B0	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x400490B4	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x400490B8	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x400490BC	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000

Table 47-50: CM41X_M0 SMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4002F00C	SMC0_B0CTL	SMC0 Bank 0 Control Register	0x01000000
0x4002F010	SMC0_B0TIM	SMC0 Bank 0 Timing Register	0x01010101
0x4002F014	SMC0_B0ETIM	SMC0 Bank 0 Extended Timing Register	0x00020200
0x4002F01C	SMC0_B1CTL	SMC0 Bank 1 Control Register	0x00000000
0x4002F020	SMC0_B1TIM	SMC0 Bank 1 Timing Register	0x01010101
0x4002F024	SMC0_B1ETIM	SMC0 Bank 1 Extended Timing Register	0x00020200
0x4002F02C	SMC0_B2CTL	SMC0 Bank 2 Control Register	0x00000000
0x4002F030	SMC0_B2TIM	SMC0 Bank 2 Timing Register	0x01010101
0x4002F034	SMC0_B2ETIM	SMC0 Bank 2 Extended Timing Register	0x00020200
0x4002F03C	SMC0_B3CTL	SMC0 Bank 3 Control Register	0x00000000
0x4002F040	SMC0_B3TIM	SMC0 Bank 3 Timing Register	0x01010101

Table 47-50: CM41X_M0 SMC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002F044	SMC0_B3ETIM	SMC0 Bank 3 Extended Timing Register	0x00020200

Table 47-51: CM41X_M0 SMPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x410E0000	SMPU0_CTL	SMPU0 SMPU Control Register	0x00000000
0x410E0004	SMPU0_STAT	SMPU0 SMPU Status Register	0x00000000
0x410E0008	SMPU0_IADDR	SMPU0 Interrupt Address Register	0x00000000
0x410E000C	SMPU0_IDTLS	SMPU0 Interrupt Details Register	0x00000000
0x410E0010	SMPU0_BADDR	SMPU0 Bus Error Address Register	0x00000000
0x410E0014	SMPU0_BDTLS	SMPU0 Bus Error Details Register	0x00000000
0x410E0020	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x410E0024	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x410E0028	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x410E002C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x410E0030	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x410E0034	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x410E0038	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x410E003C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x410E0040	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x410E0044	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x410E0048	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x410E004C	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x410E0050	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x410E0054	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x410E0058	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x410E005C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x410E0060	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x410E0064	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x410E0068	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x410E006C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000

Table 47-51: CM41X_M0 SMPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410E0070	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x410E0074	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x410E0078	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x410E007C	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x410E0220	SMPU0_REVID	SMPU0 SMPU Revision ID Register	0x00000010

Table 47-52: CM41X_M0 SMPU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400E0000	SMPU1_CTL	SMPU1 SMPU Control Register	0x00000000
0x400E0004	SMPU1_STAT	SMPU1 SMPU Status Register	0x00000000
0x400E0008	SMPU1_IADDR	SMPU1 Interrupt Address Register	0x00000000
0x400E000C	SMPU1_IDTLS	SMPU1 Interrupt Details Register	0x00000000
0x400E0010	SMPU1_BADDR	SMPU1 Bus Error Address Register	0x00000000
0x400E0014	SMPU1_BDTLS	SMPU1 Bus Error Details Register	0x00000000
0x400E0020	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E0024	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0028	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E002C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0030	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E0034	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0038	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E003C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0040	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E0044	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0048	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E004C	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0050	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E0054	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0058	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E005C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000

Table 47-52: CM41X_M0 SMPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400E0060	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E0064	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0068	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E006C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0070	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E0074	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0078	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E007C	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0080	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E0084	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0088	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E008C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0090	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E0094	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0098	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E009C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E00A0	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E00A4	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E00A8	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E00AC	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E00B0	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E00B4	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E00B8	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E00BC	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E00C0	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E00C4	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E00C8	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E00CC	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E00D0	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E00D4	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E00D8	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000

Table 47-52: CM41X_M0 SMPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400E00DC	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0220	SMPU1_REVID	SMPU1 SMPU Revision ID Register	0x00000010

Table 47-53: CM41X_M0 SMPU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400E1000	SMPU2_CTL	SMPU2 SMPU Control Register	0x00000000
0x400E1004	SMPU2_STAT	SMPU2 SMPU Status Register	0x00000000
0x400E1008	SMPU2_IADDR	SMPU2 Interrupt Address Register	0x00000000
0x400E100C	SMPU2_IDTLS	SMPU2 Interrupt Details Register	0x00000000
0x400E1010	SMPU2_BADDR	SMPU2 Bus Error Address Register	0x00000000
0x400E1014	SMPU2_BDTLS	SMPU2 Bus Error Details Register	0x00000000
0x400E1020	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x400E1024	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x400E1028	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x400E102C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x400E1030	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x400E1034	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x400E1038	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x400E103C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x400E1040	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x400E1044	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x400E1048	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x400E104C	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x400E1050	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x400E1054	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x400E1058	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x400E105C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x400E1060	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x400E1064	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x400E1068	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000

Table 47-53: CM41X_M0 SMPU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400E106C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x400E1070	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x400E1074	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x400E1078	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x400E107C	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x400E1220	SMPU2_REVID	SMPU2 SMPU Revision ID Register	0x00000010

Table 47-54: CM41X_M0 SPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41009004	SPI0_CTL	SPI0 Control Register	0x00000050
0x41009008	SPI0_RXCTL	SPI0 Receive Control Register	0x00000000
0x4100900C	SPI0_TXCTL	SPI0 Transmit Control Register	0x00000000
0x41009010	SPI0_CLK	SPI0 Clock Rate Register	0x00000000
0x41009014	SPI0_DLY	SPI0 Delay Register	0x00000301
0x41009018	SPI0_SLVSEL	SPI0 Slave Select Register	0x0000FE00
0x4100901C	SPI0_RWC	SPI0 Received Word Count Register	0x00000000
0x41009020	SPI0_RWCR	SPI0 Received Word Count Reload Register	0x00000000
0x41009024	SPI0_TWC	SPI0 Transmitted Word Count Register	0x00000000
0x41009028	SPI0_TWCR	SPI0 Transmitted Word Count Reload Register	0x00000000
0x41009030	SPI0_IMSK	SPI0 Interrupt Mask Register	0x00000000
0x41009034	SPI0_IMSK_CLR	SPI0 Interrupt Mask Clear Register	0x00000000
0x41009038	SPI0_IMSK_SET	SPI0 Interrupt Mask Set Register	0x00000000
0x41009040	SPI0_STAT	SPI0 Status Register	0x00440001
0x41009044	SPI0_ILAT	SPI0 Masked Interrupt Condition Register	0x00000000
0x41009048	SPI0_ILAT_CLR	SPI0 Masked Interrupt Clear Register	0x00000000
0x41009050	SPI0_RFIFO	SPI0 Receive FIFO Data Register	0x00000000
0x41009058	SPI0_TFIFO	SPI0 Transmit FIFO Data Register	0x00000000

Table 47-55: CM41X_M0 SPI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40040004	SPI1_CTL	SPI1 Control Register	0x00000050
0x40040008	SPI1_RXCTL	SPI1 Receive Control Register	0x00000000
0x4004000C	SPI1_TXCTL	SPI1 Transmit Control Register	0x00000000
0x40040010	SPI1_CLK	SPI1 Clock Rate Register	0x00000000
0x40040014	SPI1_DLY	SPI1 Delay Register	0x00000301
0x40040018	SPI1_SLVSEL	SPI1 Slave Select Register	0x0000FE00
0x4004001C	SPI1_RWC	SPI1 Received Word Count Register	0x00000000
0x40040020	SPI1_RWCR	SPI1 Received Word Count Reload Register	0x00000000
0x40040024	SPI1_TWC	SPI1 Transmitted Word Count Register	0x00000000
0x40040028	SPI1_TWCR	SPI1 Transmitted Word Count Reload Register	0x00000000
0x40040030	SPI1_IMSK	SPI1 Interrupt Mask Register	0x00000000
0x40040034	SPI1_IMSK_CLR	SPI1 Interrupt Mask Clear Register	0x00000000
0x40040038	SPI1_IMSK_SET	SPI1 Interrupt Mask Set Register	0x00000000
0x40040040	SPI1_STAT	SPI1 Status Register	0x00440001
0x40040044	SPI1_ILAT	SPI1 Masked Interrupt Condition Register	0x00000000
0x40040048	SPI1_ILAT_CLR	SPI1 Masked Interrupt Clear Register	0x00000000
0x40040050	SPI1_RFIFO	SPI1 Receive FIFO Data Register	0x00000000
0x40040058	SPI1_TFIFO	SPI1 Transmit FIFO Data Register	0x00000000

Table 47-56: CM41X_M0 SPORT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40042000	SPORT0_CTL_A	SPORT0 Half SPORT 'A' Control Register	0x00000000
0x40042004	SPORT0_DIV_A	SPORT0 Half SPORT 'A' Divisor Register	0x00000000
0x40042008	SPORT0_MCTL_A	SPORT0 Half SPORT 'A' Multichannel Control Register	0x00000000
0x4004200C	SPORT0_CS0_A	SPORT0 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x40042010	SPORT0_CS1_A	SPORT0 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x40042014	SPORT0_CS2_A	SPORT0 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x40042018	SPORT0_CS3_A	SPORT0 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000

Table 47-56: CM41X_M0 SPORT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40042020	SPORT0_ERR_A	SPORT0 Half SPORT 'A' Error Register	0x00000000
0x40042024	SPORT0_MSTAT_A	SPORT0 Half SPORT 'A' Multichannel Status Register	0x00000000
0x40042028	SPORT0_CTL2_A	SPORT0 Half SPORT 'A' Control 2 Register	0x00000000
0x40042040	SPORT0_TXPRI_A	SPORT0 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x40042044	SPORT0_RXPRI_A	SPORT0 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x40042048	SPORT0_TXSEC_A	SPORT0 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x4004204C	SPORT0_RXSEC_A	SPORT0 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x40042080	SPORT0_CTL_B	SPORT0 Half SPORT 'B' Control Register	0x00000000
0x40042084	SPORT0_DIV_B	SPORT0 Half SPORT 'B' Divisor Register	0x00000000
0x40042088	SPORT0_MCTL_B	SPORT0 Half SPORT 'B' Multichannel Control Register	0x00000000
0x4004208C	SPORT0_CS0_B	SPORT0 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x40042090	SPORT0_CS1_B	SPORT0 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x40042094	SPORT0_CS2_B	SPORT0 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x40042098	SPORT0_CS3_B	SPORT0 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x400420A0	SPORT0_ERR_B	SPORT0 Half SPORT 'B' Error Register	0x00000000
0x400420A4	SPORT0_MSTAT_B	SPORT0 Half SPORT 'B' Multichannel Status Register	0x00000000
0x400420A8	SPORT0_CTL2_B	SPORT0 Half SPORT 'B' Control 2 Register	0x00000000
0x400420C0	SPORT0_TXPRI_B	SPORT0 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x400420C4	SPORT0_RXPRI_B	SPORT0 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x400420C8	SPORT0_TXSEC_B	SPORT0 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x400420CC	SPORT0_RXSEC_B	SPORT0 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table 47-57: CM41X_M0 SPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x410F8000	SPU0_CTL	SPU0 Control Register	0x000000AD
0x410F8004	SPU0_STAT	SPU0 Status Register	0x00000000
0x410F8040	SPU0_TIMEOUT	SPU0 Timeout Register	0x00000000
0x410F8400	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table 47-57: CM41X_M0 SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410F8404	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8408	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F840C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8410	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8414	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8418	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F841C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8420	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8424	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8428	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F842C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8430	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8434	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8438	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F843C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8440	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8444	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8448	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F844C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8800	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8804	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8808	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F880C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8810	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8814	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8818	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F881C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8820	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8824	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8828	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F882C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000

Table 47-57: CM41X_M0 SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410F8830	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8834	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8838	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F883C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8840	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8844	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8848	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F884C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F9080	SPU0_IDTLS	SPU0 Interrupt Details Register	0x00000000
0x410F9084	SPU0_IADDR	SPU0 Interrupt Address Register	0x00000000
0x410F9FC8	SPU0_DEVID	SPU0 Device Configuration Register	0x00140891

Table 47-58: CM41X_M0 SPU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F8000	SPU1_CTL	SPU1 Control Register	0x000000AD
0x400F8004	SPU1_STAT	SPU1 Status Register	0x00000000
0x400F8040	SPU1_TIMEOUT	SPU1 Timeout Register	0x00000000
0x400F8400	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8404	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8408	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F840C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8410	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8414	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8418	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F841C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8420	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8424	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8428	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F842C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8430	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000

Table 47-58: CM41X_M0 SPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F8434	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8438	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F843C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8440	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8444	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8448	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F844C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8450	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8454	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8458	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F845C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8460	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8464	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8468	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F846C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8470	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8474	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8478	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F847C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8480	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8484	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8488	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F848C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8490	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8494	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8498	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F849C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84A0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84A4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84A8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84AC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000

Table 47-58: CM41X_M0 SPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F84B0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84B4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84B8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84BC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84C0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84C4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84C8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84CC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84D0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84D4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84D8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84DC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84E0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84E4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84E8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84EC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84F0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84F4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84F8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84FC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8500	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8504	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8800	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8804	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8808	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F880C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8810	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8814	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8818	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F881C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8820	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000

Table 47-58: CM41X_M0 SPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F8824	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8828	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F882C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8830	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8834	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8838	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F883C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8840	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8844	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8848	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F884C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8850	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8854	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8858	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F885C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8860	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8864	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8868	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F886C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8870	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8874	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8878	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F887C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8880	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8884	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8888	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F888C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8890	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8894	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8898	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F889C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000

Table 47-58: CM41X_M0 SPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F88A0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88A4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88A8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88AC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88B0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88B4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88B8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88BC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88C0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88C4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88C8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88CC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88D0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88D4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88D8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88DC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88E0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88E4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88E8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88EC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88F0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88F4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88F8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88FC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8900	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8904	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F9080	SPU1_IDTLS	SPU1 Interrupt Details Register	0x00000000
0x400F9084	SPU1_IADDR	SPU1 Interrupt Address Register	0x00000000
0x400F9FC8	SPU1_DEVID	SPU1 Device Configuration Register	0x00420891

Table 47-59: CM41X_M0 SWU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x410F0000	SWU0_GCTL	SWU0 Global Control Register	0x00000000
0x410F0004	SWU0_GSTAT	SWU0 Global Status Register	0x00000000
0x410F0010	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x410F0014	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x410F0018	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x410F001C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x410F0020	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x410F0024	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x410F0028	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x410F002C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x410F0030	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x410F0034	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x410F0038	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x410F003C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x410F0040	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x410F0044	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x410F0048	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x410F004C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x410F0050	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x410F0054	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x410F0058	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x410F005C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x410F0060	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x410F0064	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x410F0068	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x410F006C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x410F0070	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x410F0074	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x410F0078	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x410F007C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x410F0080	SWU0_CNT[n]	SWU0 Count Register n	0x00000000

Table 47-59: CM41X_M0 SWU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410F0084	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x410F0088	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x410F008C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000

Table 47-60: CM41X_M0 SWU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x410F1000	SWU1_GCTL	SWU1 Global Control Register	0x00000000
0x410F1004	SWU1_GSTAT	SWU1 Global Status Register	0x00000000
0x410F1010	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x410F1014	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x410F1018	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x410F101C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x410F1020	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x410F1024	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x410F1028	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x410F102C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x410F1030	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x410F1034	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x410F1038	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x410F103C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x410F1040	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x410F1044	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x410F1048	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x410F104C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x410F1050	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x410F1054	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x410F1058	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x410F105C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x410F1060	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x410F1064	SWU1_TARG[n]	SWU1 Target Register n	0x00000000

Table 47-60: CM41X_M0 SWU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410F1068	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x410F106C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x410F1070	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x410F1074	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x410F1078	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x410F107C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x410F1080	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x410F1084	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x410F1088	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x410F108C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000

Table 47-61: CM41X_M0 SWU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F0000	SWU2_GCTL	SWU2 Global Control Register	0x00000000
0x400F0004	SWU2_GSTAT	SWU2 Global Status Register	0x00000000
0x400F0010	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x400F0014	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x400F0018	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x400F001C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x400F0020	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x400F0024	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x400F0028	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x400F002C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x400F0030	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x400F0034	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x400F0038	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x400F003C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x400F0040	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x400F0044	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x400F0048	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000

Table 47-61: CM41X_M0 SWU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F004C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x400F0050	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x400F0054	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x400F0058	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x400F005C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x400F0060	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x400F0064	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x400F0068	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x400F006C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x400F0070	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x400F0074	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x400F0078	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x400F007C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x400F0080	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x400F0084	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x400F0088	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x400F008C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000

Table 47-62: CM41X_M0 SWU3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F1000	SWU3_GCTL	SWU3 Global Control Register	0x00000000
0x400F1004	SWU3_GSTAT	SWU3 Global Status Register	0x00000000
0x400F1010	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x400F1014	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x400F1018	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x400F101C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x400F1020	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x400F1024	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x400F1028	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x400F102C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000

Table 47-62: CM41X_M0 SWU3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F1030	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x400F1034	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x400F1038	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x400F103C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x400F1040	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x400F1044	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x400F1048	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x400F104C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x400F1050	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x400F1054	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x400F1058	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x400F105C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x400F1060	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x400F1064	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x400F1068	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x400F106C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x400F1070	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x400F1074	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x400F1078	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x400F107C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x400F1080	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x400F1084	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x400F1088	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x400F108C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000

Table 47-63: CM41X_M0 SWU4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F2000	SWU4_GCTL	SWU4 Global Control Register	0x00000000
0x400F2004	SWU4_GSTAT	SWU4 Global Status Register	0x00000000
0x400F2010	SWU4_CTL[n]	SWU4 Control Register n	0x00000000

Table 47-63: CM41X_M0 SWU4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F2014	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x400F2018	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x400F201C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x400F2020	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x400F2024	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x400F2028	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x400F202C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x400F2030	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x400F2034	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x400F2038	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x400F203C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x400F2040	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x400F2044	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x400F2048	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x400F204C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x400F2050	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x400F2054	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x400F2058	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x400F205C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x400F2060	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x400F2064	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x400F2068	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x400F206C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x400F2070	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x400F2074	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x400F2078	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x400F207C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x400F2080	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x400F2084	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x400F2088	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x400F208C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000

Table 47-64: CM41X_M0 SWU5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F3000	SWU5_GCTL	SWU5 Global Control Register	0x00000000
0x400F3004	SWU5_GSTAT	SWU5 Global Status Register	0x00000000
0x400F3010	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x400F3014	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x400F3018	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x400F301C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x400F3020	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x400F3024	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x400F3028	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x400F302C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x400F3030	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x400F3034	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x400F3038	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x400F303C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x400F3040	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x400F3044	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x400F3048	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x400F304C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x400F3050	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x400F3054	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x400F3058	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x400F305C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x400F3060	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x400F3064	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x400F3068	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x400F306C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x400F3070	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x400F3074	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x400F3078	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x400F307C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x400F3080	SWU5_CNT[n]	SWU5 Count Register n	0x00000000

Table 47-64: CM41X_M0 SWU5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F3084	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x400F3088	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x400F308C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000

Table 47-65: CM41X_M0 SWU6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F4000	SWU6_GCTL	SWU6 Global Control Register	0x00000000
0x400F4004	SWU6_GSTAT	SWU6 Global Status Register	0x00000000
0x400F4010	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x400F4014	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x400F4018	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x400F401C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x400F4020	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x400F4024	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x400F4028	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x400F402C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x400F4030	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x400F4034	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x400F4038	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x400F403C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x400F4040	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x400F4044	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x400F4048	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x400F404C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x400F4050	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x400F4054	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x400F4058	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x400F405C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x400F4060	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x400F4064	SWU6_TARG[n]	SWU6 Target Register n	0x00000000

Table 47-65: CM41X_M0 SWU6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F4068	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x400F406C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x400F4070	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x400F4074	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x400F4078	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x400F407C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x400F4080	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x400F4084	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x400F4088	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x400F408C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000

Table 47-66: CM41X_M0 TAPC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40017000	TAPC_IDCODE	TAPC IDCODE Register	0x2280B0CB
0x40017004	TAPC_USERCODE	TAPC USERCODE Register	0x00000000
0x40017008	TAPC_SDBGKEY_CTL	TAPC Secure Debug Key Control Register	0x00000000
0x4001700C	TAPC_SDBGKEY_STAT	TAPC Secure Debug Key Status Register	0x00000000
0x40017010	TAPC_SDBGKEY0	TAPC Secure Debug Key 0 Register	0x00000000
0x40017014	TAPC_SDBGKEY1	TAPC Secure Debug Key 1 Register	0x00000000
0x40017018	TAPC_SDBGKEY2	TAPC Secure Debug Key 2 Register	0x00000000
0x4001701C	TAPC_SDBGKEY3	TAPC Secure Debug Key 3 Register	0x00000000
0x40017030	TAPC_DBGCTL	TAPC Debug Control Register	0x00000000
0x40017034	TAPC_DBGSTAT	TAPC Debug Status Register	0x00000000
0x40017050	TAPC_SDBGKEYCMP0	TAPC Secure Debug Key 0 Compare Register	0x00000000
0x40017054	TAPC_SDBGKEYCMP1	TAPC Secure Debug Key 1 Compare Register	0x00000000
0x40017058	TAPC_SDBGKEYCMP2	TAPC Secure Debug Key 2 Compare Register	0x00000000
0x4001705C	TAPC_SDBGKEYCMP3	TAPC Secure Debug Key 3 Compare Register	0x00000000
0x40017070	TAPC_SDBGKEYID0	TAPC Secure Debug Key 0 Identification Register	0x00000000
0x40017074	TAPC_SDBGKEYID1	TAPC Secure Debug Key 1 Key Identification Register	0x00000000
0x40017078	TAPC_SDBGKEYID2	TAPC Secure Debug Key 2 Key Identification Register	0x00000000

Table 47-66: CM41X_M0 TAPC MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4001707C	TAPC_SDBGKEYID3	TAPC Secure Debug Key 3 Key Identification Register	0x00000000
0x400170E0	TAPC_SCMSG	TAPC System Run Control Message Register	0x00000000
0x400170E4	TAPC_SCMSG_SET	TAPC System Run Control Message Set Register	0x00000000
0x400170E8	TAPC_SCMSG_CLR	TAPC System Run Control Message Clear Register	0x00000000
0x400170EC	TAPC_SCMSG_TGL	TAPC System Run Control Message Toggle Register	0x00000000
0x400170F0	TAPC_RCMSG	TAPC Run Control Message Register	0x00000000
0x400170F4	TAPC_RCMSG_SET	TAPC Run Control Message Set Register	0x00000000
0x400170F8	TAPC_RCMSG_CLR	TAPC Run Control Message Clear Register	0x00000000
0x400170FC	TAPC_RCMSG_TGL	TAPC Run Control Message Toggle Register	0x00000000

Table 47-67: CM41X_M0 PADS0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41001004	PADS0_PCFG0	PADS0 Peripheral Configuration0 Register	0x00000000
0x41001100	PADS0_PORT[n]_DS	PADS0 Multi Port Drive Strength Control Register	0x00000000
0x41001120	PADS0_PORT[n]_RCTL	PADS0 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x41001140	PADS0_PORT[n]_TRIPST	PADS0 Multi Port Trip State Register	0x00000000
0x41001160	PADS0_PORT[n]_TRIPSEL	PADS0 Multi Port Trip Select Register	0x00000000

Table 47-68: CM41X_M0 PADS1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40001004	PADS1_PCFG0	PADS1 Peripheral Configuration0 Register	0x00000000
0x40001048	PADS1_FOCP_DIV	PADS1 Fast Over Current Protection Clock Divisor Register	0x00000000
0x40001070	PADS1_DBC_PRESCALE	PADS1 Debounce Prescale Register	0x00000000
0x40001080	PADS1_DBC[n]_CTL	PADS1 Debounce Control Register(s)	0x00000000
0x40001084	PADS1_DBC[n]_CTL	PADS1 Debounce Control Register(s)	0x00000000
0x40001088	PADS1_DBC[n]_CTL	PADS1 Debounce Control Register(s)	0x00000000
0x4000108C	PADS1_DBC[n]_CTL	PADS1 Debounce Control Register(s)	0x00000000
0x400010A0	PADS1_VMU_CTL	PADS1 Voltage Monitor Unit Control Register	0x00000000
0x400010A8	PADS1_VMU_TRIPEN	PADS1 Voltage Monitor Unit Trip Enable Register	0x00000000

Table 47-68: CM41X_M0 PADS1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400010AC	PADS1_NVWR_RSTCTL	PADS1 Non-Volatile Write Reset Control Register	0x00000000
0x400010B0	PADS1_MONOSC_CFG	PADS1 Monitor Oscillator Control Register	0x00000000
0x400010BC	PADS1_VMU_TRIM	PADS1 Voltage Monitor Unit Trim Register	0x00000000
0x40001100	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x40001104	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x40001108	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x4000110C	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x40001110	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x40001120	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x40001124	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x40001128	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x4000112C	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x40001130	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x40001140	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x40001144	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x40001148	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x4000114C	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x40001150	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x40001160	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000
0x40001164	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000
0x40001168	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000
0x4000116C	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000
0x40001170	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000

Table 47-69: CM41X_M0 SYSBLK0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41000040	SYSBLK0_SYSSTAT	SYSBLK0 System Status Register	0x00000000

Table 47-69: CM41X_M0 SYSBLK0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41000070	SYSBLK0_SCB_RESP_CFG	SYSBLK0 SCB Response Configuration Register	0x00000000
0x41000074	SYSBLK0_SCB_TIMEOUT_VALUE	SYSBLK0 SCB Timeout Value Register	0x00000000
0x41000080	SYSBLK0_ENG_MODE_CFG0	SYSBLK0 Engineering Mode Configuration Register 0	0x00000000
0x410000B4	SYSBLK0_M0_VTOR	SYSBLK0 Vector Table Base Offset Register	0x200F0000
0x410000E0	SYSBLK0_SRAM0_ECC	SYSBLK0 M0 SRAM ECC Register	0x00000000
0x410000E4	SYSBLK0_SYS_STCALIB	SYSBLK0 System Register	0x02000000
0x410000E8	SYSBLK0_IRQ_LATENCY	SYSBLK0 M0 IRQ Latency Register	0x0000000D
0x41000200	SYSBLK0_SISTAT0	SYSBLK0 Shared Interrupt 0 Status Register	0x00000000
0x4100020C	SYSBLK0_SISTAT3	SYSBLK0 Shared Interrupt 3 Status Register	0x00000000
0x41000214	SYSBLK0_SISTAT5	SYSBLK0 Shared Interrupt 5 Status Register	0x00000000
0x41000218	SYSBLK0_SISTAT6	SYSBLK0 Shared Interrupt 6 Status Register	0x00000000
0x4100021C	SYSBLK0_SISTAT7	SYSBLK0 Shared Interrupt 7 Status Register	0x00000000
0x41000220	SYSBLK0_SISTAT8	SYSBLK0 Shared Interrupt 8 Status Register	0x00000000
0x41000228	SYSBLK0_SISTAT10	SYSBLK0 Shared Interrupt 10 Status Register	0x00000000
0x4100022C	SYSBLK0_SISTAT11	SYSBLK0 Shared Interrupt 11 Status Register	0x00000000
0x41000230	SYSBLK0_SISTAT12	SYSBLK0 Shared Interrupt 12 Status Register	0x00000000
0x4100023C	SYSBLK0_SISTAT15	SYSBLK0 Shared Interrupt 15 Status Register	0x00000000
0x41000240	SYSBLK0_SISTAT16	SYSBLK0 Shared Interrupt 16 Status Register	0x00000000
0x41000244	SYSBLK0_SISTAT17	SYSBLK0 Shared Interrupt 17 Status Register	0x00000000
0x41000248	SYSBLK0_SISTAT18	SYSBLK0 Shared Interrupt 18 Status Register	0x00000000
0x4100024C	SYSBLK0_SISTAT19	SYSBLK0 Shared Interrupt 19 Status Register	0x00000000
0x41000258	SYSBLK0_SISTAT22	SYSBLK0 Shared Interrupt 22 Status Register	0x00000000
0x41000264	SYSBLK0_SISTAT25	SYSBLK0 Shared Interrupt 25 Status Register	0x00000000
0x41000270	SYSBLK0_SISTAT28	SYSBLK0 Shared Interrupt 28 Status Register	0x00000000

Table 47-70: CM41X_M0 SYSBLK1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40000010	SYSBLK1_DMA_MUXCTL	SYSBLK1 Peripheral DMA Multiplexer Control	0x00000000
0x40000014	SYSBLK1_PWM_SYS_CFG	SYSBLK1 PWM System Configuration Register	0x00000000

Table 47-70: CM41X_M0 SYSBLK1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40000018	SYSBLK1_MEMST_CTL	SYSBLK1 Memory Self-Test Control Register	0x00000000
0x4000001C	SYSBLK1_SINC_TEST	SYSBLK1 SINC Test Register	0x00000000
0x40000040	SYSBLK1_SYSSTAT	SYSBLK1 System Status Register	0x00000000
0x40000050	SYSBLK1_CLKNG_TRIPEN	SYSBLK1 Clock Not Good Trip Register	0x00000000
0x40000058	SYSBLK1_FAULT_TRIPEN	SYSBLK1 Fault Trip Register	0x00000010
0x40000060	SYSBLK1_ROT_UPDN_CFG	SYSBLK1 Rotary Counter Up/Down Configuration Register	0x00000000
0x40000070	SYSBLK1_SCB_RESP_CFG	SYSBLK1 SCB Response Configuration Register	0x00000000
0x40000074	SYSBLK1_SCB_TIMEOUT_VALUE	SYSBLK1 SCB Timeout Value Register	0x00000000
0x40000080	SYSBLK1_ENG_MODE_CFG0	SYSBLK1 Engineering Mode Configuration Register 0	0x00000000
0x40000084	SYSBLK1_LROM_STAT	SYSBLK1 Logic ROM Status Register	0x00000000

Table 47-71: CM41X_M0 TIMER0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41007004	TIMER0_RUN	TIMER0 Run Register	0x00000000
0x41007008	TIMER0_RUN_SET	TIMER0 Run Set Register	0x00000000
0x4100700C	TIMER0_RUN_CLR	TIMER0 Run Clear Register	0x00000000
0x41007010	TIMER0_STOP_CFG	TIMER0 Stop Configuration Register	0x00000000
0x41007014	TIMER0_STOP_CFG_SET	TIMER0 Stop Configuration Set Register	0x00000000
0x41007018	TIMER0_STOP_CFG_CLR	TIMER0 Stop Configuration Clear Register	0x00000000
0x4100701C	TIMER0_DATA_IMSK	TIMER0 Data Interrupt Mask Register	0x000000FF
0x41007020	TIMER0_STAT_IMSK	TIMER0 Status Interrupt Mask Register	0x000000FF
0x41007024	TIMER0_TRG_MSK	TIMER0 Trigger Master Mask Register	0x000000FF
0x41007028	TIMER0_TRG_IE	TIMER0 Trigger Slave Enable Register	0x00000000
0x4100702C	TIMER0_DATA_ILAT	TIMER0 Data Interrupt Latch Register	0x00000000
0x41007030	TIMER0_STAT_ILAT	TIMER0 Status Interrupt Latch Register	0x00000000
0x41007034	TIMER0_ERR_TYPE	TIMER0 Error Type Status Register	0x00000000
0x41007038	TIMER0_BCAST_PER	TIMER0 Broadcast Period Register	0x00000000
0x4100703C	TIMER0_BCAST_WID	TIMER0 Broadcast Width Register	0x00000000

Table 47-71: CM41X_M0 TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41007040	TIMER0_BCAST_DLY	TIMER0 Broadcast Delay Register	0x00000000
0x41007060	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007064	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007068	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100706C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007070	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x41007080	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007084	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007088	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100708C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007090	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x410070A0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x410070A4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x410070A8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x410070AC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x410070B0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x410070C0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x410070C4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x410070C8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x410070CC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x410070D0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x410070E0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x410070E4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x410070E8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x410070EC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x410070F0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x41007100	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007104	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007108	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100710C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007110	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table 47-71: CM41X_M0 TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41007120	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007124	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007128	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100712C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007130	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x41007140	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007144	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007148	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100714C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007150	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table 47-72: CM41X_M0 TIMER1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40021004	TIMER1_RUN	TIMER1 Run Register	0x00000000
0x40021008	TIMER1_RUN_SET	TIMER1 Run Set Register	0x00000000
0x4002100C	TIMER1_RUN_CLR	TIMER1 Run Clear Register	0x00000000
0x40021010	TIMER1_STOP_CFG	TIMER1 Stop Configuration Register	0x00000000
0x40021014	TIMER1_STOP_CFG_SET	TIMER1 Stop Configuration Set Register	0x00000000
0x40021018	TIMER1_STOP_CFG_CLR	TIMER1 Stop Configuration Clear Register	0x00000000
0x4002101C	TIMER1_DATA_IMSK	TIMER1 Data Interrupt Mask Register	0x000000FF
0x40021020	TIMER1_STAT_IMSK	TIMER1 Status Interrupt Mask Register	0x000000FF
0x40021024	TIMER1_TRG_MSK	TIMER1 Trigger Master Mask Register	0x000000FF
0x40021028	TIMER1_TRG_IE	TIMER1 Trigger Slave Enable Register	0x00000000
0x4002102C	TIMER1_DATA_ILAT	TIMER1 Data Interrupt Latch Register	0x00000000
0x40021030	TIMER1_STAT_ILAT	TIMER1 Status Interrupt Latch Register	0x00000000
0x40021034	TIMER1_ERR_TYPE	TIMER1 Error Type Status Register	0x00000000
0x40021038	TIMER1_BCAST_PER	TIMER1 Broadcast Period Register	0x00000000
0x4002103C	TIMER1_BCAST_WID	TIMER1 Broadcast Width Register	0x00000000
0x40021040	TIMER1_BCAST_DLY	TIMER1 Broadcast Delay Register	0x00000000
0x40021060	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000

Table 47-72: CM41X_M0 TIMER1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40021064	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x40021068	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002106C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021070	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x40021080	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x40021084	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x40021088	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002108C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021090	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x400210A0	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x400210A4	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x400210A8	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x400210AC	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x400210B0	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x400210C0	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x400210C4	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x400210C8	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x400210CC	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x400210D0	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x400210E0	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x400210E4	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x400210E8	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x400210EC	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x400210F0	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x40021100	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x40021104	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x40021108	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002110C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021110	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x40021120	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x40021124	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001

Table 47-72: CM41X_M0 TIMER1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40021128	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002112C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021130	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x40021140	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x40021144	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x40021148	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002114C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021150	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000

Table 47-73: CM41X_M0 TRU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41006000	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006004	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006008	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100600C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006010	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006014	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006018	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100601C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006020	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006024	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006028	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100602C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006030	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006034	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006038	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100603C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006040	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006044	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006048	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table 47-73: CM41X_M0 TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100604C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006050	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006054	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006058	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100605C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006060	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006064	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006068	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100606C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006070	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006074	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006078	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100607C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006080	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006084	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006088	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100608C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006090	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006094	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006098	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100609C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060A0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060A4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060A8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060AC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060B0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060B4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060B8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410067E0	TRU0_MTR	TRU0 Master Trigger Register	0x00000000
0x410067E8	TRU0_ERRADDR	TRU0 Error Address Register	0x00000000
0x410067EC	TRU0_STAT	TRU0 Status Information Register	0x00000000

Table 47-73: CM41X_M0 TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410067F4	TRU0_GCTL	TRU0 Global Control Register	0x00000000

Table 47-74: CM41X_M0 TRU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40020000	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020004	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020008	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002000C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020010	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020014	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020018	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002001C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020020	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020024	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020028	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002002C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020030	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020034	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020038	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002003C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020040	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020044	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020048	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002004C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020050	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020054	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020058	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002005C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020060	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020064	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000

Table 47-74: CM41X_M0 TRU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40020068	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002006C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020070	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020074	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020078	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002007C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020080	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020084	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020088	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002008C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020090	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020094	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020098	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002009C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200A0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200A4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200A8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200AC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200B0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200B4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200B8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200BC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200C0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200C4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200C8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200CC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200D0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200D4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200D8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200DC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200E0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000

Table 47-74: CM41X_M0 TRU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400200E4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200E8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200EC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200F0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200F4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200F8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200FC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020100	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020104	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020108	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002010C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020110	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020114	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020118	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002011C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020120	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020124	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020128	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002012C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020130	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020134	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020138	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002013C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020140	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020144	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020148	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002014C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020150	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020154	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020158	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002015C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000

Table 47-74: CM41X_M0 TRU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40020160	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020164	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020168	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002016C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020170	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020174	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020178	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002017C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020180	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020184	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020188	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002018C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020190	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020194	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020198	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002019C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201A0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201A4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201A8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201AC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201B0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201B4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201B8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201BC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201C0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201C4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201C8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201CC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201D0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201D4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201D8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000

Table 47-74: CM41X_M0 TRU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400201DC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201E0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201E4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201E8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201EC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201F0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201F4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201F8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201FC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020200	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020204	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020208	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002020C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020210	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020214	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020218	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400207E0	TRU1_MTR	TRU1 Master Trigger Register	0x00000000
0x400207E8	TRU1_ERRADDR	TRU1 Error Address Register	0x00000000
0x400207EC	TRU1_STAT	TRU1 Status Information Register	0x00000000
0x400207F4	TRU1_GCTL	TRU1 Global Control Register	0x00000000

Table 47-75: CM41X_M0 TTU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4002C000	TTU0_CTL	TTU0 TTU Control Register	0x00000000
0x4002C004	TTU0_REVID	TTU0 Revision ID Register	0x00000010
0x4002C008	TTU0_STOP	TTU0 Counter Stop Request Register	0x00000000
0x4002C00C	TTU0_STAT	TTU0 Counter Status Register	0x00000000
0x4002C010	TTU0_CHK	TTU0 Counter Check Register	0x00000000
0x4002C020	TTU0_CNT[n]	TTU0 Counter Current Value	0x00000000
0x4002C024	TTU0_CNT[n]	TTU0 Counter Current Value	0x00000000

Table 47-75: CM41X_M0 TTU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002C028	TTU0_CNT[n]	TTU0 Counter Current Value	0x00000000
0x4002C02C	TTU0_CNT[n]	TTU0 Counter Current Value	0x00000000
0x4002C040	TTU0_PER[n]	TTU0 Counter Period Register	0x00000000
0x4002C044	TTU0_PER[n]	TTU0 Counter Period Register	0x00000000
0x4002C048	TTU0_PER[n]	TTU0 Counter Period Register	0x00000000
0x4002C04C	TTU0_PER[n]	TTU0 Counter Period Register	0x00000000
0x4002C080	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C084	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C088	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C08C	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C090	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C094	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C098	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C09C	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C0C0	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0C4	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0C8	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0CC	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0D0	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0D4	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0D8	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0DC	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000

Table 47-76: CM41X_M0 TWI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40023000	TWI0_CLKDIV	TWI0 SCL Clock Divider Register	0x00000000
0x40023004	TWI0_CTL	TWI0 Control Register	0x00000000
0x40023008	TWI0_SLVCTL	TWI0 Slave Mode Control Register	0x00000000
0x4002300C	TWI0_SLVSTAT	TWI0 Slave Mode Status Register	0x00000000
0x40023010	TWI0_SLVADDR	TWI0 Slave Mode Address Register	0x00000000

Table 47-76: CM41X_M0 TWI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40023014	TWI0_MSTRCTL	TWI0 Master Mode Control Registers	0x00000000
0x40023018	TWI0_MSTRSTAT	TWI0 Master Mode Status Register	0x00000000
0x4002301C	TWI0_MSTRADDR	TWI0 Master Mode Address Register	0x00000000
0x40023020	TWI0_ISTAT	TWI0 Interrupt Status Register	0x00000000
0x40023024	TWI0_IMSK	TWI0 Interrupt Mask Register	0x00000000
0x40023028	TWI0_FIFOCTL	TWI0 FIFO Control Register	0x00000000
0x4002302C	TWI0_FIFOSTAT	TWI0 FIFO Status Register	0x00000000
0x40023080	TWI0_TXDATA8	TWI0 Tx Data Single-Byte Register	0x00000000
0x40023084	TWI0_TXDATA16	TWI0 Tx Data Double-Byte Register	0x00000000
0x40023088	TWI0_RXDATA8	TWI0 Rx Data Single-Byte Register	0x00000000
0x4002308C	TWI0_RXDATA16	TWI0 Rx Data Double-Byte Register	0x00000000

Table 47-77: CM41X_M0 UART0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100A004	UART0_CTL	UART0 Control Register	0x00000000
0x4100A008	UART0_STAT	UART0 Status Register	0x000000A0
0x4100A00C	UART0_SCR	UART0 Scratch Register	0x00000000
0x4100A010	UART0_CLK	UART0 Clock Rate Register	0x0000FFFF
0x4100A014	UART0_IMSK	UART0 Interrupt Mask Register	0x00000000
0x4100A018	UART0_IMSK_SET	UART0 Interrupt Mask Set Register	0x00000000
0x4100A01C	UART0_IMSK_CLR	UART0 Interrupt Mask Clear Register	0x00000000
0x4100A020	UART0_RBR	UART0 Receive Buffer Register	0x00000000
0x4100A024	UART0_THR	UART0 Transmit Hold Register	0x00000000
0x4100A028	UART0_TAIP	UART0 Transmit Address/Insert Pulse Register	0x00000000
0x4100A02C	UART0_TSR	UART0 Transmit Shift Register	0x000007FF
0x4100A030	UART0_RSR	UART0 Receive Shift Register	0x00000000
0x4100A034	UART0_TXCNT	UART0 Transmit Counter Register	0x00000000
0x4100A038	UART0_RXCNT	UART0 Receive Counter Register	0x00000000

Table 47-78: CM41X_M0 UART1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40044004	UART1_CTL	UART1 Control Register	0x00000000
0x40044008	UART1_STAT	UART1 Status Register	0x000000A0
0x4004400C	UART1_SCR	UART1 Scratch Register	0x00000000
0x40044010	UART1_CLK	UART1 Clock Rate Register	0x0000FFFF
0x40044014	UART1_IMSK	UART1 Interrupt Mask Register	0x00000000
0x40044018	UART1_IMSK_SET	UART1 Interrupt Mask Set Register	0x00000000
0x4004401C	UART1_IMSK_CLR	UART1 Interrupt Mask Clear Register	0x00000000
0x40044020	UART1_RBR	UART1 Receive Buffer Register	0x00000000
0x40044024	UART1_THR	UART1 Transmit Hold Register	0x00000000
0x40044028	UART1_TAIP	UART1 Transmit Address/Insert Pulse Register	0x00000000
0x4004402C	UART1_TSR	UART1 Transmit Shift Register	0x000007FF
0x40044030	UART1_RSR	UART1 Receive Shift Register	0x00000000
0x40044034	UART1_TXCNT	UART1 Transmit Counter Register	0x00000000
0x40044038	UART1_RXCNT	UART1 Receive Counter Register	0x00000000

Table 47-79: CM41X_M0 UART2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40045004	UART2_CTL	UART2 Control Register	0x00000000
0x40045008	UART2_STAT	UART2 Status Register	0x000000A0
0x4004500C	UART2_SCR	UART2 Scratch Register	0x00000000
0x40045010	UART2_CLK	UART2 Clock Rate Register	0x0000FFFF
0x40045014	UART2_IMSK	UART2 Interrupt Mask Register	0x00000000
0x40045018	UART2_IMSK_SET	UART2 Interrupt Mask Set Register	0x00000000
0x4004501C	UART2_IMSK_CLR	UART2 Interrupt Mask Clear Register	0x00000000
0x40045020	UART2_RBR	UART2 Receive Buffer Register	0x00000000
0x40045024	UART2_THR	UART2 Transmit Hold Register	0x00000000
0x40045028	UART2_TAIP	UART2 Transmit Address/Insert Pulse Register	0x00000000
0x4004502C	UART2_TSR	UART2 Transmit Shift Register	0x000007FF
0x40045030	UART2_RSR	UART2 Receive Shift Register	0x00000000
0x40045034	UART2_TXCNT	UART2 Transmit Counter Register	0x00000000

Table 47-79: CM41X_M0 UART2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40045038	UART2_RXCNT	UART2 Receive Counter Register	0x00000000

Table 47-80: CM41X_M0 UART3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40046004	UART3_CTL	UART3 Control Register	0x00000000
0x40046008	UART3_STAT	UART3 Status Register	0x000000A0
0x4004600C	UART3_SCR	UART3 Scratch Register	0x00000000
0x40046010	UART3_CLK	UART3 Clock Rate Register	0x0000FFFF
0x40046014	UART3_IMSK	UART3 Interrupt Mask Register	0x00000000
0x40046018	UART3_IMSK_SET	UART3 Interrupt Mask Set Register	0x00000000
0x4004601C	UART3_IMSK_CLR	UART3 Interrupt Mask Clear Register	0x00000000
0x40046020	UART3_RBR	UART3 Receive Buffer Register	0x00000000
0x40046024	UART3_THR	UART3 Transmit Hold Register	0x00000000
0x40046028	UART3_TAIP	UART3 Transmit Address/Insert Pulse Register	0x00000000
0x4004602C	UART3_TSR	UART3 Transmit Shift Register	0x000007FF
0x40046030	UART3_RSR	UART3 Receive Shift Register	0x00000000
0x40046034	UART3_TXCNT	UART3 Transmit Counter Register	0x00000000
0x40046038	UART3_RXCNT	UART3 Receive Counter Register	0x00000000

Table 47-81: CM41X_M0 UART4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40047004	UART4_CTL	UART4 Control Register	0x00000000
0x40047008	UART4_STAT	UART4 Status Register	0x000000A0
0x4004700C	UART4_SCR	UART4 Scratch Register	0x00000000
0x40047010	UART4_CLK	UART4 Clock Rate Register	0x0000FFFF
0x40047014	UART4_IMSK	UART4 Interrupt Mask Register	0x00000000
0x40047018	UART4_IMSK_SET	UART4 Interrupt Mask Set Register	0x00000000
0x4004701C	UART4_IMSK_CLR	UART4 Interrupt Mask Clear Register	0x00000000
0x40047020	UART4_RBR	UART4 Receive Buffer Register	0x00000000
0x40047024	UART4_THR	UART4 Transmit Hold Register	0x00000000

Table 47-81: CM41X_M0 UART4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40047028	UART4_TAIP	UART4 Transmit Address/Insert Pulse Register	0x00000000
0x4004702C	UART4_TSR	UART4 Transmit Shift Register	0x000007FF
0x40047030	UART4_RSR	UART4 Receive Shift Register	0x00000000
0x40047034	UART4_TXCNT	UART4 Transmit Counter Register	0x00000000
0x40047038	UART4_RXCNT	UART4 Receive Counter Register	0x00000000

Table 47-82: CM41X_M0 WDOG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41005000	WDOG0_CTL	WDOG0 Control Register	0x00000AD0
0x41005004	WDOG0_CNT	WDOG0 Count Register	0x00000000
0x41005008	WDOG0_STAT	WDOG0 Watchdog Timer Status Register	0x00000000

Table 47-83: CM41X_M0 WDOG1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40011000	WDOG1_CTL	WDOG1 Control Register	0x00000AD0
0x40011004	WDOG1_CNT	WDOG1 Count Register	0x00000000
0x40011008	WDOG1_STAT	WDOG1 Watchdog Timer Status Register	0x00000000

48 CM41X_M4 Register List

This appendix lists Memory-Mapped Register address and register names. The modules are presented in alphabetical order.

Table 48-1: CM41X_M4 ADCC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B000	ADCC0_CTL	ADCC0 Control Register	0x00000000
0x4100B004	ADCC0_ERRSTAT	ADCC0 Error Status Register	0x00000000
0x4100B008	ADCC0_ERRMSK	ADCC0 Error Mask Register	0x00000000
0x4100B00C	ADCC0_ERRMSK_SET	ADCC0 Error Mask Set Register	0x00000000
0x4100B010	ADCC0_ERRMSK_CLR	ADCC0 Error Mask Clear Register	0x00000000
0x4100B014	ADCC0_EISTAT	ADCC0 Event Interrupt Status Register	0x00000000
0x4100B018	ADCC0_EIMSK	ADCC0 Event Interrupt Mask Register	0x00000000
0x4100B01C	ADCC0_EIMSK_SET	ADCC0 Event Interrupt Mask Set Register	0x00000000
0x4100B020	ADCC0_EIMSK_CLR	ADCC0 Event Interrupt Mask Clear Register	0x00000000
0x4100B024	ADCC0_FISTAT	ADCC0 Frame Interrupt Status Register	0x00000000
0x4100B028	ADCC0_FIMSK	ADCC0 Frame Interrupt Mask Register	0x00000000
0x4100B02C	ADCC0_FIMSK_SET	ADCC0 Frame Interrupt Mask Set Register	0x00000000
0x4100B030	ADCC0_FIMSK_CLR	ADCC0 Frame Interrupt Mask Clear Register	0x00000000
0x4100B034	ADCC0_EVTEN	ADCC0 Event Enable Register	0x00000000
0x4100B038	ADCC0_EVTEN_SET	ADCC0 Event Enable Set Register	0x00000000
0x4100B03C	ADCC0_EVTEN_CLR	ADCC0 Event Enable Clear Register	0x00000000
0x4100B040	ADCC0_ECOL	ADCC0 Event Collision Status Register	0x00000000
0x4100B044	ADCC0_EMISS	ADCC0 Event Miss Status Register	0x00000000
0x4100B048	ADCC0_BPTR0	ADCC0 Base Pointer 0 Register	0x00000000
0x4100B04C	ADCC0_FRINC0	ADCC0 Frame Increment 0 Register	0x00000000

Table 48-1: CM41X_M4 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B050	ADCC0_CBSIZ0	ADCC0 Circular Buffer Size 0 Register	0x00000000
0x4100B054	ADCC0_TCA0	ADCC0 Timing Control A (ADC0) Register	0x00000000
0x4100B058	ADCC0_TCB0	ADCC0 Timing Control B (ADC0) Register	0x00000000
0x4100B05C	ADCC0_BWMON0	ADCC0 Bandwidth Monitor 0 Register	0x00000000
0x4100B064	ADCC0_BPTR1	ADCC0 DMA Base Pointer 1 Register	0x00000000
0x4100B068	ADCC0_FRINC1	ADCC0 Frame Increment 1 Register	0x00000000
0x4100B06C	ADCC0_CBSIZ1	ADCC0 Circular Buffer Size 1 Register	0x00000000
0x4100B070	ADCC0_TCA1	ADCC0 Timing Control A (ADC1) Register	0x00000000
0x4100B074	ADCC0_TCB1	ADCC0 Timing Control B (ADC1) Register	0x00000000
0x4100B078	ADCC0_BWMON1	ADCC0 Bandwidth Monitor 1 Register	0x00000000
0x4100B07C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B080	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B084	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B088	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B08C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B090	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B094	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B098	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B09C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0A0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0A4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0A8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0AC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0B0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0B4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0B8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0BC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0C0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0C4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0C8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0CC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000

Table 48-1: CM41X_M4 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B0D0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0D4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0D8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0DC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0E0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0E4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0E8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0EC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0F0	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0F4	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B0F8	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B0FC	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B100	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B104	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B108	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B10C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B110	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B114	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B118	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B11C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B120	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B124	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B128	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B12C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B130	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B134	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B138	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B13C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B140	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B144	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B148	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000

Table 48-1: CM41X_M4 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B14C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B150	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B154	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B158	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B15C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B160	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B164	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B168	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B16C	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B170	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B174	ADCC0_EVT[nn]	ADCC0 Event n Time Register	0x00000000
0x4100B178	ADCC0_EVCTL[nn]	ADCC0 Event n Control Register	0x00000000
0x4100B17C	ADCC0_NUMFRAM0	ADCC0 Timer0 Frame Limit Count Register	0x00000000
0x4100B180	ADCC0_NUMFRAM1	ADCC0 Timer1 Frame Limit Count Register	0x00000000
0x4100B184	ADCC0_DATOVF	ADCC0 Data Overflow Indication Register	0x00000000
0x4100B200	ADCC0_EPND	ADCC0 Pending Events Status Register	0x00000000
0x4100B204	ADCC0_T0STAT	ADCC0 Timer 0 Status Register	0x00000000
0x4100B208	ADCC0_TMR0	ADCC0 Timer 0 Current Count Register	0x00000000
0x4100B20C	ADCC0_T1STAT	ADCC0 Timer 1 Status Register	0x00000000
0x4100B210	ADCC0_TMR1	ADCC0 Timer 1 Current Count Register	0x00000000
0x4100B214	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B218	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B21C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B220	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B224	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B228	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B22C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B230	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B234	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B238	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B23C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000

Table 48-1: CM41X_M4 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B240	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B244	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B248	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B24C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B250	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B254	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B258	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B25C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B260	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B264	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B268	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B26C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B270	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B274	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B278	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B27C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B280	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B284	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B288	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B28C	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B290	ADCC0_EVDAT[nn]	ADCC0 Event n Data Register	0x00000000
0x4100B294	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B298	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B29C	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2A0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2A4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2A8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2AC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2B0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2B4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2B8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000

Table 48-1: CM41X_M4 ADCC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100B2BC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2C0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2C4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2C8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2CC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2D0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2D4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2D8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2DC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2E0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2E4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2E8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2EC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2F0	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2F4	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2F8	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B2FC	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B300	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B304	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B308	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B30C	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B310	ADCC0_EVSTAT[nn]	ADCC0 Event n Status Register	0x00000000
0x4100B314	ADCC0_CBNUM0	ADCC0 Timer0 Circular Buffer DMA Wrap Number Register	0x00000000
0x4100B318	ADCC0_CBNUM1	ADCC0 Timer1 Circular Buffer DMA Wrap Number Register	0x00000000
0x4100B31C	ADCC0_TRGCNT0	ADCC0 Trigger Count TIMER0 Register	0x00000000
0x4100B320	ADCC0_TRGCNT1	ADCC0 Trigger Count TIMER1 Register	0x00000000
0x4100B404	ADCC0_ADCRW0	ADCC0 ADC2 Interface RW Access Register	0x00000000
0x4100B408	ADCC0_ADCRW1	ADCC0 ADC2 Interface RW Access Register	0x00000000

Table 48-2: CM41X_M4 ADCC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A000	ADCC1_CTL	ADCC1 Control Register	0x00000000
0x4004A004	ADCC1_ERRSTAT	ADCC1 Error Status Register	0x00000000
0x4004A008	ADCC1_ERRMSK	ADCC1 Error Mask Register	0x00000000
0x4004A00C	ADCC1_ERRMSK_SET	ADCC1 Error Mask Set Register	0x00000000
0x4004A010	ADCC1_ERRMSK_CLR	ADCC1 Error Mask Clear Register	0x00000000
0x4004A014	ADCC1_EISTAT	ADCC1 Event Interrupt Status Register	0x00000000
0x4004A018	ADCC1_EIMSK	ADCC1 Event Interrupt Mask Register	0x00000000
0x4004A01C	ADCC1_EIMSK_SET	ADCC1 Event Interrupt Mask Set Register	0x00000000
0x4004A020	ADCC1_EIMSK_CLR	ADCC1 Event Interrupt Mask Clear Register	0x00000000
0x4004A024	ADCC1_FISTAT	ADCC1 Frame Interrupt Status Register	0x00000000
0x4004A028	ADCC1_FIMSK	ADCC1 Frame Interrupt Mask Register	0x00000000
0x4004A02C	ADCC1_FIMSK_SET	ADCC1 Frame Interrupt Mask Set Register	0x00000000
0x4004A030	ADCC1_FIMSK_CLR	ADCC1 Frame Interrupt Mask Clear Register	0x00000000
0x4004A034	ADCC1_EVTEN	ADCC1 Event Enable Register	0x00000000
0x4004A038	ADCC1_EVTEN_SET	ADCC1 Event Enable Set Register	0x00000000
0x4004A03C	ADCC1_EVTEN_CLR	ADCC1 Event Enable Clear Register	0x00000000
0x4004A040	ADCC1_ECOL	ADCC1 Event Collision Status Register	0x00000000
0x4004A044	ADCC1_EMISS	ADCC1 Event Miss Status Register	0x00000000
0x4004A048	ADCC1_BPTR0	ADCC1 Base Pointer 0 Register	0x00000000
0x4004A04C	ADCC1_FRINC0	ADCC1 Frame Increment 0 Register	0x00000000
0x4004A050	ADCC1_CBSIZ0	ADCC1 Circular Buffer Size 0 Register	0x00000000
0x4004A054	ADCC1_TCA0	ADCC1 Timing Control A (ADC0) Register	0x00000000
0x4004A058	ADCC1_TCB0	ADCC1 Timing Control B (ADC0) Register	0x00000000
0x4004A05C	ADCC1_BWMON0	ADCC1 Bandwidth Monitor 0 Register	0x00000000
0x4004A064	ADCC1_BPTR1	ADCC1 DMA Base Pointer 1 Register	0x00000000
0x4004A068	ADCC1_FRINC1	ADCC1 Frame Increment 1 Register	0x00000000
0x4004A06C	ADCC1_CBSIZ1	ADCC1 Circular Buffer Size 1 Register	0x00000000
0x4004A070	ADCC1_TCA1	ADCC1 Timing Control A (ADC1) Register	0x00000000
0x4004A074	ADCC1_TCB1	ADCC1 Timing Control B (ADC1) Register	0x00000000
0x4004A078	ADCC1_BWMON1	ADCC1 Bandwidth Monitor 1 Register	0x00000000
0x4004A07C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000

Table 48-2: CM41X_M4 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A080	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A084	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A088	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A08C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A090	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A094	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A098	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A09C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0A0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0A4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0A8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0AC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0B0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0B4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0B8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0BC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0C0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0C4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0C8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0CC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0D0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0D4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0D8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0DC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0E0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0E4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0E8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0EC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0F0	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A0F4	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A0F8	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000

Table 48-2: CM41X_M4 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A0FC	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A100	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A104	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A108	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A10C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A110	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A114	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A118	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A11C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A120	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A124	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A128	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A12C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A130	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A134	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A138	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A13C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A140	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A144	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A148	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A14C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A150	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A154	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A158	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A15C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A160	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A164	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A168	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A16C	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000
0x4004A170	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A174	ADCC1_EVT[nn]	ADCC1 Event n Time Register	0x00000000

Table 48-2: CM41X_M4 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A178	ADCC1_EVCTL[nn]	ADCC1 Event n Control Register	0x00000000
0x4004A17C	ADCC1_NUMFRAM0	ADCC1 Timer0 Frame Limit Count Register	0x00000000
0x4004A180	ADCC1_NUMFRAM1	ADCC1 Timer1 Frame Limit Count Register	0x00000000
0x4004A184	ADCC1_DATOVF	ADCC1 Data Overflow Indication Register	0x00000000
0x4004A200	ADCC1_EPND	ADCC1 Pending Events Status Register	0x00000000
0x4004A204	ADCC1_T0STAT	ADCC1 Timer 0 Status Register	0x00000000
0x4004A208	ADCC1_TMR0	ADCC1 Timer 0 Current Count Register	0x00000000
0x4004A20C	ADCC1_T1STAT	ADCC1 Timer 1 Status Register	0x00000000
0x4004A210	ADCC1_TMR1	ADCC1 Timer 1 Current Count Register	0x00000000
0x4004A214	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A218	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A21C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A220	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A224	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A228	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A22C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A230	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A234	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A238	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A23C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A240	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A244	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A248	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A24C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A250	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A254	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A258	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A25C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A260	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A264	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A268	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000

Table 48-2: CM41X_M4 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A26C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A270	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A274	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A278	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A27C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A280	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A284	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A288	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A28C	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A290	ADCC1_EVDAT[nn]	ADCC1 Event n Data Register	0x00000000
0x4004A294	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A298	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A29C	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2A0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2A4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2A8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2AC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2B0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2B4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2B8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2BC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2C0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2C4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2C8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2CC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2D0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2D4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2D8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2DC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2E0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2E4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000

Table 48-2: CM41X_M4 ADCC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004A2E8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2EC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2F0	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2F4	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2F8	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A2FC	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A300	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A304	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A308	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A30C	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A310	ADCC1_EVSTAT[nn]	ADCC1 Event n Status Register	0x00000000
0x4004A314	ADCC1_CBNUM0	ADCC1 Timer0 Circular Buffer DMA Wrap Number Register	0x00000000
0x4004A318	ADCC1_CBNUM1	ADCC1 Timer1 Circular Buffer DMA Wrap Number Register	0x00000000
0x4004A31C	ADCC1_TRGCNT0	ADCC1 Trigger Count TIMER0 Register	0x00000000
0x4004A320	ADCC1_TRGCNT1	ADCC1 Trigger Count TIMER1 Register	0x00000000
0x4004A404	ADCC1_ADCRW0	ADCC1 ADC2 Interface RW Access Register	0x00000000
0x4004A408	ADCC1_ADCRW1	ADCC1 ADC2 Interface RW Access Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41008200	CAN0_MC1	CAN0 Mailbox Configuration 1 Register	0x00000000
0x41008204	CAN0_MD1	CAN0 Mailbox Direction 1 Register	0x000000FF
0x41008208	CAN0_TRS1	CAN0 Transmission Request Set 1 Register	0x00000000
0x4100820C	CAN0_TRR1	CAN0 Transmission Request Reset 1 Register	0x00000000
0x41008210	CAN0_TA1	CAN0 Transmission Acknowledge 1 Register	0x00000000
0x41008214	CAN0_AA1	CAN0 Abort Acknowledge 1 Register	0x00000000
0x41008218	CAN0_RMP1	CAN0 Receive Message Pending 1 Register	0x00000000
0x4100821C	CAN0_RML1	CAN0 Receive Message Lost 1 Register	0x00000000
0x41008220	CAN0_MBTIF1	CAN0 Mailbox Transmit Interrupt Flag 1 Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008224	CAN0_MBRIF1	CAN0 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x41008228	CAN0_MBIM1	CAN0 Mailbox Interrupt Mask 1 Register	0x00000000
0x4100822C	CAN0_RFH1	CAN0 Remote Frame Handling 1 Register	0x00000000
0x41008230	CAN0_OPSS1	CAN0 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x41008240	CAN0_MC2	CAN0 Mailbox Configuration 2 Register	0x00000000
0x41008244	CAN0_MD2	CAN0 Mailbox Direction 2 Register	0x00000000
0x41008248	CAN0_TRS2	CAN0 Transmission Request Set 2 Register	0x00000000
0x4100824C	CAN0_TRR2	CAN0 Transmission Request Reset 2 Register	0x00000000
0x41008250	CAN0_TA2	CAN0 Transmission Acknowledge 2 Register	0x00000000
0x41008254	CAN0_AA2	CAN0 Abort Acknowledge 2 Register	0x00000000
0x41008258	CAN0_RMP2	CAN0 Receive Message Pending 2 Register	0x00000000
0x4100825C	CAN0_RML2	CAN0 Receive Message Lost 2 Register	0x00000000
0x41008260	CAN0_MBTIF2	CAN0 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x41008264	CAN0_MBRIF2	CAN0 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x41008268	CAN0_MBIM2	CAN0 Mailbox Interrupt Mask 2 Register	0x00000000
0x4100826C	CAN0_RFH2	CAN0 Remote Frame Handling 2 Register	0x00000000
0x41008270	CAN0_OPSS2	CAN0 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x41008280	CAN0_CLK	CAN0 Clock Register	0x00000000
0x41008284	CAN0_TIMING	CAN0 Timing Register	0x00000000
0x41008288	CAN0_DBG	CAN0 Debug Register	0x00000008
0x4100828C	CAN0_STAT	CAN0 Status Register	0x00000080
0x41008290	CAN0_CEC	CAN0 Error Counter Register	0x00000000
0x41008294	CAN0_GIS	CAN0 Global CAN Interrupt Status Register	0x00000000
0x41008298	CAN0_GIM	CAN0 Global CAN Interrupt Mask Register	0x00000000
0x4100829C	CAN0_GIF	CAN0 Global CAN Interrupt Flag Register	0x00000000
0x410082A0	CAN0_CTL	CAN0 CAN Master Control Register	0x00000080
0x410082A4	CAN0_INT	CAN0 Interrupt Pending Register	0x00000000
0x410082AC	CAN0_MBTD	CAN0 Temporary Mailbox Disable Register	0x00000000
0x410082B0	CAN0_EWR	CAN0 Error Counter Warning Level Register	0x00006060

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410082B4	CAN0_ESR	CAN0 Error Status Register	0x00000020
0x410082C4	CAN0_UCCNT	CAN0 Universal Counter Register	0x00000000
0x410082C8	CAN0_UCRC	CAN0 Universal Counter Reload/Capture Register	0x00000000
0x410082CC	CAN0_UCCNF	CAN0 Universal Counter Configuration Mode Register	0x00000000
0x41008300	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008304	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008308	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100830C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008310	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008314	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008318	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100831C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008320	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008324	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008328	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100832C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008330	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008334	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008338	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100833C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008340	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008344	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008348	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100834C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008350	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008354	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008358	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100835C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008360	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008364	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008368	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100836C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008370	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008374	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008378	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100837C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008380	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008384	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008388	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100838C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008390	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x41008394	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008398	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x4100839C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083A0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083A4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083A8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083AC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083B0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083B4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083B8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083BC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083C0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083C4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083C8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083CC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083D0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083D4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083D8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083DC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083E0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083E4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410083E8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083EC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083F0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083F4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x410083F8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x410083FC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x41008400	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008404	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008408	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100840C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008410	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008414	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008418	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100841C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008420	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008424	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008428	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100842C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008430	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008434	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008438	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100843C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008440	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008444	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008448	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100844C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008450	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008454	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008458	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100845C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008460	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008464	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008468	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100846C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008470	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008474	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008478	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100847C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008480	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008484	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008488	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100848C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008490	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008494	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008498	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100849C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410084A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410084A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410084A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410084AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410084B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410084B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410084B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410084BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410084C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410084C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410084C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410084CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410084D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410084D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410084D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410084DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410084E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410084E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410084E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410084EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410084F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410084F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410084F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410084FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008500	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008504	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008508	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100850C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008510	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008514	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008518	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100851C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008520	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008524	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008528	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100852C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008530	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008534	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008538	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100853C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008540	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008544	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008548	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100854C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008550	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008554	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008558	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100855C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008560	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008564	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008568	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100856C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008570	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008574	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008578	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100857C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008580	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008584	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008588	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100858C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008590	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008594	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008598	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100859C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410085A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410085A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410085A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410085AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410085B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410085B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410085B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410085BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410085C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410085C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410085C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410085CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410085D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410085D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410085D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410085DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410085E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410085E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410085E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410085EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410085F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410085F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410085F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410085FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008600	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008604	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008608	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100860C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008610	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008614	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008618	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100861C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008620	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008624	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008628	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100862C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008630	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008634	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008638	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100863C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008640	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008644	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008648	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100864C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008650	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008654	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008658	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100865C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008660	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008664	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008668	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100866C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008670	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008674	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008678	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100867C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008680	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008684	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008688	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100868C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008690	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008694	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008698	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100869C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410086A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410086A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410086A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410086AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410086B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410086B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410086B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410086BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410086C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410086C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410086C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410086CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410086D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410086D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410086D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410086DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410086E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410086E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410086E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410086EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410086F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410086F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410086F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410086FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008700	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008704	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008708	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100870C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008710	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008714	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008718	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100871C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008720	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008724	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008728	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100872C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008730	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008734	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008738	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100873C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008740	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008744	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008748	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100874C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008750	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008754	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008758	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100875C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008760	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008764	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008768	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100876C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008770	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41008774	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008778	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100877C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x41008780	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x41008784	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x41008788	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x4100878C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x41008790	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x41008794	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x41008798	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x4100879C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410087A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410087A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410087A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410087AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410087B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410087B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410087B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410087BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410087C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x410087C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410087C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410087CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410087D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410087D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410087D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410087DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x410087E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table 48-3: CM41X_M4 CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410087E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x410087E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x410087EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x410087F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x410087F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x410087F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x410087FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40024200	CAN1_MC1	CAN1 Mailbox Configuration 1 Register	0x00000000
0x40024204	CAN1_MD1	CAN1 Mailbox Direction 1 Register	0x000000FF
0x40024208	CAN1_TRS1	CAN1 Transmission Request Set 1 Register	0x00000000
0x4002420C	CAN1_TRR1	CAN1 Transmission Request Reset 1 Register	0x00000000
0x40024210	CAN1_TA1	CAN1 Transmission Acknowledge 1 Register	0x00000000
0x40024214	CAN1_AA1	CAN1 Abort Acknowledge 1 Register	0x00000000
0x40024218	CAN1_RMP1	CAN1 Receive Message Pending 1 Register	0x00000000
0x4002421C	CAN1_RML1	CAN1 Receive Message Lost 1 Register	0x00000000
0x40024220	CAN1_MBTIF1	CAN1 Mailbox Transmit Interrupt Flag 1 Register	0x00000000
0x40024224	CAN1_MBRIF1	CAN1 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x40024228	CAN1_MBIM1	CAN1 Mailbox Interrupt Mask 1 Register	0x00000000
0x4002422C	CAN1_RFH1	CAN1 Remote Frame Handling 1 Register	0x00000000
0x40024230	CAN1_OPSS1	CAN1 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x40024240	CAN1_MC2	CAN1 Mailbox Configuration 2 Register	0x00000000
0x40024244	CAN1_MD2	CAN1 Mailbox Direction 2 Register	0x00000000
0x40024248	CAN1_TRS2	CAN1 Transmission Request Set 2 Register	0x00000000
0x4002424C	CAN1_TRR2	CAN1 Transmission Request Reset 2 Register	0x00000000
0x40024250	CAN1_TA2	CAN1 Transmission Acknowledge 2 Register	0x00000000
0x40024254	CAN1_AA2	CAN1 Abort Acknowledge 2 Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024258	CAN1_RMP2	CAN1 Receive Message Pending 2 Register	0x00000000
0x4002425C	CAN1_RML2	CAN1 Receive Message Lost 2 Register	0x00000000
0x40024260	CAN1_MBTIF2	CAN1 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x40024264	CAN1_MBRIF2	CAN1 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x40024268	CAN1_MBIM2	CAN1 Mailbox Interrupt Mask 2 Register	0x00000000
0x4002426C	CAN1_RFH2	CAN1 Remote Frame Handling 2 Register	0x00000000
0x40024270	CAN1_OPSS2	CAN1 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x40024280	CAN1_CLK	CAN1 Clock Register	0x00000000
0x40024284	CAN1_TIMING	CAN1 Timing Register	0x00000000
0x40024288	CAN1_DBG	CAN1 Debug Register	0x00000008
0x4002428C	CAN1_STAT	CAN1 Status Register	0x00000080
0x40024290	CAN1_CEC	CAN1 Error Counter Register	0x00000000
0x40024294	CAN1_GIS	CAN1 Global CAN Interrupt Status Register	0x00000000
0x40024298	CAN1_GIM	CAN1 Global CAN Interrupt Mask Register	0x00000000
0x4002429C	CAN1_GIF	CAN1 Global CAN Interrupt Flag Register	0x00000000
0x400242A0	CAN1_CTL	CAN1 CAN Master Control Register	0x00000080
0x400242A4	CAN1_INT	CAN1 Interrupt Pending Register	0x00000000
0x400242AC	CAN1_MBTD	CAN1 Temporary Mailbox Disable Register	0x00000000
0x400242B0	CAN1_EWR	CAN1 Error Counter Warning Level Register	0x00006060
0x400242B4	CAN1_ESR	CAN1 Error Status Register	0x00000020
0x400242C4	CAN1_UCCNT	CAN1 Universal Counter Register	0x00000000
0x400242C8	CAN1_UCRC	CAN1 Universal Counter Reload/Capture Register	0x00000000
0x400242CC	CAN1_UCCNF	CAN1 Universal Counter Configuration Mode Register	0x00000000
0x40024300	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024304	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024308	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002430C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024310	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024314	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024318	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002431C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024320	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024324	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024328	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002432C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024330	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024334	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024338	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002433C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024340	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024344	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024348	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002434C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024350	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024354	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024358	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002435C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024360	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024364	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024368	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002436C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024370	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024374	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024378	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002437C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024380	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024384	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024388	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002438C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024390	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x40024394	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024398	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x4002439C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243A0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243A4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243A8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243AC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243B0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243B4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243B8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243BC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243C0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243C4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243C8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243CC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243D0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243D4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243D8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243DC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243E0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243E4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243E8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243EC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243F0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243F4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x400243F8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x400243FC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x40024400	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024404	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024408	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002440C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024410	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024414	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024418	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002441C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024420	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024424	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024428	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002442C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024430	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024434	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024438	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002443C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024440	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024444	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024448	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002444C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024450	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024454	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024458	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002445C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024460	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024464	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024468	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002446C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024470	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024474	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024478	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002447C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024480	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024484	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024488	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002448C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024490	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024494	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024498	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002449C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400244A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400244A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400244A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400244AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400244B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400244B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400244B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400244BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400244C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400244C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400244C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400244CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400244D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400244D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400244D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400244DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400244E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400244E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400244E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400244EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400244F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400244F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400244F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400244FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024500	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024504	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024508	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002450C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024510	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024514	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024518	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002451C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024520	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024524	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024528	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002452C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024530	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024534	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024538	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002453C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024540	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024544	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024548	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002454C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024550	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024554	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024558	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002455C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024560	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024564	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024568	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002456C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024570	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024574	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024578	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002457C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024580	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024584	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024588	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002458C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024590	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024594	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024598	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002459C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400245A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400245A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400245A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400245AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400245B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400245B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400245B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400245BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400245C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400245C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400245C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400245CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400245D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400245D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400245D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400245DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400245E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400245E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400245E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400245EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400245F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400245F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400245F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400245FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024600	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024604	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024608	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002460C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024610	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024614	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024618	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002461C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024620	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024624	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024628	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002462C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024630	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024634	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024638	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002463C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024640	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024644	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024648	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002464C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024650	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024654	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024658	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002465C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024660	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024664	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024668	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002466C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024670	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024674	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024678	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002467C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024680	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024684	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024688	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002468C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024690	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024694	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024698	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002469C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400246A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400246A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400246A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400246AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400246B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400246B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400246B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400246BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400246C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400246C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400246C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400246CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400246D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400246D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400246D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400246DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400246E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400246E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400246E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400246EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400246F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400246F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400246F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400246FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024700	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024704	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024708	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002470C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024710	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024714	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024718	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002471C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024720	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024724	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024728	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002472C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024730	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024734	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024738	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002473C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024740	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024744	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024748	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002474C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024750	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024754	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024758	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002475C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024760	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024764	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024768	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002476C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024770	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x40024774	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024778	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002477C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x40024780	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x40024784	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x40024788	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x4002478C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x40024790	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table 48-4: CM41X_M4 CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40024794	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x40024798	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x4002479C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400247A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400247A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400247A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400247AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400247B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400247B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400247B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400247BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400247C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400247C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400247C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400247CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400247D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400247D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400247D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400247DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x400247E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x400247E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x400247E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x400247EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x400247F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x400247F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x400247F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x400247FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000

Table 48-5: CM41X_M4 CGU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40014000	CGU0_CTL	CGU0 Control Register	0x00001200
0x40014004	CGU0_PLLCTL	CGU0 PLL Control Register	0x00000000
0x40014008	CGU0_STAT	CGU0 Status Register	0x0000000F
0x4001400C	CGU0_DIV	CGU0 Clocks Divisor Register	0x168F2926
0x40014010	CGU0_CLKOUTSEL	CGU0 CLKOUT Select Register	0x00000000
0x40014014	CGU0_OSCWDCTL	CGU0 Oscillator Watchdog Register	0x00007600
0x40014018	CGU0_TSCTL	CGU0 Time Stamp Control Register	0x00000000
0x4001401C	CGU0_TSVALUE0	CGU0 Time Stamp Counter Initial 32 LSB Value Register	0x00000000
0x40014020	CGU0_TSVALUE1	CGU0 Time Stamp Counter Initial MSB Value Register	0x00000000
0x40014024	CGU0_TSCOUNT0	CGU0 Time Stamp Counter 32 LSB Register	0x00000000
0x40014028	CGU0_TSCOUNT1	CGU0 Time Stamp Counter 32 MSB Register	0x00000000
0x4001402C	CGU0_CCBF_DIS	CGU0 Core Clock Buffer Disable Register	0x00000000
0x40014030	CGU0_CCBF_STAT	CGU0 Core Clock Buffer Status Register	0x00000000
0x40014038	CGU0_SCBF_DIS	CGU0 System Clock Buffer Disable Register	0x00000000
0x4001403C	CGU0_SCBF_STAT	CGU0 System Clock Buffer Status Register	0x00000000
0x40014048	CGU0_REVID	CGU0 Revision ID Register	0x00000020

Table 48-6: CM41X_M4 CNT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40022000	CNT0_CFG	CNT0 Configuration Register	0x00000000
0x40022004	CNT0_IMSK	CNT0 Interrupt Mask Register	0x00000000
0x40022008	CNT0_STAT	CNT0 Status Register	0x00000000
0x4002200C	CNT0_CMD	CNT0 Command Register	0x00000000
0x40022010	CNT0_DEBNCE	CNT0 Debounce Register	0x00000000
0x40022014	CNT0_CNTR	CNT0 Counter Register	0x00000000
0x40022018	CNT0_MAX	CNT0 Maximum Count Register	0x00000000
0x4002201C	CNT0_MIN	CNT0 Minimum Count Register	0x00000000
0x40022020	CNT0_MDIV	CNT0 M Value for Divider	0x00000000
0x40022024	CNT0_NDIV	CNT0 N Value for Divider	0x00000000

Table 48-7: CM41X_M4 CPTMR0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40025004	CPTMR0_RUN	CPTMR0 Run Register	0x00000000
0x40025008	CPTMR0_RUN_SET	CPTMR0 Run Set Register	0x00000000
0x4002500C	CPTMR0_RUN_CLR	CPTMR0 Run Clear Register	0x00000000
0x40025010	CPTMR0_DATA_IMSK	CPTMR0 Data Interrupt Mask Register	0x00000007
0x40025014	CPTMR0_DATA_IMSK_SET	CPTMR0 Data Interrupt Mask Set Register	0x00000007
0x40025018	CPTMR0_DATA_IMSK_CLR	CPTMR0 Data Interrupt Mask Clear Register	0x00000007
0x4002501C	CPTMR0_STAT_IMSK	CPTMR0 Status Interrupt Mask Register	0x00000007
0x40025020	CPTMR0_STAT_IMSK_SET	CPTMR0 Status Interrupt Mask Set Register	0x00000007
0x40025024	CPTMR0_STAT_IMSK_CLR	CPTMR0 Status Interrupt Mask Clear Register	0x00000007
0x40025028	CPTMR0_DATA_ILAT	CPTMR0 Data Interrupt Latch Status Register	0x00000000
0x4002502C	CPTMR0_STAT_ILAT	CPTMR0 Interrupt Latch Status Register	0x00000000
0x40025800	CPTMR0_CFG[n]	CPTMR0 Configuration Register	0x00000000
0x40025804	CPTMR0_CNT[n]	CPTMR0 Counter Register	0x00000001
0x40025808	CPTMR0_TON[n]	CPTMR0 On-time Capture Register	0x00000000
0x40025880	CPTMR0_CFG[n]	CPTMR0 Configuration Register	0x00000000
0x40025884	CPTMR0_CNT[n]	CPTMR0 Counter Register	0x00000001
0x40025888	CPTMR0_TON[n]	CPTMR0 On-time Capture Register	0x00000000
0x40025900	CPTMR0_CFG[n]	CPTMR0 Configuration Register	0x00000000
0x40025904	CPTMR0_CNT[n]	CPTMR0 Counter Register	0x00000001
0x40025908	CPTMR0_TON[n]	CPTMR0 On-time Capture Register	0x00000000

Table 48-8: CM41X_M4 CRC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C000	CRC0_CTL	CRC0 Control Register	0x00000000
0x4004C004	CRC0_DCNT	CRC0 Data Word Count Register	0x00000000
0x4004C008	CRC0_DCNTRLD	CRC0 Data Word Count Reload Register	0x00000000
0x4004C014	CRC0_COMP	CRC0 Data Compare Register	0x00000000
0x4004C018	CRC0_FILLVAL	CRC0 Fill Value Register	0x00000000

Table 48-8: CM41X_M4 CRC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4004C01C	CRC0_DFIFO	CRC0 Data FIFO Register	0x00000000
0x4004C020	CRC0_INEN	CRC0 Interrupt Enable Register	0x00000000
0x4004C024	CRC0_INEN_SET	CRC0 Interrupt Enable Set Register	0x00000000
0x4004C028	CRC0_INEN_CLR	CRC0 Interrupt Enable Clear Register	0x00000000
0x4004C02C	CRC0_POLY	CRC0 Polynomial Register	0x00000000
0x4004C040	CRC0_STAT	CRC0 Status Register	0x00000000
0x4004C044	CRC0_DCNTCAP	CRC0 Data Count Capture Register	0x00000000
0x4004C04C	CRC0_RESULT_FIN	CRC0 CRC Final Result Register	0x00000000
0x4004C050	CRC0_RESULT_CUR	CRC0 CRC Current Result Register	0x00000000

Table 48-9: CM41X_M4 CTI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xE0048000	CTI0_CTICONTROL	CTI0 CTI Control Register	0x00000000
0xE0048010	CTI0_CTIINTACK	CTI0 CTI Interrupt Acknowledge Register	0x00000000
0xE0048014	CTI0_CTIAPPSET	CTI0 CTI Application Trigger Set Register	0x00000000
0xE0048018	CTI0_CTIAPPCLEAR	CTI0 CTI Application Trigger Clear Register	0x00000000
0xE004801C	CTI0_CTIAPPPULSE	CTI0 CTI Application Pulse Register	0x00000000
0xE0048020	CTI0_CTIINEN0	CTI0 CTI Trigger 0 to Channel Enable Register	0x00000000
0xE0048024	CTI0_CTIINEN1	CTI0 CTI Trigger 1 to Channel Enable Register	0x00000000
0xE0048028	CTI0_CTIINEN2	CTI0 CTI Trigger 2 to Channel Enable Register	0x00000000
0xE004802C	CTI0_CTIINEN3	CTI0 CTI Trigger 3 to Channel Enable Register	0x00000000
0xE0048030	CTI0_CTIINEN4	CTI0 CTI Trigger 4 to Channel Enable Register	0x00000000
0xE0048034	CTI0_CTIINEN5	CTI0 CTI Trigger 5 to Channel Enable Register	0x00000000
0xE0048038	CTI0_CTIINEN6	CTI0 CTI Trigger 6 to Channel Enable Register	0x00000000
0xE004803C	CTI0_CTIINEN7	CTI0 CTI Trigger 7 to Channel Enable Register	0x00000000
0xE00480A0	CTI0_CTIOUTEN0	CTI0 CTI Channel to Trigger 0 Enable Register	0x00000000
0xE00480A4	CTI0_CTIOUTEN1	CTI0 CTI Channel to Trigger 1 Enable Register	0x00000000
0xE00480A8	CTI0_CTIOUTEN2	CTI0 CTI Channel to Trigger 2 Enable Register	0x00000000
0xE00480AC	CTI0_CTIOUTEN3	CTI0 CTI Channel to Trigger 3 Enable Register	0x00000000
0xE00480B0	CTI0_CTIOUTEN4	CTI0 CTI Channel to Trigger 4 Enable Register	0x00000000

Table 48-9: CM41X_M4 CTI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE00480B4	CTI0_CTIOUTEN5	CTI0 CTI Channel to Trigger 5 Enable Register	0x00000000
0xE00480B8	CTI0_CTIOUTEN6	CTI0 CTI Channel to Trigger 6 Enable Register	0x00000000
0xE00480BC	CTI0_CTIOUTEN7	CTI0 CTI Channel to Trigger 7 Enable Register	0x00000000
0xE0048130	CTI0_CTITRIGINSTATUS	CTI0 CTI Trigger In Status Register	0x00000000
0xE0048134	CTI0_CTITRIGOUTSTATUS	CTI0 CTI Trigger Out Status Register	0x00000000
0xE0048138	CTI0_CTICHINSTATUS	CTI0 CTI Channel In Status Register	0x00000000
0xE004813C	CTI0_CTICHOUTSTATUS	CTI0 CTI Channel Out Status Register	0x00000000
0xE0048140	CTI0_CTIGATE	CTI0 Enable CTI Channel Gate Register	0x0000000F
0xE0048144	CTI0_ASICCTL	CTI0 External Multiplexor Control Register	0x00000000
0xE0048EDC	CTI0_ITCHINACK	CTI0 ITCHINACK	0x00000000
0xE0048EE0	CTI0_ITTRIGINACK	CTI0 ITTRIGINACK	0x00000000
0xE0048EE4	CTI0_ITCHOUT	CTI0 ITCHOUT	0x00000000
0xE0048EE8	CTI0_ITTRIGOUT	CTI0 ITTRIGOUT	0x00000000
0xE0048EEC	CTI0_ITCHOUTACK	CTI0 ITCHOUTACK	0x00000000
0xE0048EF0	CTI0_ITTRIGOUTACK	CTI0 ITTRIGOUTACK	0x00000000
0xE0048EF4	CTI0_ITCHIN	CTI0 ITCHIN	0x00000000
0xE0048EF8	CTI0_ITTRIGIN	CTI0 ITTRIGIN	0x00000000
0xE0048F00	CTI0_ITCTRL	CTI0 Integration Mode Control Register	0x00000000
0xE0048FA0	CTI0_CLAIMSET	CTI0 Claim Tag Set Register	0x0000000F
0xE0048FA4	CTI0_CLAIMCLR	CTI0 Claim Tag Clear Register	0x00000000
0xE0048FB0	CTI0_LAR	CTI0 Lock Access Register	0x00000000
0xE0048FB4	CTI0_LSR	CTI0 Lock Status Register	0x00000000
0xE0048FB8	CTI0_AUTHSTATUS	CTI0 Authentication Status	0x00000005
0xE0048FC8	CTI0_DEVID	CTI0 Device ID	0x00040800
0xE0048FCC	CTI0_DEVTYPE	CTI0 Device Type	0x00000014
0xE0048FD0	CTI0_PERIPHID4	CTI0 Peripheral ID4	0x00000004
0xE0048FD4	CTI0_PERIPHID5	CTI0 Peripheral ID5	0x00000000
0xE0048FD8	CTI0_PERIPHID6	CTI0 Peripheral ID6	0x00000000
0xE0048FDC	CTI0_PERIPHID7	CTI0 Peripheral ID7	0x00000000
0xE0048FE0	CTI0_PERIPHID0	CTI0 Peripheral ID0	0x00000006

Table 48-9: CM41X_M4 CTI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE0048FE4	CTI0_PERIPHID1	CTI0 Peripheral ID1	0x000000B9
0xE0048FE8	CTI0_PERIPHID2	CTI0 Peripheral ID2	0x0000003B
0xE0048FEC	CTI0_PERIPHID3	CTI0 Peripheral ID3	0x00000000
0xE0048FF0	CTI0_COMPID0	CTI0 Component ID0	0x0000000D
0xE0048FF4	CTI0_COMPID1	CTI0 Component ID1	0x00000090
0xE0048FF8	CTI0_COMPID2	CTI0 Component ID2	0x00000005
0xE0048FFC	CTI0_COMPID3	CTI0 Component ID3	0x000000B1

Table 48-10: CM41X_M4 CTI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xE0049000	CTI1_CTICONTROL	CTI1 CTI Control Register	0x00000000
0xE0049010	CTI1_CTIINTACK	CTI1 CTI Interrupt Acknowledge Register	0x00000000
0xE0049014	CTI1_CTIAPPSET	CTI1 CTI Application Trigger Set Register	0x00000000
0xE0049018	CTI1_CTIAPPCLEAR	CTI1 CTI Application Trigger Clear Register	0x00000000
0xE004901C	CTI1_CTIAPPPULSE	CTI1 CTI Application Pulse Register	0x00000000
0xE0049020	CTI1_CTIINEN0	CTI1 CTI Trigger 0 to Channel Enable Register	0x00000000
0xE0049024	CTI1_CTIINEN1	CTI1 CTI Trigger 1 to Channel Enable Register	0x00000000
0xE0049028	CTI1_CTIINEN2	CTI1 CTI Trigger 2 to Channel Enable Register	0x00000000
0xE004902C	CTI1_CTIINEN3	CTI1 CTI Trigger 3 to Channel Enable Register	0x00000000
0xE0049030	CTI1_CTIINEN4	CTI1 CTI Trigger 4 to Channel Enable Register	0x00000000
0xE0049034	CTI1_CTIINEN5	CTI1 CTI Trigger 5 to Channel Enable Register	0x00000000
0xE0049038	CTI1_CTIINEN6	CTI1 CTI Trigger 6 to Channel Enable Register	0x00000000
0xE004903C	CTI1_CTIINEN7	CTI1 CTI Trigger 7 to Channel Enable Register	0x00000000
0xE00490A0	CTI1_CTIOUTEN0	CTI1 CTI Channel to Trigger 0 Enable Register	0x00000000
0xE00490A4	CTI1_CTIOUTEN1	CTI1 CTI Channel to Trigger 1 Enable Register	0x00000000
0xE00490A8	CTI1_CTIOUTEN2	CTI1 CTI Channel to Trigger 2 Enable Register	0x00000000
0xE00490AC	CTI1_CTIOUTEN3	CTI1 CTI Channel to Trigger 3 Enable Register	0x00000000
0xE00490B0	CTI1_CTIOUTEN4	CTI1 CTI Channel to Trigger 4 Enable Register	0x00000000
0xE00490B4	CTI1_CTIOUTEN5	CTI1 CTI Channel to Trigger 5 Enable Register	0x00000000
0xE00490B8	CTI1_CTIOUTEN6	CTI1 CTI Channel to Trigger 6 Enable Register	0x00000000

Table 48-10: CM41X_M4 CTI1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE00490BC	CTI1_CTIOUTEN7	CTI1 CTI Channel to Trigger 7 Enable Register	0x00000000
0xE0049130	CTI1_CTITRIGINSTATUS	CTI1 CTI Trigger In Status Register	0x00000000
0xE0049134	CTI1_CTITRIGOUTSTATUS	CTI1 CTI Trigger Out Status Register	0x00000000
0xE0049138	CTI1_CTICHINSTATUS	CTI1 CTI Channel In Status Register	0x00000000
0xE004913C	CTI1_CTICHOUTSTATUS	CTI1 CTI Channel Out Status Register	0x00000000
0xE0049140	CTI1_CTIGATE	CTI1 Enable CTI Channel Gate Register	0x0000000F
0xE0049144	CTI1_ASICCTL	CTI1 External Multiplexor Control Register	0x00000000
0xE0049EDC	CTI1_ITCHINACK	CTI1 ITCHINACK	0x00000000
0xE0049EE0	CTI1_ITTRIGINACK	CTI1 ITTRIGINACK	0x00000000
0xE0049EE4	CTI1_ITCHOUT	CTI1 ITCHOUT	0x00000000
0xE0049EE8	CTI1_ITTRIGOUT	CTI1 ITTRIGOUT	0x00000000
0xE0049EEC	CTI1_ITCHOUTACK	CTI1 ITCHOUTACK	0x00000000
0xE0049EF0	CTI1_ITTRIGOUTACK	CTI1 ITTRIGOUTACK	0x00000000
0xE0049EF4	CTI1_ITCHIN	CTI1 ITCHIN	0x00000000
0xE0049EF8	CTI1_ITTRIGIN	CTI1 ITTRIGIN	0x00000000
0xE0049F00	CTI1_ITCTRL	CTI1 Integration Mode Control Register	0x00000000
0xE0049FA0	CTI1_CLAIMSET	CTI1 Claim Tag Set Register	0x0000000F
0xE0049FA4	CTI1_CLAIMCLR	CTI1 Claim Tag Clear Register	0x00000000
0xE0049FB0	CTI1_LAR	CTI1 Lock Access Register	0x00000000
0xE0049FB4	CTI1_LSR	CTI1 Lock Status Register	0x00000000
0xE0049FB8	CTI1_AUTHSTATUS	CTI1 Authentication Status	0x00000005
0xE0049FC8	CTI1_DEVID	CTI1 Device ID	0x00040800
0xE0049FCC	CTI1_DEVTYPE	CTI1 Device Type	0x00000014
0xE0049FD0	CTI1_PERIPHID4	CTI1 Peripheral ID4	0x00000004
0xE0049FD4	CTI1_PERIPHID5	CTI1 Peripheral ID5	0x00000000
0xE0049FD8	CTI1_PERIPHID6	CTI1 Peripheral ID6	0x00000000
0xE0049FDC	CTI1_PERIPHID7	CTI1 Peripheral ID7	0x00000000
0xE0049FE0	CTI1_PERIPHID0	CTI1 Peripheral ID0	0x00000006
0xE0049FE4	CTI1_PERIPHID1	CTI1 Peripheral ID1	0x000000B9
0xE0049FE8	CTI1_PERIPHID2	CTI1 Peripheral ID2	0x0000003B

Table 48-10: CM41X_M4 CTI1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE0049FEC	CTI1_PERIPHID3	CTI1 Peripheral ID3	0x00000000
0xE0049FF0	CTI1_COMPID0	CTI1 Component ID0	0x0000000D
0xE0049FF4	CTI1_COMPID1	CTI1 Component ID1	0x00000090
0xE0049FF8	CTI1_COMPID2	CTI1 Component ID2	0x00000005
0xE0049FFC	CTI1_COMPID3	CTI1 Component ID3	0x000000B1

Table 48-11: CM41X_M4 CTI2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xE004A000	CTI2_CTICONTROL	CTI2 CTI Control Register	0x00000000
0xE004A010	CTI2_CTIINTACK	CTI2 CTI Interrupt Acknowledge Register	0x00000000
0xE004A014	CTI2_CTIAPPSET	CTI2 CTI Application Trigger Set Register	0x00000000
0xE004A018	CTI2_CTIAPPCLEAR	CTI2 CTI Application Trigger Clear Register	0x00000000
0xE004A01C	CTI2_CTIAPPPULSE	CTI2 CTI Application Pulse Register	0x00000000
0xE004A020	CTI2_CTIINEN0	CTI2 CTI Trigger 0 to Channel Enable Register	0x00000000
0xE004A024	CTI2_CTIINEN1	CTI2 CTI Trigger 1 to Channel Enable Register	0x00000000
0xE004A028	CTI2_CTIINEN2	CTI2 CTI Trigger 2 to Channel Enable Register	0x00000000
0xE004A02C	CTI2_CTIINEN3	CTI2 CTI Trigger 3 to Channel Enable Register	0x00000000
0xE004A030	CTI2_CTIINEN4	CTI2 CTI Trigger 4 to Channel Enable Register	0x00000000
0xE004A034	CTI2_CTIINEN5	CTI2 CTI Trigger 5 to Channel Enable Register	0x00000000
0xE004A038	CTI2_CTIINEN6	CTI2 CTI Trigger 6 to Channel Enable Register	0x00000000
0xE004A03C	CTI2_CTIINEN7	CTI2 CTI Trigger 7 to Channel Enable Register	0x00000000
0xE004A0A0	CTI2_CTIOUTEN0	CTI2 CTI Channel to Trigger 0 Enable Register	0x00000000
0xE004A0A4	CTI2_CTIOUTEN1	CTI2 CTI Channel to Trigger 1 Enable Register	0x00000000
0xE004A0A8	CTI2_CTIOUTEN2	CTI2 CTI Channel to Trigger 2 Enable Register	0x00000000
0xE004A0AC	CTI2_CTIOUTEN3	CTI2 CTI Channel to Trigger 3 Enable Register	0x00000000
0xE004A0B0	CTI2_CTIOUTEN4	CTI2 CTI Channel to Trigger 4 Enable Register	0x00000000
0xE004A0B4	CTI2_CTIOUTEN5	CTI2 CTI Channel to Trigger 5 Enable Register	0x00000000
0xE004A0B8	CTI2_CTIOUTEN6	CTI2 CTI Channel to Trigger 6 Enable Register	0x00000000
0xE004A0BC	CTI2_CTIOUTEN7	CTI2 CTI Channel to Trigger 7 Enable Register	0x00000000
0xE004A130	CTI2_CTITRIGINSTATUS	CTI2 CTI Trigger In Status Register	0x00000000

Table 48-11: CM41X_M4 CTI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE004A134	CTI2_CTITRIGOUTSTATUS	CTI2 CTI Trigger Out Status Register	0x00000000
0xE004A138	CTI2_CTICHINSTATUS	CTI2 CTI Channel In Status Register	0x00000000
0xE004A13C	CTI2_CTICHOUTSTATUS	CTI2 CTI Channel Out Status Register	0x00000000
0xE004A140	CTI2_CTIGATE	CTI2 Enable CTI Channel Gate Register	0x0000000F
0xE004A144	CTI2_ASICCTL	CTI2 External Multiplexor Control Register	0x00000000
0xE004AEDC	CTI2_ITCHINACK	CTI2 ITCHINACK	0x00000000
0xE004AEE0	CTI2_ITTRIGINACK	CTI2 ITTRIGINACK	0x00000000
0xE004AEE4	CTI2_ITCHOUT	CTI2 ITCHOUT	0x00000000
0xE004AEE8	CTI2_ITTRIGOUT	CTI2 ITTRIGOUT	0x00000000
0xE004AEEC	CTI2_ITCHOUTACK	CTI2 ITCHOUTACK	0x00000000
0xE004AEF0	CTI2_ITTRIGOUTACK	CTI2 ITTRIGOUTACK	0x00000000
0xE004AEF4	CTI2_ITCHIN	CTI2 ITCHIN	0x00000000
0xE004AEF8	CTI2_ITTRIGIN	CTI2 ITTRIGIN	0x00000000
0xE004AF00	CTI2_ITCTRL	CTI2 Integration Mode Control Register	0x00000000
0xE004AFA0	CTI2_CLAIMSET	CTI2 Claim Tag Set Register	0x0000000F
0xE004AFA4	CTI2_CLAIMCLR	CTI2 Claim Tag Clear Register	0x00000000
0xE004AFB0	CTI2_LAR	CTI2 Lock Access Register	0x00000000
0xE004AFB4	CTI2_LSR	CTI2 Lock Status Register	0x00000000
0xE004AFB8	CTI2_AUTHSTATUS	CTI2 Authentication Status	0x00000005
0xE004AFC8	CTI2_DEVID	CTI2 Device ID	0x00040800
0xE004AFCC	CTI2_DEVTYPE	CTI2 Device Type	0x00000014
0xE004AFD0	CTI2_PERIPHID4	CTI2 Peripheral ID4	0x00000004
0xE004AFD4	CTI2_PERIPHID5	CTI2 Peripheral ID5	0x00000000
0xE004AFD8	CTI2_PERIPHID6	CTI2 Peripheral ID6	0x00000000
0xE004AFDC	CTI2_PERIPHID7	CTI2 Peripheral ID7	0x00000000
0xE004AFE0	CTI2_PERIPHID0	CTI2 Peripheral ID0	0x00000006
0xE004AFE4	CTI2_PERIPHID1	CTI2 Peripheral ID1	0x000000B9
0xE004AFE8	CTI2_PERIPHID2	CTI2 Peripheral ID2	0x0000003B
0xE004AFEC	CTI2_PERIPHID3	CTI2 Peripheral ID3	0x00000000
0xE004AFF0	CTI2_COMPID0	CTI2 Component ID0	0x0000000D

Table 48-11: CM41X_M4 CTI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE004AFF4	CTI2_COMPID1	CTI2 Component ID1	0x00000090
0xE004AFF8	CTI2_COMPID2	CTI2 Component ID2	0x00000005
0xE004AFFC	CTI2_COMPID3	CTI2 Component ID3	0x000000B1

Table 48-12: CM41X_M4 CTI3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xE004B000	CTI3_CTICONTROL	CTI3 CTI Control Register	0x00000000
0xE004B010	CTI3_CTIINTACK	CTI3 CTI Interrupt Acknowledge Register	0x00000000
0xE004B014	CTI3_CTIAPPSET	CTI3 CTI Application Trigger Set Register	0x00000000
0xE004B018	CTI3_CTIAPPCLEAR	CTI3 CTI Application Trigger Clear Register	0x00000000
0xE004B01C	CTI3_CTIAPPPULSE	CTI3 CTI Application Pulse Register	0x00000000
0xE004B020	CTI3_CTIINEN0	CTI3 CTI Trigger 0 to Channel Enable Register	0x00000000
0xE004B024	CTI3_CTIINEN1	CTI3 CTI Trigger 1 to Channel Enable Register	0x00000000
0xE004B028	CTI3_CTIINEN2	CTI3 CTI Trigger 2 to Channel Enable Register	0x00000000
0xE004B02C	CTI3_CTIINEN3	CTI3 CTI Trigger 3 to Channel Enable Register	0x00000000
0xE004B030	CTI3_CTIINEN4	CTI3 CTI Trigger 4 to Channel Enable Register	0x00000000
0xE004B034	CTI3_CTIINEN5	CTI3 CTI Trigger 5 to Channel Enable Register	0x00000000
0xE004B038	CTI3_CTIINEN6	CTI3 CTI Trigger 6 to Channel Enable Register	0x00000000
0xE004B03C	CTI3_CTIINEN7	CTI3 CTI Trigger 7 to Channel Enable Register	0x00000000
0xE004B0A0	CTI3_CTIOUTEN0	CTI3 CTI Channel to Trigger 0 Enable Register	0x00000000
0xE004B0A4	CTI3_CTIOUTEN1	CTI3 CTI Channel to Trigger 1 Enable Register	0x00000000
0xE004B0A8	CTI3_CTIOUTEN2	CTI3 CTI Channel to Trigger 2 Enable Register	0x00000000
0xE004B0AC	CTI3_CTIOUTEN3	CTI3 CTI Channel to Trigger 3 Enable Register	0x00000000
0xE004B0B0	CTI3_CTIOUTEN4	CTI3 CTI Channel to Trigger 4 Enable Register	0x00000000
0xE004B0B4	CTI3_CTIOUTEN5	CTI3 CTI Channel to Trigger 5 Enable Register	0x00000000
0xE004B0B8	CTI3_CTIOUTEN6	CTI3 CTI Channel to Trigger 6 Enable Register	0x00000000
0xE004B0BC	CTI3_CTIOUTEN7	CTI3 CTI Channel to Trigger 7 Enable Register	0x00000000
0xE004B130	CTI3_CTITRIGINSTATUS	CTI3 CTI Trigger In Status Register	0x00000000
0xE004B134	CTI3_CTITRIGOUTSTATUS	CTI3 CTI Trigger Out Status Register	0x00000000
0xE004B138	CTI3_CTICHINSTATUS	CTI3 CTI Channel In Status Register	0x00000000

Table 48-12: CM41X_M4 CTI3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE004B13C	CTI3_CTICHOUTSTATUS	CTI3 CTI Channel Out Status Register	0x00000000
0xE004B140	CTI3_CTIGATE	CTI3 Enable CTI Channel Gate Register	0x0000000F
0xE004B144	CTI3_ASICCTL	CTI3 External Multiplexor Control Register	0x00000000
0xE004BEDC	CTI3_ITCHINACK	CTI3 ITCHINACK	0x00000000
0xE004BEE0	CTI3_ITTRIGINACK	CTI3 ITTRIGINACK	0x00000000
0xE004BEE4	CTI3_ITCHOUT	CTI3 ITCHOUT	0x00000000
0xE004BEE8	CTI3_ITTRIGOUT	CTI3 ITTRIGOUT	0x00000000
0xE004BEEC	CTI3_ITCHOUTACK	CTI3 ITCHOUTACK	0x00000000
0xE004BEF0	CTI3_ITTRIGOUTACK	CTI3 ITTRIGOUTACK	0x00000000
0xE004BEF4	CTI3_ITCHIN	CTI3 ITCHIN	0x00000000
0xE004BEF8	CTI3_ITTRIGIN	CTI3 ITTRIGIN	0x00000000
0xE004BF00	CTI3_ITCTRL	CTI3 Integration Mode Control Register	0x00000000
0xE004BFA0	CTI3_CLAIMSET	CTI3 Claim Tag Set Register	0x0000000F
0xE004BFA4	CTI3_CLAIMCLR	CTI3 Claim Tag Clear Register	0x00000000
0xE004BFB0	CTI3_LAR	CTI3 Lock Access Register	0x00000000
0xE004BFB4	CTI3_LSR	CTI3 Lock Status Register	0x00000000
0xE004BFB8	CTI3_AUTHSTATUS	CTI3 Authentication Status	0x00000005
0xE004BFC8	CTI3_DEVID	CTI3 Device ID	0x00040800
0xE004BFCC	CTI3_DEVTYPE	CTI3 Device Type	0x00000014
0xE004BFD0	CTI3_PERIPHID4	CTI3 Peripheral ID4	0x00000004
0xE004BFD4	CTI3_PERIPHID5	CTI3 Peripheral ID5	0x00000000
0xE004BFD8	CTI3_PERIPHID6	CTI3 Peripheral ID6	0x00000000
0xE004BFDC	CTI3_PERIPHID7	CTI3 Peripheral ID7	0x00000000
0xE004BFE0	CTI3_PERIPHID0	CTI3 Peripheral ID0	0x00000006
0xE004BFE4	CTI3_PERIPHID1	CTI3 Peripheral ID1	0x000000B9
0xE004BFE8	CTI3_PERIPHID2	CTI3 Peripheral ID2	0x0000003B
0xE004BFEC	CTI3_PERIPHID3	CTI3 Peripheral ID3	0x00000000
0xE004BFF0	CTI3_COMPID0	CTI3 Component ID0	0x0000000D
0xE004BFF4	CTI3_COMPID1	CTI3 Component ID1	0x00000090
0xE004BFF8	CTI3_COMPID2	CTI3 Component ID2	0x00000005
0xE004BFFC	CTI3_COMPID3	CTI3 Component ID3	0x000000B1

Table 48-13: CM41X_M4 DACC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4004B000	DACC0_CTL0	DACC0 Control 0 Register	0x00000000
0x4004B008	DACC0_ERRSTAT	DACC0 Error Status Register	0x00000000
0x4004B00C	DACC0_ERRMSK	DACC0 Error Mask Register	0x00000000
0x4004B010	DACC0_ERRMSK_SET	DACC0 Error Mask Set Register	0x00000000
0x4004B014	DACC0_ERRMSK_CLR	DACC0 Error Mask Clear Register	0x00000000
0x4004B018	DACC0_ISTAT	DACC0 Interrupt Status Register	0x00000000
0x4004B01C	DACC0_IMSK	DACC0 Interrupt Mask Register	0x00000000
0x4004B020	DACC0_IMSK_SET	DACC0 Interrupt Mask Set Register	0x00000000
0x4004B024	DACC0_IMSK_CLR	DACC0 Interrupt Mask Clear Register	0x00000000
0x4004B028	DACC0_TC0	DACC0 Timing Control 0 Register	0x00000000
0x4004B02C	DACC0_BPTR0	DACC0 Base Pointer 0 Register	0x00000000
0x4004B030	DACC0_MOD0	DACC0 Modify 0 Register	0x00000000
0x4004B034	DACC0_CNT0	DACC0 Count 0 Register	0x00000000
0x4004B038	DACC0_DAT0	DACC0 Data FIFO 0 Register	0x00000000
0x4004B050	DACC0_BCST_CTL	DACC0 Broadcast (Write) Control Register	0x00000000
0x4004B100	DACC0_CNTCUR0	DACC0 Current Count 0 Register	0x00000000
0x4004B108	DACC0_STAT	DACC0 Status Register	0x00000000

Table 48-14: CM41X_M4 DMA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100C000	DMA0_DSCPTR_NXT	DMA0 Pointer to Next Initial Descriptor Register	0x00000000
0x4100C004	DMA0_ADDRSTART	DMA0 Start Address of Current Buffer Register	0x00000000
0x4100C008	DMA0_CFG	DMA0 Configuration Register	0x00000000
0x4100C00C	DMA0_XCNT	DMA0 Inner Loop Count Start Value Register	0x00000000
0x4100C010	DMA0_XMOD	DMA0 Inner Loop Address Increment Register	0x00000000
0x4100C014	DMA0_YCNT	DMA0 Outer Loop Count Start Value (2D only) Register	0x00000000
0x4100C018	DMA0_YMOD	DMA0 Outer Loop Address Increment (2D only) Register	0x00000000
0x4100C024	DMA0_DSCPTR_CUR	DMA0 Current Descriptor Pointer Register	0x00000000
0x4100C028	DMA0_DSCPTR_PRV	DMA0 Previous Initial Descriptor Pointer Register	0x00000000
0x4100C02C	DMA0_ADDR_CUR	DMA0 Current Address Register	0x00000000

Table 48-14: CM41X_M4 DMA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100C030	DMA0_STAT	DMA0 Status Register	0x00006000
0x4100C034	DMA0_XCNT_CUR	DMA0 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x4100C038	DMA0_YCNT_CUR	DMA0 Current Row Count (2D only) Register	0x00000000

Table 48-15: CM41X_M4 DMA1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100D000	DMA1_DSCPTR_NXT	DMA1 Pointer to Next Initial Descriptor Register	0x00000000
0x4100D004	DMA1_ADDRSTART	DMA1 Start Address of Current Buffer Register	0x00000000
0x4100D008	DMA1_CFG	DMA1 Configuration Register	0x00000000
0x4100D00C	DMA1_XCNT	DMA1 Inner Loop Count Start Value Register	0x00000000
0x4100D010	DMA1_XMOD	DMA1 Inner Loop Address Increment Register	0x00000000
0x4100D014	DMA1_YCNT	DMA1 Outer Loop Count Start Value (2D only) Register	0x00000000
0x4100D018	DMA1_YMOD	DMA1 Outer Loop Address Increment (2D only) Register	0x00000000
0x4100D024	DMA1_DSCPTR_CUR	DMA1 Current Descriptor Pointer Register	0x00000000
0x4100D028	DMA1_DSCPTR_PRV	DMA1 Previous Initial Descriptor Pointer Register	0x00000000
0x4100D02C	DMA1_ADDR_CUR	DMA1 Current Address Register	0x00000000
0x4100D030	DMA1_STAT	DMA1 Status Register	0x00006000
0x4100D034	DMA1_XCNT_CUR	DMA1 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x4100D038	DMA1_YCNT_CUR	DMA1 Current Row Count (2D only) Register	0x00000000

Table 48-16: CM41X_M4 DMA10 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40066000	DMA10_DSCPTR_NXT	DMA10 Pointer to Next Initial Descriptor Register	0x00000000
0x40066004	DMA10_ADDRSTART	DMA10 Start Address of Current Buffer Register	0x00000000
0x40066008	DMA10_CFG	DMA10 Configuration Register	0x00000000
0x4006600C	DMA10_XCNT	DMA10 Inner Loop Count Start Value Register	0x00000000
0x40066010	DMA10_XMOD	DMA10 Inner Loop Address Increment Register	0x00000000
0x40066014	DMA10_YCNT	DMA10 Outer Loop Count Start Value (2D only) Register	0x00000000

Table 48-16: CM41X_M4 DMA10 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40066018	DMA10_YMOD	DMA10 Outer Loop Address Increment (2D only) Register	0x00000000
0x40066024	DMA10_DSCPTR_CUR	DMA10 Current Descriptor Pointer Register	0x00000000
0x40066028	DMA10_DSCPTR_PRV	DMA10 Previous Initial Descriptor Pointer Register	0x00000000
0x4006602C	DMA10_ADDR_CUR	DMA10 Current Address Register	0x00000000
0x40066030	DMA10_STAT	DMA10 Status Register	0x00006000
0x40066034	DMA10_XCNT_CUR	DMA10 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40066038	DMA10_YCNT_CUR	DMA10 Current Row Count (2D only) Register	0x00000000

Table 48-17: CM41X_M4 DMA11 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40067000	DMA11_DSCPTR_NXT	DMA11 Pointer to Next Initial Descriptor Register	0x00000000
0x40067004	DMA11_ADDRSTART	DMA11 Start Address of Current Buffer Register	0x00000000
0x40067008	DMA11_CFG	DMA11 Configuration Register	0x00000000
0x4006700C	DMA11_XCNT	DMA11 Inner Loop Count Start Value Register	0x00000000
0x40067010	DMA11_XMOD	DMA11 Inner Loop Address Increment Register	0x00000000
0x40067014	DMA11_YCNT	DMA11 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40067018	DMA11_YMOD	DMA11 Outer Loop Address Increment (2D only) Register	0x00000000
0x40067024	DMA11_DSCPTR_CUR	DMA11 Current Descriptor Pointer Register	0x00000000
0x40067028	DMA11_DSCPTR_PRV	DMA11 Previous Initial Descriptor Pointer Register	0x00000000
0x4006702C	DMA11_ADDR_CUR	DMA11 Current Address Register	0x00000000
0x40067030	DMA11_STAT	DMA11 Status Register	0x00006000
0x40067034	DMA11_XCNT_CUR	DMA11 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40067038	DMA11_YCNT_CUR	DMA11 Current Row Count (2D only) Register	0x00000000

Table 48-18: CM41X_M4 DMA12 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40068000	DMA12_DSCPTR_NXT	DMA12 Pointer to Next Initial Descriptor Register	0x00000000

Table 48-18: CM41X_M4 DMA12 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40068004	DMA12_ADDRSTART	DMA12 Start Address of Current Buffer Register	0x00000000
0x40068008	DMA12_CFG	DMA12 Configuration Register	0x00000000
0x4006800C	DMA12_XCNT	DMA12 Inner Loop Count Start Value Register	0x00000000
0x40068010	DMA12_XMOD	DMA12 Inner Loop Address Increment Register	0x00000000
0x40068014	DMA12_YCNT	DMA12 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40068018	DMA12_YMOD	DMA12 Outer Loop Address Increment (2D only) Register	0x00000000
0x40068024	DMA12_DSCPTR_CUR	DMA12 Current Descriptor Pointer Register	0x00000000
0x40068028	DMA12_DSCPTR_PRV	DMA12 Previous Initial Descriptor Pointer Register	0x00000000
0x4006802C	DMA12_ADDR_CUR	DMA12 Current Address Register	0x00000000
0x40068030	DMA12_STAT	DMA12 Status Register	0x00006000
0x40068034	DMA12_XCNT_CUR	DMA12 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40068038	DMA12_YCNT_CUR	DMA12 Current Row Count (2D only) Register	0x00000000
0x40068040	DMA12_BWLCNT	DMA12 Bandwidth Limit Count Register	0x00000000
0x40068044	DMA12_BWLCNT_CUR	DMA12 Bandwidth Limit Count Current Register	0x00000000
0x40068048	DMA12_BWMCNT	DMA12 Bandwidth Monitor Count Register	0x00000000
0x4006804C	DMA12_BWMCNT_CUR	DMA12 Bandwidth Monitor Count Current Register	0x00000000

Table 48-19: CM41X_M4 DMA13 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40069000	DMA13_DSCPTR_NXT	DMA13 Pointer to Next Initial Descriptor Register	0x00000000
0x40069004	DMA13_ADDRSTART	DMA13 Start Address of Current Buffer Register	0x00000000
0x40069008	DMA13_CFG	DMA13 Configuration Register	0x00000000
0x4006900C	DMA13_XCNT	DMA13 Inner Loop Count Start Value Register	0x00000000
0x40069010	DMA13_XMOD	DMA13 Inner Loop Address Increment Register	0x00000000
0x40069014	DMA13_YCNT	DMA13 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40069018	DMA13_YMOD	DMA13 Outer Loop Address Increment (2D only) Register	0x00000000
0x40069024	DMA13_DSCPTR_CUR	DMA13 Current Descriptor Pointer Register	0x00000000

Table 48-19: CM41X_M4 DMA13 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40069028	DMA13_DSCPTR_PRV	DMA13 Previous Initial Descriptor Pointer Register	0x00000000
0x4006902C	DMA13_ADDR_CUR	DMA13 Current Address Register	0x00000000
0x40069030	DMA13_STAT	DMA13 Status Register	0x00006000
0x40069034	DMA13_XCNT_CUR	DMA13 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40069038	DMA13_YCNT_CUR	DMA13 Current Row Count (2D only) Register	0x00000000
0x40069040	DMA13_BWLCNT	DMA13 Bandwidth Limit Count Register	0x00000000
0x40069044	DMA13_BWLCNT_CUR	DMA13 Bandwidth Limit Count Current Register	0x00000000
0x40069048	DMA13_BWMCNT	DMA13 Bandwidth Monitor Count Register	0x00000000
0x4006904C	DMA13_BWMCNT_CUR	DMA13 Bandwidth Monitor Count Current Register	0x00000000

Table 48-20: CM41X_M4 DMA2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100E000	DMA2_DSCPTR_NXT	DMA2 Pointer to Next Initial Descriptor Register	0x00000000
0x4100E004	DMA2_ADDRSTART	DMA2 Start Address of Current Buffer Register	0x00000000
0x4100E008	DMA2_CFG	DMA2 Configuration Register	0x00000000
0x4100E00C	DMA2_XCNT	DMA2 Inner Loop Count Start Value Register	0x00000000
0x4100E010	DMA2_XMOD	DMA2 Inner Loop Address Increment Register	0x00000000
0x4100E014	DMA2_YCNT	DMA2 Outer Loop Count Start Value (2D only) Register	0x00000000
0x4100E018	DMA2_YMOD	DMA2 Outer Loop Address Increment (2D only) Register	0x00000000
0x4100E024	DMA2_DSCPTR_CUR	DMA2 Current Descriptor Pointer Register	0x00000000
0x4100E028	DMA2_DSCPTR_PRV	DMA2 Previous Initial Descriptor Pointer Register	0x00000000
0x4100E02C	DMA2_ADDR_CUR	DMA2 Current Address Register	0x00000000
0x4100E030	DMA2_STAT	DMA2 Status Register	0x00006000
0x4100E034	DMA2_XCNT_CUR	DMA2 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x4100E038	DMA2_YCNT_CUR	DMA2 Current Row Count (2D only) Register	0x00000000

Table 48-21: CM41X_M4 DMA3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100F000	DMA3_DSCPTR_NXT	DMA3 Pointer to Next Initial Descriptor Register	0x00000000

Table 48-21: CM41X_M4 DMA3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100F004	DMA3_ADDRSTART	DMA3 Start Address of Current Buffer Register	0x00000000
0x4100F008	DMA3_CFG	DMA3 Configuration Register	0x00000000
0x4100F00C	DMA3_XCNT	DMA3 Inner Loop Count Start Value Register	0x00000000
0x4100F010	DMA3_XMOD	DMA3 Inner Loop Address Increment Register	0x00000000
0x4100F014	DMA3_YCNT	DMA3 Outer Loop Count Start Value (2D only) Register	0x00000000
0x4100F018	DMA3_YMOD	DMA3 Outer Loop Address Increment (2D only) Register	0x00000000
0x4100F024	DMA3_DSCPTR_CUR	DMA3 Current Descriptor Pointer Register	0x00000000
0x4100F028	DMA3_DSCPTR_PRV	DMA3 Previous Initial Descriptor Pointer Register	0x00000000
0x4100F02C	DMA3_ADDR_CUR	DMA3 Current Address Register	0x00000000
0x4100F030	DMA3_STAT	DMA3 Status Register	0x00006000
0x4100F034	DMA3_XCNT_CUR	DMA3 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x4100F038	DMA3_YCNT_CUR	DMA3 Current Row Count (2D only) Register	0x00000000

Table 48-22: CM41X_M4 DMA4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40060000	DMA4_DSCPTR_NXT	DMA4 Pointer to Next Initial Descriptor Register	0x00000000
0x40060004	DMA4_ADDRSTART	DMA4 Start Address of Current Buffer Register	0x00000000
0x40060008	DMA4_CFG	DMA4 Configuration Register	0x00000000
0x4006000C	DMA4_XCNT	DMA4 Inner Loop Count Start Value Register	0x00000000
0x40060010	DMA4_XMOD	DMA4 Inner Loop Address Increment Register	0x00000000
0x40060014	DMA4_YCNT	DMA4 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40060018	DMA4_YMOD	DMA4 Outer Loop Address Increment (2D only) Register	0x00000000
0x40060024	DMA4_DSCPTR_CUR	DMA4 Current Descriptor Pointer Register	0x00000000
0x40060028	DMA4_DSCPTR_PRV	DMA4 Previous Initial Descriptor Pointer Register	0x00000000
0x4006002C	DMA4_ADDR_CUR	DMA4 Current Address Register	0x00000000
0x40060030	DMA4_STAT	DMA4 Status Register	0x00006000
0x40060034	DMA4_XCNT_CUR	DMA4 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40060038	DMA4_YCNT_CUR	DMA4 Current Row Count (2D only) Register	0x00000000

Table 48-23: CM41X_M4 DMA5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40061000	DMA5_DSCPTR_NXT	DMA5 Pointer to Next Initial Descriptor Register	0x00000000
0x40061004	DMA5_ADDRSTART	DMA5 Start Address of Current Buffer Register	0x00000000
0x40061008	DMA5_CFG	DMA5 Configuration Register	0x00000000
0x4006100C	DMA5_XCNT	DMA5 Inner Loop Count Start Value Register	0x00000000
0x40061010	DMA5_XMOD	DMA5 Inner Loop Address Increment Register	0x00000000
0x40061014	DMA5_YCNT	DMA5 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40061018	DMA5_YMOD	DMA5 Outer Loop Address Increment (2D only) Register	0x00000000
0x40061024	DMA5_DSCPTR_CUR	DMA5 Current Descriptor Pointer Register	0x00000000
0x40061028	DMA5_DSCPTR_PRV	DMA5 Previous Initial Descriptor Pointer Register	0x00000000
0x4006102C	DMA5_ADDR_CUR	DMA5 Current Address Register	0x00000000
0x40061030	DMA5_STAT	DMA5 Status Register	0x00006000
0x40061034	DMA5_XCNT_CUR	DMA5 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40061038	DMA5_YCNT_CUR	DMA5 Current Row Count (2D only) Register	0x00000000

Table 48-24: CM41X_M4 DMA6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40062000	DMA6_DSCPTR_NXT	DMA6 Pointer to Next Initial Descriptor Register	0x00000000
0x40062004	DMA6_ADDRSTART	DMA6 Start Address of Current Buffer Register	0x00000000
0x40062008	DMA6_CFG	DMA6 Configuration Register	0x00000000
0x4006200C	DMA6_XCNT	DMA6 Inner Loop Count Start Value Register	0x00000000
0x40062010	DMA6_XMOD	DMA6 Inner Loop Address Increment Register	0x00000000
0x40062014	DMA6_YCNT	DMA6 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40062018	DMA6_YMOD	DMA6 Outer Loop Address Increment (2D only) Register	0x00000000
0x40062024	DMA6_DSCPTR_CUR	DMA6 Current Descriptor Pointer Register	0x00000000
0x40062028	DMA6_DSCPTR_PRV	DMA6 Previous Initial Descriptor Pointer Register	0x00000000
0x4006202C	DMA6_ADDR_CUR	DMA6 Current Address Register	0x00000000
0x40062030	DMA6_STAT	DMA6 Status Register	0x00006000
0x40062034	DMA6_XCNT_CUR	DMA6 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40062038	DMA6_YCNT_CUR	DMA6 Current Row Count (2D only) Register	0x00000000

Table 48-25: CM41X_M4 DMA7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40063000	DMA7_DSCPTR_NXT	DMA7 Pointer to Next Initial Descriptor Register	0x00000000
0x40063004	DMA7_ADDRSTART	DMA7 Start Address of Current Buffer Register	0x00000000
0x40063008	DMA7_CFG	DMA7 Configuration Register	0x00000000
0x4006300C	DMA7_XCNT	DMA7 Inner Loop Count Start Value Register	0x00000000
0x40063010	DMA7_XMOD	DMA7 Inner Loop Address Increment Register	0x00000000
0x40063014	DMA7_YCNT	DMA7 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40063018	DMA7_YMOD	DMA7 Outer Loop Address Increment (2D only) Register	0x00000000
0x40063024	DMA7_DSCPTR_CUR	DMA7 Current Descriptor Pointer Register	0x00000000
0x40063028	DMA7_DSCPTR_PRV	DMA7 Previous Initial Descriptor Pointer Register	0x00000000
0x4006302C	DMA7_ADDR_CUR	DMA7 Current Address Register	0x00000000
0x40063030	DMA7_STAT	DMA7 Status Register	0x00006000
0x40063034	DMA7_XCNT_CUR	DMA7 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40063038	DMA7_YCNT_CUR	DMA7 Current Row Count (2D only) Register	0x00000000

Table 48-26: CM41X_M4 DMA8 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40064000	DMA8_DSCPTR_NXT	DMA8 Pointer to Next Initial Descriptor Register	0x00000000
0x40064004	DMA8_ADDRSTART	DMA8 Start Address of Current Buffer Register	0x00000000
0x40064008	DMA8_CFG	DMA8 Configuration Register	0x00000000
0x4006400C	DMA8_XCNT	DMA8 Inner Loop Count Start Value Register	0x00000000
0x40064010	DMA8_XMOD	DMA8 Inner Loop Address Increment Register	0x00000000
0x40064014	DMA8_YCNT	DMA8 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40064018	DMA8_YMOD	DMA8 Outer Loop Address Increment (2D only) Register	0x00000000
0x40064024	DMA8_DSCPTR_CUR	DMA8 Current Descriptor Pointer Register	0x00000000
0x40064028	DMA8_DSCPTR_PRV	DMA8 Previous Initial Descriptor Pointer Register	0x00000000
0x4006402C	DMA8_ADDR_CUR	DMA8 Current Address Register	0x00000000
0x40064030	DMA8_STAT	DMA8 Status Register	0x00006000
0x40064034	DMA8_XCNT_CUR	DMA8 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40064038	DMA8_YCNT_CUR	DMA8 Current Row Count (2D only) Register	0x00000000

Table 48-27: CM41X_M4 DMA9 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40065000	DMA9_DSCPTR_NXT	DMA9 Pointer to Next Initial Descriptor Register	0x00000000
0x40065004	DMA9_ADDRSTART	DMA9 Start Address of Current Buffer Register	0x00000000
0x40065008	DMA9_CFG	DMA9 Configuration Register	0x00000000
0x4006500C	DMA9_XCNT	DMA9 Inner Loop Count Start Value Register	0x00000000
0x40065010	DMA9_XMOD	DMA9 Inner Loop Address Increment Register	0x00000000
0x40065014	DMA9_YCNT	DMA9 Outer Loop Count Start Value (2D only) Register	0x00000000
0x40065018	DMA9_YMOD	DMA9 Outer Loop Address Increment (2D only) Register	0x00000000
0x40065024	DMA9_DSCPTR_CUR	DMA9 Current Descriptor Pointer Register	0x00000000
0x40065028	DMA9_DSCPTR_PRV	DMA9 Previous Initial Descriptor Pointer Register	0x00000000
0x4006502C	DMA9_ADDR_CUR	DMA9 Current Address Register	0x00000000
0x40065030	DMA9_STAT	DMA9 Status Register	0x00006000
0x40065034	DMA9_XCNT_CUR	DMA9 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x40065038	DMA9_YCNT_CUR	DMA9 Current Row Count (2D only) Register	0x00000000

Table 48-28: CM41X_M4 DPM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40013000	DPM0_CTL	DPM0 Control Register	0x00000000
0x40013004	DPM0_STAT	DPM0 Status Register	0x00000001
0x4001301C	DPM0_WAKE_EN	DPM0 Wakeup Enable Register	0x00000000
0x40013020	DPM0_WAKE_POL	DPM0 Wakeup Polarity Register	0x00000000
0x40013024	DPM0_WAKE_STAT	DPM0 Wakeup Status Register	0x00000000
0x40013070	DPM0_PER_DIS0	DPM0 Peripherals Disable Register 0	0x00000000
0x40013084	DPM0_REVID	DPM0 Revision ID	0x00000020

Table 48-29: CM41X_M4 EMUID0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40002020	EMUID0_ADIID	EMUID0 Analog Devices Identification	0x00004144
0x40002024	EMUID0_CHIPID	EMUID0 Chip Identification	0x00000402

Table 48-30: CM41X_M4 FFTB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40048000	FFTB0_CTL	FFTB0 FFTB Control Register	0x00500000
0x40048004	FFTB0_STAT	FFTB0 FFTB Status Register	0x00000000
0x40048008	FFTB0_FMTCTL	FFTB0 Format Converter Control Register	0x0000FFF0
0x4004800C	FFTB0_INOFST	FFTB0 Input Offset Register	0x00000000
0x40048010	FFTB0_COMB	FFTB0 FFTB Input Comb Filter Control Register	0x00000000
0x40048014	FFTB0_DMABASE	FFTB0 DMA Output Base Address Register	0x00000000
0x40048018	FFTB0_DMAWR	FFTB0 DMA Output Write Address Register	0x00000000
0x40048020	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048024	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048028	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x4004802C	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048030	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048034	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048038	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x4004803C	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048040	FFTB0_MAGSEL[n]	FFTB0 Magnitude Pin Select Registers	0xFFFFFFFF
0x40048070	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048074	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048078	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x4004807C	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048080	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048084	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048088	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x4004808C	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x40048090	FFTB0_LIMSTAT[n]	FFTB0 Limit Status Registers	0x00000000
0x400480C0	FFTB0_MAXMAG	FFTB0 Maximum Magnitude Register	0x00000000
0x400480C4	FFTB0_MINMAG	FFTB0 Minimum Magnitude Register	0x00000000

Table 48-31: CM41X_M4 FLC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
	FLC0_CTL	FLC0 Control Register	0x00000000

Table 48-31: CM41X_M4 FLC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xF8010000			
0xF8010004	FLC0_CMD	FLC0 Command Register	0x00000000
0xF8010008	FLC0_IMSK	FLC0 Interrupt Enable Register	0x00000000
0xF801000C	FLC0_ADDR	FLC0 Command Address Register	0x00000000
0xF8010010	FLC0_DATA0	FLC0 Command Data Register 0	0x00000000
0xF8010014	FLC0_DATA1	FLC0 Command Data Register 1	0x00000000
0xF8010018	FLC0_ECC	FLC0 Command ECC Register	0x00000000
0xF801001C	FLC0_STAT	FLC0 Status Register	0x00000000
0xF8010020	FLC0_CMD_KEY	FLC0 Command Key Register	0x00000000
0xF8010024	FLC0_SBERR_THR	FLC0 ECC Single Bit Error Threshold Register	0x00000000
0xF8010030	FLC0_TIM_PWR	FLC0 Power Timing Register	0x000147D0
0xF8010040	FLC0_PRFCTL	FLC0 Prefetch Control Register	0x0000000D
0xF8010044	FLC0_FILL_CNT	FLC0 Pre-fetcher Fill Count Register	0x00000000
0xF8010048	FLC0_MISS_CNT	FLC0 Pre-fetcher Miss Count Register	0x00000000
0xF801004C	FLC0_REF_CNT	FLC0 Pre-fetcher Reference Count Register	0x00000000
0xF8010054	FLC0_REVID	FLC0 Revision ID Register	0x00000000

Table 48-32: CM41X_M4 FLC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xF8011000	FLC1_CTL	FLC1 Control Register	0x00000000
0xF8011004	FLC1_CMD	FLC1 Command Register	0x00000000
0xF8011008	FLC1_IMSK	FLC1 Interrupt Enable Register	0x00000000
0xF801100C	FLC1_ADDR	FLC1 Command Address Register	0x00000000
0xF8011010	FLC1_DATA0	FLC1 Command Data Register 0	0x00000000
0xF8011014	FLC1_DATA1	FLC1 Command Data Register 1	0x00000000
0xF8011018	FLC1_ECC	FLC1 Command ECC Register	0x00000000
0xF801101C	FLC1_STAT	FLC1 Status Register	0x00000000
0xF8011020	FLC1_CMD_KEY	FLC1 Command Key Register	0x00000000
0xF8011024	FLC1_SBERR_THR	FLC1 ECC Single Bit Error Threshold Register	0x00000000
0xF8011030	FLC1_TIM_PWR	FLC1 Power Timing Register	0x000147D0

Table 48-32: CM41X_M4 FLC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xF8011040	FLC1_PRFCTL	FLC1 Prefetch Control Register	0x0000000D
0xF8011044	FLC1_FILL_CNT	FLC1 Pre-fetcher Fill Count Register	0x00000000
0xF8011048	FLC1_MISS_CNT	FLC1 Pre-fetcher Miss Count Register	0x00000000
0xF801104C	FLC1_REF_CNT	FLC1 Pre-fetcher Reference Count Register	0x00000000
0xF8011054	FLC1_REVID	FLC1 Revision ID Register	0x00000000

Table 48-33: CM41X_M4 HAE0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400500A0	HAE0_RUN	HAE0 Run Register	0x00000000
0x40050B00	HAE0_CFG0	HAE0 Configuration 0 Register	0x00000000
0x40050B04	HAE0_CFG1	HAE0 Configuration 1 Register	0x00000000
0x40050B08	HAE0_CFG2	HAE0 Configuration 2 Register	0x00000000
0x40050B0C	HAE0_CFG3	HAE0 Configuration 3 Register	0x00000000
0x40050B20	HAE0_STAT	HAE0 Status Register	0x00000000
0x40050B40	HAE0_ISAMPLE	HAE0 I (Current) Sample Register	0x00000000
0x40050B44	HAE0_VSAMPLE	HAE0 V (Voltage) Sample Register	0x00000000
0x40050B80	HAE0_IWAVEFORM	HAE0 I (Current) Waveform Register	0x00000000
0x40050B84	HAE0_VWAVEFORM	HAE0 V (Voltage) Waveform Register	0x00000000
0x40051E00	HAE0_CFG4	HAE0 Configuration 4 Register	0x00000000
0x40051E04	HAE0_DIDT_GAIN	HAE0 DIDT Gain Register	0x00000000
0x40051E08	HAE0_DIDT_COEF	HAE0 DIDT Coefficient Register	0x00000000
0x40051E10	HAE0_VLEVEL	HAE0 Voltage Level Register	0x00000000
0x40051E2C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E30	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E34	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E38	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E3C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E40	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E44	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E48	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000

Table 48-33: CM41X_M4 HAE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40051E4C	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E50	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E54	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000
0x40051E58	HAE0_H[nn]_INDX	HAE0 Harmonic n Index Register	0x00000000

Table 48-34: CM41X_M4 LBA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4000F004	LBA0_CLK_CTL	LBA0 Logic Block Array Clock Divisor Register	0x00000000
0x4000F008	LBA0_DIR	LBA0 Logic Block Array Pin Direction Register	0x00000000
0x4000F00C	LBA0_DIN	LBA0 Logic Block Array Data Input Register	0x00000000
0x4000F010	LBA0_DOUT	LBA0 Logic Block Array Data Output Register	0x00000000
0x4000F014	LBA0_CFG	LBA0 Logic Block Array Configuration Register	0x00000000
0x4000F018	LBA0_SYNC_CTL	LBA0 Logic Block Array GPIO Input Synchronization Control Register	0x00000000
0x4000F020	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F024	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F028	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F02C	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F030	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F034	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F038	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F03C	LBA0_BLK[n]_CTL	LBA0 Logic Block Control Register	0x00000000
0x4000F100	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F104	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F108	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F10C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F110	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F114	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F118	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F11C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F120	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000

Table 48-34: CM41X_M4 LBA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4000F124	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F128	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F12C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F130	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F134	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F138	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F13C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F140	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F144	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F148	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F14C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F150	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F154	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F158	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F15C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F160	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F164	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F168	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F16C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F170	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F174	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F178	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F17C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F180	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F184	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F188	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F18C	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F190	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F194	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F198	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F19C	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000

Table 48-34: CM41X_M4 LBA0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4000F1A0	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F1A4	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F1A8	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F1AC	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F1B0	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F1B4	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F1B8	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F1BC	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F1C0	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F1C4	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F1C8	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F1CC	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F1D0	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F1D4	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F1D8	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F1DC	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000
0x4000F1E0	LBA0_BLK[n]_FN0	LBA0 Logic Block Function 0 Register	0x00000000
0x4000F1E4	LBA0_BLK[n]_FN1	LBA0 Logic Block Function 1 Register	0x00000000
0x4000F1E8	LBA0_BLK[n]_FN2	LBA0 Logic Block Function 2 Register	0x00000000
0x4000F1EC	LBA0_BLK[n]_FN3	LBA0 Logic Block Function 3 Register	0x00000000
0x4000F1F0	LBA0_BLK[n]_FN4	LBA0 Logic Block Function 4 Register	0x00000000
0x4000F1F4	LBA0_BLK[n]_FN5	LBA0 Logic Block Function 5 Register	0x00000000
0x4000F1F8	LBA0_BLK[n]_FN6	LBA0 Logic Block Function 6 Register	0x00000000
0x4000F1FC	LBA0_BLK[n]_FN7	LBA0 Logic Block Function 7 Register	0x00000000

Table 48-35: CM41X_M4 SCS0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xE000E004	SCS0_ICTR	SCS0 Interrupt Control Type Register	0x00000000
0xE000E008	SCS0_ACTLR	SCS0 Auxiliary Control Register	0x00000000
0xE000E010	SCS0_STCSR	SCS0 SysTick Control and Status Register	0x00000000

Table 48-35: CM41X_M4 SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE000E014	SCS0_STRVR	SCS0 SysTick Reload Value Register	0x00000000
0xE000E018	SCS0_STCVR	SCS0 SysTick Current Value Register	0x00000000
0xE000E01C	SCS0_STCR	SCS0 SysTick Calibration Value Register	0x00000000
0xE000E100	SCS0_NVIC_ISER[n]	SCS0 Irq Set Enable Register n	0x00000000
0xE000E104	SCS0_NVIC_ISER[n]	SCS0 Irq Set Enable Register n	0x00000000
0xE000E108	SCS0_NVIC_ISER[n]	SCS0 Irq Set Enable Register n	0x00000000
0xE000E10C	SCS0_NVIC_ISER[n]	SCS0 Irq Set Enable Register n	0x00000000
0xE000E110	SCS0_NVIC_ISER[n]	SCS0 Irq Set Enable Register n	0x00000000
0xE000E114	SCS0_NVIC_ISER[n]	SCS0 Irq Set Enable Register n	0x00000000
0xE000E180	SCS0_NVIC_ICER[n]	SCS0 Irq Clear Enable Register n	0x00000000
0xE000E184	SCS0_NVIC_ICER[n]	SCS0 Irq Clear Enable Register n	0x00000000
0xE000E188	SCS0_NVIC_ICER[n]	SCS0 Irq Clear Enable Register n	0x00000000
0xE000E18C	SCS0_NVIC_ICER[n]	SCS0 Irq Clear Enable Register n	0x00000000
0xE000E190	SCS0_NVIC_ICER[n]	SCS0 Irq Clear Enable Register n	0x00000000
0xE000E194	SCS0_NVIC_ICER[n]	SCS0 Irq Clear Enable Register n	0x00000000
0xE000E200	SCS0_NVIC_ISPR[n]	SCS0 Irq Set Pending Register n	0x00000000
0xE000E204	SCS0_NVIC_ISPR[n]	SCS0 Irq Set Pending Register n	0x00000000
0xE000E208	SCS0_NVIC_ISPR[n]	SCS0 Irq Set Pending Register n	0x00000000
0xE000E20C	SCS0_NVIC_ISPR[n]	SCS0 Irq Set Pending Register n	0x00000000
0xE000E210	SCS0_NVIC_ISPR[n]	SCS0 Irq Set Pending Register n	0x00000000
0xE000E214	SCS0_NVIC_ISPR[n]	SCS0 Irq Set Pending Register n	0x00000000
0xE000E280	SCS0_NVIC_ICPR[n]	SCS0 Irq Clear Pending Register n	0x00000000
0xE000E284	SCS0_NVIC_ICPR[n]	SCS0 Irq Clear Pending Register n	0x00000000
0xE000E288	SCS0_NVIC_ICPR[n]	SCS0 Irq Clear Pending Register n	0x00000000
0xE000E28C	SCS0_NVIC_ICPR[n]	SCS0 Irq Clear Pending Register n	0x00000000
0xE000E290	SCS0_NVIC_ICPR[n]	SCS0 Irq Clear Pending Register n	0x00000000
0xE000E294	SCS0_NVIC_ICPR[n]	SCS0 Irq Clear Pending Register n	0x00000000
0xE000E300	SCS0_NVIC_IABR[n]	SCS0 Irq Active Bit Register n	0x00000000
0xE000E304	SCS0_NVIC_IABR[n]	SCS0 Irq Active Bit Register n	0x00000000
0xE000E308	SCS0_NVIC_IABR[n]	SCS0 Irq Active Bit Register n	0x00000000
0xE000E30C	SCS0_NVIC_IABR[n]	SCS0 Irq Active Bit Register n	0x00000000

Table 48-35: CM41X_M4 SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE000E310	SCS0_NVIC_IABR[n]	SCS0 Irq Active Bit Register n	0x00000000
0xE000E314	SCS0_NVIC_IABR[n]	SCS0 Irq Active Bit Register n	0x00000000
0xE000E400	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E404	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E408	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E40C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E410	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E414	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E418	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E41C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E420	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E424	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E428	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E42C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E430	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E434	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E438	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E43C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E440	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E444	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E448	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E44C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E450	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E454	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E458	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E45C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E460	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E464	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E468	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E46C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E470	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000

Table 48-35: CM41X_M4 SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE000E474	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E478	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E47C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E480	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E484	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E488	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E48C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E490	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E494	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E498	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E49C	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000E4A0	SCS0_NVIC_IPR[nn]	SCS0 Irq Priority Register nn	0x00000000
0xE000ED00	SCS0_CPUID	SCS0 CPUID Base Register	0x410FC241
0xE000ED04	SCS0_ICSR	SCS0 Interrupt Control State Register	0x00000000
0xE000ED08	SCS0_VTOR	SCS0 Vector Table Offset Register	0x00000000
0xE000ED0C	SCS0_AIRCR	SCS0 Application Interrupt/Reset Control Register	0xFA050000
0xE000ED10	SCS0_SCR	SCS0 System Control Register	0x00000000
0xE000ED14	SCS0_CCR	SCS0 Configuration Control Register	0x00000200
0xE000ED18	SCS0_SHPR1	SCS0 System Handlers 4-7 Priority Register	0x00000000
0xE000ED1C	SCS0_SHPR2	SCS0 System Handlers 8-11 Priority Register	0x00000000
0xE000ED20	SCS0_SHPR3	SCS0 System Handlers 12-15 Priority Register	0x00000000
0xE000ED24	SCS0_SHCSR	SCS0 System Handler Control and State Register	0x00000000
0xE000ED28	SCS0_CFSR	SCS0 Configurable Fault Status Registers	0x00000000
0xE000ED2C	SCS0_HFSR	SCS0 Hard Fault Status Register	0x00000000
0xE000ED30	SCS0_DFSR	SCS0 Debug Fault Status Register	0x00000000
0xE000ED34	SCS0_MMFAR	SCS0 Mem Manage Fault Address Register	0x00000000
0xE000ED38	SCS0_BFAR	SCS0 Bus Fault Address Register	0x00000000
0xE000ED3C	SCS0_AFSR	SCS0 Auxiliary Fault Status Register	0x00000000
0xE000ED40	SCS0_ID_PFR0	SCS0 Processor Feature register0	0x00000030
0xE000ED44	SCS0_ID_PFR1	SCS0 Processor Feature register1	0x00000200
0xE000ED48	SCS0_ID_DFR0	SCS0 Debug Feature register0	0x00100000

Table 48-35: CM41X_M4 SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE000ED4C	SCS0_ID_AFR0	SCS0 Auxiliary Feature register0	0x00000000
0xE000ED50	SCS0_ID_MMFR0	SCS0 Memory Model Feature register0	0x00100030
0xE000ED54	SCS0_ID_MMFR1	SCS0 Memory Model Feature register1	0x00000000
0xE000ED58	SCS0_ID_MMFR2	SCS0 Memory Model Feature register2	0x00000000
0xE000ED5C	SCS0_ID_MMFR3	SCS0 Memory Model Feature register3	0x00000000
0xE000ED60	SCS0_ISAR0	SCS0 ISA Feature register0	0x01141110
0xE000ED64	SCS0_ID_ISAR1	SCS0 ISA Feature register1	0x02112000
0xE000ED68	SCS0_ID_ISAR2	SCS0 ISA Feature register2	0x20232231
0xE000ED6C	SCS0_ID_ISAR3	SCS0 ISA Feature register3	0x01111131
0xE000ED70	SCS0_ID_ISAR4	SCS0 ISA Feature register4	0x01310102
0xE000ED88	SCS0_CPACR	SCS0 Coprocessor Access Control Register	0x00000000
0xE000ED90	SCS0_MPU_TYPE	SCS0 MPU Type Register	0x00000800
0xE000ED94	SCS0_MPU_CTRL	SCS0 MPU Control Register	0x00000000
0xE000ED98	SCS0_MPU_RNR	SCS0 MPU Region Number Register	0x00000000
0xE000ED9C	SCS0_MPU_RBAR	SCS0 MPU Region Base Address Register	0x00000000
0xE000EDA0	SCS0_MPU_RASR	SCS0 MPU Region Attribute and Size Register	0x00000000
0xE000EDA4	SCS0_MPU_RBAR_A1	SCS0 MPU Alias 1 Region Base Address register	0x00000000
0xE000EDA8	SCS0_MPU_RASR_A1	SCS0 MPU Alias 1 Region Attribute and Size register	0x00000000
0xE000EDAC	SCS0_MPU_RBAR_A2	SCS0 MPU Alias 2 Region Base Address register	0x00000000
0xE000EDB0	SCS0_MPU_RASR_A2	SCS0 MPU Alias 2 Region Attribute and Size register	0x00000000
0xE000EDB4	SCS0_MPU_RBAR_A3	SCS0 MPU Alias 3 Region Base Address register	0x00000000
0xE000EDB8	SCS0_MPU_RASR_A3	SCS0 MPU Alias 3 Region Attribute and Size register	0x00000000
0xE000EDF0	SCS0_DHCSR	SCS0 Debug Halting Control and Status Register	0x00000000
0xE000EDF4	SCS0_DCRSR	SCS0 Deubg Core Register Selector Register	0x00000000
0xE000EDF8	SCS0_DCRDR	SCS0 Debug Core Register Data Register	0x00000000
0xE000EDFC	SCS0_DEMCR	SCS0 Debug Exception and Monitor Control Register	0x00000000
0xE000EF00	SCS0_STIR	SCS0 Software Trigger Interrupt Register	0x00000000
0xE000EF34	SCS0_FPCCR	SCS0 Floating Point Context Control Register	0xC0000000
0xE000EF38	SCS0_FPCAR	SCS0 Floating-Point Context Address Register	0x00000000
0xE000EF3C	SCS0_FPDSCR	SCS0 Floating Point Default Status Control Register	0x00000000
0xE000EF40	SCS0_MVFR0	SCS0 Media and FP Feature Register 0 (MVFR0)	0x10110021

Table 48-35: CM41X_M4 SCS0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE000EF44	SCS0_MVFR1	SCS0 Media and FP Feature Register 1 (MVFR1)	0x11000011
0xE000EFD0	SCS0_PID4	SCS0 Peripheral identification register (PID4)	0x00000000
0xE000EFD4	SCS0_PID5	SCS0 Peripheral identification register (PID5)	0x00000000
0xE000EFD8	SCS0_PID6	SCS0 Peripheral identification register (PID6)	0x00000000
0xE000EFDC	SCS0_PID7	SCS0 Peripheral identification register (PID7)	0x00000000
0xE000EFE0	SCS0_PID0	SCS0 Peripheral identification register Bits 7:0 (PID0)	0x00000000
0xE000EFE4	SCS0_PID1	SCS0 Peripheral identification register Bits 15:8 (PID1)	0x00000000
0xE000EFE8	SCS0_PID2	SCS0 Peripheral identification register Bits 23:16 (PID2)	0x00000000
0xE000EFEC	SCS0_PID3	SCS0 Peripheral identification register Bits 31:24 (PID3)	0x00000000
0xE000EFF0	SCS0_CID0	SCS0 Component identification register Bits 7:0 (CID0)	0x00000000
0xE000EFF4	SCS0_CID1	SCS0 Component identification register Bits 15:8 (CID1)	0x00000000
0xE000EFF8	SCS0_CID2	SCS0 Component identification register Bits 23:16 (CID2)	0x00000000
0xE000EFFC	SCS0_CID3	SCS0 Component identification register Bits 31:24 (CID3)	0x00000000

Table 48-36: CM41X_M4 M4P MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xF8008020	M4P_SRAM_CFG	M4P SRAM Configuration Register	0x08000002
0xF800802C	M4P_BUSFLT	M4P Bus Fault Error Information Register	0x00000000
0xF8008030	M4P_STCALIB	M4P SysTick Calibration Register	0x801E8480
0xF800805C	M4P_SRAM_INITDONE	M4P SRAM Initialization Done Status Register	0x00000000
0xF8008060	M4P_SRAM_EEADDR_CORE	M4P SRAM ECC Error Address (Core) Register	0x00000000
0xF8008064	M4P_SRAM_EEADDR_DMA	M4P SRAM ECC Error Address (DMA) Register	0x00000000
0xF8008068	M4P_SRAM_RFRPER	M4P SRAM Refresh Period Register	0x00000000
0xF800806C	M4P_SRAM_RFRADDR	M4P SRAM Refresh Address	0x00000000
0xF8008070	M4P_BROMERR	M4P Boot ROM Error Register	0x00000000

Table 48-37: CM41X_M4 MATH0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xF8009000	MATH0_CTL	MATH0 Math Unit Interrupt Enable Control	0x00000000
0xF8009004	MATH0_EXPF	MATH0 exp(x) Function Register	0x00000000
0xF800900C	MATH0_EXP2F	MATH0 exp2(x) Function Register	0x00000000
0xF8009014	MATH0_RECIPF	MATH0 Reciprocal(x) or 1/x Function Register	0x00000000
0xF800901C	MATH0_SQRTF	MATH0 Square Root or sqrt(x) Function	0x00000000
0xF8009024	MATH0_LNF	MATH0 ln(x) Function Register	0x00000000
0xF800902C	MATH0_LOG2F	MATH0 log2(x) Function Register	0x00000000
0xF8009034	MATH0_SINF	MATH0 sin(x) Function Register	0x00000000
0xF800903C	MATH0_COSF	MATH0 cos(x) Function Register	0x00000000
0xF8009044	MATH0_TANF	MATH0 tan(x) Function Register	0x00000000
0xF800904C	MATH0_ASINF	MATH0 arcsin(x) Function Register	0x00000000
0xF8009054	MATH0_ACOSF	MATH0 arccos(x) Function Register	0x00000000
0xF800905C	MATH0_ATANF	MATH0 arctan(x) Function Register	0x00000000
0xF8009080	MATH0_ATAN2F_Y	MATH0 arctan(y/x) Function y Operand Register	0x00000000
0xF8009084	MATH0_ATAN2F_X	MATH0 arctan(y/x) Function x Operand Register	0x00000000
0xF8009088	MATH0_HYPOTF_Y	MATH0 hypot(x,y) Function y Operand Register	0x00000000
0xF800908C	MATH0_HYPOTF_X	MATH0 hypot(x,y) Function x Operand Register	0x00000000
0xF8009090	MATH0_RTOPF_Y	MATH0 Rectangular to Polar Function y Operand Register	0x00000000
0xF8009094	MATH0_RTOPF_X	MATH0 Rectangular to Polar Function x Operand Register	0x00000000
0xF8009098	MATH0_PTORF_R	MATH0 Polar to Rectangular Function r Operand Register	0x00000000
0xF800909C	MATH0_PTORF_A	MATH0 Polar to Rectangular Function a Operand Register	0x00000000
0xF8009200	MATH0_GY	MATH0 Generic Y Function Register (GY)	0x00000000
0xF8009204	MATH0_GX	MATH0 Generic X Function Register (GX)	0x00000000
0xF8009208	MATH0_RES1	MATH0 Math Unit Function Result 1	0x00000000
0xF800920C	MATH0_RES2	MATH0 Math Unit Function Result 2	0x00000000
0xF8009210	MATH0_FSTAT	MATH0 Math Unit Function Status Register	0x001F0000
0xF8009214	MATH0_ISTAT	MATH0 Math Unit Interrupt Register	0x00000000
0xF8009218	MATH0_SSTAT	MATH0 Math Unit Function State Status Register	0x00000000

Table 48-38: CM41X_M4 MBOX0_PORT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400D1000	MBOX0_PORT1_CTL	MBOX0_PORT1 Port 1 Control Register	0x00000000
0x400D1004	MBOX0_PORT1_FORCE-IRQ	MBOX0_PORT1 Port 1 Force IRQ Register	0x00000000
0x400D1008	MBOX0_PORT1_STAT	MBOX0_PORT1 Port 1 Status Register	0x00000000
0x400D100C	MBOX0_PORT1_ESTAT	MBOX0_PORT1 Port 1 ECC Status Register	0x00000000
0x400D1010	MBOX0_PORT1_RFRPER	MBOX0_PORT1 Auto-refresh Period Register	0x00000000
0x400D1014	MBOX0_PORT1_RFRADDR	MBOX0_PORT1 Auto-refresh Address Register	0x00000000
0x400D1018	MBOX0_PORT1_RFRCNT	MBOX0_PORT1 Auto-refresh Counter Register	0x00000001

Table 48-39: CM41X_M4 OCU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40015000	OCU0_CTL	OCU0 Control Register	0x00000000
0x40015004	OCU0_STAT	OCU0 Status Register	0x00000000
0x40015008	OCU0_CLKCNT	OCU0 Frequency Measured Count Register	0x00000000
0x4001500C	OCU0_REFCNT	OCU0 Reference Count Register	0x00000000
0x40015010	OCU0_MINCNT	OCU0 Minimum Limit Register	0x00000000
0x40015014	OCU0_MAXCNT	OCU0 Maximum Count Register	0x00000000
0x40015018	OCU0_LMONCNT	OCU0 LFO Monitor Reference Count Register	0x00000000
0x4001501C	OCU0_CMONCNT	OCU0 CLKIN Monitor Reference Count Register	0x00000000

Table 48-40: CM41X_M4 PINT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41002000	PINT0_MSK_SET	PINT0 PINT Mask Set Register	0x00000000
0x41002004	PINT0_MSK_CLR	PINT0 PINT Mask Clear Register	0x00000000
0x41002008	PINT0_REQ	PINT0 PINT Request Register	0x00000000
0x4100200C	PINT0_ASSIGN	PINT0 PINT Assign Register	0x00000101
0x41002010	PINT0_EDGE_SET	PINT0 PINT Edge Set Register	0x00000000
0x41002014	PINT0_EDGE_CLR	PINT0 PINT Edge Clear Register	0x00000000
0x41002018	PINT0_INV_SET	PINT0 PINT Invert Set Register	0x00000000
0x4100201C	PINT0_INV_CLR	PINT0 PINT Invert Clear Register	0x00000000

Table 48-40: CM41X_M4 PINT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41002020	PINT0_PINSTATE	PINT0 PINT Pin State Register	0x00000000
0x41002024	PINT0_LATCH	PINT0 PINT Latch Register	0x00000000

Table 48-41: CM41X_M4 PINT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40003000	PINT1_MSK_SET	PINT1 PINT Mask Set Register	0x00000000
0x40003004	PINT1_MSK_CLR	PINT1 PINT Mask Clear Register	0x00000000
0x40003008	PINT1_REQ	PINT1 PINT Request Register	0x00000000
0x4000300C	PINT1_ASSIGN	PINT1 PINT Assign Register	0x00000101
0x40003010	PINT1_EDGE_SET	PINT1 PINT Edge Set Register	0x00000000
0x40003014	PINT1_EDGE_CLR	PINT1 PINT Edge Clear Register	0x00000000
0x40003018	PINT1_INV_SET	PINT1 PINT Invert Set Register	0x00000000
0x4000301C	PINT1_INV_CLR	PINT1 PINT Invert Clear Register	0x00000000
0x40003020	PINT1_PINSTATE	PINT1 PINT Pin State Register	0x00000000
0x40003024	PINT1_LATCH	PINT1 PINT Latch Register	0x00000000

Table 48-42: CM41X_M4 PINT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40004000	PINT2_MSK_SET	PINT2 PINT Mask Set Register	0x00000000
0x40004004	PINT2_MSK_CLR	PINT2 PINT Mask Clear Register	0x00000000
0x40004008	PINT2_REQ	PINT2 PINT Request Register	0x00000000
0x4000400C	PINT2_ASSIGN	PINT2 PINT Assign Register	0x00000101
0x40004010	PINT2_EDGE_SET	PINT2 PINT Edge Set Register	0x00000000
0x40004014	PINT2_EDGE_CLR	PINT2 PINT Edge Clear Register	0x00000000
0x40004018	PINT2_INV_SET	PINT2 PINT Invert Set Register	0x00000000
0x4000401C	PINT2_INV_CLR	PINT2 PINT Invert Clear Register	0x00000000
0x40004020	PINT2_PINSTATE	PINT2 PINT Pin State Register	0x00000000
0x40004024	PINT2_LATCH	PINT2 PINT Latch Register	0x00000000

Table 48-43: CM41X_M4 PINT3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40005000	PINT3_MSK_SET	PINT3 PINT Mask Set Register	0x00000000
0x40005004	PINT3_MSK_CLR	PINT3 PINT Mask Clear Register	0x00000000
0x40005008	PINT3_REQ	PINT3 PINT Request Register	0x00000000
0x4000500C	PINT3_ASSIGN	PINT3 PINT Assign Register	0x00000101
0x40005010	PINT3_EDGE_SET	PINT3 PINT Edge Set Register	0x00000000
0x40005014	PINT3_EDGE_CLR	PINT3 PINT Edge Clear Register	0x00000000
0x40005018	PINT3_INV_SET	PINT3 PINT Invert Set Register	0x00000000
0x4000501C	PINT3_INV_CLR	PINT3 PINT Invert Clear Register	0x00000000
0x40005020	PINT3_PINSTATE	PINT3 PINT Pin State Register	0x00000000
0x40005024	PINT3_LATCH	PINT3 PINT Latch Register	0x00000000

Table 48-44: CM41X_M4 PINT4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40006000	PINT4_MSK_SET	PINT4 PINT Mask Set Register	0x00000000
0x40006004	PINT4_MSK_CLR	PINT4 PINT Mask Clear Register	0x00000000
0x40006008	PINT4_REQ	PINT4 PINT Request Register	0x00000000
0x4000600C	PINT4_ASSIGN	PINT4 PINT Assign Register	0x00000101
0x40006010	PINT4_EDGE_SET	PINT4 PINT Edge Set Register	0x00000000
0x40006014	PINT4_EDGE_CLR	PINT4 PINT Edge Clear Register	0x00000000
0x40006018	PINT4_INV_SET	PINT4 PINT Invert Set Register	0x00000000
0x4000601C	PINT4_INV_CLR	PINT4 PINT Invert Clear Register	0x00000000
0x40006020	PINT4_PINSTATE	PINT4 PINT Pin State Register	0x00000000
0x40006024	PINT4_LATCH	PINT4 PINT Latch Register	0x00000000

Table 48-45: CM41X_M4 PINT5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40007000	PINT5_MSK_SET	PINT5 PINT Mask Set Register	0x00000000
0x40007004	PINT5_MSK_CLR	PINT5 PINT Mask Clear Register	0x00000000
0x40007008	PINT5_REQ	PINT5 PINT Request Register	0x00000000
0x4000700C	PINT5_ASSIGN	PINT5 PINT Assign Register	0x00000101

Table 48-45: CM41X_M4 PINT5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40007010	PINT5_EDGE_SET	PINT5 PINT Edge Set Register	0x00000000
0x40007014	PINT5_EDGE_CLR	PINT5 PINT Edge Clear Register	0x00000000
0x40007018	PINT5_INV_SET	PINT5 PINT Invert Set Register	0x00000000
0x4000701C	PINT5_INV_CLR	PINT5 PINT Invert Clear Register	0x00000000
0x40007020	PINT5_PINSTATE	PINT5 PINT Pin State Register	0x00000000
0x40007024	PINT5_LATCH	PINT5 PINT Latch Register	0x00000000

Table 48-46: CM41X_M4 PORTA MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41003000	PORTA_FER	PORTA Port x Function Enable Register	0x00000000
0x41003004	PORTA_FER_SET	PORTA Port x Function Enable Set Register	0x00000000
0x41003008	PORTA_FER_CLR	PORTA Port x Function Enable Clear Register	0x00000000
0x4100300C	PORTA_DATA	PORTA Port x GPIO Data Register	0x00000000
0x41003010	PORTA_DATA_SET	PORTA Port x GPIO Data Set Register	0x00000000
0x41003014	PORTA_DATA_CLR	PORTA Port x GPIO Data Clear Register	0x00000000
0x41003018	PORTA_DIR	PORTA Port x GPIO Direction Register	0x00000000
0x4100301C	PORTA_DIR_SET	PORTA Port x GPIO Direction Set Register	0x00000000
0x41003020	PORTA_DIR_CLR	PORTA Port x GPIO Direction Clear Register	0x00000000
0x41003024	PORTA_INEN	PORTA Port x GPIO Input Enable Register	0x00000000
0x41003028	PORTA_INEN_SET	PORTA Port x GPIO Input Enable Set Register	0x00000000
0x4100302C	PORTA_INEN_CLR	PORTA Port x GPIO Input Enable Clear Register	0x00000000
0x41003030	PORTA_MUX	PORTA Port x Multiplexer Control Register	0x00000000
0x41003034	PORTA_DATA_TGL	PORTA Port x GPIO Output Toggle Register	0x00000000
0x41003038	PORTA_POL	PORTA Port x GPIO Polarity Invert Register	0x00000000
0x4100303C	PORTA_POL_SET	PORTA Port x GPIO Polarity Invert Set Register	0x00000000
0x41003040	PORTA_POL_CLR	PORTA Port x GPIO Polarity Invert Clear Register	0x00000000
0x41003044	PORTA_LOCK	PORTA Port x GPIO Lock Register	0x00000000
0x41003048	PORTA_TRIG_TGL	PORTA Port x GPIO Trigger Toggle Register	0x00000000

Table 48-47: CM41X_M4 PORTB MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40008000	PORTB_FER	PORTB Port x Function Enable Register	0x00000000
0x40008004	PORTB_FER_SET	PORTB Port x Function Enable Set Register	0x00000000
0x40008008	PORTB_FER_CLR	PORTB Port x Function Enable Clear Register	0x00000000
0x4000800C	PORTB_DATA	PORTB Port x GPIO Data Register	0x00000000
0x40008010	PORTB_DATA_SET	PORTB Port x GPIO Data Set Register	0x00000000
0x40008014	PORTB_DATA_CLR	PORTB Port x GPIO Data Clear Register	0x00000000
0x40008018	PORTB_DIR	PORTB Port x GPIO Direction Register	0x00000000
0x4000801C	PORTB_DIR_SET	PORTB Port x GPIO Direction Set Register	0x00000000
0x40008020	PORTB_DIR_CLR	PORTB Port x GPIO Direction Clear Register	0x00000000
0x40008024	PORTB_INEN	PORTB Port x GPIO Input Enable Register	0x00000000
0x40008028	PORTB_INEN_SET	PORTB Port x GPIO Input Enable Set Register	0x00000000
0x4000802C	PORTB_INEN_CLR	PORTB Port x GPIO Input Enable Clear Register	0x00000000
0x40008030	PORTB_MUX	PORTB Port x Multiplexer Control Register	0x00000000
0x40008034	PORTB_DATA_TGL	PORTB Port x GPIO Output Toggle Register	0x00000000
0x40008038	PORTB_POL	PORTB Port x GPIO Polarity Invert Register	0x00000000
0x4000803C	PORTB_POL_SET	PORTB Port x GPIO Polarity Invert Set Register	0x00000000
0x40008040	PORTB_POL_CLR	PORTB Port x GPIO Polarity Invert Clear Register	0x00000000
0x40008044	PORTB_LOCK	PORTB Port x GPIO Lock Register	0x00000000
0x40008048	PORTB_TRIG_TGL	PORTB Port x GPIO Trigger Toggle Register	0x00000000

Table 48-48: CM41X_M4 PORTC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40009000	PORTC_FER	PORTC Port x Function Enable Register	0x00000000
0x40009004	PORTC_FER_SET	PORTC Port x Function Enable Set Register	0x00000000
0x40009008	PORTC_FER_CLR	PORTC Port x Function Enable Clear Register	0x00000000
0x4000900C	PORTC_DATA	PORTC Port x GPIO Data Register	0x00000000
0x40009010	PORTC_DATA_SET	PORTC Port x GPIO Data Set Register	0x00000000
0x40009014	PORTC_DATA_CLR	PORTC Port x GPIO Data Clear Register	0x00000000
0x40009018	PORTC_DIR	PORTC Port x GPIO Direction Register	0x00000000
0x4000901C	PORTC_DIR_SET	PORTC Port x GPIO Direction Set Register	0x00000000

Table 48-48: CM41X_M4 PORTC MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40009020	PORTC_DIR_CLR	PORTC Port x GPIO Direction Clear Register	0x00000000
0x40009024	PORTC_INEN	PORTC Port x GPIO Input Enable Register	0x00000000
0x40009028	PORTC_INEN_SET	PORTC Port x GPIO Input Enable Set Register	0x00000000
0x4000902C	PORTC_INEN_CLR	PORTC Port x GPIO Input Enable Clear Register	0x00000000
0x40009030	PORTC_MUX	PORTC Port x Multiplexer Control Register	0x00000000
0x40009034	PORTC_DATA_TGL	PORTC Port x GPIO Output Toggle Register	0x00000000
0x40009038	PORTC_POL	PORTC Port x GPIO Polarity Invert Register	0x00000000
0x4000903C	PORTC_POL_SET	PORTC Port x GPIO Polarity Invert Set Register	0x00000000
0x40009040	PORTC_POL_CLR	PORTC Port x GPIO Polarity Invert Clear Register	0x00000000
0x40009044	PORTC_LOCK	PORTC Port x GPIO Lock Register	0x00000000
0x40009048	PORTC_TRIG_TGL	PORTC Port x GPIO Trigger Toggle Register	0x00000000

Table 48-49: CM41X_M4 PORTD MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4000A000	PORTD_FER	PORTD Port x Function Enable Register	0x00000000
0x4000A004	PORTD_FER_SET	PORTD Port x Function Enable Set Register	0x00000000
0x4000A008	PORTD_FER_CLR	PORTD Port x Function Enable Clear Register	0x00000000
0x4000A00C	PORTD_DATA	PORTD Port x GPIO Data Register	0x00000000
0x4000A010	PORTD_DATA_SET	PORTD Port x GPIO Data Set Register	0x00000000
0x4000A014	PORTD_DATA_CLR	PORTD Port x GPIO Data Clear Register	0x00000000
0x4000A018	PORTD_DIR	PORTD Port x GPIO Direction Register	0x00000000
0x4000A01C	PORTD_DIR_SET	PORTD Port x GPIO Direction Set Register	0x00000000
0x4000A020	PORTD_DIR_CLR	PORTD Port x GPIO Direction Clear Register	0x00000000
0x4000A024	PORTD_INEN	PORTD Port x GPIO Input Enable Register	0x00000000
0x4000A028	PORTD_INEN_SET	PORTD Port x GPIO Input Enable Set Register	0x00000000
0x4000A02C	PORTD_INEN_CLR	PORTD Port x GPIO Input Enable Clear Register	0x00000000
0x4000A030	PORTD_MUX	PORTD Port x Multiplexer Control Register	0x00000000
0x4000A034	PORTD_DATA_TGL	PORTD Port x GPIO Output Toggle Register	0x00000000
0x4000A038	PORTD_POL	PORTD Port x GPIO Polarity Invert Register	0x00000000
0x4000A03C	PORTD_POL_SET	PORTD Port x GPIO Polarity Invert Set Register	0x00000000

Table 48-49: CM41X_M4 PORTD MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4000A040	PORTD_POL_CLR	PORTD Port x GPIO Polarity Invert Clear Register	0x00000000
0x4000A044	PORTD_LOCK	PORTD Port x GPIO Lock Register	0x00000000
0x4000A048	PORTD_TRIG_TGL	PORTD Port x GPIO Trigger Toggle Register	0x00000000

Table 48-50: CM41X_M4 PORTE MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4000B000	PORTE_FER	PORTE Port x Function Enable Register	0x00000000
0x4000B004	PORTE_FER_SET	PORTE Port x Function Enable Set Register	0x00000000
0x4000B008	PORTE_FER_CLR	PORTE Port x Function Enable Clear Register	0x00000000
0x4000B00C	PORTE_DATA	PORTE Port x GPIO Data Register	0x00000000
0x4000B010	PORTE_DATA_SET	PORTE Port x GPIO Data Set Register	0x00000000
0x4000B014	PORTE_DATA_CLR	PORTE Port x GPIO Data Clear Register	0x00000000
0x4000B018	PORTE_DIR	PORTE Port x GPIO Direction Register	0x00000000
0x4000B01C	PORTE_DIR_SET	PORTE Port x GPIO Direction Set Register	0x00000000
0x4000B020	PORTE_DIR_CLR	PORTE Port x GPIO Direction Clear Register	0x00000000
0x4000B024	PORTE_INEN	PORTE Port x GPIO Input Enable Register	0x00000000
0x4000B028	PORTE_INEN_SET	PORTE Port x GPIO Input Enable Set Register	0x00000000
0x4000B02C	PORTE_INEN_CLR	PORTE Port x GPIO Input Enable Clear Register	0x00000000
0x4000B030	PORTE_MUX	PORTE Port x Multiplexer Control Register	0x00000000
0x4000B034	PORTE_DATA_TGL	PORTE Port x GPIO Output Toggle Register	0x00000000
0x4000B038	PORTE_POL	PORTE Port x GPIO Polarity Invert Register	0x00000000
0x4000B03C	PORTE_POL_SET	PORTE Port x GPIO Polarity Invert Set Register	0x00000000
0x4000B040	PORTE_POL_CLR	PORTE Port x GPIO Polarity Invert Clear Register	0x00000000
0x4000B044	PORTE_LOCK	PORTE Port x GPIO Lock Register	0x00000000
0x4000B048	PORTE_TRIG_TGL	PORTE Port x GPIO Trigger Toggle Register	0x00000000

Table 48-51: CM41X_M4 PORTF MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4000C000	PORTF_FER	PORTF Port x Function Enable Register	0x00000000
0x4000C004	PORTF_FER_SET	PORTF Port x Function Enable Set Register	0x00000000

Table 48-51: CM41X_M4 PORTF MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4000C008	PORTF_FER_CLR	PORTF Port x Function Enable Clear Register	0x00000000
0x4000C00C	PORTF_DATA	PORTF Port x GPIO Data Register	0x00000000
0x4000C010	PORTF_DATA_SET	PORTF Port x GPIO Data Set Register	0x00000000
0x4000C014	PORTF_DATA_CLR	PORTF Port x GPIO Data Clear Register	0x00000000
0x4000C018	PORTF_DIR	PORTF Port x GPIO Direction Register	0x00000000
0x4000C01C	PORTF_DIR_SET	PORTF Port x GPIO Direction Set Register	0x00000000
0x4000C020	PORTF_DIR_CLR	PORTF Port x GPIO Direction Clear Register	0x00000000
0x4000C024	PORTF_INEN	PORTF Port x GPIO Input Enable Register	0x00000000
0x4000C028	PORTF_INEN_SET	PORTF Port x GPIO Input Enable Set Register	0x00000000
0x4000C02C	PORTF_INEN_CLR	PORTF Port x GPIO Input Enable Clear Register	0x00000000
0x4000C030	PORTF_MUX	PORTF Port x Multiplexer Control Register	0x00000000
0x4000C034	PORTF_DATA_TGL	PORTF Port x GPIO Output Toggle Register	0x00000000
0x4000C038	PORTF_POL	PORTF Port x GPIO Polarity Invert Register	0x00000000
0x4000C03C	PORTF_POL_SET	PORTF Port x GPIO Polarity Invert Set Register	0x00000000
0x4000C040	PORTF_POL_CLR	PORTF Port x GPIO Polarity Invert Clear Register	0x00000000
0x4000C044	PORTF_LOCK	PORTF Port x GPIO Lock Register	0x00000000
0x4000C048	PORTF_TRIG_TGL	PORTF Port x GPIO Trigger Toggle Register	0x00000000

Table 48-52: CM41X_M4 PWM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40028000	PWM0_CTL	PWM0 Control Register	0x00020000
0x40028004	PWM0_CHANCFG	PWM0 Channel Configuration Register	0x00000000
0x40028008	PWM0_TRIPCFG	PWM0 Trip Configuration Register	0x00000000
0x4002800C	PWM0_STAT	PWM0 Status Register	0x00000000
0x40028010	PWM0_IMSK	PWM0 Interrupt Mask Register	0x00000000
0x40028014	PWM0_ILAT	PWM0 Interrupt Latch Register	0x00000000
0x40028018	PWM0_CHOPCFG	PWM0 Chop Configuration Register	0x00000000
0x40028020	PWM0_SYNC_WID	PWM0 Sync Pulse Width Register	0x000003FF
0x40028024	PWM0_TM0	PWM0 Timer 0 Period Register	0x00000000
0x40028028	PWM0_TM1	PWM0 Timer 1 Period Register	0x00000000

Table 48-52: CM41X_M4 PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002802C	PWM0_TM2	PWM0 Timer 2 Period Register	0x00000000
0x40028030	PWM0_TM3	PWM0 Timer 3 Period Register	0x00000000
0x40028034	PWM0_TM4	PWM0 Timer 4 Period Register	0x00000000
0x40028038	PWM0_DLYA	PWM0 Channel A Delay Register	0x00000000
0x4002803C	PWM0_DLYB	PWM0 Channel B Delay Register	0x00000000
0x40028040	PWM0_DLYC	PWM0 Channel C Delay Register	0x00000000
0x40028044	PWM0_DLYD	PWM0 Channel D Delay Register	0x00000000
0x40028048	PWM0_ACTL	PWM0 Channel A Control Register	0x00000000
0x4002804C	PWM0_AH0	PWM0 Channel A-High Duty-0 Register	0x00000000
0x40028050	PWM0_AH1	PWM0 Channel A-High Duty-1 Register	0x00000000
0x40028054	PWM0_AH0_HP	PWM0 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x40028058	PWM0_AH1_HP	PWM0 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4002805C	PWM0_AL0	PWM0 Channel A-Low Duty-0 Register	0x00000000
0x40028060	PWM0_AL1	PWM0 Channel A-Low Duty-1 Register	0x00000000
0x40028064	PWM0_AL0_HP	PWM0 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x40028068	PWM0_AL1_HP	PWM0 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002806C	PWM0_BCTL	PWM0 Channel B Control Register	0x00000000
0x40028070	PWM0_BH0	PWM0 Channel B-High Duty-0 Register	0x00000000
0x40028074	PWM0_BH1	PWM0 Channel B-High Duty-1 Register	0x00000000
0x40028078	PWM0_BH0_HP	PWM0 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x4002807C	PWM0_BH1_HP	PWM0 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x40028080	PWM0_BL0	PWM0 Channel B-Low Duty-0 Register	0x00000000
0x40028084	PWM0_BL1	PWM0 Channel B-Low Duty-1 Register	0x00000000
0x40028088	PWM0_BL0_HP	PWM0 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002808C	PWM0_BL1_HP	PWM0 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000

Table 48-52: CM41X_M4 PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40028090	PWM0_CCTL	PWM0 Channel C Control Register	0x00000000
0x40028094	PWM0_CH0	PWM0 Channel C-High Pulse Duty Register 0	0x00000000
0x40028098	PWM0_CH1	PWM0 Channel C-High Pulse Duty Register 1	0x00000000
0x4002809C	PWM0_CH0_HP	PWM0 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400280A0	PWM0_CH1_HP	PWM0 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400280A4	PWM0_CL0	PWM0 Channel C-Low Pulse Duty Register 0	0x00000000
0x400280A8	PWM0_CL1	PWM0 Channel C-Low Duty-1 Register	0x00000000
0x400280AC	PWM0_CL0_HP	PWM0 Channel C-Low Pulse Duty Register 1	0x00000000
0x400280B0	PWM0_CL1_HP	PWM0 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x400280B4	PWM0_DCTL	PWM0 Channel D Control Register	0x00000000
0x400280B8	PWM0_DH0	PWM0 Channel D-High Duty-0 Register	0x00000000
0x400280BC	PWM0_DH1	PWM0 Channel D-High Pulse Duty Register 1	0x00000000
0x400280C0	PWM0_DH0_HP	PWM0 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400280C4	PWM0_DH1_HP	PWM0 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400280C8	PWM0_DL0	PWM0 Channel D-Low Pulse Duty Register 0	0x00000000
0x400280CC	PWM0_DL1	PWM0 Channel D-Low Pulse Duty Register 1	0x00000000
0x400280D0	PWM0_DL0_HP	PWM0 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x400280D4	PWM0_DL1_HP	PWM0 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x400280D8	PWM0_AH_DUTY0	PWM0 Channel A-High Full Duty0 Register	0x00000000
0x400280DC	PWM0_AH_DUTY1	PWM0 Channel A-High Full Duty1 Register	0x00000000
0x400280E0	PWM0_AL_DUTY0	PWM0 Channel A-Low Full Duty0 Register	0x00000000
0x400280E4	PWM0_AL_DUTY1	PWM0 Channel A-Low Full Duty1 Register	0x00000000
0x400280E8	PWM0_BH_DUTY0	PWM0 Channel B-High Full Duty0 Register	0x00000000
0x400280EC	PWM0_BH_DUTY1	PWM0 Channel B-High Full Duty1 Register	0x00000000
0x400280F0	PWM0_BL_DUTY0	PWM0 Channel B-Low Full Duty0 Register	0x00000000
0x400280F4	PWM0_BL_DUTY1	PWM0 Channel B-Low Full Duty1 Register	0x00000000

Table 48-52: CM41X_M4 PWM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400280F8	PWM0_CH_DUTY0	PWM0 Channel C-High Full Duty0 Register	0x00000000
0x400280FC	PWM0_CH_DUTY1	PWM0 Channel C-High Full Duty1 Register	0x00000000
0x40028100	PWM0_CL_DUTY0	PWM0 Channel C-Low Full Duty0 Register	0x00000000
0x40028104	PWM0_CL_DUTY1	PWM0 Channel C-Low Full Duty1 Register	0x00000000
0x40028108	PWM0_DH_DUTY0	PWM0 Channel D-High Full Duty0 Register	0x00000000
0x4002810C	PWM0_DH_DUTY1	PWM0 Channel D-High Full Duty1 Register	0x00000000
0x40028110	PWM0_DL_DUTY0	PWM0 Channel D-Low Full Duty0 Register	0x00000000
0x40028114	PWM0_DL_DUTY1	PWM0 Channel D-Low Full Duty1 Register	0x00000000
0x40028118	PWM0_CHA_DT	PWM0 Channel A Dead-time Register	0x00000000
0x4002811C	PWM0_CHB_DT	PWM0 Channel B Dead-time Register	0x00000000
0x40028120	PWM0_CHC_DT	PWM0 Channel C Dead-time Register	0x00000000
0x40028124	PWM0_CHD_DT	PWM0 Channel D Dead-time Register	0x00000000
0x40028130	PWM0_SWTRIP	PWM0 Software Trip Register	0x00000000
0x40028134	PWM0_TRIP_POL	PWM0 Trip Polarity Register	0x00000000

Table 48-53: CM41X_M4 PWM1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40029000	PWM1_CTL	PWM1 Control Register	0x00020000
0x40029004	PWM1_CHANCFG	PWM1 Channel Configuration Register	0x00000000
0x40029008	PWM1_TRIPCFG	PWM1 Trip Configuration Register	0x00000000
0x4002900C	PWM1_STAT	PWM1 Status Register	0x00000000
0x40029010	PWM1_IMSK	PWM1 Interrupt Mask Register	0x00000000
0x40029014	PWM1_ILAT	PWM1 Interrupt Latch Register	0x00000000
0x40029018	PWM1_CHOPCFG	PWM1 Chop Configuration Register	0x00000000
0x40029020	PWM1_SYNC_WID	PWM1 Sync Pulse Width Register	0x000003FF
0x40029024	PWM1_TM0	PWM1 Timer 0 Period Register	0x00000000
0x40029028	PWM1_TM1	PWM1 Timer 1 Period Register	0x00000000
0x4002902C	PWM1_TM2	PWM1 Timer 2 Period Register	0x00000000
0x40029030	PWM1_TM3	PWM1 Timer 3 Period Register	0x00000000
0x40029034	PWM1_TM4	PWM1 Timer 4 Period Register	0x00000000

Table 48-53: CM41X_M4 PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40029038	PWM1_DLYA	PWM1 Channel A Delay Register	0x00000000
0x4002903C	PWM1_DLYB	PWM1 Channel B Delay Register	0x00000000
0x40029040	PWM1_DLYC	PWM1 Channel C Delay Register	0x00000000
0x40029044	PWM1_DLYD	PWM1 Channel D Delay Register	0x00000000
0x40029048	PWM1_ACTL	PWM1 Channel A Control Register	0x00000000
0x4002904C	PWM1_AH0	PWM1 Channel A-High Duty-0 Register	0x00000000
0x40029050	PWM1_AH1	PWM1 Channel A-High Duty-1 Register	0x00000000
0x40029054	PWM1_AH0_HP	PWM1 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x40029058	PWM1_AH1_HP	PWM1 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4002905C	PWM1_AL0	PWM1 Channel A-Low Duty-0 Register	0x00000000
0x40029060	PWM1_AL1	PWM1 Channel A-Low Duty-1 Register	0x00000000
0x40029064	PWM1_AL0_HP	PWM1 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x40029068	PWM1_AL1_HP	PWM1 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002906C	PWM1_BCTL	PWM1 Channel B Control Register	0x00000000
0x40029070	PWM1_BH0	PWM1 Channel B-High Duty-0 Register	0x00000000
0x40029074	PWM1_BH1	PWM1 Channel B-High Duty-1 Register	0x00000000
0x40029078	PWM1_BH0_HP	PWM1 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x4002907C	PWM1_BH1_HP	PWM1 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x40029080	PWM1_BL0	PWM1 Channel B-Low Duty-0 Register	0x00000000
0x40029084	PWM1_BL1	PWM1 Channel B-Low Duty-1 Register	0x00000000
0x40029088	PWM1_BL0_HP	PWM1 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002908C	PWM1_BL1_HP	PWM1 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x40029090	PWM1_CCTL	PWM1 Channel C Control Register	0x00000000
0x40029094	PWM1_CH0	PWM1 Channel C-High Pulse Duty Register 0	0x00000000
0x40029098	PWM1_CH1	PWM1 Channel C-High Pulse Duty Register 1	0x00000000

Table 48-53: CM41X_M4 PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002909C	PWM1_CH0_HP	PWM1 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400290A0	PWM1_CH1_HP	PWM1 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400290A4	PWM1_CL0	PWM1 Channel C-Low Pulse Duty Register 0	0x00000000
0x400290A8	PWM1_CL1	PWM1 Channel C-Low Duty-1 Register	0x00000000
0x400290AC	PWM1_CL0_HP	PWM1 Channel C-Low Pulse Duty Register 1	0x00000000
0x400290B0	PWM1_CL1_HP	PWM1 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x400290B4	PWM1_DCTL	PWM1 Channel D Control Register	0x00000000
0x400290B8	PWM1_DH0	PWM1 Channel D-High Duty-0 Register	0x00000000
0x400290BC	PWM1_DH1	PWM1 Channel D-High Pulse Duty Register 1	0x00000000
0x400290C0	PWM1_DH0_HP	PWM1 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x400290C4	PWM1_DH1_HP	PWM1 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x400290C8	PWM1_DL0	PWM1 Channel D-Low Pulse Duty Register 0	0x00000000
0x400290CC	PWM1_DL1	PWM1 Channel D-Low Pulse Duty Register 1	0x00000000
0x400290D0	PWM1_DL0_HP	PWM1 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x400290D4	PWM1_DL1_HP	PWM1 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x400290D8	PWM1_AH_DUTY0	PWM1 Channel A-High Full Duty0 Register	0x00000000
0x400290DC	PWM1_AH_DUTY1	PWM1 Channel A-High Full Duty1 Register	0x00000000
0x400290E0	PWM1_AL_DUTY0	PWM1 Channel A-Low Full Duty0 Register	0x00000000
0x400290E4	PWM1_AL_DUTY1	PWM1 Channel A-Low Full Duty1 Register	0x00000000
0x400290E8	PWM1_BH_DUTY0	PWM1 Channel B-High Full Duty0 Register	0x00000000
0x400290EC	PWM1_BH_DUTY1	PWM1 Channel B-High Full Duty1 Register	0x00000000
0x400290F0	PWM1_BL_DUTY0	PWM1 Channel B-Low Full Duty0 Register	0x00000000
0x400290F4	PWM1_BL_DUTY1	PWM1 Channel B-Low Full Duty1 Register	0x00000000
0x400290F8	PWM1_CH_DUTY0	PWM1 Channel C-High Full Duty0 Register	0x00000000
0x400290FC	PWM1_CH_DUTY1	PWM1 Channel C-High Full Duty1 Register	0x00000000
0x40029100	PWM1_CL_DUTY0	PWM1 Channel C-Low Full Duty0 Register	0x00000000

Table 48-53: CM41X_M4 PWM1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40029104	PWM1_CL_DUTY1	PWM1 Channel C-Low Full Duty1 Register	0x00000000
0x40029108	PWM1_DH_DUTY0	PWM1 Channel D-High Full Duty0 Register	0x00000000
0x4002910C	PWM1_DH_DUTY1	PWM1 Channel D-High Full Duty1 Register	0x00000000
0x40029110	PWM1_DL_DUTY0	PWM1 Channel D-Low Full Duty0 Register	0x00000000
0x40029114	PWM1_DL_DUTY1	PWM1 Channel D-Low Full Duty1 Register	0x00000000
0x40029118	PWM1_CHA_DT	PWM1 Channel A Dead-time Register	0x00000000
0x4002911C	PWM1_CHB_DT	PWM1 Channel B Dead-time Register	0x00000000
0x40029120	PWM1_CHC_DT	PWM1 Channel C Dead-time Register	0x00000000
0x40029124	PWM1_CHD_DT	PWM1 Channel D Dead-time Register	0x00000000
0x40029130	PWM1_SWTRIP	PWM1 Software Trip Register	0x00000000
0x40029134	PWM1_TRIP_POL	PWM1 Trip Polarity Register	0x00000000

Table 48-54: CM41X_M4 PWM2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4002A000	PWM2_CTL	PWM2 Control Register	0x00020000
0x4002A004	PWM2_CHANCFG	PWM2 Channel Configuration Register	0x00000000
0x4002A008	PWM2_TRIPCFG	PWM2 Trip Configuration Register	0x00000000
0x4002A00C	PWM2_STAT	PWM2 Status Register	0x00000000
0x4002A010	PWM2_IMSK	PWM2 Interrupt Mask Register	0x00000000
0x4002A014	PWM2_ILAT	PWM2 Interrupt Latch Register	0x00000000
0x4002A018	PWM2_CHOPCFG	PWM2 Chop Configuration Register	0x00000000
0x4002A020	PWM2_SYNC_WID	PWM2 Sync Pulse Width Register	0x000003FF
0x4002A024	PWM2_TM0	PWM2 Timer 0 Period Register	0x00000000
0x4002A028	PWM2_TM1	PWM2 Timer 1 Period Register	0x00000000
0x4002A02C	PWM2_TM2	PWM2 Timer 2 Period Register	0x00000000
0x4002A030	PWM2_TM3	PWM2 Timer 3 Period Register	0x00000000
0x4002A034	PWM2_TM4	PWM2 Timer 4 Period Register	0x00000000
0x4002A038	PWM2_DLYA	PWM2 Channel A Delay Register	0x00000000
0x4002A03C	PWM2_DLYB	PWM2 Channel B Delay Register	0x00000000
0x4002A040	PWM2_DLYC	PWM2 Channel C Delay Register	0x00000000

Table 48-54: CM41X_M4 PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002A044	PWM2_DLYD	PWM2 Channel D Delay Register	0x00000000
0x4002A048	PWM2_ACTL	PWM2 Channel A Control Register	0x00000000
0x4002A04C	PWM2_AH0	PWM2 Channel A-High Duty-0 Register	0x00000000
0x4002A050	PWM2_AH1	PWM2 Channel A-High Duty-1 Register	0x00000000
0x4002A054	PWM2_AH0_HP	PWM2 Channel A-High Heightened-Precision Duty-0 Register	0x00000000
0x4002A058	PWM2_AH1_HP	PWM2 Channel A-High Heightened-Precision Duty-1 Register	0x00000000
0x4002A05C	PWM2_AL0	PWM2 Channel A-Low Duty-0 Register	0x00000000
0x4002A060	PWM2_AL1	PWM2 Channel A-Low Duty-1 Register	0x00000000
0x4002A064	PWM2_AL0_HP	PWM2 Channel A-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002A068	PWM2_AL1_HP	PWM2 Channel A-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002A06C	PWM2_BCTL	PWM2 Channel B Control Register	0x00000000
0x4002A070	PWM2_BH0	PWM2 Channel B-High Duty-0 Register	0x00000000
0x4002A074	PWM2_BH1	PWM2 Channel B-High Duty-1 Register	0x00000000
0x4002A078	PWM2_BH0_HP	PWM2 Channel B-High Heightened-Precision Duty-0 Register	0x00000000
0x4002A07C	PWM2_BH1_HP	PWM2 Channel B-High Heightened-Precision Duty-1 Register	0x00000000
0x4002A080	PWM2_BL0	PWM2 Channel B-Low Duty-0 Register	0x00000000
0x4002A084	PWM2_BL1	PWM2 Channel B-Low Duty-1 Register	0x00000000
0x4002A088	PWM2_BL0_HP	PWM2 Channel B-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002A08C	PWM2_BL1_HP	PWM2 Channel B-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002A090	PWM2_CCTL	PWM2 Channel C Control Register	0x00000000
0x4002A094	PWM2_CH0	PWM2 Channel C-High Pulse Duty Register 0	0x00000000
0x4002A098	PWM2_CH1	PWM2 Channel C-High Pulse Duty Register 1	0x00000000
0x4002A09C	PWM2_CH0_HP	PWM2 Channel C-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x4002A0A0	PWM2_CH1_HP	PWM2 Channel C-High Pulse Heightened-Precision Duty Register 1	0x00000000

Table 48-54: CM41X_M4 PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002A0A4	PWM2_CL0	PWM2 Channel C-Low Pulse Duty Register 0	0x00000000
0x4002A0A8	PWM2_CL1	PWM2 Channel C-Low Duty-1 Register	0x00000000
0x4002A0AC	PWM2_CL0_HP	PWM2 Channel C-Low Pulse Duty Register 1	0x00000000
0x4002A0B0	PWM2_CL1_HP	PWM2 Channel C-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002A0B4	PWM2_DCTL	PWM2 Channel D Control Register	0x00000000
0x4002A0B8	PWM2_DH0	PWM2 Channel D-High Duty-0 Register	0x00000000
0x4002A0BC	PWM2_DH1	PWM2 Channel D-High Pulse Duty Register 1	0x00000000
0x4002A0C0	PWM2_DH0_HP	PWM2 Channel D-High Pulse Heightened-Precision Duty Register 0	0x00000000
0x4002A0C4	PWM2_DH1_HP	PWM2 Channel D High Pulse Heightened-Precision Duty Register 1	0x00000000
0x4002A0C8	PWM2_DL0	PWM2 Channel D-Low Pulse Duty Register 0	0x00000000
0x4002A0CC	PWM2_DL1	PWM2 Channel D-Low Pulse Duty Register 1	0x00000000
0x4002A0D0	PWM2_DL0_HP	PWM2 Channel D-Low Heightened-Precision Duty-0 Register	0x00000000
0x4002A0D4	PWM2_DL1_HP	PWM2 Channel D-Low Heightened-Precision Duty-1 Register	0x00000000
0x4002A0D8	PWM2_AH_DUTY0	PWM2 Channel A-High Full Duty0 Register	0x00000000
0x4002A0DC	PWM2_AH_DUTY1	PWM2 Channel A-High Full Duty1 Register	0x00000000
0x4002A0E0	PWM2_AL_DUTY0	PWM2 Channel A-Low Full Duty0 Register	0x00000000
0x4002A0E4	PWM2_AL_DUTY1	PWM2 Channel A-Low Full Duty1 Register	0x00000000
0x4002A0E8	PWM2_BH_DUTY0	PWM2 Channel B-High Full Duty0 Register	0x00000000
0x4002A0EC	PWM2_BH_DUTY1	PWM2 Channel B-High Full Duty1 Register	0x00000000
0x4002A0F0	PWM2_BL_DUTY0	PWM2 Channel B-Low Full Duty0 Register	0x00000000
0x4002A0F4	PWM2_BL_DUTY1	PWM2 Channel B-Low Full Duty1 Register	0x00000000
0x4002A0F8	PWM2_CH_DUTY0	PWM2 Channel C-High Full Duty0 Register	0x00000000
0x4002A0FC	PWM2_CH_DUTY1	PWM2 Channel C-High Full Duty1 Register	0x00000000
0x4002A100	PWM2_CL_DUTY0	PWM2 Channel C-Low Full Duty0 Register	0x00000000
0x4002A104	PWM2_CL_DUTY1	PWM2 Channel C-Low Full Duty1 Register	0x00000000
0x4002A108	PWM2_DH_DUTY0	PWM2 Channel D-High Full Duty0 Register	0x00000000
0x4002A10C	PWM2_DH_DUTY1	PWM2 Channel D-High Full Duty1 Register	0x00000000

Table 48-54: CM41X_M4 PWM2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002A110	PWM2_DL_DUTY0	PWM2 Channel D-Low Full Duty0 Register	0x00000000
0x4002A114	PWM2_DL_DUTY1	PWM2 Channel D-Low Full Duty1 Register	0x00000000
0x4002A118	PWM2_CHA_DT	PWM2 Channel A Dead-time Register	0x00000000
0x4002A11C	PWM2_CHB_DT	PWM2 Channel B Dead-time Register	0x00000000
0x4002A120	PWM2_CHC_DT	PWM2 Channel C Dead-time Register	0x00000000
0x4002A124	PWM2_CHD_DT	PWM2 Channel D Dead-time Register	0x00000000
0x4002A130	PWM2_SWTRIP	PWM2 Software Trip Register	0x00000000
0x4002A134	PWM2_TRIP_POL	PWM2 Trip Polarity Register	0x00000000

Table 48-55: CM41X_M4 RCU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40012000	RCU0_CTL	RCU0 Control Register	0x00000700
0x40012004	RCU0_STAT	RCU0 Status Register	0x00000021
0x40012008	RCU0_CRCTL	RCU0 Core Reset Outputs Control Register	0x00000002
0x4001200C	RCU0_CRSTAT	RCU0 Core Reset Outputs Status Register	0x00000007
0x40012018	RCU0_SRRQSTAT	RCU0 System Reset Request Status Register	0x00000000
0x40012028	RCU0_BCODE	RCU0 Boot Code Register	0x00000000
0x4001206C	RCU0_MSG	RCU0 Message Register	0x00000000
0x40012070	RCU0_MSG_SET	RCU0 Message Set Bits Register	0x00000000
0x40012074	RCU0_MSG_CLR	RCU0 Message Clear Bits Register	0x00000000

Table 48-56: CM41X_M4 SCB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4F042100	SCB0_DMA0_RQOS	SCB0 DMA0 Read Quality of Service Register	0x00000001
0x4F042104	SCB0_DMA0_WQOS	SCB0 DMA0 Write Quality of Service Register	0x00000001
0x4F043100	SCB0_DMA1_RQOS	SCB0 DMA1 Read Quality of Service Register	0x00000001
0x4F043104	SCB0_DMA1_WQOS	SCB0 DMA1 Write Quality of Service Register	0x00000001
0x4F044100	SCB0_DMA2_RQOS	SCB0 DMA2 Read Quality of Service Register	0x00000001
0x4F044104	SCB0_DMA2_WQOS	SCB0 DMA2 Write Quality of Service Register	0x00000001
0x4F045100	SCB0_DMA3_RQOS	SCB0 DMA3 Read Quality of Service Register	0x00000001

Table 48-56: CM41X_M4 SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4F045104	SCB0_DMA3_WQOS	SCB0 DMA3 Write Quality of Service Register	0x00000001
0x4F046100	SCB0_DMA4_RQOS	SCB0 DMA4 Read Quality of Service Register	0x00000001
0x4F046104	SCB0_DMA4_WQOS	SCB0 DMA4 Write Quality of Service Register	0x00000001
0x4F047100	SCB0_DMA5_RQOS	SCB0 DMA5 Read Quality of Service Register	0x00000001
0x4F047104	SCB0_DMA5_WQOS	SCB0 DMA5 Write Quality of Service Register	0x00000001
0x4F048100	SCB0_DMA6_RQOS	SCB0 DMA6 Read Quality of Service Register	0x00000001
0x4F048104	SCB0_DMA6_WQOS	SCB0 DMA6 Write Quality of Service Register	0x00000001
0x4F049100	SCB0_DMA7_RQOS	SCB0 DMA7 Read Quality of Service Register	0x00000001
0x4F049104	SCB0_DMA7_WQOS	SCB0 DMA7 Write Quality of Service Register	0x00000001
0x4F04A100	SCB0_DMA8_RQOS	SCB0 DMA8 Read Quality of Service Register	0x00000000
0x4F04A104	SCB0_DMA8_WQOS	SCB0 DMA8 Write Quality of Service Register	0x00000000
0x4F04B100	SCB0_DMA9_RQOS	SCB0 DMA9 Read Quality of Service Register	0x00000000
0x4F04B104	SCB0_DMA9_WQOS	SCB0 DMA9 Write Quality of Service Register	0x00000000
0x4F04C100	SCB0_AFE0_RQOS	SCB0 AFE0 Read Quality of Service Register	0x00000001
0x4F04C104	SCB0_AFE0_WQOS	SCB0 AFE0 Write Quality of Service Register	0x00000001
0x4F04D100	SCB0_FFT0_RQOS	SCB0 FFT0 Read Quality of Service Register	0x00000001
0x4F04D104	SCB0_FFT0_WQOS	SCB0 FFT0 Write Quality of Service Register	0x00000001
0x4F04E100	SCB0_M4_RQOS	SCB0 M4 Core Read Quality of Service Register	0x00000002
0x4F04E104	SCB0_M4_WQOS	SCB0 M4 Core Write Quality of Service Register	0x00000002
0x4F04F100	SCB0_M0_RQOS	SCB0 M0 Core Read Quality of Service Register	0x00000003
0x4F04F104	SCB0_M0_WQOS	SCB0 M0 Core Write Quality of Service Register	0x00000003
0x4F051100	SCB0_SINC0_RQOS	SCB0 SINC0 Read Quality of Service Register	0x00000001
0x4F051104	SCB0_SINC0_WQOS	SCB0 SINC0 Write Quality of Service Register	0x00000001

Table 48-57: CM41X_M4 SEC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41004000	SEC0_GCTL	SEC0 Global Control Register	0x00000000
0x41004004	SEC0_GSTAT	SEC0 Global Status Register	0x00000000
0x41004008	SEC0_RAISE	SEC0 Global Raise Register	0x00000000
0x41004010	SEC0_FCTL	SEC0 Fault Control Register	0x00000000

Table 48-57: CM41X_M4 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41004014	SEC0_FSTAT	SEC0 Fault Status Register	0x00000000
0x41004018	SEC0_FSID	SEC0 Fault Source ID Register	0x00000000
0x4100401C	SEC0_FEND	SEC0 Fault End Register	0x00000000
0x41004020	SEC0_FDLY	SEC0 Fault Delay Register	0x00000000
0x41004024	SEC0_FDLY_CUR	SEC0 Fault Delay Current Register	0x00000000
0x41004028	SEC0_FSRDLY	SEC0 Fault System Reset Delay Register	0x00000000
0x4100402C	SEC0_FSRDLY_CUR	SEC0 Fault System Reset Delay Current Register	0x00000000
0x41004030	SEC0_FCOPP	SEC0 Fault COP Period Register	0x00000000
0x41004034	SEC0_FCOPP_CUR	SEC0 Fault COP Period Current Register	0x00000000
0x41004800	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004804	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004808	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100480C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004810	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004814	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004818	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100481C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004820	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004824	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004828	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100482C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004830	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004834	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004838	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100483C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004840	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004844	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004848	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100484C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004850	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004854	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table 48-57: CM41X_M4 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41004858	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100485C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004860	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004864	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004868	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100486C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004870	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004874	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004878	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100487C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004880	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004884	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004888	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100488C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004890	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004894	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004898	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100489C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table 48-57: CM41X_M4 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410048D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410048F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410048FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004900	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004904	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004908	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100490C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004910	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004914	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004918	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100491C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004920	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004924	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004928	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100492C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004930	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004934	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004938	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100493C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004940	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004944	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004948	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100494C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table 48-57: CM41X_M4 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41004950	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004954	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004958	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100495C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004960	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004964	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004968	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100496C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004970	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004974	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004978	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100497C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004980	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004984	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004988	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100498C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004990	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004994	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004998	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x4100499C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table 48-57: CM41X_M4 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410049CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x410049F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x410049FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table 48-57: CM41X_M4 SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41004A48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x41004A58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x41004A5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40010000	SEC1_GCTL	SEC1 Global Control Register	0x00000000
0x40010004	SEC1_GSTAT	SEC1 Global Status Register	0x00000000
0x40010008	SEC1_RAISE	SEC1 Global Raise Register	0x00000000
0x40010010	SEC1_FCTL	SEC1 Fault Control Register	0x00000000
0x40010014	SEC1_FSTAT	SEC1 Fault Status Register	0x00000000
0x40010018	SEC1_FSID	SEC1 Fault Source ID Register	0x00000000
0x4001001C	SEC1_FEND	SEC1 Fault End Register	0x00000000
0x40010020	SEC1_FDLY	SEC1 Fault Delay Register	0x00000000
0x40010024	SEC1_FDLY_CUR	SEC1 Fault Delay Current Register	0x00000000
0x40010028	SEC1_FSRDLY	SEC1 Fault System Reset Delay Register	0x00000000
0x4001002C	SEC1_FSRDLY_CUR	SEC1 Fault System Reset Delay Current Register	0x00000000
0x40010030	SEC1_FCOPP	SEC1 Fault COP Period Register	0x00000000
0x40010034	SEC1_FCOPP_CUR	SEC1 Fault COP Period Current Register	0x00000000
0x40010800	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010804	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010808	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001080C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010810	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010814	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010818	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001081C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010820	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010824	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010828	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001082C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010830	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010834	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010838	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001083C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010840	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010844	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010848	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001084C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010850	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010854	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010858	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001085C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010860	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010864	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010868	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001086C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010870	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010874	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010878	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001087C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010880	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010884	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010888	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001088C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010890	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010894	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010898	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4001089C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108A0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108A4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108A8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108AC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108B0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108B4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108B8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108BC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108C0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108C4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108C8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108CC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108D0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108D4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108D8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108DC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108E0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108E4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108E8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108EC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108F0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108F4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400108F8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400108FC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010900	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010904	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010908	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001090C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010910	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010914	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010918	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001091C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010920	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010924	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010928	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001092C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010930	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010934	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010938	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001093C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010940	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010944	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010948	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001094C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010950	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010954	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010958	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001095C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010960	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010964	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010968	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001096C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010970	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010974	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010978	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001097C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010980	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010984	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010988	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001098C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010990	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010994	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010998	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x4001099C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109A0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109A4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109A8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109AC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109B0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109B4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109B8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109BC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109C0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109C4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109C8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109CC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109D0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109D4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109D8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109DC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109E0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109E4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109E8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109EC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109F0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109F4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x400109F8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x400109FC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A00	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A04	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A08	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A0C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010A10	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A14	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A18	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A1C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A20	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A24	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A28	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A2C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A30	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A34	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A38	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A3C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A40	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A44	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A48	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A4C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A50	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A54	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A58	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A5C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A60	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A64	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A68	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A6C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A70	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A74	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A78	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A7C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A80	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A84	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A88	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010A8C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A90	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A94	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010A98	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010A9C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AA0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AA4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AA8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AAC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AB0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AB4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AB8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010ABC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AC0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AC4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AC8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010ACC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AD0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AD4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AD8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010ADC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AE0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AE4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AE8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AEC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AF0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AF4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010AF8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010AFC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B00	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B04	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010B08	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B0C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B10	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B14	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B18	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B1C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B20	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B24	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B28	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B2C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B30	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B34	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B38	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B3C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B40	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B44	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B48	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B4C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B50	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B54	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B58	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B5C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B60	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B64	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B68	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B6C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B70	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B74	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B78	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B7C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B80	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010B84	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B88	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B8C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B90	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B94	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010B98	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010B9C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BA0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BA4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BA8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BAC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BB0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BB4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BB8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BBC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BC0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BC4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BC8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BCC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BD0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BD4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BD8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BDC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BE0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BE4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BE8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BEC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BF0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BF4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010BF8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010BFC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010C00	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C04	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C08	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C0C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C10	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C14	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C18	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C1C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C20	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C24	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C28	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C2C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C30	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C34	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C38	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C3C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C40	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C44	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C48	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C4C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C50	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C54	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C58	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C5C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C60	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C64	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C68	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C6C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C70	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C74	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C78	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010C7C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C80	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C84	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C88	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C8C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C90	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C94	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010C98	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010C9C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CA0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CA4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CA8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CAC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CB0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CB4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CB8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CBC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CC0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CC4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CC8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CCC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CD0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CD4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CD8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CDC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CE0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CE4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CE8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CEC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010CF0	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CF4	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 48-58: CM41X_M4 SEC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40010CF8	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010CFC	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010D00	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010D04	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010D08	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010D0C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010D10	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010D14	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000
0x40010D18	SEC1_SCTL[n]	SEC1 Source Control Register n	0x00000000
0x40010D1C	SEC1_SSTAT[n]	SEC1 Source Status Register n	0x00000000

Table 48-59: CM41X_M4 SINC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40049000	SINC0_CTL	SINC0 Control Register	0x00000000
0x40049004	SINC0_STAT	SINC0 Status Register	0x00000000
0x40049008	SINC0_CLK	SINC0 Clock Control Register	0x10001000
0x40049010	SINC0_RATE0	SINC0 Rate Control for Group 0 Register	0x00000804
0x40049014	SINC0_RATE1	SINC0 Rate Control for Group 1 Register	0x00000804
0x40049018	SINC0_LEVEL0	SINC0 Level Control for Group 0 Register	0x04000000
0x4004901C	SINC0_LEVEL1	SINC0 Level Control for Group 1 Register	0x04000000
0x40049020	SINC0_LIMIT0	SINC0 (Amplitude) Limits for Secondary Filter 0 Register	0x00000000
0x40049024	SINC0_LIMIT1	SINC0 (Amplitude) Limits for Secondary Filter 1 Register	0x00000000
0x40049028	SINC0_LIMIT2	SINC0 (Amplitude) Limits for Secondary Filter 2 Register	0x00000000
0x4004902C	SINC0_LIMIT3	SINC0 (Amplitude) Limits for Secondary Filter 3 Register	0x00000000
0x40049030	SINC0_BIAS0	SINC0 Bias for Group 0 Register	0x00000000
0x40049034	SINC0_BIAS1	SINC0 Bias for Group 1 Register	0x00000000
0x40049038	SINC0_PPTR0	SINC0 Primary (Filters) Pointer for Group 0 Register	0x00000000
0x4004903C	SINC0_PPTR1	SINC0 Primary (Filters) Pointer for Group 1 Register	0x00000000
0x40049040	SINC0_PHEAD0	SINC0 Primary (Filters) Head for Group 0 Register	0x00000000
0x40049044	SINC0_PHEAD1	SINC0 Primary (Filters) Head for Group 1 Register	0x00000000

Table 48-59: CM41X_M4 SINC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40049048	SINC0_PTAIL0	SINC0 Primary (Filters) Tail for Group 0 Register	0x00000000
0x4004904C	SINC0_PTAIL1	SINC0 Primary (Filters) Tail for Group 1 Register	0x00000000
0x40049050	SINC0_HIS_STAT	SINC0 History Status Register	0x00000000
0x40049080	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x40049084	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x40049088	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x4004908C	SINC0_P0SEC_HIST[n]	SINC0 Pair 0 Secondary (Filter) History n Register	0x00000000
0x40049090	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x40049094	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x40049098	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x4004909C	SINC0_P1SEC_HIST[n]	SINC0 Pair 1 Secondary (Filter) History n Register	0x00000000
0x400490A0	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x400490A4	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x400490A8	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x400490AC	SINC0_P2SEC_HIST[n]	SINC0 Pair 2 Secondary (Filter) History n Register	0x00000000
0x400490B0	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x400490B4	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x400490B8	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000
0x400490BC	SINC0_P3SEC_HIST[n]	SINC0 Pair 3 Secondary (Filter) History n Register	0x00000000

Table 48-60: CM41X_M4 SMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4002F00C	SMC0_B0CTL	SMC0 Bank 0 Control Register	0x01000000
0x4002F010	SMC0_B0TIM	SMC0 Bank 0 Timing Register	0x01010101
0x4002F014	SMC0_B0ETIM	SMC0 Bank 0 Extended Timing Register	0x00020200
0x4002F01C	SMC0_B1CTL	SMC0 Bank 1 Control Register	0x00000000
0x4002F020	SMC0_B1TIM	SMC0 Bank 1 Timing Register	0x01010101
0x4002F024	SMC0_B1ETIM	SMC0 Bank 1 Extended Timing Register	0x00020200
0x4002F02C	SMC0_B2CTL	SMC0 Bank 2 Control Register	0x00000000
0x4002F030	SMC0_B2TIM	SMC0 Bank 2 Timing Register	0x01010101

Table 48-60: CM41X_M4 SMC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002F034	SMC0_B2ETIM	SMC0 Bank 2 Extended Timing Register	0x00020200
0x4002F03C	SMC0_B3CTL	SMC0 Bank 3 Control Register	0x00000000
0x4002F040	SMC0_B3TIM	SMC0 Bank 3 Timing Register	0x01010101
0x4002F044	SMC0_B3ETIM	SMC0 Bank 3 Extended Timing Register	0x00020200

Table 48-61: CM41X_M4 SMPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x410E0000	SMPU0_CTL	SMPU0 SMPU Control Register	0x00000000
0x410E0004	SMPU0_STAT	SMPU0 SMPU Status Register	0x00000000
0x410E0008	SMPU0_IADDR	SMPU0 Interrupt Address Register	0x00000000
0x410E000C	SMPU0_IDTLS	SMPU0 Interrupt Details Register	0x00000000
0x410E0010	SMPU0_BADDR	SMPU0 Bus Error Address Register	0x00000000
0x410E0014	SMPU0_BDTLS	SMPU0 Bus Error Details Register	0x00000000
0x410E0020	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x410E0024	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x410E0028	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x410E002C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x410E0030	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x410E0034	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x410E0038	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x410E003C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x410E0040	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x410E0044	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x410E0048	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x410E004C	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x410E0050	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x410E0054	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x410E0058	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x410E005C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x410E0060	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000

Table 48-61: CM41X_M4 SMPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410E0064	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x410E0068	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x410E006C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x410E0070	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x410E0074	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x410E0078	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x410E007C	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x410E0220	SMPU0_REVID	SMPU0 SMPU Revision ID Register	0x00000010

Table 48-62: CM41X_M4 SMPU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400E0000	SMPU1_CTL	SMPU1 SMPU Control Register	0x00000000
0x400E0004	SMPU1_STAT	SMPU1 SMPU Status Register	0x00000000
0x400E0008	SMPU1_IADDR	SMPU1 Interrupt Address Register	0x00000000
0x400E000C	SMPU1_IDTLS	SMPU1 Interrupt Details Register	0x00000000
0x400E0010	SMPU1_BADDR	SMPU1 Bus Error Address Register	0x00000000
0x400E0014	SMPU1_BDTLS	SMPU1 Bus Error Details Register	0x00000000
0x400E0020	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E0024	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0028	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E002C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0030	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E0034	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0038	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E003C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0040	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E0044	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0048	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E004C	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0050	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000

Table 48-62: CM41X_M4 SMPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400E0054	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0058	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E005C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0060	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E0064	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0068	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E006C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0070	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E0074	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0078	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E007C	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0080	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E0084	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E0088	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E008C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E0090	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E0094	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0098	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E009C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E00A0	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E00A4	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E00A8	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E00AC	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E00B0	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E00B4	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x400E00B8	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E00BC	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E00C0	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E00C4	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E00C8	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x400E00CC	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000

Table 48-62: CM41X_M4 SMPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400E00D0	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x400E00D4	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x400E00D8	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x400E00DC	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x400E0220	SMPU1_REVID	SMPU1 SMPU Revision ID Register	0x00000010

Table 48-63: CM41X_M4 SMPU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400E1000	SMPU2_CTL	SMPU2 SMPU Control Register	0x00000000
0x400E1004	SMPU2_STAT	SMPU2 SMPU Status Register	0x00000000
0x400E1008	SMPU2_IADDR	SMPU2 Interrupt Address Register	0x00000000
0x400E100C	SMPU2_IDTLS	SMPU2 Interrupt Details Register	0x00000000
0x400E1010	SMPU2_BADDR	SMPU2 Bus Error Address Register	0x00000000
0x400E1014	SMPU2_BDTLS	SMPU2 Bus Error Details Register	0x00000000
0x400E1020	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x400E1024	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x400E1028	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x400E102C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x400E1030	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x400E1034	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x400E1038	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x400E103C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x400E1040	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x400E1044	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x400E1048	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x400E104C	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x400E1050	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x400E1054	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x400E1058	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x400E105C	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000

Table 48-63: CM41X_M4 SMPU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400E1060	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x400E1064	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x400E1068	SMPU2_RCTL[n]	SMPU2 Region n Control Register	0x00000000
0x400E106C	SMPU2_RADDR[n]	SMPU2 Region n Address Register	0x00000000
0x400E1070	SMPU2_RIDA[n]	SMPU2 Region n ID A Register	0x00000000
0x400E1074	SMPU2_RIDMSKA[n]	SMPU2 Region n ID Mask A Register	0x00000000
0x400E1078	SMPU2_RIDB[n]	SMPU2 Region n ID B Register	0x00000000
0x400E107C	SMPU2_RIDMSKB[n]	SMPU2 Region n ID Mask B Register	0x00000000
0x400E1220	SMPU2_REVID	SMPU2 SMPU Revision ID Register	0x00000010

Table 48-64: CM41X_M4 SPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41009004	SPI0_CTL	SPI0 Control Register	0x00000050
0x41009008	SPI0_RXCTL	SPI0 Receive Control Register	0x00000000
0x4100900C	SPI0_TXCTL	SPI0 Transmit Control Register	0x00000000
0x41009010	SPI0_CLK	SPI0 Clock Rate Register	0x00000000
0x41009014	SPI0_DLY	SPI0 Delay Register	0x00000301
0x41009018	SPI0_SLVSEL	SPI0 Slave Select Register	0x0000FE00
0x4100901C	SPI0_RWC	SPI0 Received Word Count Register	0x00000000
0x41009020	SPI0_RWCR	SPI0 Received Word Count Reload Register	0x00000000
0x41009024	SPI0_TWC	SPI0 Transmitted Word Count Register	0x00000000
0x41009028	SPI0_TWCR	SPI0 Transmitted Word Count Reload Register	0x00000000
0x41009030	SPI0_IMSK	SPI0 Interrupt Mask Register	0x00000000
0x41009034	SPI0_IMSK_CLR	SPI0 Interrupt Mask Clear Register	0x00000000
0x41009038	SPI0_IMSK_SET	SPI0 Interrupt Mask Set Register	0x00000000
0x41009040	SPI0_STAT	SPI0 Status Register	0x00440001
0x41009044	SPI0_ILAT	SPI0 Masked Interrupt Condition Register	0x00000000
0x41009048	SPI0_ILAT_CLR	SPI0 Masked Interrupt Clear Register	0x00000000
0x41009050	SPI0_RFIFO	SPI0 Receive FIFO Data Register	0x00000000
0x41009058	SPI0_TFIFO	SPI0 Transmit FIFO Data Register	0x00000000

Table 48-65: CM41X_M4 SPI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40040004	SPI1_CTL	SPI1 Control Register	0x00000050
0x40040008	SPI1_RXCTL	SPI1 Receive Control Register	0x00000000
0x4004000C	SPI1_TXCTL	SPI1 Transmit Control Register	0x00000000
0x40040010	SPI1_CLK	SPI1 Clock Rate Register	0x00000000
0x40040014	SPI1_DLY	SPI1 Delay Register	0x00000301
0x40040018	SPI1_SLVSEL	SPI1 Slave Select Register	0x0000FE00
0x4004001C	SPI1_RWC	SPI1 Received Word Count Register	0x00000000
0x40040020	SPI1_RWCR	SPI1 Received Word Count Reload Register	0x00000000
0x40040024	SPI1_TWC	SPI1 Transmitted Word Count Register	0x00000000
0x40040028	SPI1_TWCR	SPI1 Transmitted Word Count Reload Register	0x00000000
0x40040030	SPI1_IMSK	SPI1 Interrupt Mask Register	0x00000000
0x40040034	SPI1_IMSK_CLR	SPI1 Interrupt Mask Clear Register	0x00000000
0x40040038	SPI1_IMSK_SET	SPI1 Interrupt Mask Set Register	0x00000000
0x40040040	SPI1_STAT	SPI1 Status Register	0x00440001
0x40040044	SPI1_ILAT	SPI1 Masked Interrupt Condition Register	0x00000000
0x40040048	SPI1_ILAT_CLR	SPI1 Masked Interrupt Clear Register	0x00000000
0x40040050	SPI1_RFIFO	SPI1 Receive FIFO Data Register	0x00000000
0x40040058	SPI1_TFIFO	SPI1 Transmit FIFO Data Register	0x00000000

Table 48-66: CM41X_M4 SPORT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40042000	SPORT0_CTL_A	SPORT0 Half SPORT 'A' Control Register	0x00000000
0x40042004	SPORT0_DIV_A	SPORT0 Half SPORT 'A' Divisor Register	0x00000000
0x40042008	SPORT0_MCTL_A	SPORT0 Half SPORT 'A' Multichannel Control Register	0x00000000
0x4004200C	SPORT0_CS0_A	SPORT0 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x40042010	SPORT0_CS1_A	SPORT0 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x40042014	SPORT0_CS2_A	SPORT0 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x40042018	SPORT0_CS3_A	SPORT0 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000

Table 48-66: CM41X_M4 SPORT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40042020	SPORT0_ERR_A	SPORT0 Half SPORT 'A' Error Register	0x00000000
0x40042024	SPORT0_MSTAT_A	SPORT0 Half SPORT 'A' Multichannel Status Register	0x00000000
0x40042028	SPORT0_CTL2_A	SPORT0 Half SPORT 'A' Control 2 Register	0x00000000
0x40042040	SPORT0_TXPRI_A	SPORT0 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x40042044	SPORT0_RXPRI_A	SPORT0 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x40042048	SPORT0_TXSEC_A	SPORT0 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x4004204C	SPORT0_RXSEC_A	SPORT0 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x40042080	SPORT0_CTL_B	SPORT0 Half SPORT 'B' Control Register	0x00000000
0x40042084	SPORT0_DIV_B	SPORT0 Half SPORT 'B' Divisor Register	0x00000000
0x40042088	SPORT0_MCTL_B	SPORT0 Half SPORT 'B' Multichannel Control Register	0x00000000
0x4004208C	SPORT0_CS0_B	SPORT0 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x40042090	SPORT0_CS1_B	SPORT0 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x40042094	SPORT0_CS2_B	SPORT0 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x40042098	SPORT0_CS3_B	SPORT0 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x400420A0	SPORT0_ERR_B	SPORT0 Half SPORT 'B' Error Register	0x00000000
0x400420A4	SPORT0_MSTAT_B	SPORT0 Half SPORT 'B' Multichannel Status Register	0x00000000
0x400420A8	SPORT0_CTL2_B	SPORT0 Half SPORT 'B' Control 2 Register	0x00000000
0x400420C0	SPORT0_TXPRI_B	SPORT0 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x400420C4	SPORT0_RXPRI_B	SPORT0 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x400420C8	SPORT0_TXSEC_B	SPORT0 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x400420CC	SPORT0_RXSEC_B	SPORT0 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table 48-67: CM41X_M4 SPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x410F8000	SPU0_CTL	SPU0 Control Register	0x000000AD
0x410F8004	SPU0_STAT	SPU0 Status Register	0x00000000
0x410F8040	SPU0_TIMEOUT	SPU0 Timeout Register	0x00000000
0x410F8400	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table 48-67: CM41X_M4 SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410F8404	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8408	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F840C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8410	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8414	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8418	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F841C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8420	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8424	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8428	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F842C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8430	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8434	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8438	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F843C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8440	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8444	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8448	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F844C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x410F8800	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8804	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8808	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F880C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8810	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8814	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8818	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F881C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8820	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8824	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8828	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F882C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000

Table 48-67: CM41X_M4 SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410F8830	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8834	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8838	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F883C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8840	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8844	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F8848	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F884C	SPU0_AP[n]	SPU0 Access Protect Register n	0x00000000
0x410F9080	SPU0_IDTLS	SPU0 Interrupt Details Register	0x00000000
0x410F9084	SPU0_IADDR	SPU0 Interrupt Address Register	0x00000000
0x410F9FC8	SPU0_DEVID	SPU0 Device Configuration Register	0x00140891

Table 48-68: CM41X_M4 SPU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F8000	SPU1_CTL	SPU1 Control Register	0x000000AD
0x400F8004	SPU1_STAT	SPU1 Status Register	0x00000000
0x400F8040	SPU1_TIMEOUT	SPU1 Timeout Register	0x00000000
0x400F8400	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8404	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8408	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F840C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8410	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8414	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8418	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F841C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8420	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8424	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8428	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F842C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8430	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000

Table 48-68: CM41X_M4 SPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F8434	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8438	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F843C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8440	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8444	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8448	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F844C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8450	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8454	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8458	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F845C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8460	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8464	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8468	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F846C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8470	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8474	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8478	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F847C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8480	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8484	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8488	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F848C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8490	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8494	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8498	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F849C	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84A0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84A4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84A8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84AC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000

Table 48-68: CM41X_M4 SPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F84B0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84B4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84B8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84BC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84C0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84C4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84C8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84CC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84D0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84D4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84D8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84DC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84E0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84E4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84E8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84EC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84F0	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84F4	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84F8	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F84FC	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8500	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8504	SPU1_WP[n]	SPU1 Write Protect Register n	0x00000000
0x400F8800	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8804	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8808	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F880C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8810	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8814	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8818	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F881C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8820	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000

Table 48-68: CM41X_M4 SPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F8824	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8828	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F882C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8830	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8834	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8838	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F883C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8840	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8844	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8848	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F884C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8850	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8854	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8858	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F885C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8860	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8864	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8868	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F886C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8870	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8874	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8878	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F887C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8880	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8884	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8888	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F888C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8890	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8894	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8898	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F889C	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000

Table 48-68: CM41X_M4 SPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F88A0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88A4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88A8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88AC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88B0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88B4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88B8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88BC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88C0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88C4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88C8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88CC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88D0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88D4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88D8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88DC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88E0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88E4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88E8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88EC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88F0	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88F4	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88F8	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F88FC	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8900	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F8904	SPU1_AP[n]	SPU1 Access Protect Register n	0x00000000
0x400F9080	SPU1_IDTLS	SPU1 Interrupt Details Register	0x00000000
0x400F9084	SPU1_IADDR	SPU1 Interrupt Address Register	0x00000000
0x400F9FC8	SPU1_DEVID	SPU1 Device Configuration Register	0x00420891

Table 48-69: CM41X_M4 SWU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x410F0000	SWU0_GCTL	SWU0 Global Control Register	0x00000000
0x410F0004	SWU0_GSTAT	SWU0 Global Status Register	0x00000000
0x410F0010	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x410F0014	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x410F0018	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x410F001C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x410F0020	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x410F0024	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x410F0028	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x410F002C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x410F0030	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x410F0034	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x410F0038	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x410F003C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x410F0040	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x410F0044	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x410F0048	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x410F004C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x410F0050	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x410F0054	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x410F0058	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x410F005C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x410F0060	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x410F0064	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x410F0068	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x410F006C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x410F0070	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x410F0074	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x410F0078	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x410F007C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x410F0080	SWU0_CNT[n]	SWU0 Count Register n	0x00000000

Table 48-69: CM41X_M4 SWU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410F0084	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x410F0088	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x410F008C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000

Table 48-70: CM41X_M4 SWU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x410F1000	SWU1_GCTL	SWU1 Global Control Register	0x00000000
0x410F1004	SWU1_GSTAT	SWU1 Global Status Register	0x00000000
0x410F1010	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x410F1014	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x410F1018	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x410F101C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x410F1020	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x410F1024	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x410F1028	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x410F102C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x410F1030	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x410F1034	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x410F1038	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x410F103C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x410F1040	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x410F1044	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x410F1048	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x410F104C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x410F1050	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x410F1054	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x410F1058	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x410F105C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x410F1060	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x410F1064	SWU1_TARG[n]	SWU1 Target Register n	0x00000000

Table 48-70: CM41X_M4 SWU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410F1068	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x410F106C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x410F1070	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x410F1074	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x410F1078	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x410F107C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x410F1080	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x410F1084	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x410F1088	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x410F108C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000

Table 48-71: CM41X_M4 SWU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F0000	SWU2_GCTL	SWU2 Global Control Register	0x00000000
0x400F0004	SWU2_GSTAT	SWU2 Global Status Register	0x00000000
0x400F0010	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x400F0014	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x400F0018	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x400F001C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x400F0020	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x400F0024	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x400F0028	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x400F002C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x400F0030	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x400F0034	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x400F0038	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x400F003C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x400F0040	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x400F0044	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x400F0048	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000

Table 48-71: CM41X_M4 SWU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F004C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x400F0050	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x400F0054	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x400F0058	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x400F005C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x400F0060	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x400F0064	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x400F0068	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x400F006C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x400F0070	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x400F0074	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x400F0078	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x400F007C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x400F0080	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x400F0084	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x400F0088	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x400F008C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000

Table 48-72: CM41X_M4 SWU3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F1000	SWU3_GCTL	SWU3 Global Control Register	0x00000000
0x400F1004	SWU3_GSTAT	SWU3 Global Status Register	0x00000000
0x400F1010	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x400F1014	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x400F1018	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x400F101C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x400F1020	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x400F1024	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x400F1028	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x400F102C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000

Table 48-72: CM41X_M4 SWU3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F1030	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x400F1034	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x400F1038	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x400F103C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x400F1040	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x400F1044	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x400F1048	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x400F104C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x400F1050	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x400F1054	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x400F1058	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x400F105C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x400F1060	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x400F1064	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x400F1068	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x400F106C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x400F1070	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x400F1074	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x400F1078	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x400F107C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x400F1080	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x400F1084	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x400F1088	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x400F108C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000

Table 48-73: CM41X_M4 SWU4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F2000	SWU4_GCTL	SWU4 Global Control Register	0x00000000
0x400F2004	SWU4_GSTAT	SWU4 Global Status Register	0x00000000
0x400F2010	SWU4_CTL[n]	SWU4 Control Register n	0x00000000

Table 48-73: CM41X_M4 SWU4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F2014	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x400F2018	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x400F201C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x400F2020	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x400F2024	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x400F2028	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x400F202C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x400F2030	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x400F2034	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x400F2038	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x400F203C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x400F2040	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x400F2044	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x400F2048	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x400F204C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x400F2050	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x400F2054	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x400F2058	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x400F205C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x400F2060	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x400F2064	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x400F2068	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x400F206C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x400F2070	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x400F2074	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x400F2078	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x400F207C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x400F2080	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x400F2084	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x400F2088	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x400F208C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000

Table 48-74: CM41X_M4 SWU5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F3000	SWU5_GCTL	SWU5 Global Control Register	0x00000000
0x400F3004	SWU5_GSTAT	SWU5 Global Status Register	0x00000000
0x400F3010	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x400F3014	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x400F3018	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x400F301C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x400F3020	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x400F3024	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x400F3028	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x400F302C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x400F3030	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x400F3034	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x400F3038	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x400F303C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x400F3040	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x400F3044	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x400F3048	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x400F304C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x400F3050	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x400F3054	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x400F3058	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x400F305C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x400F3060	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x400F3064	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x400F3068	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x400F306C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x400F3070	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x400F3074	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x400F3078	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x400F307C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x400F3080	SWU5_CNT[n]	SWU5 Count Register n	0x00000000

Table 48-74: CM41X_M4 SWU5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F3084	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x400F3088	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x400F308C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000

Table 48-75: CM41X_M4 SWU6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x400F4000	SWU6_GCTL	SWU6 Global Control Register	0x00000000
0x400F4004	SWU6_GSTAT	SWU6 Global Status Register	0x00000000
0x400F4010	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x400F4014	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x400F4018	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x400F401C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x400F4020	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x400F4024	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x400F4028	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x400F402C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x400F4030	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x400F4034	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x400F4038	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x400F403C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x400F4040	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x400F4044	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x400F4048	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x400F404C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x400F4050	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x400F4054	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x400F4058	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x400F405C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x400F4060	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x400F4064	SWU6_TARG[n]	SWU6 Target Register n	0x00000000

Table 48-75: CM41X_M4 SWU6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400F4068	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x400F406C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x400F4070	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x400F4074	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x400F4078	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x400F407C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x400F4080	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x400F4084	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x400F4088	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x400F408C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000

Table 48-76: CM41X_M4 TAPC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40017000	TAPC_IDCODE	TAPC IDCODE Register	0x2280B0CB
0x40017004	TAPC_USERCODE	TAPC USERCODE Register	0x00000000
0x40017008	TAPC_SDBGKEY_CTL	TAPC Secure Debug Key Control Register	0x00000000
0x4001700C	TAPC_SDBGKEY_STAT	TAPC Secure Debug Key Status Register	0x00000000
0x40017010	TAPC_SDBGKEY0	TAPC Secure Debug Key 0 Register	0x00000000
0x40017014	TAPC_SDBGKEY1	TAPC Secure Debug Key 1 Register	0x00000000
0x40017018	TAPC_SDBGKEY2	TAPC Secure Debug Key 2 Register	0x00000000
0x4001701C	TAPC_SDBGKEY3	TAPC Secure Debug Key 3 Register	0x00000000
0x40017030	TAPC_DBGCTL	TAPC Debug Control Register	0x00000000
0x40017034	TAPC_DBGSTAT	TAPC Debug Status Register	0x00000000
0x40017050	TAPC_SDBGKEYCMP0	TAPC Secure Debug Key 0 Compare Register	0x00000000
0x40017054	TAPC_SDBGKEYCMP1	TAPC Secure Debug Key 1 Compare Register	0x00000000
0x40017058	TAPC_SDBGKEYCMP2	TAPC Secure Debug Key 2 Compare Register	0x00000000
0x4001705C	TAPC_SDBGKEYCMP3	TAPC Secure Debug Key 3 Compare Register	0x00000000
0x40017070	TAPC_SDBGKEYID0	TAPC Secure Debug Key 0 Identification Register	0x00000000
0x40017074	TAPC_SDBGKEYID1	TAPC Secure Debug Key 1 Key Identification Register	0x00000000
0x40017078	TAPC_SDBGKEYID2	TAPC Secure Debug Key 2 Key Identification Register	0x00000000

Table 48-76: CM41X_M4 TAPC MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4001707C	TAPC_SDBGKEYID3	TAPC Secure Debug Key 3 Key Identification Register	0x00000000
0x400170E0	TAPC_SCMSG	TAPC System Run Control Message Register	0x00000000
0x400170E4	TAPC_SCMSG_SET	TAPC System Run Control Message Set Register	0x00000000
0x400170E8	TAPC_SCMSG_CLR	TAPC System Run Control Message Clear Register	0x00000000
0x400170EC	TAPC_SCMSG_TGL	TAPC System Run Control Message Toggle Register	0x00000000
0x400170F0	TAPC_RCMSG	TAPC Run Control Message Register	0x00000000
0x400170F4	TAPC_RCMSG_SET	TAPC Run Control Message Set Register	0x00000000
0x400170F8	TAPC_RCMSG_CLR	TAPC Run Control Message Clear Register	0x00000000
0x400170FC	TAPC_RCMSG_TGL	TAPC Run Control Message Toggle Register	0x00000000

Table 48-77: CM41X_M4 PADS0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41001004	PADS0_PCFG0	PADS0 Peripheral Configuration0 Register	0x00000000
0x41001100	PADS0_PORT[n]_DS	PADS0 Multi Port Drive Strength Control Register	0x00000000
0x41001120	PADS0_PORT[n]_RCTL	PADS0 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x41001140	PADS0_PORT[n]_TRIPST	PADS0 Multi Port Trip State Register	0x00000000
0x41001160	PADS0_PORT[n]_TRIPSEL	PADS0 Multi Port Trip Select Register	0x00000000

Table 48-78: CM41X_M4 PADS1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40001004	PADS1_PCFG0	PADS1 Peripheral Configuration0 Register	0x00000000
0x40001048	PADS1_FOCP_DIV	PADS1 Fast Over Current Protection Clock Divisor Register	0x00000000
0x40001070	PADS1_DBC_PRESCALE	PADS1 Debounce Prescale Register	0x00000000
0x40001080	PADS1_DBC[n]_CTL	PADS1 Debounce Control Register(s)	0x00000000
0x40001084	PADS1_DBC[n]_CTL	PADS1 Debounce Control Register(s)	0x00000000
0x40001088	PADS1_DBC[n]_CTL	PADS1 Debounce Control Register(s)	0x00000000
0x4000108C	PADS1_DBC[n]_CTL	PADS1 Debounce Control Register(s)	0x00000000
0x400010A0	PADS1_VMU_CTL	PADS1 Voltage Monitor Unit Control Register	0x00000000
0x400010A8	PADS1_VMU_TRIPEN	PADS1 Voltage Monitor Unit Trip Enable Register	0x00000000

Table 48-78: CM41X_M4 PADS1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400010AC	PADS1_NVWR_RSTCTL	PADS1 Non-Volatile Write Reset Control Register	0x00000000
0x400010B0	PADS1_MONOSC_CFG	PADS1 Monitor Oscillator Control Register	0x00000000
0x400010BC	PADS1_VMU_TRIM	PADS1 Voltage Monitor Unit Trim Register	0x00000000
0x40001100	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x40001104	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x40001108	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x4000110C	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x40001110	PADS1_PORT[n]_DS	PADS1 Multi Port Drive Strength Control Register	0x00000000
0x40001120	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x40001124	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x40001128	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x4000112C	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x40001130	PADS1_PORT[n]_RCTL	PADS1 Multi Port Pull-up/Pull-down Resistor Control Register	0x00000000
0x40001140	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x40001144	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x40001148	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x4000114C	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x40001150	PADS1_PORT[n]_TRIPST	PADS1 Multi Port Trip State Register	0x00000000
0x40001160	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000
0x40001164	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000
0x40001168	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000
0x4000116C	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000
0x40001170	PADS1_PORT[n]_TRIPSEL	PADS1 Multi Port Trip Select Register	0x00000000

Table 48-79: CM41X_M4 SYSBLK0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41000040	SYSBLK0_SYSSTAT	SYSBLK0 System Status Register	0x00000000

Table 48-79: CM41X_M4 SYSBLK0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41000070	SYSBLK0_SCB_RESP_CFG	SYSBLK0 SCB Response Configuration Register	0x00000000
0x41000074	SYSBLK0_SCB_TIMEOUT_VALUE	SYSBLK0 SCB Timeout Value Register	0x00000000
0x41000080	SYSBLK0_ENG_MODE_CFG0	SYSBLK0 Engineering Mode Configuration Register 0	0x00000000
0x410000B4	SYSBLK0_M0_VTOR	SYSBLK0 Vector Table Base Offset Register	0x200F0000
0x410000E0	SYSBLK0_SRAM0_ECC	SYSBLK0 M0 SRAM ECC Register	0x00000000
0x410000E4	SYSBLK0_SYS_STCALIB	SYSBLK0 System Register	0x02000000
0x410000E8	SYSBLK0_IRQ_LATENCY	SYSBLK0 M0 IRQ Latency Register	0x0000000D
0x41000200	SYSBLK0_SISTAT0	SYSBLK0 Shared Interrupt 0 Status Register	0x00000000
0x4100020C	SYSBLK0_SISTAT3	SYSBLK0 Shared Interrupt 3 Status Register	0x00000000
0x41000214	SYSBLK0_SISTAT5	SYSBLK0 Shared Interrupt 5 Status Register	0x00000000
0x41000218	SYSBLK0_SISTAT6	SYSBLK0 Shared Interrupt 6 Status Register	0x00000000
0x4100021C	SYSBLK0_SISTAT7	SYSBLK0 Shared Interrupt 7 Status Register	0x00000000
0x41000220	SYSBLK0_SISTAT8	SYSBLK0 Shared Interrupt 8 Status Register	0x00000000
0x41000228	SYSBLK0_SISTAT10	SYSBLK0 Shared Interrupt 10 Status Register	0x00000000
0x4100022C	SYSBLK0_SISTAT11	SYSBLK0 Shared Interrupt 11 Status Register	0x00000000
0x41000230	SYSBLK0_SISTAT12	SYSBLK0 Shared Interrupt 12 Status Register	0x00000000
0x4100023C	SYSBLK0_SISTAT15	SYSBLK0 Shared Interrupt 15 Status Register	0x00000000
0x41000240	SYSBLK0_SISTAT16	SYSBLK0 Shared Interrupt 16 Status Register	0x00000000
0x41000244	SYSBLK0_SISTAT17	SYSBLK0 Shared Interrupt 17 Status Register	0x00000000
0x41000248	SYSBLK0_SISTAT18	SYSBLK0 Shared Interrupt 18 Status Register	0x00000000
0x4100024C	SYSBLK0_SISTAT19	SYSBLK0 Shared Interrupt 19 Status Register	0x00000000
0x41000258	SYSBLK0_SISTAT22	SYSBLK0 Shared Interrupt 22 Status Register	0x00000000
0x41000264	SYSBLK0_SISTAT25	SYSBLK0 Shared Interrupt 25 Status Register	0x00000000
0x41000270	SYSBLK0_SISTAT28	SYSBLK0 Shared Interrupt 28 Status Register	0x00000000

Table 48-80: CM41X_M4 SYSBLK1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40000010	SYSBLK1_DMA_MUXCTL	SYSBLK1 Peripheral DMA Multiplexer Control	0x00000000
0x40000014	SYSBLK1_PWM_SYS_CFG	SYSBLK1 PWM System Configuration Register	0x00000000

Table 48-80: CM41X_M4 SYSBLK1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40000018	SYSBLK1_MEMST_CTL	SYSBLK1 Memory Self-Test Control Register	0x00000000
0x4000001C	SYSBLK1_SINC_TEST	SYSBLK1 SINC Test Register	0x00000000
0x40000040	SYSBLK1_SYSSTAT	SYSBLK1 System Status Register	0x00000000
0x40000050	SYSBLK1_CLKNG_TRIPEN	SYSBLK1 Clock Not Good Trip Register	0x00000000
0x40000058	SYSBLK1_FAULT_TRIPEN	SYSBLK1 Fault Trip Register	0x00000010
0x40000060	SYSBLK1_ROT_UPDN_CFG	SYSBLK1 Rotary Counter Up/Down Configuration Register	0x00000000
0x40000070	SYSBLK1_SCB_RESP_CFG	SYSBLK1 SCB Response Configuration Register	0x00000000
0x40000074	SYSBLK1_SCB_TIMEOUT_VALUE	SYSBLK1 SCB Timeout Value Register	0x00000000
0x40000080	SYSBLK1_ENG_MODE_CFG0	SYSBLK1 Engineering Mode Configuration Register 0	0x00000000
0x40000084	SYSBLK1_LROM_STAT	SYSBLK1 Logic ROM Status Register	0x00000000

Table 48-81: CM41X_M4 TIMER0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41007004	TIMER0_RUN	TIMER0 Run Register	0x00000000
0x41007008	TIMER0_RUN_SET	TIMER0 Run Set Register	0x00000000
0x4100700C	TIMER0_RUN_CLR	TIMER0 Run Clear Register	0x00000000
0x41007010	TIMER0_STOP_CFG	TIMER0 Stop Configuration Register	0x00000000
0x41007014	TIMER0_STOP_CFG_SET	TIMER0 Stop Configuration Set Register	0x00000000
0x41007018	TIMER0_STOP_CFG_CLR	TIMER0 Stop Configuration Clear Register	0x00000000
0x4100701C	TIMER0_DATA_IMSK	TIMER0 Data Interrupt Mask Register	0x000000FF
0x41007020	TIMER0_STAT_IMSK	TIMER0 Status Interrupt Mask Register	0x000000FF
0x41007024	TIMER0_TRG_MSK	TIMER0 Trigger Master Mask Register	0x000000FF
0x41007028	TIMER0_TRG_IE	TIMER0 Trigger Slave Enable Register	0x00000000
0x4100702C	TIMER0_DATA_ILAT	TIMER0 Data Interrupt Latch Register	0x00000000
0x41007030	TIMER0_STAT_ILAT	TIMER0 Status Interrupt Latch Register	0x00000000
0x41007034	TIMER0_ERR_TYPE	TIMER0 Error Type Status Register	0x00000000
0x41007038	TIMER0_BCAST_PER	TIMER0 Broadcast Period Register	0x00000000
0x4100703C	TIMER0_BCAST_WID	TIMER0 Broadcast Width Register	0x00000000

Table 48-81: CM41X_M4 TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41007040	TIMER0_BCAST_DLY	TIMER0 Broadcast Delay Register	0x00000000
0x41007060	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007064	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007068	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100706C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007070	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x41007080	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007084	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007088	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100708C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007090	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x410070A0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x410070A4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x410070A8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x410070AC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x410070B0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x410070C0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x410070C4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x410070C8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x410070CC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x410070D0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x410070E0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x410070E4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x410070E8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x410070EC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x410070F0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x41007100	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007104	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007108	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100710C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007110	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table 48-81: CM41X_M4 TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41007120	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007124	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007128	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100712C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007130	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x41007140	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x41007144	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x41007148	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x4100714C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x41007150	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table 48-82: CM41X_M4 TIMER1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40021004	TIMER1_RUN	TIMER1 Run Register	0x00000000
0x40021008	TIMER1_RUN_SET	TIMER1 Run Set Register	0x00000000
0x4002100C	TIMER1_RUN_CLR	TIMER1 Run Clear Register	0x00000000
0x40021010	TIMER1_STOP_CFG	TIMER1 Stop Configuration Register	0x00000000
0x40021014	TIMER1_STOP_CFG_SET	TIMER1 Stop Configuration Set Register	0x00000000
0x40021018	TIMER1_STOP_CFG_CLR	TIMER1 Stop Configuration Clear Register	0x00000000
0x4002101C	TIMER1_DATA_IMSK	TIMER1 Data Interrupt Mask Register	0x000000FF
0x40021020	TIMER1_STAT_IMSK	TIMER1 Status Interrupt Mask Register	0x000000FF
0x40021024	TIMER1_TRG_MSK	TIMER1 Trigger Master Mask Register	0x000000FF
0x40021028	TIMER1_TRG_IE	TIMER1 Trigger Slave Enable Register	0x00000000
0x4002102C	TIMER1_DATA_ILAT	TIMER1 Data Interrupt Latch Register	0x00000000
0x40021030	TIMER1_STAT_ILAT	TIMER1 Status Interrupt Latch Register	0x00000000
0x40021034	TIMER1_ERR_TYPE	TIMER1 Error Type Status Register	0x00000000
0x40021038	TIMER1_BCAST_PER	TIMER1 Broadcast Period Register	0x00000000
0x4002103C	TIMER1_BCAST_WID	TIMER1 Broadcast Width Register	0x00000000
0x40021040	TIMER1_BCAST_DLY	TIMER1 Broadcast Delay Register	0x00000000
0x40021060	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000

Table 48-82: CM41X_M4 TIMER1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40021064	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x40021068	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002106C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021070	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x40021080	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x40021084	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x40021088	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002108C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021090	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x400210A0	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x400210A4	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x400210A8	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x400210AC	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x400210B0	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x400210C0	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x400210C4	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x400210C8	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x400210CC	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x400210D0	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x400210E0	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x400210E4	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x400210E8	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x400210EC	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x400210F0	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x40021100	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x40021104	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x40021108	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002110C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021110	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x40021120	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x40021124	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001

Table 48-82: CM41X_M4 TIMER1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40021128	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002112C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021130	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000
0x40021140	TIMER1_TMR[n]_CFG	TIMER1 Timer n Configuration Register	0x00000000
0x40021144	TIMER1_TMR[n]_CNT	TIMER1 Timer n Counter Register	0x00000001
0x40021148	TIMER1_TMR[n]_PER	TIMER1 Timer n Period Register	0x00000000
0x4002114C	TIMER1_TMR[n]_WID	TIMER1 Timer n Width Register	0x00000000
0x40021150	TIMER1_TMR[n]_DLY	TIMER1 Timer n Delay Register	0x00000000

Table 48-83: CM41X_M4 TRACE0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0xE0040000	TRACE0_SSPSR	TRACE0 Supported Sync Port Sizes Register.	0x00000000
0xE0040004	TRACE0_CSPSR	TRACE0 Current Sync Port Size Register.	0x00000001
0xE0040010	TRACE0_ACPR	TRACE0 Async Clock Prescaler Register.	0x00000000
0xE00400F0	TRACE0_SPPR	TRACE0 Selected Pin Protocol Register.	0x00000001
0xE0040300	TRACE0_FFSR	TRACE0 Formatter and Flush Status Register.	0x00000008
0xE0040304	TRACE0_FFCR	TRACE0 Formatter and Flush Control Register.	0x00000102
0xE0040308	TRACE0_FSCR	TRACE0 Formatter Synchronization Counter Register.	0x00000000
0xE0040EE8	TRACE0_TRIGGER	TRACE0 Integration Register: TRIGGER.	0x00000000
0xE0040EEC	TRACE0_FIFO_DATA_0	TRACE0 Integration register: FIFO data 0.	0x00000000
0xE0040EF0	TRACE0_ITATBCTR2	TRACE0 Integration Register: ITATBCTR2.	0x00000000
0xE0040EF8	TRACE0_ITATBCTR0	TRACE0 Integration Register: ITATBCTR0.	0x00000000
0xE0040EFC	TRACE0_FIFO_DATA_1	TRACE0 Integration register: FIFO data 1.	0x00000000
0xE0040F00	TRACE0_ITCTRL	TRACE0 Integration Mode Control Register.	0x00000000
0xE0040FA0	TRACE0_CLAIMSET	TRACE0 Claim tag set register.	0x0000000F
0xE0040FA4	TRACE0_CLAIMCLR	TRACE0 Claim tag clear register.	0x00000000
0xE0040FC8	TRACE0_DEVID	TRACE0 Device ID register.	0x00000CA1
0xE0040FD0	TRACE0_PID4	TRACE0 Peripheral identification register (PID4).	0x00000004
0xE0040FD4	TRACE0_PID5	TRACE0 Peripheral identification register (PID5).	0x00000000
0xE0040FD8	TRACE0_PID6	TRACE0 Peripheral identification register (PID6).	0x00000000

Table 48-83: CM41X_M4 TRACE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0xE0040FDC	TRACE0_PID7	TRACE0 Peripheral identification register (PID7).	0x00000000
0xE0040FE0	TRACE0_PID0	TRACE0 Peripheral identification register Bits 7:0 (PID0).	0x00000023
0xE0040FE4	TRACE0_PID1	TRACE0 Peripheral identification register Bits 15:8 (PID1).	0x000000B9
0xE0040FE8	TRACE0_PID2	TRACE0 Peripheral identification register Bits 23:16 (PID2).	0x0000003B
0xE0040FEC	TRACE0_PID3	TRACE0 Peripheral identification register Bits 31:24 (PID3).	0x00000000
0xE0040FF0	TRACE0_CID0	TRACE0 Component identification register Bits 7:0 (CID0).	0x0000000D
0xE0040FF4	TRACE0_CID1	TRACE0 Component identification register Bits 15:8 (CID1).	0x00000090
0xE0040FF8	TRACE0_CID2	TRACE0 Component identification register Bits 23:16 (CID2).	0x00000005
0xE0040FFC	TRACE0_CID3	TRACE0 Component identification register Bits 31:24 (CID3).	0x000000B1

Table 48-84: CM41X_M4 TRU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41006000	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006004	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006008	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100600C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006010	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006014	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006018	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100601C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006020	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006024	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006028	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100602C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006030	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table 48-84: CM41X_M4 TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x41006034	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006038	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100603C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006040	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006044	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006048	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100604C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006050	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006054	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006058	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100605C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006060	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006064	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006068	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100606C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006070	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006074	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006078	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100607C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006080	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006084	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006088	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100608C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006090	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006094	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x41006098	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x4100609C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060A0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060A4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060A8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060AC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table 48-84: CM41X_M4 TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x410060B0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060B4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410060B8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x410067E0	TRU0_MTR	TRU0 Master Trigger Register	0x00000000
0x410067E8	TRU0_ERRADDR	TRU0 Error Address Register	0x00000000
0x410067EC	TRU0_STAT	TRU0 Status Information Register	0x00000000
0x410067F4	TRU0_GCTL	TRU0 Global Control Register	0x00000000

Table 48-85: CM41X_M4 TRU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40020000	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020004	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020008	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002000C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020010	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020014	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020018	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002001C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020020	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020024	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020028	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002002C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020030	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020034	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020038	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002003C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020040	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020044	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020048	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002004C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000

Table 48-85: CM41X_M4 TRU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40020050	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020054	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020058	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002005C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020060	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020064	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020068	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002006C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020070	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020074	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020078	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002007C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020080	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020084	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020088	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002008C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020090	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020094	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020098	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002009C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200A0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200A4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200A8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200AC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200B0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200B4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200B8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200BC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200C0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200C4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200C8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000

Table 48-85: CM41X_M4 TRU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400200CC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200D0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200D4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200D8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200DC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200E0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200E4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200E8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200EC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200F0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200F4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200F8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400200FC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020100	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020104	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020108	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002010C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020110	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020114	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020118	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002011C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020120	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020124	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020128	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002012C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020130	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020134	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020138	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002013C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020140	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020144	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000

Table 48-85: CM41X_M4 TRU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40020148	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002014C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020150	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020154	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020158	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002015C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020160	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020164	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020168	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002016C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020170	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020174	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020178	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002017C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020180	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020184	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020188	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002018C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020190	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020194	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020198	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002019C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201A0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201A4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201A8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201AC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201B0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201B4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201B8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201BC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201C0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000

Table 48-85: CM41X_M4 TRU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x400201C4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201C8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201CC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201D0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201D4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201D8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201DC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201E0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201E4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201E8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201EC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201F0	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201F4	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201F8	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400201FC	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020200	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020204	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020208	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x4002020C	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020210	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020214	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x40020218	TRU1_SSR[n]	TRU1 Slave Select Register	0x00000000
0x400207E0	TRU1_MTR	TRU1 Master Trigger Register	0x00000000
0x400207E8	TRU1_ERRADDR	TRU1 Error Address Register	0x00000000
0x400207EC	TRU1_STAT	TRU1 Status Information Register	0x00000000
0x400207F4	TRU1_GCTL	TRU1 Global Control Register	0x00000000

Table 48-86: CM41X_M4 TTU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4002C000	TTU0_CTL	TTU0 TTU Control Register	0x00000000

Table 48-86: CM41X_M4 TTU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4002C004	TTU0_REVID	TTU0 Revision ID Register	0x00000010
0x4002C008	TTU0_STOP	TTU0 Counter Stop Request Register	0x00000000
0x4002C00C	TTU0_STAT	TTU0 Counter Status Register	0x00000000
0x4002C010	TTU0_CHK	TTU0 Counter Check Register	0x00000000
0x4002C020	TTU0_CNT[n]	TTU0 Counter Current Value	0x00000000
0x4002C024	TTU0_CNT[n]	TTU0 Counter Current Value	0x00000000
0x4002C028	TTU0_CNT[n]	TTU0 Counter Current Value	0x00000000
0x4002C02C	TTU0_CNT[n]	TTU0 Counter Current Value	0x00000000
0x4002C040	TTU0_PER[n]	TTU0 Counter Period Register	0x00000000
0x4002C044	TTU0_PER[n]	TTU0 Counter Period Register	0x00000000
0x4002C048	TTU0_PER[n]	TTU0 Counter Period Register	0x00000000
0x4002C04C	TTU0_PER[n]	TTU0 Counter Period Register	0x00000000
0x4002C080	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C084	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C088	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C08C	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C090	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C094	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C098	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C09C	TTU0_DLY[m]	TTU0 Trigger Output Delay Register	0x00000000
0x4002C0C0	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0C4	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0C8	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0CC	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0D0	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0D4	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0D8	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000
0x4002C0DC	TTU0_DGRP[m]	TTU0 Trigger Output Counter Assignment	0x00000000

Table 48-87: CM41X_M4 TWI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40023000	TWI0_CLKDIV	TWI0 SCL Clock Divider Register	0x00000000
0x40023004	TWI0_CTL	TWI0 Control Register	0x00000000
0x40023008	TWI0_SLVCTL	TWI0 Slave Mode Control Register	0x00000000
0x4002300C	TWI0_SLVSTAT	TWI0 Slave Mode Status Register	0x00000000
0x40023010	TWI0_SLVADDR	TWI0 Slave Mode Address Register	0x00000000
0x40023014	TWI0_MSTRCTL	TWI0 Master Mode Control Registers	0x00000000
0x40023018	TWI0_MSTRSTAT	TWI0 Master Mode Status Register	0x00000000
0x4002301C	TWI0_MSTRADDR	TWI0 Master Mode Address Register	0x00000000
0x40023020	TWI0_ISTAT	TWI0 Interrupt Status Register	0x00000000
0x40023024	TWI0_IMSK	TWI0 Interrupt Mask Register	0x00000000
0x40023028	TWI0_FIFOCTL	TWI0 FIFO Control Register	0x00000000
0x4002302C	TWI0_FIFOSTAT	TWI0 FIFO Status Register	0x00000000
0x40023080	TWI0_TXDATA8	TWI0 Tx Data Single-Byte Register	0x00000000
0x40023084	TWI0_TXDATA16	TWI0 Tx Data Double-Byte Register	0x00000000
0x40023088	TWI0_RXDATA8	TWI0 Rx Data Single-Byte Register	0x00000000
0x4002308C	TWI0_RXDATA16	TWI0 Rx Data Double-Byte Register	0x00000000

Table 48-88: CM41X_M4 UART0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x4100A004	UART0_CTL	UART0 Control Register	0x00000000
0x4100A008	UART0_STAT	UART0 Status Register	0x000000A0
0x4100A00C	UART0_SCR	UART0 Scratch Register	0x00000000
0x4100A010	UART0_CLK	UART0 Clock Rate Register	0x0000FFFF
0x4100A014	UART0_IMSK	UART0 Interrupt Mask Register	0x00000000
0x4100A018	UART0_IMSK_SET	UART0 Interrupt Mask Set Register	0x00000000
0x4100A01C	UART0_IMSK_CLR	UART0 Interrupt Mask Clear Register	0x00000000
0x4100A020	UART0_RBR	UART0 Receive Buffer Register	0x00000000
0x4100A024	UART0_THR	UART0 Transmit Hold Register	0x00000000
0x4100A028	UART0_TAIP	UART0 Transmit Address/Insert Pulse Register	0x00000000
0x4100A02C	UART0_TSR	UART0 Transmit Shift Register	0x000007FF

Table 48-88: CM41X_M4 UART0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x4100A030	UART0_RSR	UART0 Receive Shift Register	0x00000000
0x4100A034	UART0_TXCNT	UART0 Transmit Counter Register	0x00000000
0x4100A038	UART0_RXCNT	UART0 Receive Counter Register	0x00000000

Table 48-89: CM41X_M4 UART1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40044004	UART1_CTL	UART1 Control Register	0x00000000
0x40044008	UART1_STAT	UART1 Status Register	0x000000A0
0x4004400C	UART1_SCR	UART1 Scratch Register	0x00000000
0x40044010	UART1_CLK	UART1 Clock Rate Register	0x0000FFFF
0x40044014	UART1_IMSK	UART1 Interrupt Mask Register	0x00000000
0x40044018	UART1_IMSK_SET	UART1 Interrupt Mask Set Register	0x00000000
0x4004401C	UART1_IMSK_CLR	UART1 Interrupt Mask Clear Register	0x00000000
0x40044020	UART1_RBR	UART1 Receive Buffer Register	0x00000000
0x40044024	UART1_THR	UART1 Transmit Hold Register	0x00000000
0x40044028	UART1_TAIP	UART1 Transmit Address/Insert Pulse Register	0x00000000
0x4004402C	UART1_TSR	UART1 Transmit Shift Register	0x000007FF
0x40044030	UART1_RSR	UART1 Receive Shift Register	0x00000000
0x40044034	UART1_TXCNT	UART1 Transmit Counter Register	0x00000000
0x40044038	UART1_RXCNT	UART1 Receive Counter Register	0x00000000

Table 48-90: CM41X_M4 UART2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40045004	UART2_CTL	UART2 Control Register	0x00000000
0x40045008	UART2_STAT	UART2 Status Register	0x000000A0
0x4004500C	UART2_SCR	UART2 Scratch Register	0x00000000
0x40045010	UART2_CLK	UART2 Clock Rate Register	0x0000FFFF
0x40045014	UART2_IMSK	UART2 Interrupt Mask Register	0x00000000
0x40045018	UART2_IMSK_SET	UART2 Interrupt Mask Set Register	0x00000000
0x4004501C	UART2_IMSK_CLR	UART2 Interrupt Mask Clear Register	0x00000000

Table 48-90: CM41X_M4 UART2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40045020	UART2_RBR	UART2 Receive Buffer Register	0x00000000
0x40045024	UART2_THR	UART2 Transmit Hold Register	0x00000000
0x40045028	UART2_TAIP	UART2 Transmit Address/Insert Pulse Register	0x00000000
0x4004502C	UART2_TSR	UART2 Transmit Shift Register	0x000007FF
0x40045030	UART2_RSR	UART2 Receive Shift Register	0x00000000
0x40045034	UART2_TXCNT	UART2 Transmit Counter Register	0x00000000
0x40045038	UART2_RXCNT	UART2 Receive Counter Register	0x00000000

Table 48-91: CM41X_M4 UART3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40046004	UART3_CTL	UART3 Control Register	0x00000000
0x40046008	UART3_STAT	UART3 Status Register	0x000000A0
0x4004600C	UART3_SCR	UART3 Scratch Register	0x00000000
0x40046010	UART3_CLK	UART3 Clock Rate Register	0x0000FFFF
0x40046014	UART3_IMSK	UART3 Interrupt Mask Register	0x00000000
0x40046018	UART3_IMSK_SET	UART3 Interrupt Mask Set Register	0x00000000
0x4004601C	UART3_IMSK_CLR	UART3 Interrupt Mask Clear Register	0x00000000
0x40046020	UART3_RBR	UART3 Receive Buffer Register	0x00000000
0x40046024	UART3_THR	UART3 Transmit Hold Register	0x00000000
0x40046028	UART3_TAIP	UART3 Transmit Address/Insert Pulse Register	0x00000000
0x4004602C	UART3_TSR	UART3 Transmit Shift Register	0x000007FF
0x40046030	UART3_RSR	UART3 Receive Shift Register	0x00000000
0x40046034	UART3_TXCNT	UART3 Transmit Counter Register	0x00000000
0x40046038	UART3_RXCNT	UART3 Receive Counter Register	0x00000000

Table 48-92: CM41X_M4 UART4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40047004	UART4_CTL	UART4 Control Register	0x00000000
0x40047008	UART4_STAT	UART4 Status Register	0x000000A0
0x4004700C	UART4_SCR	UART4 Scratch Register	0x00000000

Table 48-92: CM41X_M4 UART4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x40047010	UART4_CLK	UART4 Clock Rate Register	0x0000FFFF
0x40047014	UART4_IMSK	UART4 Interrupt Mask Register	0x00000000
0x40047018	UART4_IMSK_SET	UART4 Interrupt Mask Set Register	0x00000000
0x4004701C	UART4_IMSK_CLR	UART4 Interrupt Mask Clear Register	0x00000000
0x40047020	UART4_RBR	UART4 Receive Buffer Register	0x00000000
0x40047024	UART4_THR	UART4 Transmit Hold Register	0x00000000
0x40047028	UART4_TAIP	UART4 Transmit Address/Insert Pulse Register	0x00000000
0x4004702C	UART4_TSR	UART4 Transmit Shift Register	0x000007FF
0x40047030	UART4_RSR	UART4 Receive Shift Register	0x00000000
0x40047034	UART4_TXCNT	UART4 Transmit Counter Register	0x00000000
0x40047038	UART4_RXCNT	UART4 Receive Counter Register	0x00000000

Table 48-93: CM41X_M4 WDOG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x41005000	WDOG0_CTL	WDOG0 Control Register	0x00000AD0
0x41005004	WDOG0_CNT	WDOG0 Count Register	0x00000000
0x41005008	WDOG0_STAT	WDOG0 Watchdog Timer Status Register	0x00000000

Table 48-94: CM41X_M4 WDOG1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x40011000	WDOG1_CTL	WDOG1 Control Register	0x00000AD0
0x40011004	WDOG1_CNT	WDOG1 Count Register	0x00000000
0x40011008	WDOG1_STAT	WDOG1 Watchdog Timer Status Register	0x00000000