ADSP-BF70x Blackfin+ Processor Hardware Reference

Revision 1.0, October 2016

Part Number
82-100124-01

,

ANALOG DEVICES

Notices

Copyright Information

Printed in the USA.

Disclaimer

Trademark and Service Mark Notice

Contents

Clock Generation Unit (CGU)

System Protection Unit (SPU)

System Memory Protection Unit (SMPU)

System Security

Dynamic Power Management (DPM)

System Event Controller (SEC)

Trigger Routing Unit (TRU)

Static Memory Controller (SMC)

L2 System Memory

Dynamic Memory Controller (DMC)

One-Time Programmable Memory Controller (OTPC)

Cyclic Redundancy Check (CRC)

Security Packet Engine (PKTE)

Public Key Accelerator (PKA)

Public Key Interrupt Controller (PKIC)

True Random Number Generator (TRNG)

Direct Memory Access (DMA)

General-Purpose Ports (PORT)

General-Purpose Timer (TIMER)

Watchdog Timer (WDOG)

Real Time Clock (RTC)

General-Purpose Counter (CNT)

Universal Asynchronous Receiver/Transmitter (UART)

Two-Wire Interface (TWI)

Controller Area Network (CAN)

Universal Serial Bus (USB)

Serial Peripheral Interface (SPI)

Serial Peripheral Interface Host Port (SPIHP)

Serial Port (SPORT)

Enhanced Parallel Peripheral Interface (EPPI)

Mobile Storage Interface (MSI)

Housekeeping ADC (HADC)

Reset Control Unit (RCU)

Boot ROM and Booting the Processor

System Debug and Trace Unit (DBG)

System Watchpoint Unit (SWU)

ADSP-BF70x Register List

1 Preface

Purpose of This Manual

The *ADSP-BF70x Blackfin+ Processor Hardware Reference* provides architectural information about the ADSP-BF70x 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 programming information, see the *Blackfin+ Processor Programming Reference*. For timing, electrical, and package specifications, see the *ADSP-BF70x Blackfin+ Processor Data Sheet*.

Intended Audience

The primary audience for this manual is a programmer who is familiar with Analog Devices processors. The manual assumes the audience has a working knowledge of the appropriate processor architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors can use this manual, but should supplement it with other texts, such as programming reference books and data sheets, that describe their target architecture.

What's New in This Manual

This revision (1.0) is the first publicly released version of the *ADSP-BF70x Blackfin+ Processor Hardware Reference*. This revision includes corrected errata associated with this processor. Also, this revision adds an MMR register appendix.

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)
- Phone questions to *1-800-ANALOGD* (USA only)
- 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 and CrossCore Embedded Studio online Help system.

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 supported by the CrossCore Embedded Studio® development tools suite.

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-2106x, 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.

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.

2 System Crossbars (SCB)

Modern Systems-on-Chip (SoCs) contain 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
- Protection model (privileged or secure) support for selective bus interconnect protection
- QoS based arbitration model
- Programmable quality of service

SCB Functional Description

The following sections provide a functional description of the SCB.

- SCB Architectural Concepts

ADSP-BF70x SCB Register List

Table 2-1: ADSP-BF70x SCB Register List

Name	Description
SCB_IB6_SYNC	Interface Block Sync Register
SCB_IB7_SYNC	DDR Interface Block Sync Mode
SCB_MST00_RQOS	Master 00 Read Quality of Service Register
SCB_MST00_WQOS	Master 00 Write Quality of Service Register
SCB_MST01_RQOS	Master 01 Read Quality of Service Register
SCB_MST01_WQOS	Master 01 Write Quality of Service Register
SCB_MST02_RQOS	Master 02 Read Quality of Service Register
SCB_MST02_WQOS	Master 02 Write Quality of Service Register
SCB_MST03_RQOS	Master 03 Read Quality of Service Register
SCB_MST03_WQOS	Master 03 Write Quality of Service Register
SCB_MST04_RQOS	Master 04 Read Quality of Service Register
SCB_MST04_WQOS	Master 04 Write Quality of Service Register
SCB_MST05_RQOS	Master 05 Read Quality of Service Register
SCB_MST05_WQOS	Master 05 Write Quality of Service Register
SCB_MST06_RQOS	Master 06 Read Quality of Service Register
SCB_MST06_WQOS	Master 06 Write Quality of Service Register
SCB_MST07_RQOS	Master 07 Read Quality of Service Register
SCB_MST07_WQOS	Master 07 Write Quality of Service Register
SCB_MST08_RQOS	Master 08 Read Quality of Service Register
SCB_MST08_WQOS	Master 08 Write Quality of Service Register
SCB_MST09_RQOS	Master 09 Read Quality of Service Register
SCB_MST09_WQOS	Master 09 Write Quality of Service Register
SCB_MST10_RQOS	Master 10 Read Quality of Service Register
SCB_MST10_WQOS	Master 10 Write Quality of Service Register
SCB_MST11_RQOS	Master 11 Read Quality of Service Register
SCB_MST11_WQOS	Master 11 Write Quality of Service Register
SCB_MST12_RQOS	Master 12 Read Quality of Service Register
SCB_MST12_WQOS	Master 12 Write Quality of Service Register

Table 2-1: ADSP-BF70x SCB Register List (Continued)

Name	Description
SCB_MST13_RQOS	Master 13 Read Quality of Service Register
SCB_MST13_WQOS	Master 13 Write Quality of Service Register
SCB_MST14_RQOS	Master 14 Read Quality of Service Register
SCB_MST14_WQOS	Master 14 Write Quality of Service Register
SCB_MST15_RQOS	Master 15 Read Quality of Service Register
SCB_MST15_WQOS	Master 15 Write Quality of Service Register
SCB_MST16_RQOS	Master 16 Read Quality of Service Register
SCB_MST16_WQOS	Master 16 Write Quality of Service Register
SCB_MST17_RQOS	Master 17 Read Quality of Service Register
SCB_MST17_WQOS	Master 17 Write Quality of Service Register
SCB_MST18_RQOS	Master 18 Read Quality of Service Register
SCB_MST18_WQOS	Master 18 Write Quality of Service Register
SCB_MST19_RQOS	Master 19 Read Quality of Service Register
SCB_MST19_WQOS	Master 19 Write Quality of Service Register
SCB_MST20_RQOS	Master 20 Read Quality of Service Register
SCB_MST20_WQOS	Master 20 Write Quality of Service Register
SCB_MST21_RQOS	Master 21 Read Quality of Service Register
SCB_MST21_WQOS	Master 21 Write Quality of Service Register
SCB_MST22_RQOS	Master 22 Read Quality of Service Register
SCB_MST22_WQOS	Master 22 Write Quality of Service Register
SCB_MST23_RQOS	Master 23 Read Quality of Service Register
SCB_MST23_WQOS	Master 23 Write Quality of Service Register
SCB_MST24_RQOS	Master 24 Read Quality of Service Register
SCB_MST24_WQOS	Master 24 Write Quality of Service Register
SCB_MST25_RQOS	Master 25 Read Quality of Service Register
SCB_MST25_WQOS	Master 25 Write Quality of Service Register
SCB_MST26_RQOS	Master 26 Read Quality of Service Register
SCB_MST26_WQOS	Master 26 Write Quality of Service Register
SCB_MST27_RQOS	Master 27 Read Quality of Service Register
SCB_MST27_WQOS	Master 27 Write Quality of Service Register
SCB_MST28_RQOS	Master 28 Read Quality of Service Register

Table 2-1: ADSP-BF70x SCB Register List (Continued)

Name	Description
SCB_MST28_WQOS	Master 28 Write Quality of Service Register
SCB_MST29_RQOS	Master 29 Read Quality of Service Register
SCB_MST29_WQOS	Master 29 Write Quality of Service Register
SCB_MST30_SYNC	Interface Block IB4 Sync Mode
SCB_MST31_SYNC	Interface Block IB5 Sync Mode

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), the processor cores, and the L1 code fill block.

Slaves

Slaves respond to the requests raised by any SCB Master. The system bus slaves include on-chip and off-chip memory devices and controllers such as L1 SRAM , and the System Memory-Mapped registers (MMRs). Each system bus slave has its own latency characteristics, operating in a given clock domain. For example, an L1 SRAM access is faster than memory accesses.

Master Interface

The master interface connects the slave to the SCB which in turn connects to the master on the other end. In effect, the master interface connects the slave to the master concealing underlying cross bar details.

Slave Interface

The slave interface provides the interface for the masters to the SCB.

SCB Block Diagrams

The SCB architectural model is illustrated in the ***SCB Overview*** figure. This figure shows a high-level overview of the SCB and associated connections to system masters and slaves. 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.

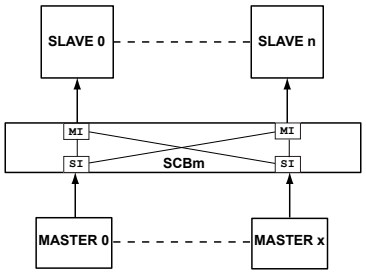

Figure 2-1: SCB Overview

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, reaches its intended slave.

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

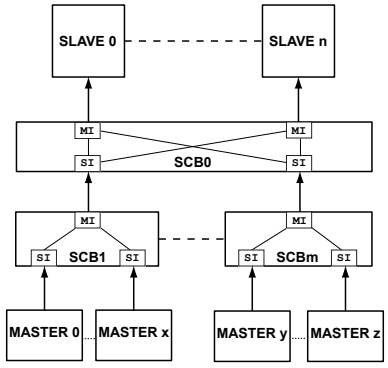

Figure 2-2: SCB Hierarchy Overview

NOTE: For an overall diagram of all SCB interconnections, see the ADSP-BF70x SCB Block Diagram.

ADSP-BF70x SCB Block Diagram

The following figure shows the SCB block diagram for the ADSP-BF70x processors.

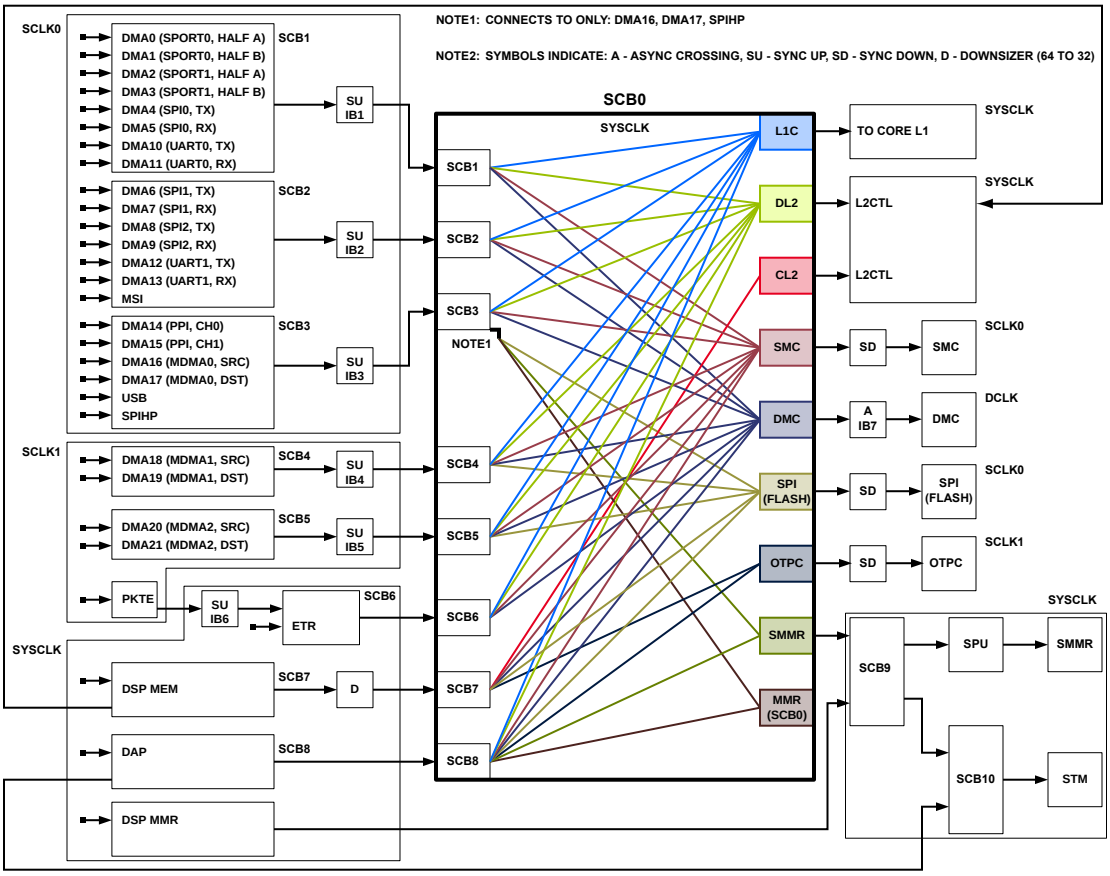

Figure 2-3: ADSP-BF70x SCB Block Diagram

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 (such as USB). The connections also support memory-mapped register access for internal memory (L1 and L2) and for external memory (DDR, FLASH, and others).

  The slave interface of the crossbar (where the masters such as DMA connect to) performs two functions. The first function is arbitration. SCBx handles arbitration. The second function is clock conversion. IBx handles clock conversion. The programmable QoS registers can be viewed as being associated with the SCBx. For example, the programmable QoS registers for DMA 14-17, USB, SPIHP can be viewed as residing in SCB3. For example, whenever a transaction is received at DMA14, the programmed QoS value is associated with that transaction and is arbitrated with the rest of the masters at SCB3.

  IB1–3 perform clock domain crossing from *SCLK0* to *SYSCLK*. *SCLK0* is always assumed to be an integer ratio (1:n) to *SYSCLK* and so passes through a one stage synchronizer. Therefore, there are no programmable registers for IB1–3. IB4–5 and PKTE IB perform clock domain crossing from *SCLK1* to *SYSCLK*. The clock domain crossing from *SCLK1* to *SYSCLK* is programmable through the sync mode register associated with IB4,

IB5, and PKTE IB. Similarly, the IB preset at the DDR boundary performs clock domain crossing from *SYSCLK* to *DCLK*. This crossing is also programmable through the sync mode register present in DDR IB.

- Most of the peripherals, MDMA0 stream, and their SCBs are in the *SCLK0* domain. MDMA1 and MDMA2 streams and their SCBs are in *SCLK1* domain. Crypto is in SCLK1 domain. ETR, DAP, SCB0, SCB5, DAP, DSP_MMR, SCB10 are in *SYSCLK* domain. The processor core is in its own clock domain.

  Synchronizations across clock domains affect the SCB performance. Interface blocks IB4 (MDMA1), IB5 (MDMA2), IB6 (PKTE Crypto), and IB7 (DDR) are programmed depending upon the clock ratios of the two clock domains.

- 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.

  MDMA channels are unidirectional. For example, MDMA0 SRC is read-only and MDMA0 DST is write-only. Access to the MMR space of SCB0 is allowed only in secure mode. The MMR space of SCB0 has registers for programming the QoS of various masters and controlling the clock domain crossing.

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

DMA0-DMA21

Indicate DMA channels for peripherals supporting DMA transfers.

SCB0-10

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

SCLK0, SCLK1, SYSCLK, DCLK

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, L2

Indicate on-core (L1) internal memory and off-core (L2) internal memory.

SMC, DMC

Indicate static memory controller (SMC), off-core (L2) memory controller, and dynamic memory controller (DMC) interfaces.

PPI0

Indicates the parallel peripheral port interface using either internal or external frame sync.

SP0 A/B, SP1 A/B

Indicate serial port interfaces and their full-duplex halves.

SPI0, SPI1 - RX/TX

Indicate serial peripheral interfaces ports with receive or transmit paths.

MSI

Indicates the mass storage interface (MSI) interface.

DAP, ETR

Indicate Debug Access Port (DAP) and Embedded Trace Router (ETR) which provide access to debug and trace capabilities.

SPU

Indicates the system protection unit (SPU).

MDMA0, MDMA1, MDMA2

Indicate memory DMA 0 through 2 interfaces.

USB

Indicates the universal serial bus (USB) interface.

SMMR

Indicates the system memory-mapped register interface.

ADSP-BF70x SCB Bus Master IDs

The SCB bus master ID tables indicate which masters are connected to each of the slave ports of SCBn. The tables also indicate the precise value of the ID as seen by the slave. These values are useful for SWU programming.

NOTE: For an overall diagram of all SCB interconnections, see the ADSP-BF70x SCB Block Diagram.

Table 2-2: ADSP-BF70x SCB Bus Master IDs

min-SCB	Master	Hexadecimal Values
SCB1	DMA0	0x0, 0x8 (Descriptor fetch)
	DMA1	0x1, 0x9 (Descriptor fetch)

Table 2-2: ADSP-BF70x SCB Bus Master IDs (Continued)

min-SCB	Master	Hexadecimal Values
	DMA2	0x2, 0xA (Descriptor fetch)
	DMA3	0x3, 0xB (Descriptor fetch)
	DMA4	0x4, 0xC (Descriptor fetch)
	DMA5	0x5, 0xD (Descriptor fetch)
	DMA10	0x6, 0xE (Descriptor fetch)
	DMA11	0x7, 0xF (Descriptor fetch)
SCB2	DMA6	0x45, 0x4D (Descriptor fetch)
	DMA7	0x46, 0x4E (Descriptor fetch)
	DMA8	0x40, 0x48 (Descriptor fetch)
	DMA9	0x41, 0x49 (Descriptor fetch)
	DMA12	0x42, 0x4A (Descriptor fetch)
	DMA13	0x43, 0x4B (Descriptor fetch)
	MSI	0x044, 0x04C (Descriptor fetch)
SCB3	DMA14	0x80, 0x88 (Descriptor fetch)
	DMA15	0x81, 0x89 (Descriptor fetch)
	DMA16	0x82 0x8A (Descriptor fetch)
	DMA17	0x83, 0x8B (Descriptor fetch)
	USB	0x84,0x8C (Descriptor fetch)
	SPIHP	0x085
SCB4	DMA18	0xC, 0xC8 (Descriptor fetch)
	DMA19	0xC1, 0xC9 (Descriptor fetch)
SCB5	DMA20	0x100, 0x108 (Descriptor fetch)
	DMA21	0x101, 0x109 (Descriptor fetch)
SCB6	PKTE	0x1C0, 0x1C8 (Descriptor fetch)
	ETR	0x1C1
DSP MEM [Core Access]		0x140 (L2 ROM), 0x148 (L2 SRAM), 0x150 (Reserved), 0x158 (System MMR),

Table 2-2: ADSP-BF70x SCB Bus Master IDs (Continued)

min-SCB	Master	Hexadecimal Values
		0x160 (SPI Flash), 0x168 (SMC), 0x170 (DMC), 0x178 (IO or non-speculative accesses for example FIFO reads)
DSP MMR		0x181
DAP		0x200

The ***ADSP-BF70x MSTxx IDs***s table shows the MSTxx IDs associated with various masters. The IDs are used for naming the corresponding SCB registers (such as QOS registers, sync mode registers etc.). For example, the `SCB_MST00_RQOS` register corresponds to the master DMA0 and the `SCB_MST22_RQOS` register corresponds to the master USB.

Table 2-3: ADSP-BF70x MSTxx IDs

Master	MSTxx ID
DMA0	0
DMA1	1
DMA2	2
DMA3	3
DMA4	4
DMA5	5
DMA6	6
DMA7	7
DMA8	8
DMA9	9
DMA10	10
DMA11	11
DMA12	12
DMA13	13
DMA14	14
DMA15	15
DMA16	16
DMA17	17

Table 2-3: ADSP-BF70x MSTxx IDs (Continued)

Master	MSTxx ID
DMA18	18
DMA19	19
DMA20	20
DMA21	21
USB	22
DSP MEM	23
DSP MMR	24
MSI	25
PKTE	26
ETR	27
SPIH	28
DAP	29
IB4	30
IB5	31

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 Mbps 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.

See ADSP-BF70x SCB Block Diagram.

All other SCBs in the chip connect to SCB0 through different slave interfaces. Other primary masters (DMAs, cores, and so on) in the system are distributed across these small SCBs. For a given SCB, all the master and slaves share the total bandwidth of the SCB. (Only SCB0 is the exception). Since different DMA channels are scattered across different SCBs (SCB1, SCB2 SCB3, and so on), they do not conflict for the bandwidth as long as they are in different SCB and are accessing different slaves. SCB0 allows for concurrent data transfer between multiple bus

masters and multiple bus slaves, providing flexibility, and full-duplex operation. For example, the data transfer between SCB4 (one of the MDMA channels), and SMC controller (accessing SRAM memory) can happen in parallel to SCB2 (SPI RX/TX DMA) accessing memory mapped SPI memory with both the transfers occurring at up to 400 MBPS. If system accesses are carefully architected, SCB has a potential of providing sufficient sustained bandwidth in the end system.

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.

Clock Domain Synchronization

Most of the masters in the system operate at the same clock as the SCB, which is SCLK. So, there is no need to synchronize. The M4 core and L1 code fill blocks (used for cache fill from SPI flash memory or SMC memory) operate in CCLK domain. The SCB provides the option to program the different synchronization schemes for these masters through the sync mode registers (SCB_IB6_SYNC, SCB_IB7_SYNC).

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

ADSP-BF70x SCB Programming Model

The following sections provide information for programming SCB properly.

Programming SCB Arbitration

Each slave interface has a QoS value (priority) associated with both read and write channels. These values are 4 bits present in the SCB0_MSTx_RQOS and SCB0_MSTx_WQOS registers. At the entry point to the infrastructure, all transactions are allocated this 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 that require arbitration, then the network uses a Least Recently Granted (LRG) algorithm. At each switch, the master with the highest QoS acquires access and that switch output takes the QoS value of the winner for that transaction. At the next switch slave

interface, the master uses the QoS value of the winner. QoS can have values from 0 (lowest priority) to 15 (highest priority).

For example in the following figure, ***SCB Arbitration***:

1. At SCB1, masters (1, 2, 3) have RQOS values of (6, 4, 2)
2. At SCB2, masters (4, 5, 6) have RQOS values of (12, 13, 1)

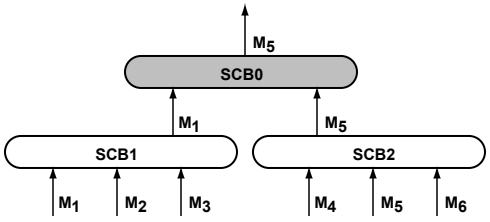

Figure 2-4: SCB Arbitration

In this case, master 1 wins at SCB1, and master 5 wins at SCB2. However, in a perfect competition at SCB0, masters 4 and 5 had the highest overall RQOS values. Masters 4 and 5 would have fought for arbitration directly at SCB0. However, because of the mini-SCBs, master 1, at a much lower RQOS value, is able to win against master 4 and make it all the way to SCB0.

Programming Clock Domain Crossing Registers

In addition to the QoS registers, the interconnect has a set of registers to program the clock domain crossing in the interface blocks. The clock domain crossing can be programmed to be one of the following:

- SYNC 1:1
- SYNC 1:n
- SYNC n:1
- ASYNC

The clock domain crossing can be programmed at:

- the DMC interface (IB7) between DCLK and the SYSCLK
- the MDMA1 and MDMA2 interface block (IB4, IB5) between SCLK1 and SYSCLK
- the PKTE synchronizer (IB6) between SCLK1 and SYSCLK clocks

The clock domain crossing defaults to ASYNC, where no relation between the two clocks is assumed.

At IB7, the sync ratio is programmed as:

- SYNC 1:1 when SYSCLK and DCLK are the same
- SYNC n:1 when the SYSCLK frequency is an integer multiple of DCLK SYNC 1:n
- SYNC m:n selection is not supported at the IB7

Similarly, at IB4, IB5, and IB6, the clock domain crossing is specified to SYNC 1:1, when SCLK1 clock and SYSCLK are the same. When the SYSCLK clock frequency is an integer multiple of SCLK1 frequency, the clock domain crossing is specified as SYNC 1:n, SYNC n:1, and SYNC m:n. SYNC m:n is not supported at IB4, IB5, and IB6 for the ADSP-BF70x processors.

NOTE: The clock domain crossing between SCLK0 and SYSCLK clock is always synchronous n:1. As discussed, the clock domain crossing defaults to ASYNC. To change the clock domain crossing mode, follow the actions described in the ***Changing Clock Domain Crossing Modes*** table.

Table 2-4: Changing Clock Domain Crossing Modes

Original Mode	Required Mode	Action
ASYNC	Any other mode	Change the clocks then change the MMR register
Any Mode	ASYNC	Change the MMR register then change clocks to ASYNC.
m:n	1:1	Change the clocks, then change the register.
1:1	m:n	Change the register, then change the clocks.

ADSP-BF70x SCB Register Descriptions

(SCB) contains the following registers.

Table 2-5: ADSP-BF70x SCB Register List

Name	Description
SCB_IB6_SYNC	Interface Block Sync Register
SCB_IB7_SYNC	DDR Interface Block Sync Mode
SCB_MST00_RQOS	Master 00 Read Quality of Service Register
SCB_MST00_WQOS	Master 00 Write Quality of Service Register
SCB_MST01_RQOS	Master 01 Read Quality of Service Register
SCB_MST01_WQOS	Master 01 Write Quality of Service Register
SCB_MST02_RQOS	Master 02 Read Quality of Service Register
SCB_MST02_WQOS	Master 02 Write Quality of Service Register
SCB_MST03_RQOS	Master 03 Read Quality of Service Register
SCB_MST03_WQOS	Master 03 Write Quality of Service Register
SCB_MST04_RQOS	Master 04 Read Quality of Service Register
SCB_MST04_WQOS	Master 04 Write Quality of Service Register
SCB_MST05_RQOS	Master 05 Read Quality of Service Register
SCB_MST05_WQOS	Master 05 Write Quality of Service Register
SCB_MST06_RQOS	Master 06 Read Quality of Service Register

Table 2-5: ADSP-BF70x SCB Register List (Continued)

Name	Description
SCB_MST06_WQOS	Master 06 Write Quality of Service Register
SCB_MST07_RQOS	Master 07 Read Quality of Service Register
SCB_MST07_WQOS	Master 07 Write Quality of Service Register
SCB_MST08_RQOS	Master 08 Read Quality of Service Register
SCB_MST08_WQOS	Master 08 Write Quality of Service Register
SCB_MST09_RQOS	Master 09 Read Quality of Service Register
SCB_MST09_WQOS	Master 09 Write Quality of Service Register
SCB_MST10_RQOS	Master 10 Read Quality of Service Register
SCB_MST10_WQOS	Master 10 Write Quality of Service Register
SCB_MST11_RQOS	Master 11 Read Quality of Service Register
SCB_MST11_WQOS	Master 11 Write Quality of Service Register
SCB_MST12_RQOS	Master 12 Read Quality of Service Register
SCB_MST12_WQOS	Master 12 Write Quality of Service Register
SCB_MST13_RQOS	Master 13 Read Quality of Service Register
SCB_MST13_WQOS	Master 13 Write Quality of Service Register
SCB_MST14_RQOS	Master 14 Read Quality of Service Register
SCB_MST14_WQOS	Master 14 Write Quality of Service Register
SCB_MST15_RQOS	Master 15 Read Quality of Service Register
SCB_MST15_WQOS	Master 15 Write Quality of Service Register
SCB_MST16_RQOS	Master 16 Read Quality of Service Register
SCB_MST16_WQOS	Master 16 Write Quality of Service Register
SCB_MST17_RQOS	Master 17 Read Quality of Service Register
SCB_MST17_WQOS	Master 17 Write Quality of Service Register
SCB_MST18_RQOS	Master 18 Read Quality of Service Register
SCB_MST18_WQOS	Master 18 Write Quality of Service Register
SCB_MST19_RQOS	Master 19 Read Quality of Service Register
SCB_MST19_WQOS	Master 19 Write Quality of Service Register
SCB_MST20_RQOS	Master 20 Read Quality of Service Register
SCB_MST20_WQOS	Master 20 Write Quality of Service Register
SCB_MST21_RQOS	Master 21 Read Quality of Service Register
SCB_MST21_WQOS	Master 21 Write Quality of Service Register

Table 2-5: ADSP-BF70x SCB Register List (Continued)

Name	Description
SCB_MST22_RQOS	Master 22 Read Quality of Service Register
SCB_MST22_WQOS	Master 22 Write Quality of Service Register
SCB_MST23_RQOS	Master 23 Read Quality of Service Register
SCB_MST23_WQOS	Master 23 Write Quality of Service Register
SCB_MST24_RQOS	Master 24 Read Quality of Service Register
SCB_MST24_WQOS	Master 24 Write Quality of Service Register
SCB_MST25_RQOS	Master 25 Read Quality of Service Register
SCB_MST25_WQOS	Master 25 Write Quality of Service Register
SCB_MST26_RQOS	Master 26 Read Quality of Service Register
SCB_MST26_WQOS	Master 26 Write Quality of Service Register
SCB_MST27_RQOS	Master 27 Read Quality of Service Register
SCB_MST27_WQOS	Master 27 Write Quality of Service Register
SCB_MST28_RQOS	Master 28 Read Quality of Service Register
SCB_MST28_WQOS	Master 28 Write Quality of Service Register
SCB_MST29_RQOS	Master 29 Read Quality of Service Register
SCB_MST29_WQOS	Master 29 Write Quality of Service Register
SCB_MST30_SYNC	Interface Block IB4 Sync Mode
SCB_MST31_SYNC	Interface Block IB5 Sync Mode

Interface Block Sync Register

This register is used to program the clock domain crossing in the interface blocks.

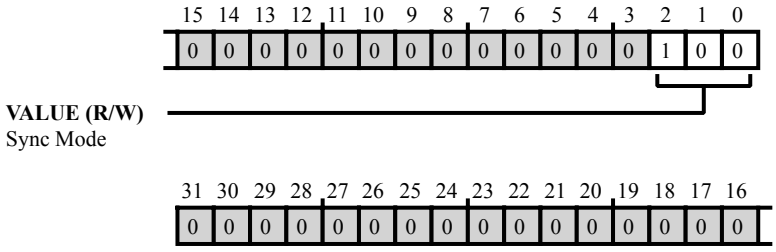

Figure 2-5: SCB_IB6_SYNC Register Diagram

Table 2-6: SCB_IB6_SYNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	VALUE	Sync Mode. The `SCB_IB6_SYNC.VALUE` bit field is used to program the clock domain crossing in the interface blocks.	
		0	1:1
		1	Sync n:1
		2	Sync 1:n
		3	SYNC m:n
		4	ASYNC (Default)

DDR Interface Block Sync Mode

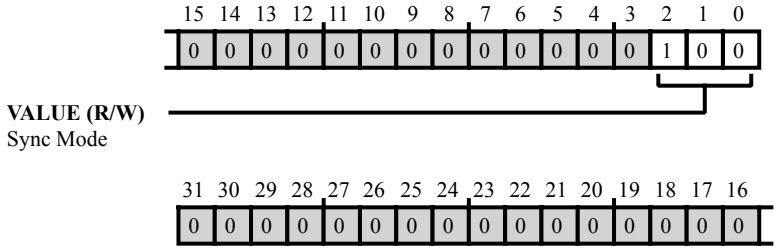

Figure 2-6: SCB_IB7_SYNC Register Diagram

Table 2-7: SCB_IB7_SYNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2:0 (R/W)	VALUE	Sync Mode.

Master 00 Read Quality of Service Register

The SCB_MST00_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

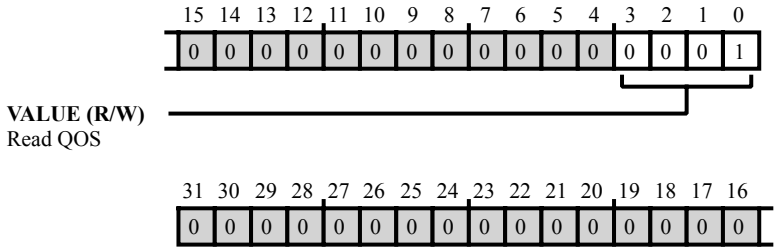

Figure 2-7: SCB_MST00_RQOS Register Diagram

Table 2-8: SCB_MST00_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST00_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 00 Write Quality of Service Register

The SCB_MST00_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

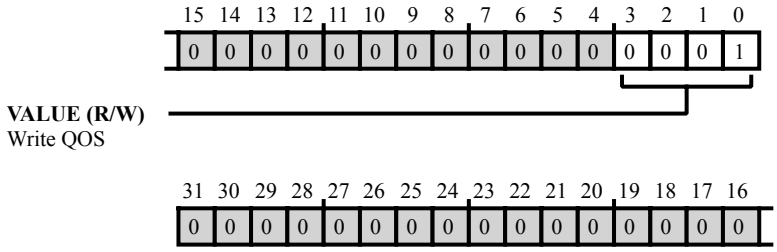

Figure 2-8: SCB_MST00_WQOS Register Diagram

Table 2-9: SCB_MST00_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST00_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 01 Read Quality of Service Register

The SCB_MST01_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

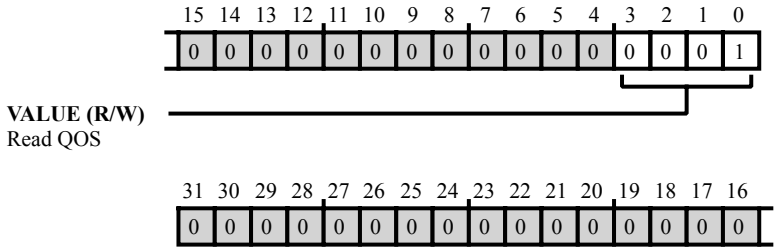

Figure 2-9: SCB_MST01_RQOS Register Diagram

Table 2-10: SCB_MST01_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST01_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 01 Write Quality of Service Register

The SCB_MST01_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

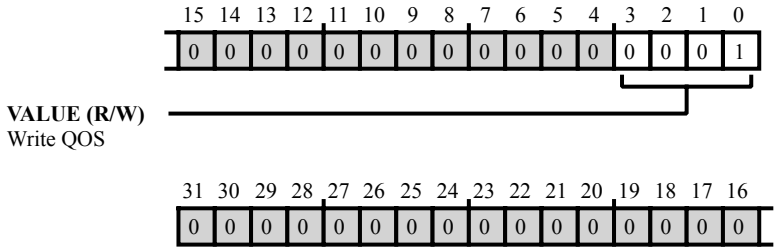

Figure 2-10: SCB_MST01_WQOS Register Diagram

Table 2-11: SCB_MST01_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST01_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 02 Read Quality of Service Register

The SCB_MST02_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

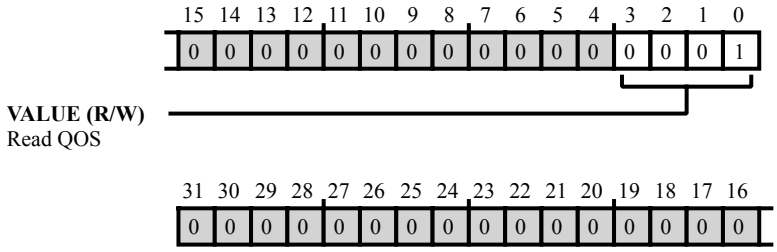

Figure 2-11: SCB_MST02_RQOS Register Diagram

Table 2-12: SCB_MST02_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST02_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 02 Write Quality of Service Register

The SCB_MST02_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

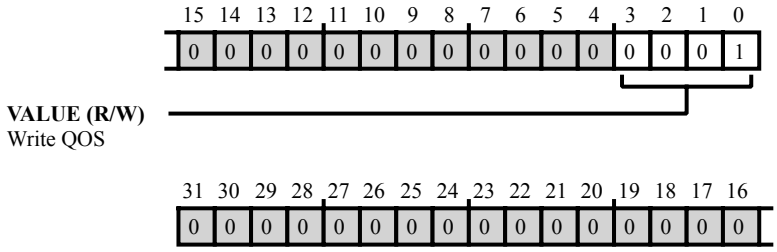

Figure 2-12: SCB_MST02_WQOS Register Diagram

Table 2-13: SCB_MST02_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST02_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 03 Read Quality of Service Register

The SCB_MST03_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

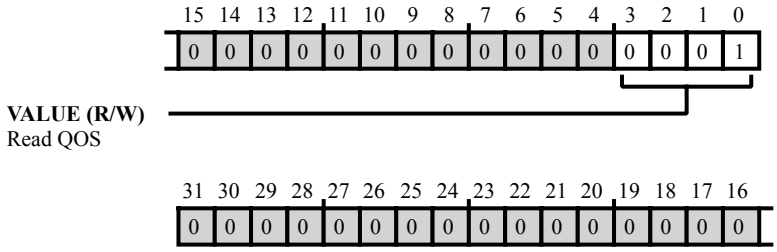

Figure 2-13: SCB_MST03_RQOS Register Diagram

Table 2-14: SCB_MST03_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST03_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 03 Write Quality of Service Register

The SCB_MST03_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

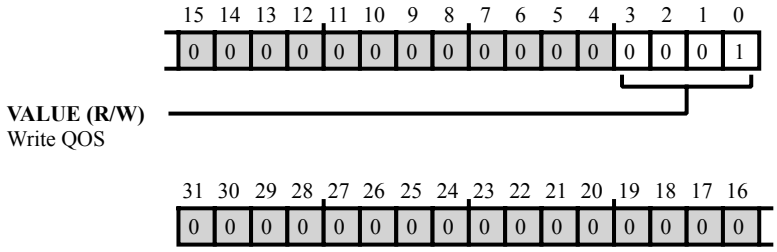

Figure 2-14: SCB_MST03_WQOS Register Diagram

Table 2-15: SCB_MST03_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST03_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 04 Read Quality of Service Register

The SCB_MST04_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

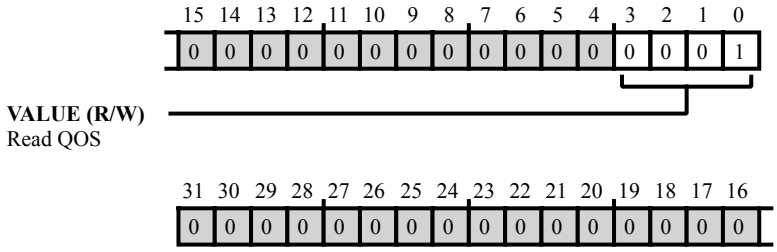

Figure 2-15: SCB_MST04_RQOS Register Diagram

Table 2-16: SCB_MST04_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST04_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 04 Write Quality of Service Register

The SCB_MST04_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

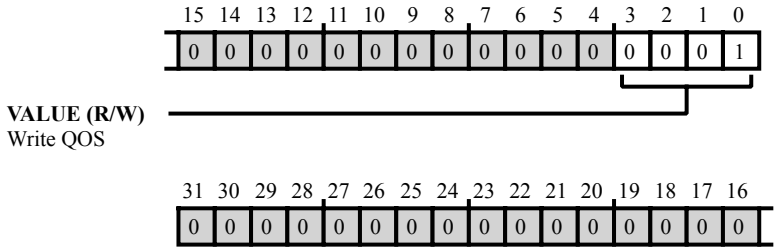

Figure 2-16: SCB_MST04_WQOS Register Diagram

Table 2-17: SCB_MST04_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST04_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 05 Read Quality of Service Register

The SCB_MST05_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

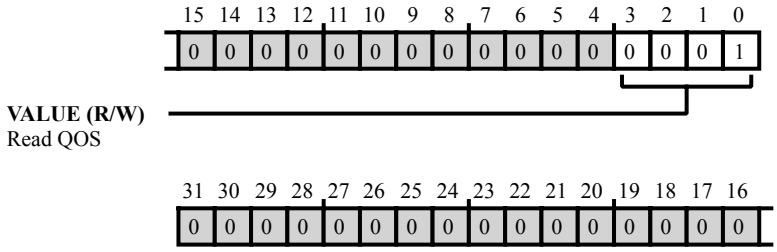

Figure 2-17: SCB_MST05_RQOS Register Diagram

Table 2-18: SCB_MST05_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST05_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 05 Write Quality of Service Register

The SCB_MST05_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

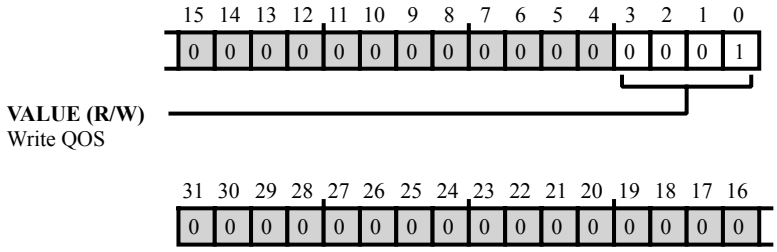

Figure 2-18: SCB_MST05_WQOS Register Diagram

Table 2-19: SCB_MST05_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST05_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 06 Read Quality of Service Register

The SCB_MST06_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

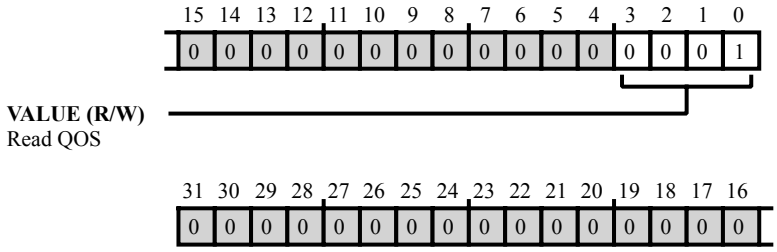

Figure 2-19: SCB_MST06_RQOS Register Diagram

Table 2-20: SCB_MST06_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST06_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 06 Write Quality of Service Register

The SCB_MST06_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

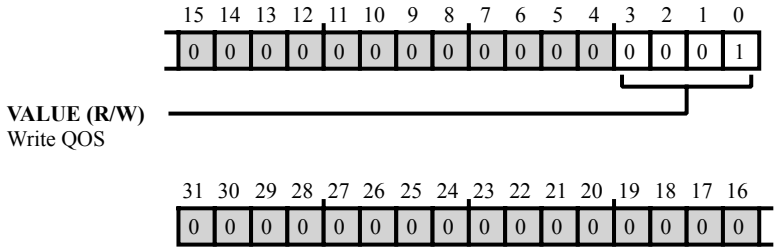

Figure 2-20: SCB_MST06_WQOS Register Diagram

Table 2-21: SCB_MST06_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST06_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 07 Read Quality of Service Register

The SCB_MST07_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

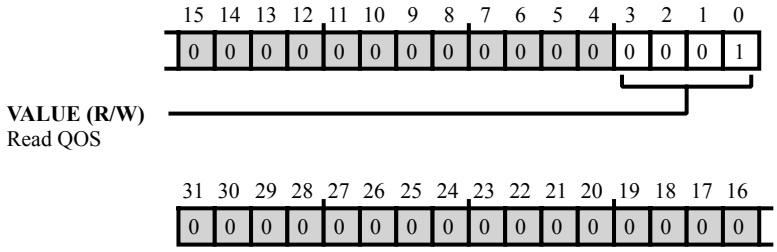

Figure 2-21: SCB_MST07_RQOS Register Diagram

Table 2-22: SCB_MST07_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST07_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 07 Write Quality of Service Register

The SCB_MST07_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

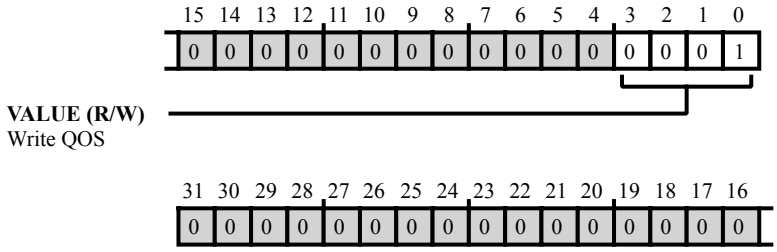

Figure 2-22: SCB_MST07_WQOS Register Diagram

Table 2-23: SCB_MST07_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST07_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 08 Read Quality of Service Register

The SCB_MST08_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

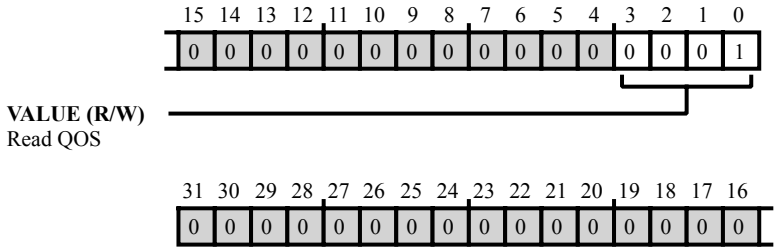

Figure 2-23: SCB_MST08_RQOS Register Diagram

Table 2-24: SCB_MST08_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST08_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 08 Write Quality of Service Register

The SCB_MST08_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

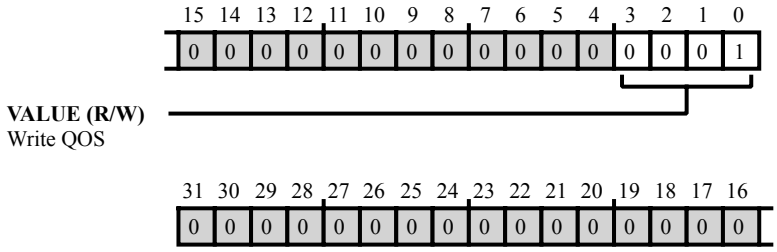

Figure 2-24: SCB_MST08_WQOS Register Diagram

Table 2-25: SCB_MST08_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST08_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 09 Read Quality of Service Register

The SCB_MST09_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

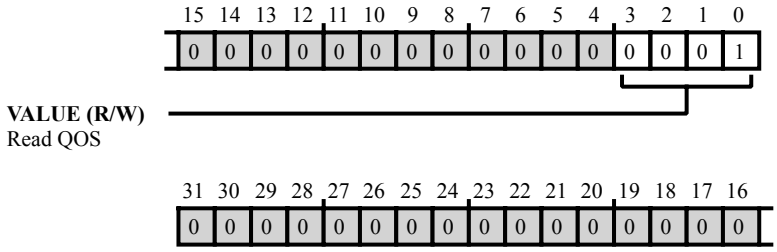

Figure 2-25: SCB_MST09_RQOS Register Diagram

Table 2-26: SCB_MST09_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST09_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 09 Write Quality of Service Register

The SCB_MST09_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

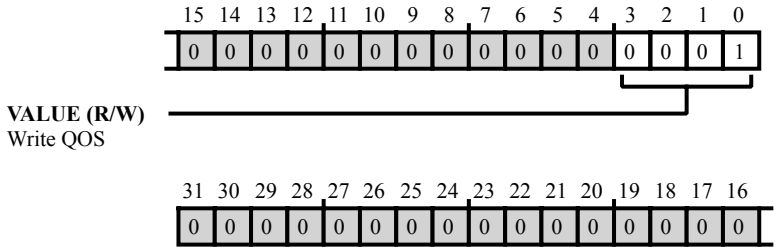

Figure 2-26: SCB_MST09_WQOS Register Diagram

Table 2-27: SCB_MST09_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST09_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 10 Read Quality of Service Register

The SCB_MST10_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

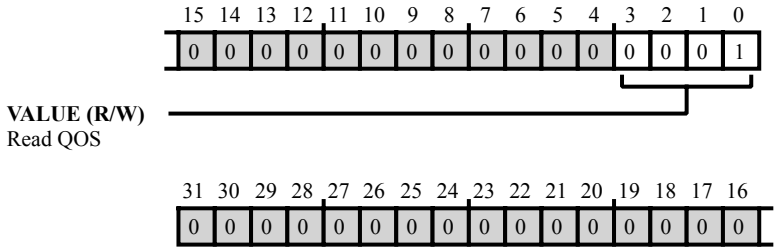

Figure 2-27: SCB_MST10_RQOS Register Diagram

Table 2-28: SCB_MST10_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST10_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 10 Write Quality of Service Register

The SCB_MST10_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

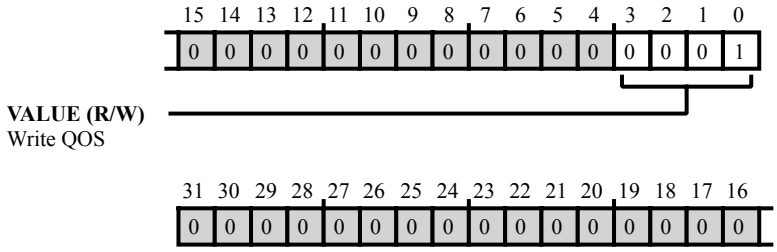

Figure 2-28: SCB_MST10_WQOS Register Diagram

Table 2-29: SCB_MST10_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST10_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 11 Read Quality of Service Register

The SCB_MST11_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

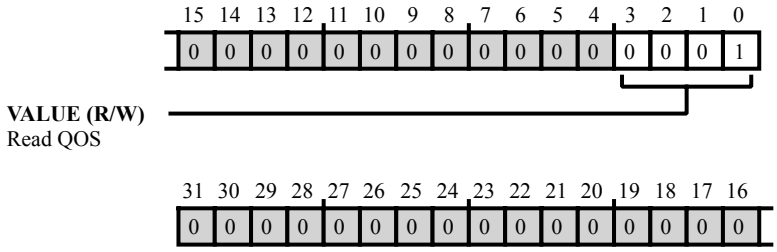

Figure 2-29: SCB_MST11_RQOS Register Diagram

Table 2-30: SCB_MST11_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST11_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 11 Write Quality of Service Register

The SCB_MST11_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

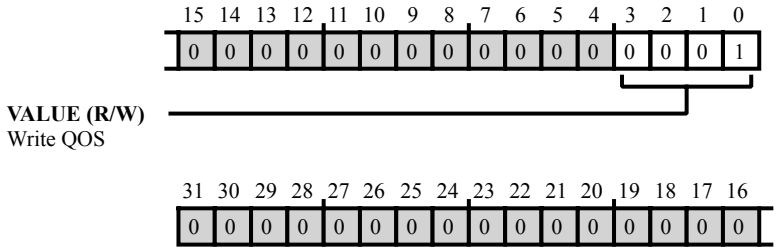

Figure 2-30: SCB_MST11_WQOS Register Diagram

Table 2-31: SCB_MST11_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST11_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 12 Read Quality of Service Register

The SCB_MST12_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

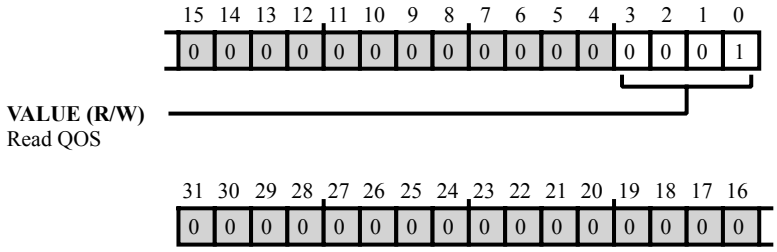

Figure 2-31: SCB_MST12_RQOS Register Diagram

Table 2-32: SCB_MST12_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST12_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 12 Write Quality of Service Register

The SCB_MST12_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

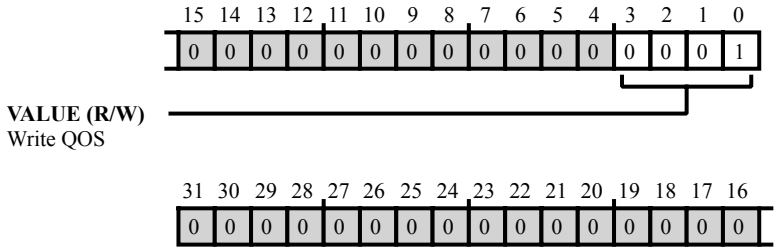

Figure 2-32: SCB_MST12_WQOS Register Diagram

Table 2-33: SCB_MST12_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST12_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 13 Read Quality of Service Register

The SCB_MST13_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

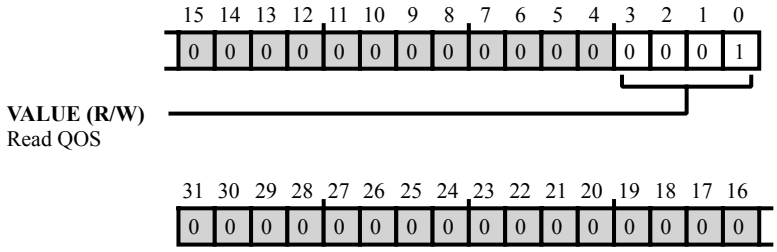

Figure 2-33: SCB_MST13_RQOS Register Diagram

Table 2-34: SCB_MST13_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST13_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 13 Write Quality of Service Register

The SCB_MST13_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

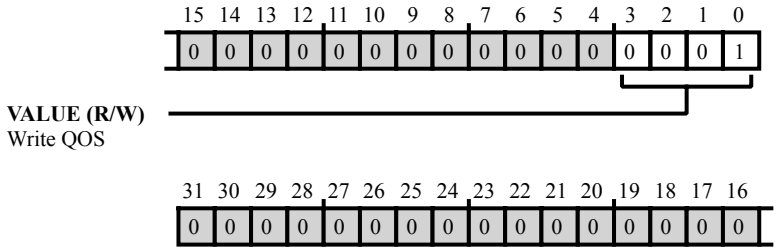

Figure 2-34: SCB_MST13_WQOS Register Diagram

Table 2-35: SCB_MST13_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST13_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 14 Read Quality of Service Register

The SCB_MST14_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

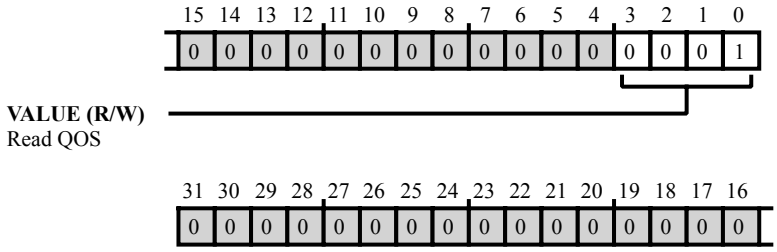

Figure 2-35: SCB_MST14_RQOS Register Diagram

Table 2-36: SCB_MST14_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST14_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 14 Write Quality of Service Register

The SCB_MST14_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

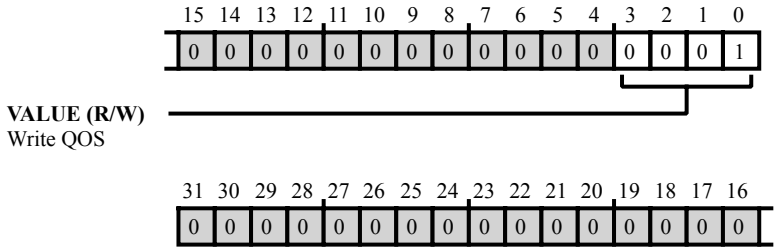

Figure 2-36: SCB_MST14_WQOS Register Diagram

Table 2-37: SCB_MST14_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST14_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 15 Read Quality of Service Register

The SCB_MST15_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

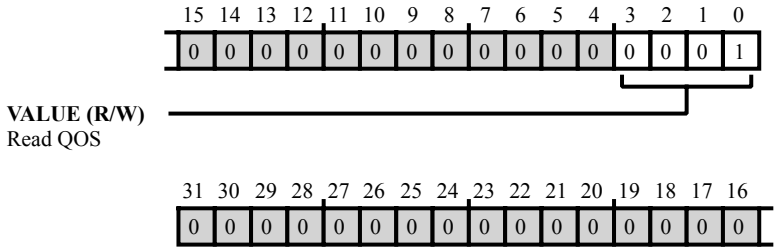

Figure 2-37: SCB_MST15_RQOS Register Diagram

Table 2-38: SCB_MST15_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST15_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 15 Write Quality of Service Register

The SCB_MST15_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

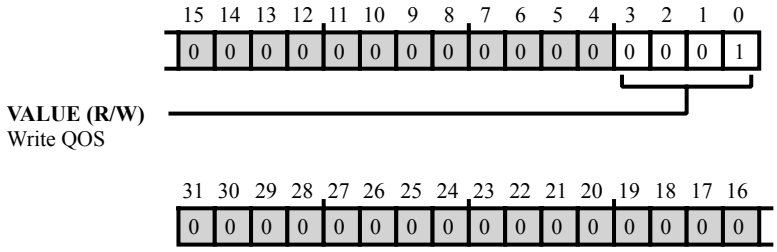

Figure 2-38: SCB_MST15_WQOS Register Diagram

Table 2-39: SCB_MST15_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST15_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 16 Read Quality of Service Register

The SCB_MST16_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

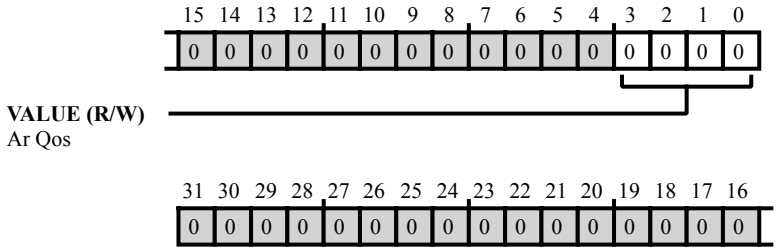

Figure 2-39: SCB_MST16_RQOS Register Diagram

Table 2-40: SCB_MST16_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Ar Qos. The SCB_MST16_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 16 Write Quality of Service Register

The SCB_MST16_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

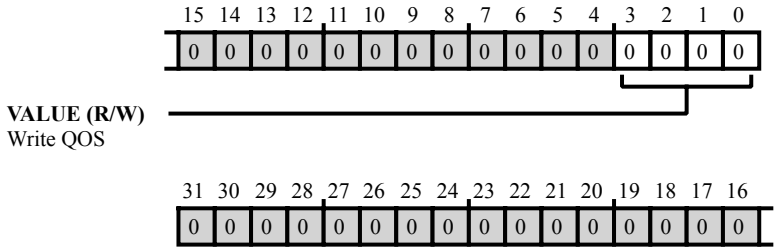

Figure 2-40: SCB_MST16_WQOS Register Diagram

Table 2-41: SCB_MST16_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST16_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 17 Read Quality of Service Register

The SCB_MST17_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

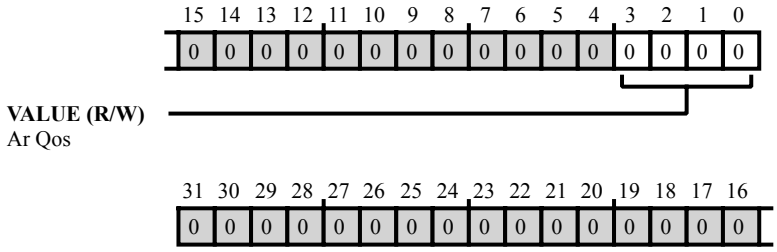

Figure 2-41: SCB_MST17_RQOS Register Diagram

Table 2-42: SCB_MST17_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Ar Qos. The SCB_MST17_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 17 Write Quality of Service Register

The SCB_MST17_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

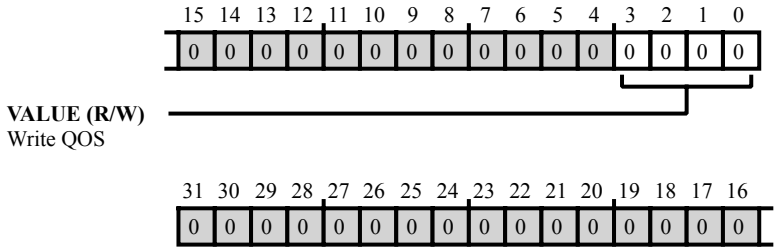

Figure 2-42: SCB_MST17_WQOS Register Diagram

Table 2-43: SCB_MST17_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST17_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 18 Read Quality of Service Register

The SCB_MST18_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

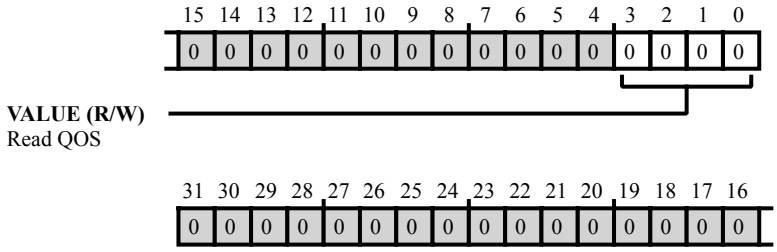

Figure 2-43: SCB_MST18_RQOS Register Diagram

Table 2-44: SCB_MST18_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST18_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 18 Write Quality of Service Register

The SCB_MST18_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

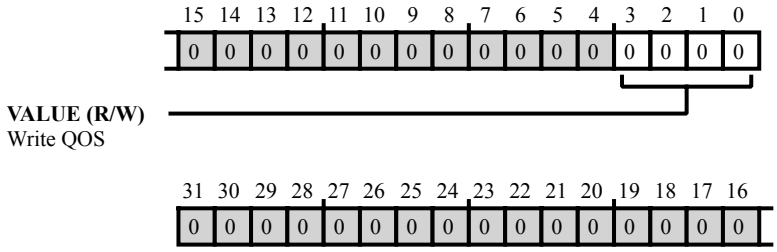

Figure 2-44: SCB_MST18_WQOS Register Diagram

Table 2-45: SCB_MST18_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST18_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 19 Read Quality of Service Register

The SCB_MST19_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

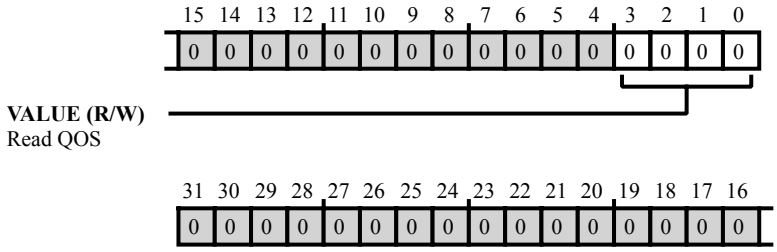

Figure 2-45: SCB_MST19_RQOS Register Diagram

Table 2-46: SCB_MST19_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST19_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 19 Write Quality of Service Register

The SCB_MST19_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

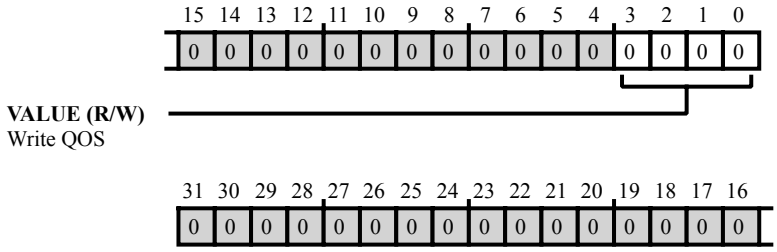

Figure 2-46: SCB_MST19_WQOS Register Diagram

Table 2-47: SCB_MST19_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST19_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 20 Read Quality of Service Register

The SCB_MST20_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

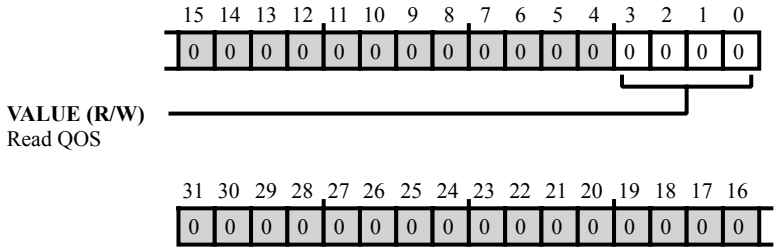

Figure 2-47: SCB_MST20_RQOS Register Diagram

Table 2-48: SCB_MST20_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST20_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 20 Write Quality of Service Register

The SCB_MST20_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

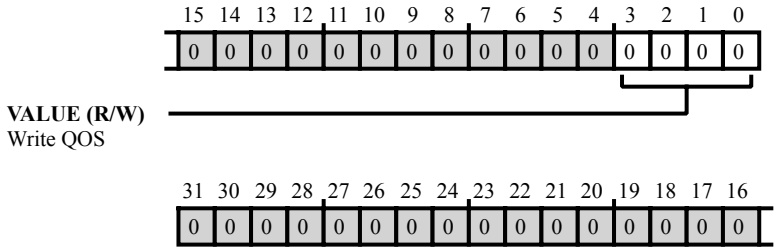

Figure 2-48: SCB_MST20_WQOS Register Diagram

Table 2-49: SCB_MST20_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST20_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 21 Read Quality of Service Register

The SCB_MST21_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

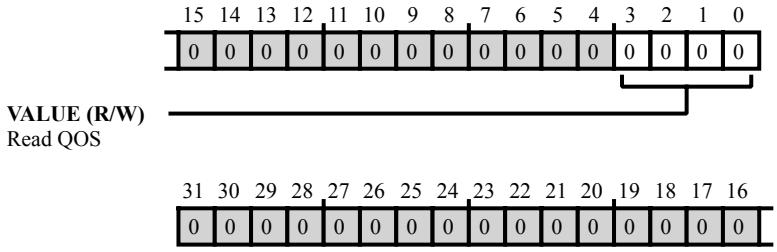

Figure 2-49: SCB_MST21_RQOS Register Diagram

Table 2-50: SCB_MST21_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST21_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 21 Write Quality of Service Register

The SCB_MST21_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

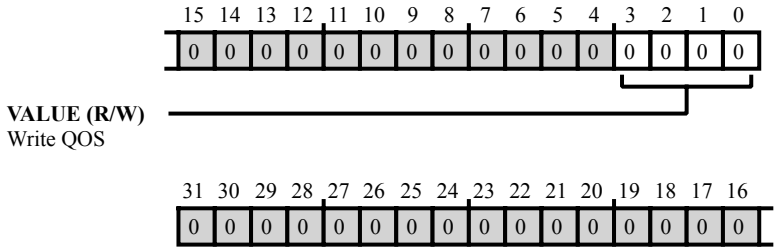

Figure 2-50: SCB_MST21_WQOS Register Diagram

Table 2-51: SCB_MST21_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST21_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 22 Read Quality of Service Register

The SCB_MST22_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

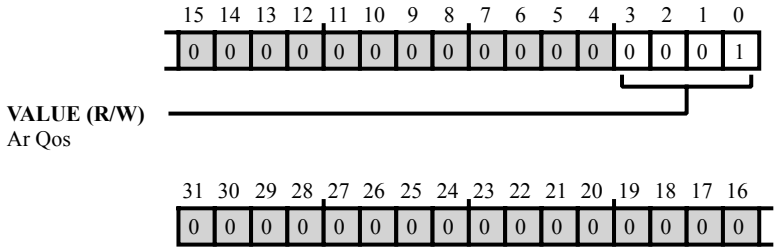

Figure 2-51: SCB_MST22_RQOS Register Diagram

Table 2-52: SCB_MST22_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Ar Qos. The SCB_MST22_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 22 Write Quality of Service Register

The SCB_MST22_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

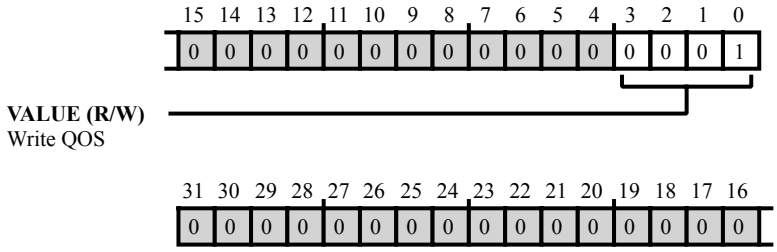

Figure 2-52: SCB_MST22_WQOS Register Diagram

Table 2-53: SCB_MST22_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST22_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 23 Read Quality of Service Register

The SCB_MST23_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

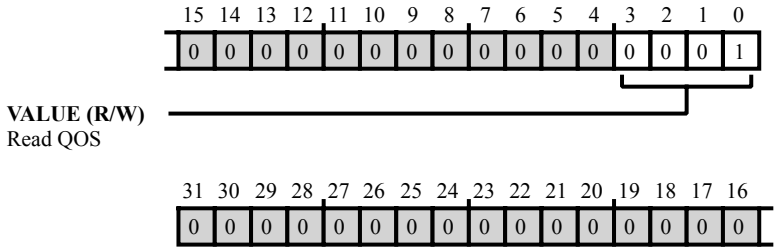

Figure 2-53: SCB_MST23_RQOS Register Diagram

Table 2-54: SCB_MST23_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST23_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 23 Write Quality of Service Register

The SCB_MST23_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

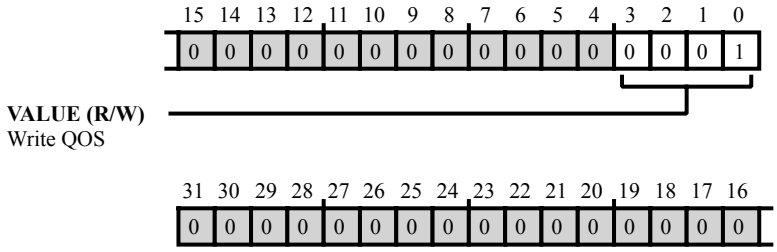

Figure 2-54: SCB_MST23_WQOS Register Diagram

Table 2-55: SCB_MST23_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST23_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 24 Read Quality of Service Register

The SCB_MST24_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

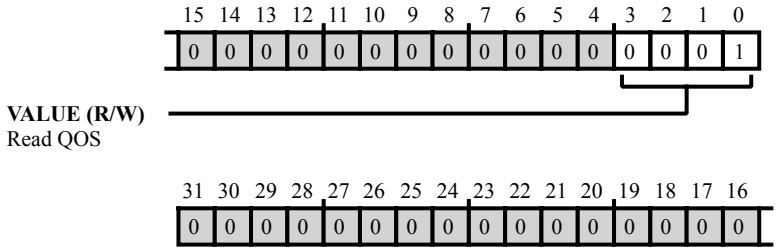

Figure 2-55: SCB_MST24_RQOS Register Diagram

Table 2-56: SCB_MST24_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST24_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 24 Write Quality of Service Register

The SCB_MST24_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

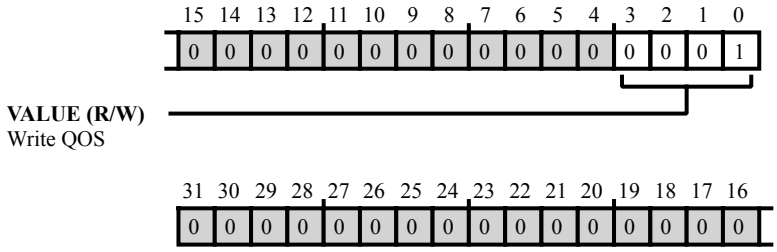

Figure 2-56: SCB_MST24_WQOS Register Diagram

Table 2-57: SCB_MST24_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST24_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 25 Read Quality of Service Register

The SCB_MST25_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

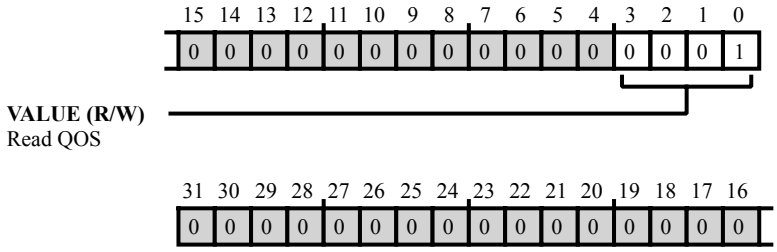

Figure 2-57: SCB_MST25_RQOS Register Diagram

Table 2-58: SCB_MST25_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST25_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 25 Write Quality of Service Register

The SCB_MST25_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

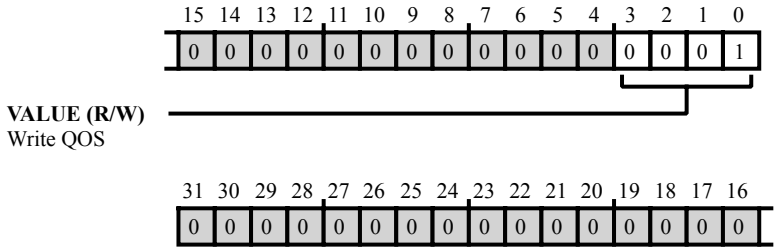

Figure 2-58: SCB_MST25_WQOS Register Diagram

Table 2-59: SCB_MST25_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST25_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 26 Read Quality of Service Register

The SCB_MST26_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

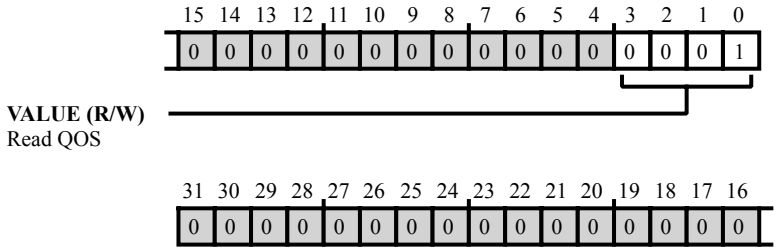

Figure 2-59: SCB_MST26_RQOS Register Diagram

Table 2-60: SCB_MST26_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST26_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 26 Write Quality of Service Register

The SCB_MST26_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

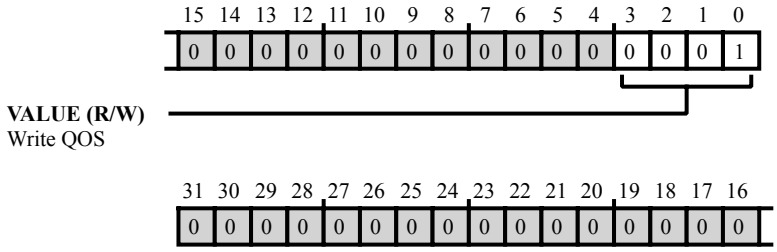

Figure 2-60: SCB_MST26_WQOS Register Diagram

Table 2-61: SCB_MST26_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST26_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 27 Read Quality of Service Register

The SCB_MST27_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

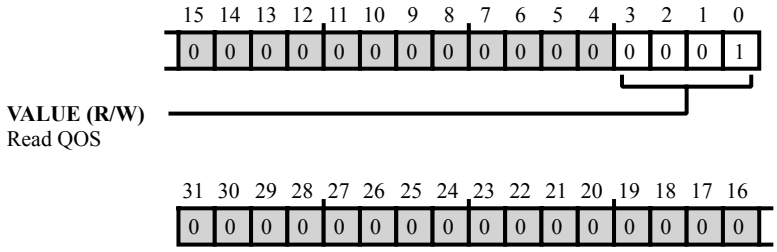

Figure 2-61: SCB_MST27_RQOS Register Diagram

Table 2-62: SCB_MST27_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST27_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 27 Write Quality of Service Register

The SCB_MST27_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

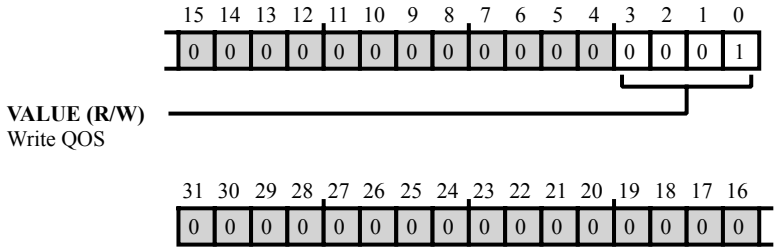

Figure 2-62: SCB_MST27_WQOS Register Diagram

Table 2-63: SCB_MST27_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST27_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 28 Read Quality of Service Register

The SCB_MST28_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

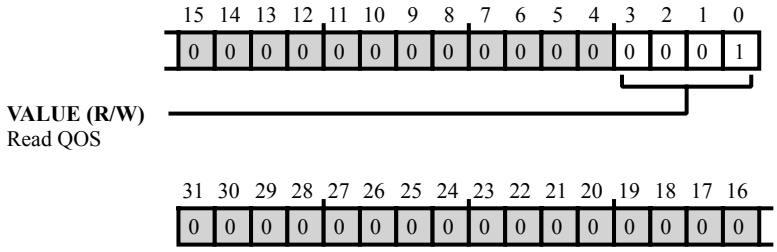

Figure 2-63: SCB_MST28_RQOS Register Diagram

Table 2-64: SCB_MST28_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST28_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 28 Write Quality of Service Register

The SCB_MST28_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

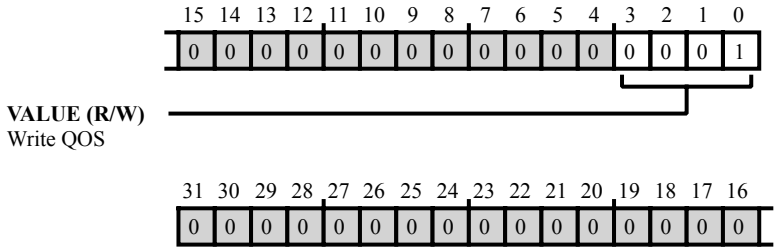

Figure 2-64: SCB_MST28_WQOS Register Diagram

Table 2-65: SCB_MST28_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST28_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Master 29 Read Quality of Service Register

The SCB_MST29_RQOS register indicates the read QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting read channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

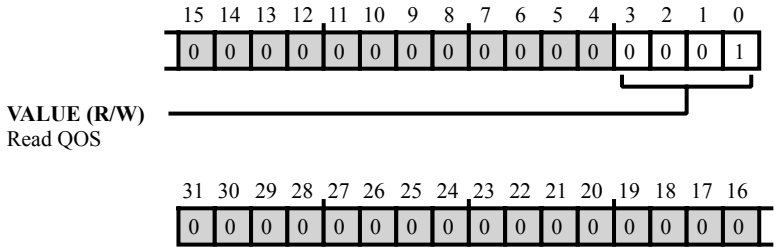

Figure 2-65: SCB_MST29_RQOS Register Diagram

Table 2-66: SCB_MST29_RQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Read QOS. The SCB_MST29_RQOS.VALUE bit field indicates the read QOS or priority value for the indicated master.

Master 29 Write Quality of Service Register

The SCB_MST29_WQOS register indicates the write QOS or priority value for the indicated master. This value is used by the SCBs at different levels to arbitrate among different masters requesting write channel accesses. For mapping of master IDs to peripherals, see the SCB Bus Master IDs table.

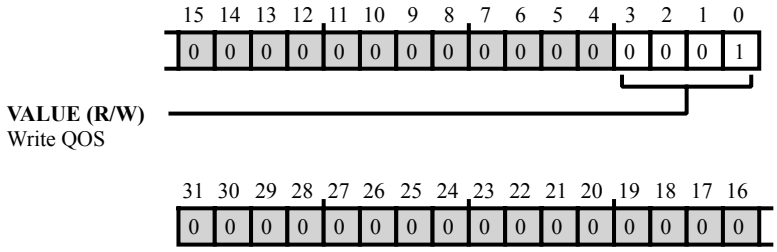

Figure 2-66: SCB_MST29_WQOS Register Diagram

Table 2-67: SCB_MST29_WQOS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	VALUE	Write QOS. The SCB_MST29_WQOS.VALUE bit field indicates the write QOS or priority value for the indicated master.

Interface Block IB4 Sync Mode

This register is used to program the clock domain crossing in the interface blocks.

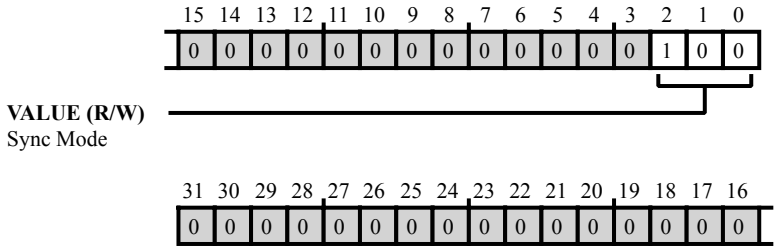

Figure 2-67: SCB_MST30_SYNC Register Diagram

Table 2-68: SCB_MST30_SYNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	VALUE	Sync Mode.	
		0	1:1
		1	Sync n:1
		2	Sync 1:n
		3	SYNC m:n
		4	ASYNC (Default)

Interface Block IB5 Sync Mode

This register is used to program the clock domain crossing in the interface blocks.

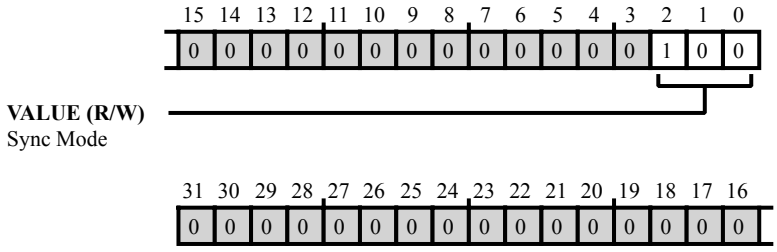

Figure 2-68: SCB_MST31_SYNC Register Diagram

Table 2-69: SCB_MST31_SYNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2:0 (R/W)	VALUE	Sync Mode. The `SCB_MST31_SYNC.VALUE` bit field is used to program the clock domain crossing in the interface blocks.	
		0	1:1
		1	Sync n:1
		2	Sync 1:n
		3	SYNC m:n
		4	ASYNC (Default)

3 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 clock that runs at a frequency that is a multiple of the CLKIN input clock frequency. It also generates all on-chip clocks and synchronization signals. The PCU allows the application software to control the PLL module operation.

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
- Supports programmable options for the `SYS_CLKOUT` output, providing output divided-down versions of the on-chip clocks.

  By default, the `SYS_CLKOUT` pin drives a buffered version of the `SYS_CLKIN` input.
- Provides PLL and clock domain status reporting for event management
- Supports the capability to bypass the PLL for power savings
- Maximizes power management flexibility with the DPM
- Manages power dynamically through software, allowing the dynamic control of the core clock frequency (f_{CCLK}) of the processor
- Provides clock generation support for multiple operating or sleep modes, permitting a custom model for power usage; modes include full-on mode, deep sleep mode, and hibernate mode
- 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 CGU Programming Model section.

ADSP-BF70x 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 3-1: ADSP-BF70x 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_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

Table 3-1: ADSP-BF70x CGU Register List (Continued)

Name	Description
CGU_TSVALUE0	Time Stamp Counter Initial 32 LSB Value Register
CGU_TSVALUE1	Time Stamp Counter Initial MSB Value Register

ADSP-BF70x CGU Interrupt List

Table 3-2: ADSP-BF70x CGU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
2	CGU0_EVT	CGU0 Event	Edge	
83	CGU0_ERR	CGU0 Error	Level	

ADSP-BF70x CGU Trigger List

Table 3-3: ADSP-BF70x CGU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
1	CGU0_EVT	CGU0 Event	Edge

Table 3-4: ADSP-BF70x 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. The CGU generates the clocks listed in the ***Clock Descriptions*** table.

Table 3-5: Clock Descriptions

Clock	Description
PLLCLK	Phase-locked loop clock provides the source from which all clocks listed in this table are derived from unless the PLL is bypassed
CCLK0	Core Clock 0
SYSCLK	Clock for system buses and provides the source from which SCLK0 and SCLK1 are derived
SCLK0	All other peripherals not clocked by SCLK1
SCLK1	MDMA1, MDMA2, and CRYPTOGRAPHIC ACCELERATORS
DCLK	Dynamic memory clock
OCLK	Output clock is a programmable source clock for the SYS_CLKOUT pin

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_CLKIN` input to a higher frequency.

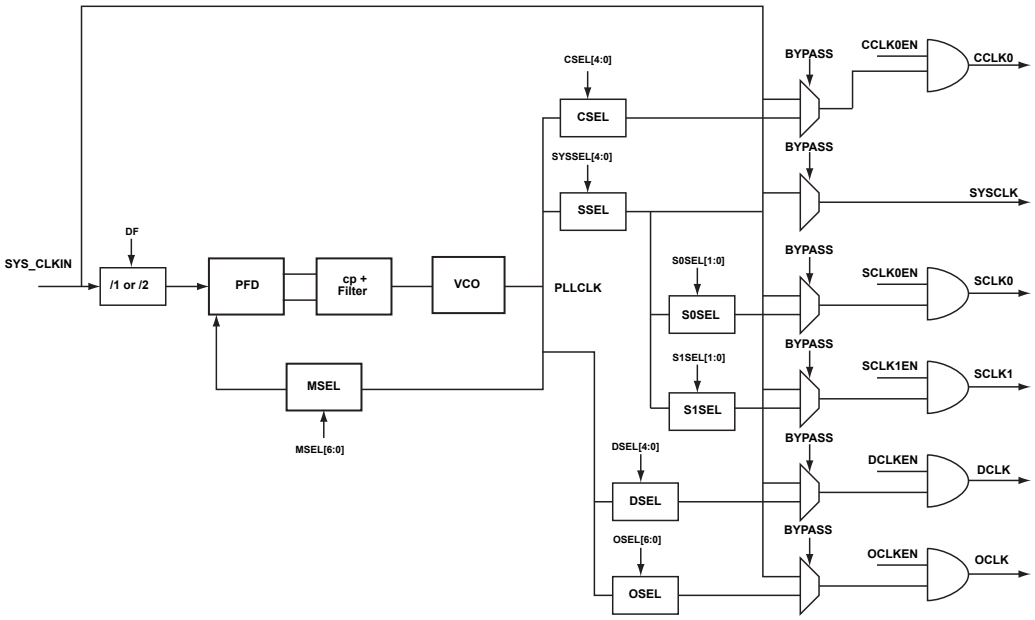

Figure 3-1: CGU PLL Block Diagram

The output of these blocks is called *PLLCLK*. The *PLLCLK* is divided to form *CCLK0*, *SYSCLK*, *DCLK*, and *OCLK*. The *SYSCLK* is further divided to form *SCLK0* and *SCLK1*.

The ***SYS_CLKOUT Generation*** figure is a conceptual representation of the CLKOUT module. Different clocks that originate from the CGU blocks are available on the SYS_CLKOUT output pin. The selection of which clock output on the SYS_CLKOUT pin is controlled by the `CGU_CLKOUTSEL.CLKOUTSEL` bit field.

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

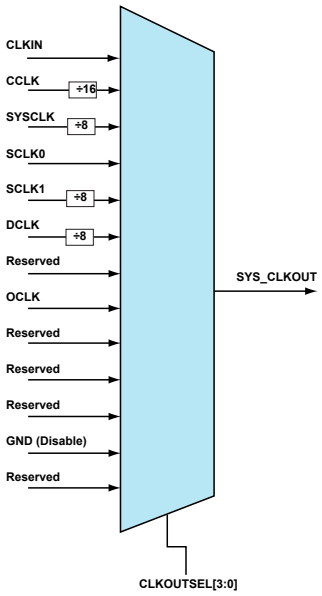

Figure 3-2: SYS_CLKOUT Generation

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

The power-up scenario requires that all power supplies and PLL input clocks are valid before the system HW reset pin is deasserted. The deassertion starts to lock the PLL with default settings. For details, refer to the specific product data sheet.

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
- Short Clocks Alignment Time - CGU_DIV.S0SEL, CGU_DIV.S1SEL
- DCLK Divisor - CGU_DIV.DSEL

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.

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 perform the following sequence only once, after coming out of reset, inside the application, before changing clocks. This sequence clears the CGU_STAT.CLKSALGN bit:

```
*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 register to trigger an alignment sequence
- A write access to the CGU_DIV register 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. But, the new PLLCLK frequency is not used until the PLL is no longer bypassed.

CAUTION: Changing the PLL frequency causes the *DCLK* frequency to change. Either accessing dynamic memory (for example, DDR) or accessing the dynamic memory controller (DMC) registers while PLL or clock frequency changes are in progress can have unpredictable results.

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.

Writing to the CGU_DIV register is allowed while the processor is in active (PLL bypassed) mode. But, the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Accessing the DDR memory while changing the *SYSCLK* frequency is not supported and can have unpredictable results.

Changing the ADSP-BF70x SCLKn Frequency without Modifying the PLLCLK Frequency

To change the clock frequency, write new values to CGU_DIV.S0SEL or CGU_DIV.S1SEL bit fields. The frequency change occurs only when the PLL is not bypassed. Any time the *SCLKn* clock frequency is changed, it exits 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.S0SEL or CGU_DIV.S1SEL bit values with the CGU_DIV.UPDT bit = 1.

   ADDITIONAL INFORMATION: This write updates the CGU_DIV register, changes the *SCLKn* 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. The CGU_STAT.CLKSALGN bit can be polled to discover when the clocks are aligned. Any write attempt to change the CGU_DIV.S0SEL or CGU_DIV.S1SEL bit fields while the 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.

Writing to the CGU_DIV register is allowed while the processor is in active (PLL bypassed) mode. But, the effect of the write is visible only after the transition to full-on (PLL not bypassed) mode.

Accessing the DDR memory while changing the *SYSCLK* frequency is not supported and can have unpredictable results.

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.

   ADDITIONAL INFORMATION: 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 register 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. But, 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 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. And, 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.

The CGU does not support accessing the DDR memory while changing the *SYSCLK* or *DCLK* frequencies. This type of access can have unpredictable results.

Shutting Off CCLK0 From Core 0

Core 0 can shut off its own clock to save power when it is not in use.

1. Program the trigger routing unit (TRU) such that the core 0 system interface disable acknowledge C0_SI_DIS_ACK is assigned the master trigger for the MDMA0_SRC and MDMA0_DST slave triggers.
2. Set up two MDMA0 descriptors. Program one descriptor in list mode, program both descriptors with `DMA_CFG.TWAIT` set such that no action is taken until a trigger is received, and store the data in L2 or L3. Use the following steps to program the descriptors:
    a. Descriptor A - Uses 2D DMA. First pass sets the `CGU_CCBF_DIS.CCBF0` bit. Second pass sets the `TRU_SSR[n]` register for MDMA0 to the desired *CCLK* wake up source. For this example, PINT0 is used.
    b. Descriptor B - Uses 2D DMA to clear the `CGU_CCBF_DIS.CCBF0` bit and clear the RCU disable request.
3. Set up PINT0 to generate a trigger on the desired event to reenable the *CCLK*.
4. Execute code on core n to program the disable request in the RCU (`RCU_SIDIS.SI[n]`).
5. Go to IDLE or JUMP (PC, 0) loop.

    As a result of this disable request, core 0 completes any outstanding transactions on its interfaces (MMR bus, MEM bus, and input DMA bus) and asserts the core disable ACK. This acknowledge signal is connected to the C0_SI_DIS_ACK master trigger of the TRU. The acknowledge event causes the MDMA0 descriptor A to shut off the core clock and set up PINT0 triggering for descriptor A.

The *CCLK* remains shut off until the PINT0 trigger asserts. At that point execution on the core resumes.

Shutting Off CCLKn From Another Master

CCLKn can be shut off to save power when it is not in use.

1. Disable interrupts to core n.
2. Set the `RCU_SIDIS.SI[n]` bit to disable the interfaces of core n in order to:
    a. Stop DMA accesses to its L1.
    b. Stop accesses to memory to core 0.
    c. Stop accesses to MMRs.

3. Test the RCU_SISTAT.SI[n] bit to detect when accesses to core n have been disabled and all the pending transactions have completed.
4. Set the CGU_CCBF_DIS.CCBF0 bit to disable the *CCLKn* buffer.
5. Check the CGU_CCBF_STAT.CCBF0 bit.

If the CGU_CCBF_STAT.CCBF0 bit is set, continue.

Reenable CCLKn From Another Master

1. Clear the CGU_CCBF_DIS.CCBF0 to enable *CCLKn*.
2. Check the CGU_CCBF_STAT.CCBF0 bit.
    a. If the CGU_CCBF_DIS.CCBF0 bit is cleared, continue.
3. Clear the RCU_SIDIS.SI[n] bit. The core deasserts its acknowledge signal in response to the RCU_SYSRST0signal. This operation clears the RCU_SISTAT.SI[n] bit.

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.

ADSP-BF70x CGU Register Descriptions

Clock Generation Unit (CGU) contains the following registers.

Table 3-6: ADSP-BF70x 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_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

Table 3-6: ADSP-BF70x CGU Register List (Continued)

Name	Description
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.

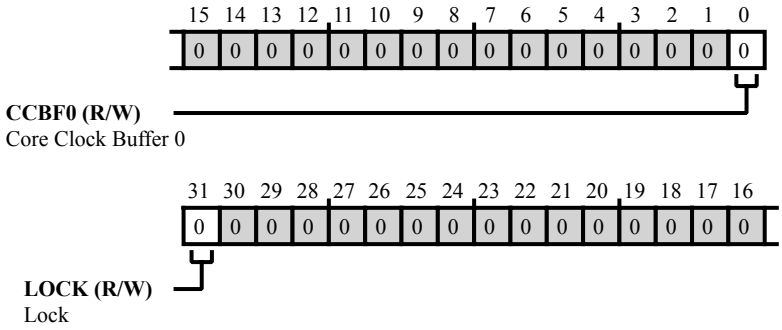

Figure 3-3: CGU_CCBF_DIS Register Diagram

Table 3-7: 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.

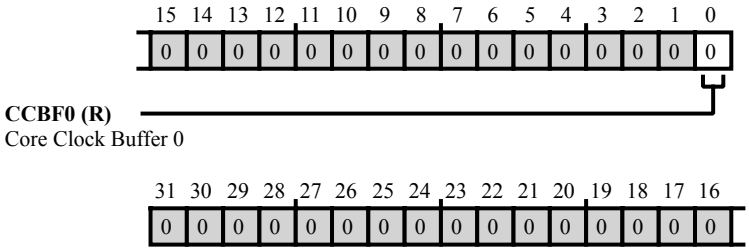

Figure 3-4: CGU_CCBF_STAT Register Diagram

Table 3-8: 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.

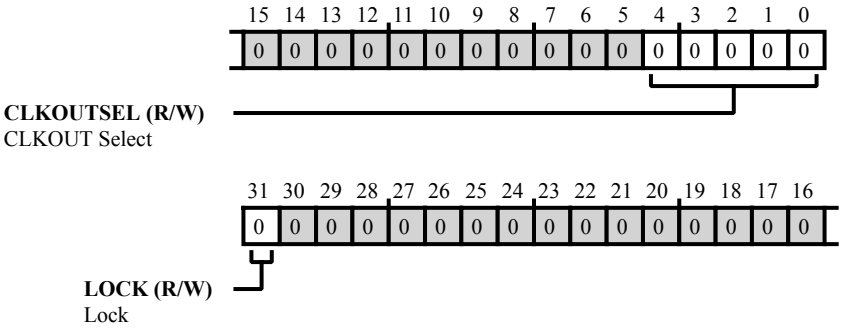

Figure 3-5: CGU_CLKOUTSEL Register Diagram

Table 3-9: 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.	
		0	CLKIN
		1	CCLK (Divided by 16)
		2	SYSCLK (Divided by 8)
		3	SCLK0 (Divided by 1)
		4	SCLK1 (Divided by 8)
		5	DCLK (Divided by 8)
		6	Reserved
		7	OCLK
		8	Reserved
		9	Reserved
		10	Reserved
		11	GND

Table 3-9: CGU_CLKOUTSEL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		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.

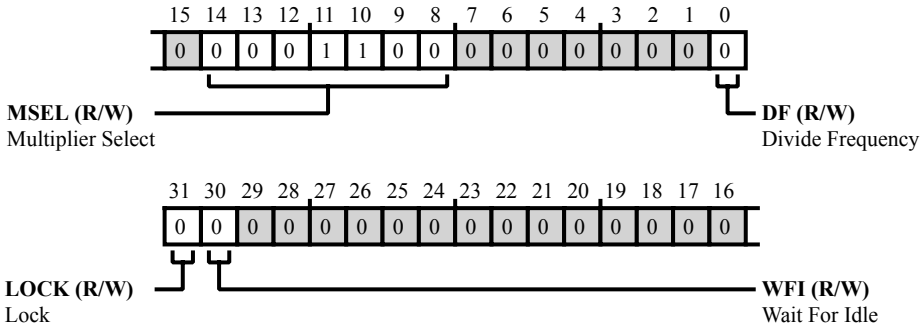

Figure 3-6: CGU_CTL Register Diagram

Table 3-10: 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 3-10: 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.

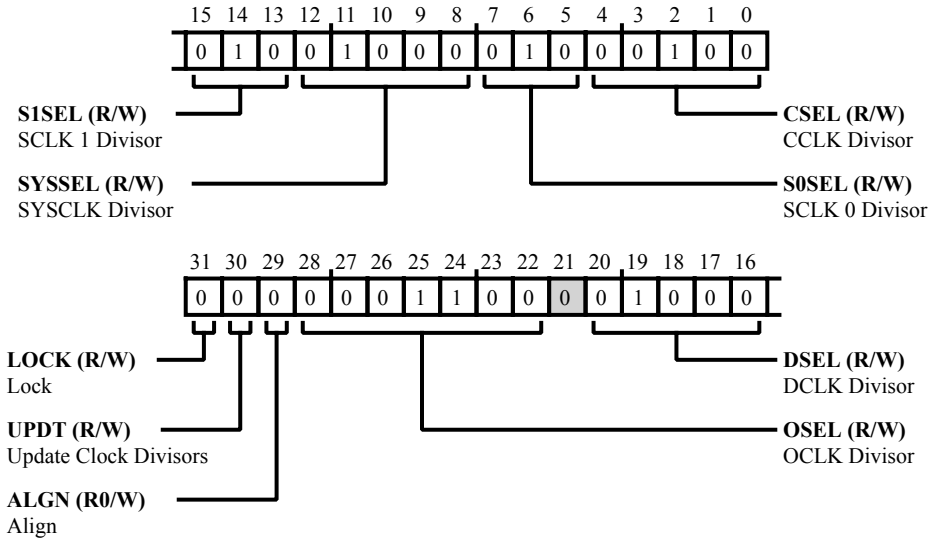

Figure 3-7: CGU_DIV Register Diagram

Table 3-11: 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 3-11: 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	SCLK 1 Divisor. The CGU_DIV.S1SEL selects the divisor in the SCLK1 equation: SCLK1 frequency = (SYSCLK frequency) / CGU_DIV.S1SEL Where the value of CGU_DIV.S1SEL is between 1 and 7.	
		0	S1SEL = 8
		1-7	S1SEL = 1 to 7
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 3-11: 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

PLL Control Register

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

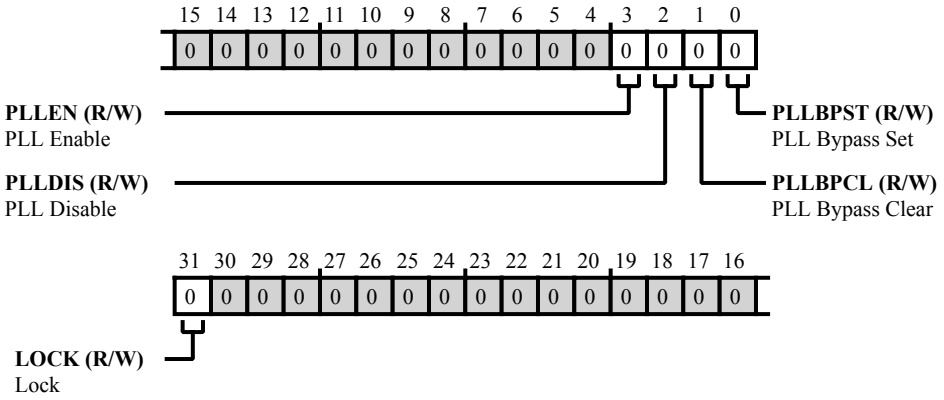

Figure 3-8: CGU_PLLCTL Register Diagram

Table 3-12: 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 3-12: 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.

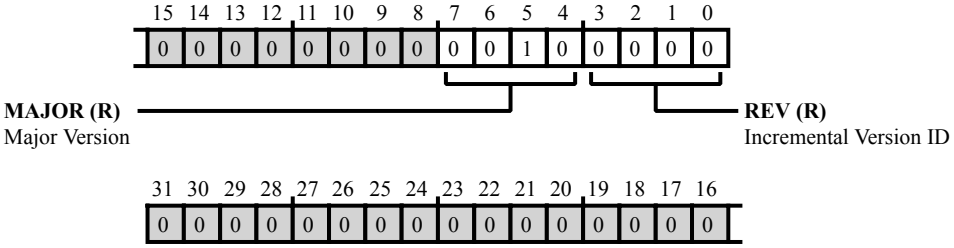

Figure 3-9: CGU_REVID Register Diagram

Table 3-13: 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.

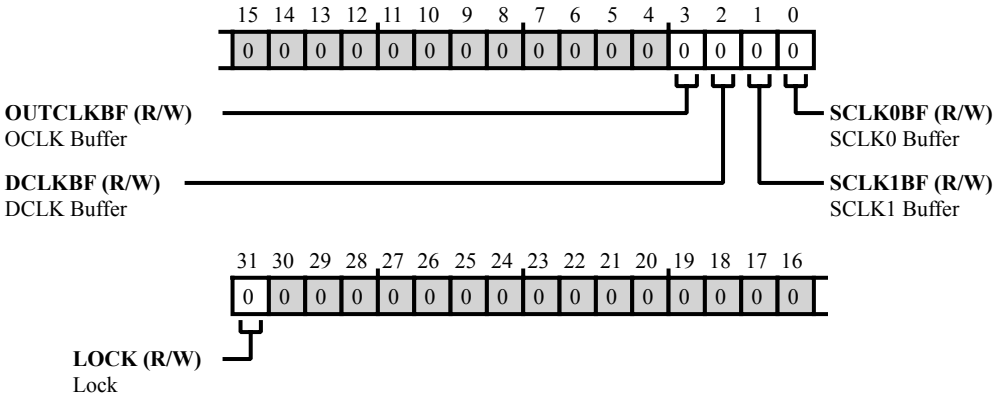

Figure 3-10: CGU_SCBF_DIS Register Diagram

Table 3-14: 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 3-14: CGU_SCBF_DIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	SCLK1BF	SCLK1 Buffer. The `CGU_SCBF_DIS.SCLK1BF` bit enables (=0, default) or disables (=1) SCLK1s buffer.	
		0	Enable buffer
		1	Disable buffer
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.

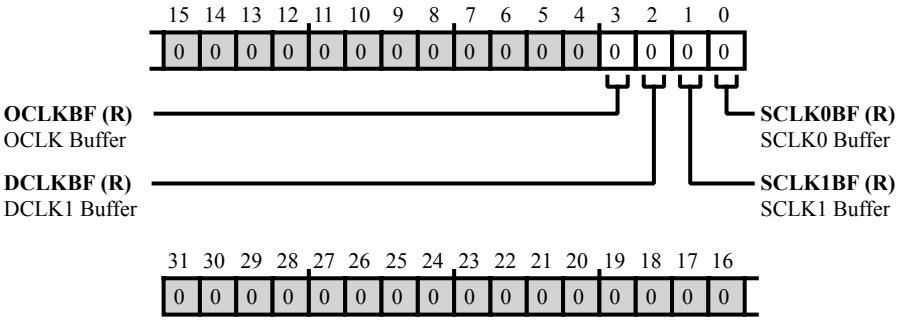

Figure 3-11: CGU_SCBF_STAT Register Diagram

Table 3-15: 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
1 (R/NW)	SCLK1BF	SCLK1 Buffer. The CGU_SCBF_STAT.SCLK1BF bit reports the status of the CGU_SCBF_DIS.SCLK1BF bit where 0 = enabled and 1 = disabled.	
		0	Enabled
		1	Disabled

Table 3-15: CGU_SCBF_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.

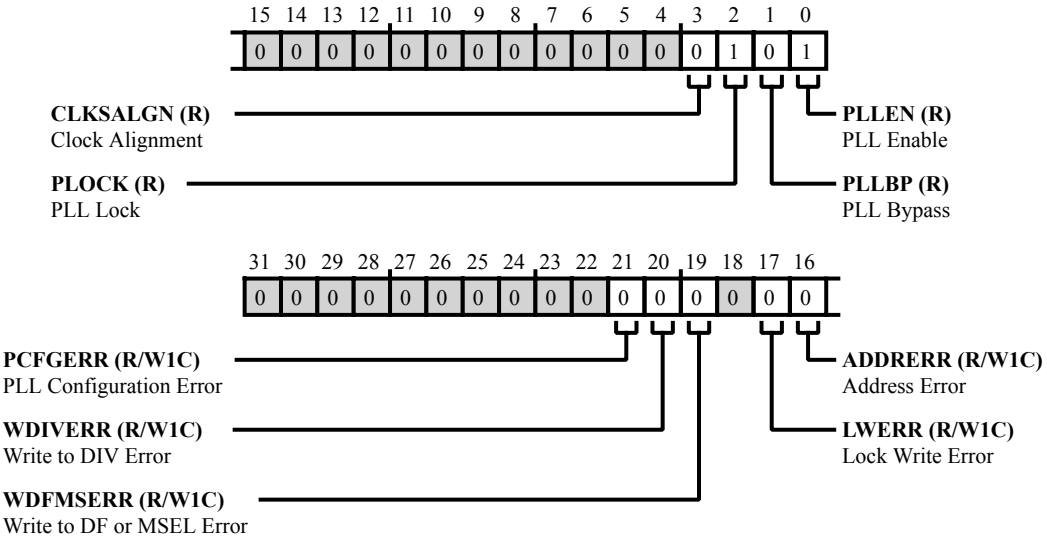

Figure 3-12: CGU_STAT Register Diagram

Table 3-16: 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
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

Table 3-16: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

Table 3-16: CGU_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.

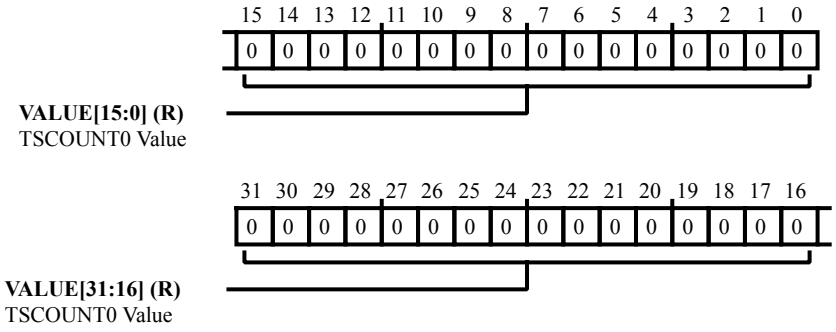

Figure 3-13: CGU_TSCOUNT0 Register Diagram

Table 3-17: 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.

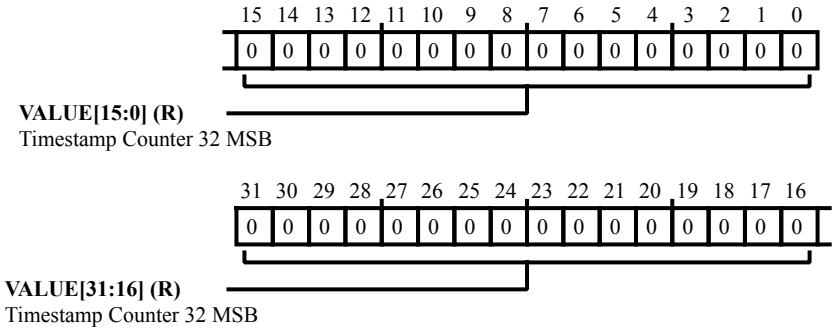

Figure 3-14: CGU_TSCOUNT1 Register Diagram

Table 3-18: 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.

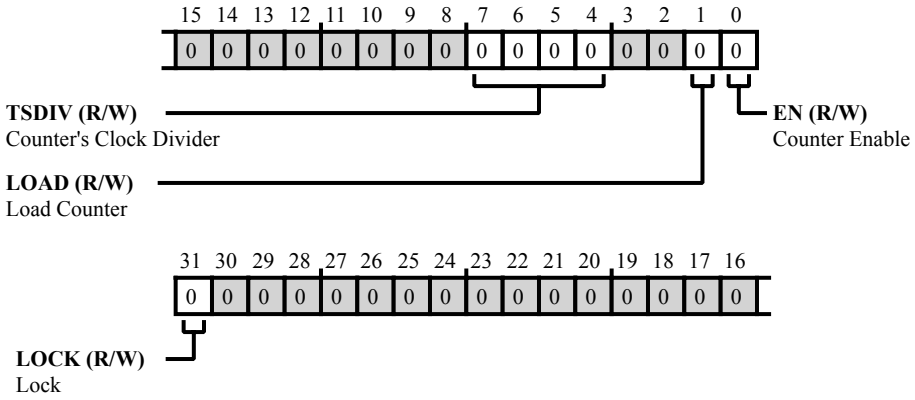

Figure 3-15: CGU_TSCTL Register Diagram

Table 3-19: 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 Core-Sight time stamp counter.

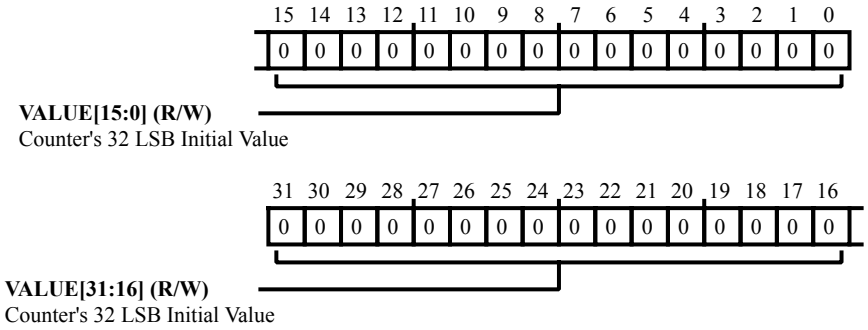

Figure 3-16: CGU_TSVALUE0 Register Diagram

Table 3-20: 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 Core-Sight time stamp counter.

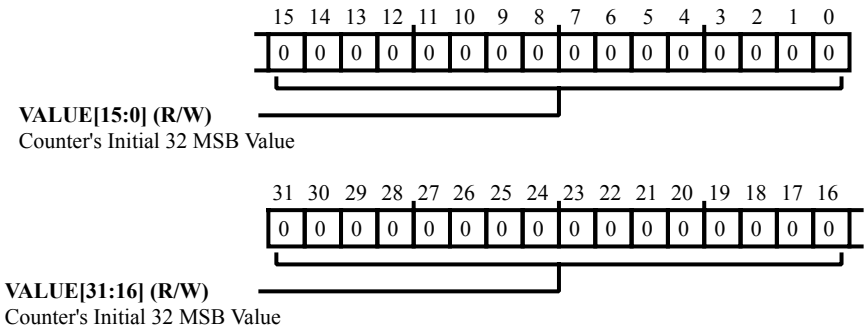

Figure 3-17: CGU_TSVALUE1 Register Diagram

Table 3-21: 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.

4 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 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.

SPU Features

The SPU has the following features:

- Write-protect system MMR from certain system masters and core masters.
- Simultaneously lock multiple peripheral configuration registers through a global lock mechanism.
- Write-protect and block access to its own write-protection registers from other system masters.
- Defined security privileges to peripherals and system resources.
- Security protection to guard secure peripheral MMRs against non-secure accesses.

SPU Functional Description

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

ADSP-BF70x 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 4-1: ADSP-BF70x SPU Register List

Name	Description
SPU_CTL	Control Register
SPU_SECURECHK	Secure Check Register

Table 4-1: ADSP-BF70x SPU Register List (Continued)

Name	Description
SPU_SECURECTL	Secure Control Register
SPU_SECUREC[n]	Secure Core Registers
SPU_SECUREP[n]	Secure Peripheral Register
SPU_STAT	Status Register
SPU_WP[n]	Write Protect Register n

ADSP-BF70x SPU Interrupt List

Table 4-2: ADSP-BF70x SPU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
76	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 (Core 0, Core 1, MDMA, for example). 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.

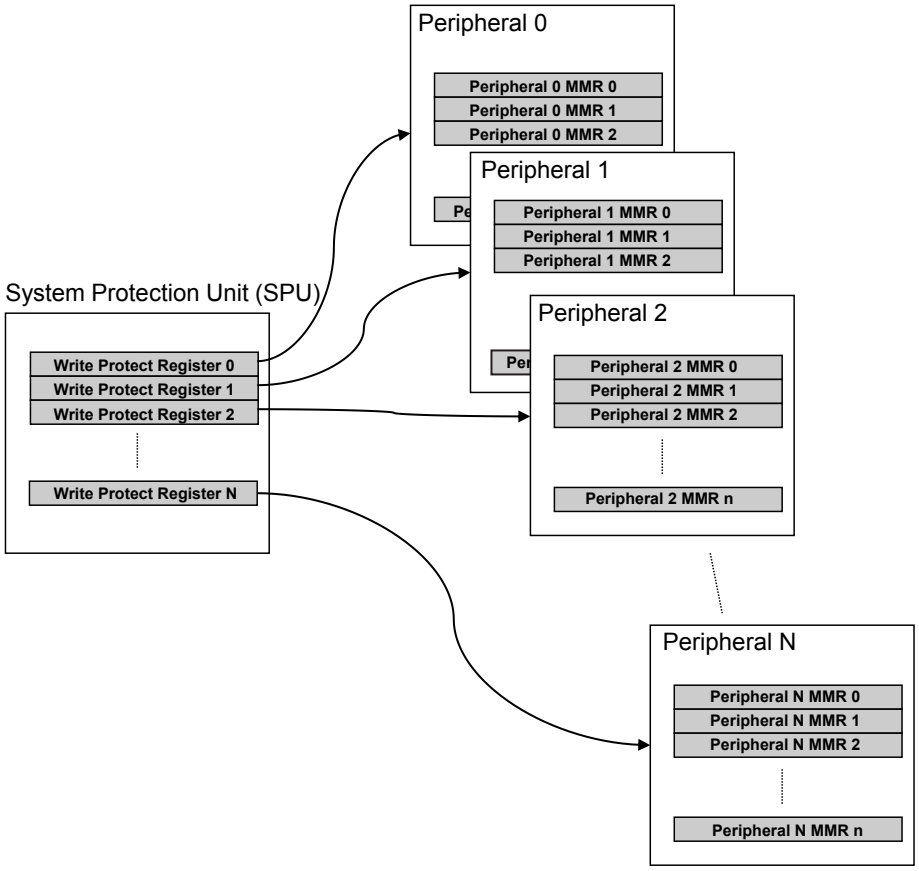

Figure 4-1: SPU Write Protect Registers

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.

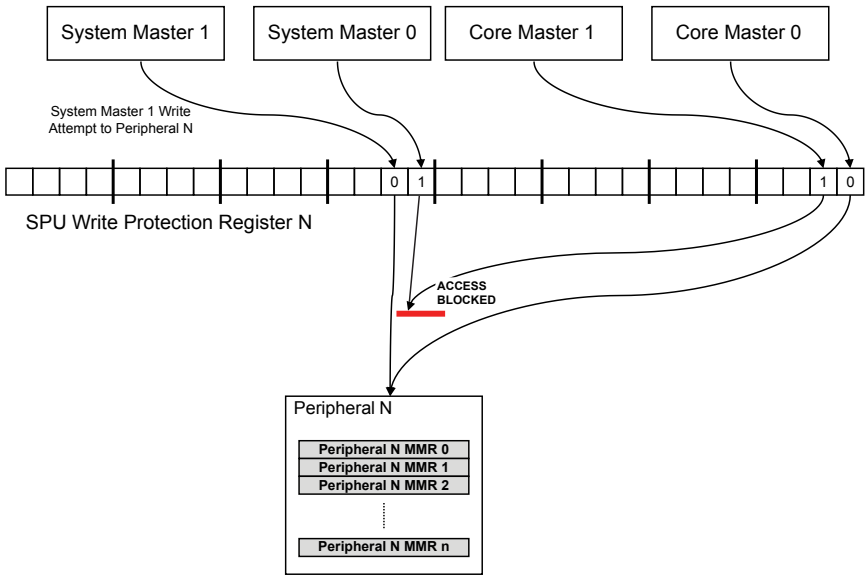

Figure 4-2: SPU Write-Protect Register Blocking Access from System Master 0 and Core Master 1

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

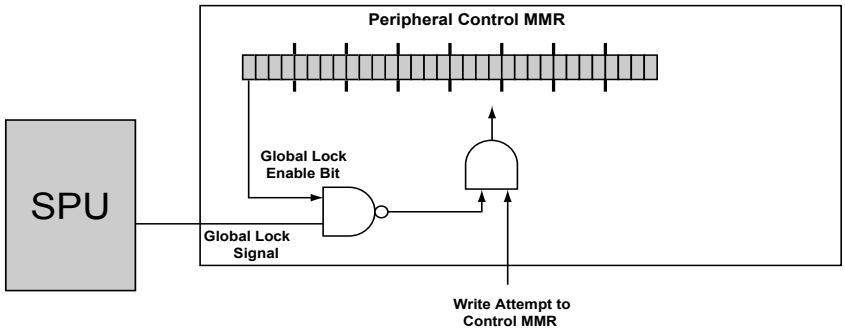

Figure 4-3: Global Locking

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.

Security Protection

The SPU also offers security protection. Just as each peripheral has write-protection associated with it, each peripheral also has a security control register (SPU_SECUREP[n]) associated with it for security protection. The SPU_SECUREP[n] registers all have two bits:

- SPU_SECUREP[n].MSEC bit
- SPU_SECUREP[n].SSEC bit

If set, the SPU_SECUREP[n].SSEC bit configures the SPU to protect the associated peripheral from non-secure transactions (reads and writes). If the SPU_SECUREP[n].MSEC bit is set, the associated peripheral is configured to generate secure transactions (reads and writes).

If the SPU_SECUREP[n].SSEC bit is set for the SPU_SECUREP[n] register of the peripheral, only secure transactions are allowed through to the MMR space of a peripheral.

If the SPU_SECUREP[n].SSEC bit of the peripheral is cleared (disabling slave security), the SPU allows both secure and non-secure transactions. If write-protection was enabled, a write access is still blocked.

The SPU_SECUREC[n] registers are used to configure the security of a core feature such as if a core is a secure slave and or secure master. Refer to the product-specific information section for bit definitions.

NOTE: The SPU_SECUREC[n] registers only pertain to DSP processor cores from Analog Devices in the product.

The write-protection and security protection are mutually exclusive. These features can be used separately or in conjunction but security protection takes precedence over write protection.

NOTE: To maintain a chain of trust in security, ensure that only secure system masters can modify the security settings in the SPU_SECUREP[n] register.

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.

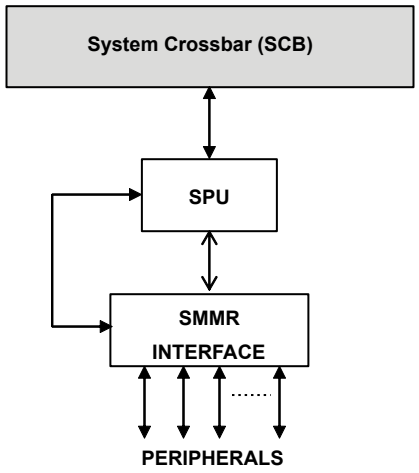

Figure 4-4: SPU System-Level Block Diagram

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.

The SPU also checks whether the transaction is a secure or non-secure transaction and blocks it according to the configured security setting for the target destination. A secure master can generate secure read or secure write transactions which can access secure or non-secure slaves. A non-secure master can generate non-secure read or non-secure write transactions and can only access non-secure slaves.

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.

The SPU can be configured to generate an interrupt for the write-protection violation by setting the `SPU_CTL.PINTEN` bit. The SPU can also be configured to generate an interrupt for a security violation by setting the `SPU_SECURECTL.SINTEN` bit. If either one or both bits is triggered, the `SPU_STAT.VIRQ` bit is set.

The SPU can also lock its own registers from write attempts. If a write-attempt is made to a locked register in the SPU, the SPU blocks it and records it as an error in the `SPU_STAT.LWERR` bit. Again, the SPU generates a bus error to the master that attempted the write.

The master does or does not generate an event, based on the returned error.

The SPU does not generate an event for a blocked write access to an SPU register. If the SPU_CTL.PINTEN bit is set, the SPU triggers an interrupt for this blocked access attempt.

The global lock is enabled by setting the SPU_CTL.GLCK bit to something other than 0xAD. If the lock bit is set in that same configuration register, a peripheral can block write access to its configuration register. When the SPU blocks a write attempt, the peripheral logs and reports the failed attempt. The SPU is unaware and therefore does not provide any indication of a failed write attempt to the configuration register of the peripheral.

SPU Programming Model

The system protection unit (SPU) consists of write-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.

For security, the SPU provides a set of SPU_SECUREC[n] registers (one for each processor core from Analog Devices) to configure their security settings. The SPU also provides a set of SPU_SECUREP[n] registers (one for each peripheral instance) to configure their security settings.

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.

Checking the Security State

In some cases while running a peripheral, an application system master does not know whether they are a secure master generating secure transactions or not. The SPU provides a means for checking the security state of the master through the SPU_SECURECHK register. When read by a secure master, the register reads 0xFFFFFFFF and when read by a non-secure master, the value is 0x00000000.

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.

Configuring Security Privileges of a Peripheral

Use the following procedure to configure the security privileges of a peripheral.

1. Determine the peripheral and its corresponding secure peripheral register (SPU_SECUREP[n]) in the SPU. See the "Write-Protect and Secure Peripheral Registers" section.
2. If the peripheral is to be a secure slave only accepting secure transactions, set bit 0 (SPU_SECUREP[n].SSEC).

3. If the peripheral is to be a secure master that generates secure transactions (keeping in mind not all peripherals can be masters), set bit 1 (`SPU_SECUREP[n].MSEC`).

This procedure sets the security privileges of a peripheral.

NOTE: Only a secure master can set security privileges, keeping the chain of trust intact. If a non-secure master configures the security privileges, it can undermine security protection.

ADSP-BF70x SPU Register Descriptions

System Protection Unit (SPU) contains the following registers.

Table 4-3: ADSP-BF70x SPU Register List

Name	Description
SPU_CTL	Control Register
SPU_SECURECHK	Secure Check Register
SPU_SECURECTL	Secure Control Register
SPU_SECUREC[n]	Secure Core Registers
SPU_SECUREP[n]	Secure Peripheral Register
SPU_STAT	Status Register
SPU_WP[n]	Write Protect Register n

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.

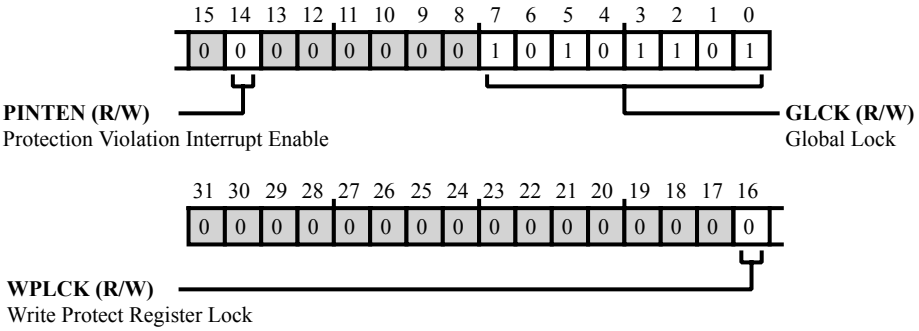

Figure 4-5: SPU_CTL Register Diagram

Table 4-4: SPU_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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.	

Secure Check Register

The SPU_SECURECHK register reads by secure masters return 0xFFFFFFFF. Reads by non-secure masters return 0x00000000.

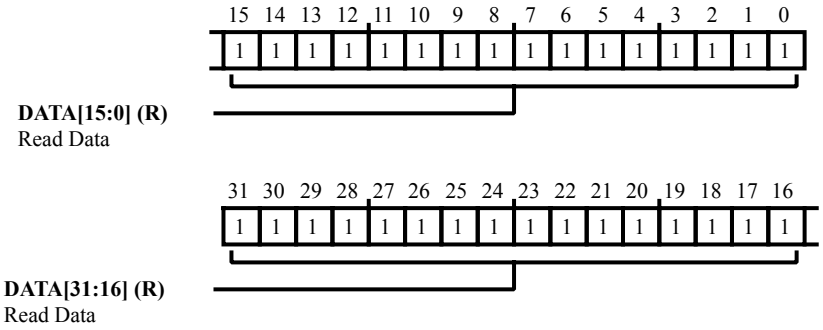

Figure 4-6: SPU_SECURECHK Register Diagram

Table 4-5: SPU_SECURECHK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	DATA	Read Data. The SPU_SECURECHK.DATA bit field performs reads. Reads by secure masters return 0xFFFFFFFF. Reads by non-secure masters return 0x00000000.

Secure Control Register

The SPU Secure Control Register (SPU_SECURECTL) allows the user to lock write access to all the SPU_SECUREC[n] and SPU_SECUREP[n] registers as well as configure the interrupt generation in an event of a security error. It also allows bulk clear of the SSEC bits and/or MSEC bits in the SPU_SECUREP[n] registers.

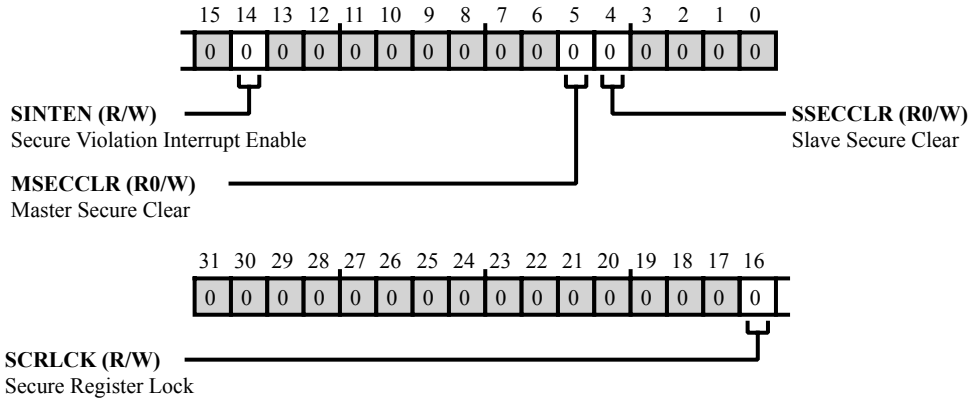

Figure 4-7: SPU_SECURECTL Register Diagram

Table 4-6: SPU_SECURECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	SCRLCK	Secure Register Lock. When the SPU_SECURECTL.SCRLCK bit is set in combination with the SPU_CTL.GLCK bit, writes to the Security Configuration registers (SPU_SECUREC[n] and SPU_SECUREP[n]) are blocked and return an error which is captured in the SPU_STAT.LWERR bit.	
		0	Disable
		1	Enable
14 (R/W)	SINTEN	Secure Violation Interrupt Enable. The SPU_SECURECTL.SINTEN bit generates an interrupt if a security violation was captured. Interrupt status is provided in the SPU_STAT.VIRQ bit.	
		0	Disable
		1	Enable
5 (R0/W)	MSECCLR	Master Secure Clear. When the SPU_SECURECTL.MSECCLR bit is set, the SPU_SECUREP[n].MSEC bits in all SPU_SECUREP[n] registers are cleared. The SPU_SECURECTL.MSECCLR bit always reads back as a 0.	
		0	No Action
		1	Clear All Master Secure Control Bits

Table 4-6: SPU_SECURECTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R0/W)	SSECCLR	Slave Secure Clear. When the `SPU_SECURECTL.SSECCLR` bit is set, the `SPU_SECUREP[n].SSEC` bits in all `SPU_SECUREP[n]` registers are cleared. The `SPU_SECURECTL.SSECCLR` bit always reads back as a 0.	
		0	No Action
		1	Clear All Slave Secure Control Bits

Secure Core Registers

A SPU register exists for every DSP core in the system. The bits enable or disable security for features in the core.

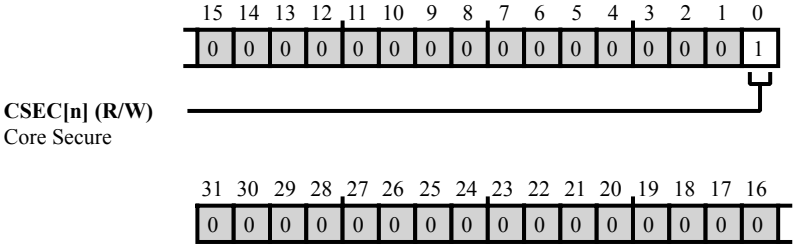

Figure 4-8: SPU_SECUREC[n] Register Diagram

Table 4-7: SPU_SECUREC[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	CSEC[n]	Core Secure. The `SPU_SECUREC[n].CSEC[n]` bit controls whether non-secure accesses are allowed to L1 memory of the processor core. When =1, the core (as a slave) is set as secure meaning only secure transactions are allowed to L1.	
		0	Disable
		1	Enable

Secure Peripheral Register

In the system, each SPU_SECUREP[n] register is assigned to a specific MMR address range associated with one peripheral. Each SPU_SECUREP[n] has a Slave Secure (SSEC) bit and a Master Secure (MSEC) bit. When the Slave Secure (SSEC) bit is set, the SPU will only allow Secure Masters generating secure transactions to access the peripheral's MMR address space. When the Master Secure (MSEC) bit is set, the associated peripheral will be secure and will generate secure transactions.

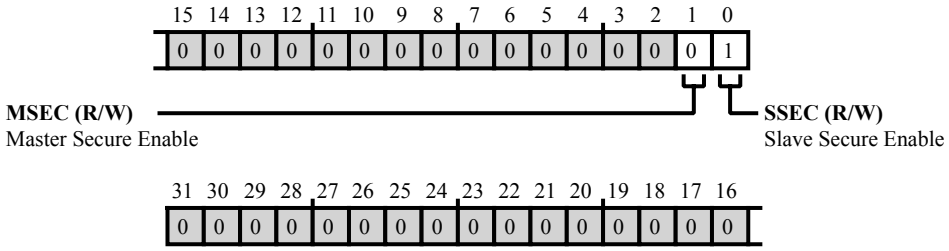

Figure 4-9: SPU_SECUREP[n] Register Diagram

Table 4-8: SPU_SECUREP[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	MSEC	Master Secure Enable. The SPU_SECUREP[n].MSEC bit controls whether the peripheral generates secure transactions as a master. When clear (=0), the peripheral generates non-secure transactions as a master (if applicable). When set (=1), the peripheral generates secure transactions as a master.	
		0	Disable
		1	Enable
0 (R/W)	SSEC	Slave Secure Enable. The SPU_SECUREP[n].SSEC bit controls whether the peripheral is protected from non-secure transactions. When clear (=0), the security status of the transaction is ignored. When set (=1), only secure transactions are allowed to access the address space of the peripheral and non-secure transactions are blocked.	
		0	Disable
		1	Enable

Status Register

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

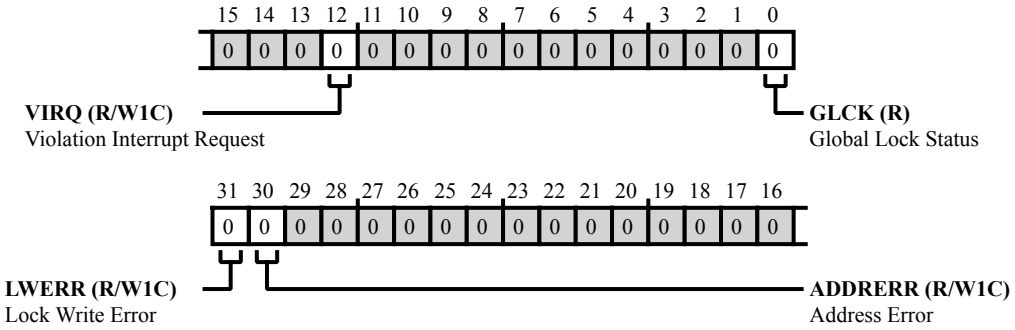

Figure 4-10: SPU_STAT Register Diagram

Table 4-9: 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
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)

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.

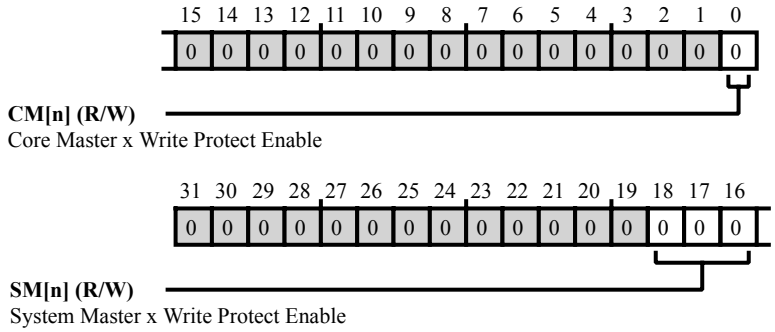

Figure 4-11: SPU_WP[n] Register Diagram

Table 4-10: SPU_WP[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18:16 (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.
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-BF7xx 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-BF7xx is configured with 76 write-protect registers and 76 secure 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 4-11: SPU_WPn Registers and SPU_SECUREPn and Related Peripherals

Write-Protect Register and Secure Peripheral Number (n)	Peripheral
0	RCU0
1	TRU0
2	CGU0
3	DPM0
4	SEC0
5	SPU0
6	SMPU0
8	L2CTL0
9	SWU0
10	SWU3
11	SWU1
12	SWU2
13	DEBUG
14	TIMER0
15	WDOG0
16	JTAG0
17	TWI0
18	SPORT0
19	SPORT1
20	SPI0
21	SPI1
22	SPI2
23	SPIHP0
24	SMC0
25	EPPI0
26	CAN0
27	CAN1
28	CNT0
29	HADC0
30	OPTC0
31	MSI0

Table 4-11: SPU_WPn Registers and SPU_SECUREPn and Related Peripherals (Continued)

Write-Protect Register and Secure Peripheral Number (n)	Peripheral
32	SWU5
33	SWU6
34	SWU7
35	DMA0
36	DMA1
37	DMA2
38	DMA3
39	DMA4
40	DMA5
41	DMA6
42	DMA7
43	DMA8
44	DMA9
45	DMA10
46	DMA11
47	DMA12
48	DMA13
49	DMA14
50	DMA15
51	DMA16
52	DMA17
53	DMA18
54	DMA19
55	DMA20
56	DMA21
57	CRC0
58	CRC1
59	DDR0
60	DMCPHY0
61	SMPU1
62	SWU4

Table 4-11: SPU_WPn Registers and SPU_SECUREPn and Related Peripherals (Continued)

Write-Protect Register and Secure Peripheral Number (n)	Peripheral
63	SPI1
64	USB0
65	RTC0
66	UART0
67	UART1
68	PORTA
69	PORTB
70	PORTC
71	PADS0
72	PINT0
73	PINT1
74	PINT2
75	TRNG0
76	PKA0
77	PKIC0

ADSP-BF7xx Specific Information

The information in this section applies specifically to the ADSP-BF7xx processor models.

Global Locking

The global lock signal from the SPU along with the peripheral lock bit can be used to provide lock functionality for the control MMR of the peripheral. The global lock `SPU_CTL.GLCK` bit field determines whether global lock is enabled or not. Global lock is disabled if the `SPU_CTL.GLCK` field is 0xAD (default value), otherwise it is enabled. The following is a list of peripherals that have the global lock bit in their control MMR.

- General-Purpose IO (GPIO)
- System Event Controller (SEC)
- Trigger Routing Unit (TRU)
- Clock Generation Unit (CGU)
- Dynamic Power Management (DPM-LV)
- Reset Control Unit (RCU)
- System Protection Unit (SPU)

- L2 Memory Controller (L2)
- System Memory Protection Unit (SMPU)

5 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.

  NOTE: On the ADSP-BF70x, there are two SMPU instances. One for L2 (DMA port) and one for DDR memory. The SMPU that guards L2 can be configured for up to four regions while the SMPU for DDR can be configured for up to eight regions.

- 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.

ADSP-BF70x 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 5-1: ADSP-BF70x SMPU Register List

Name	Description
SMPU_BADDR	Bus Error Address Register
SMPU_BDTLS	Bus Error Details Register
SMPU_CTL	SMPU Control Register

Table 5-1: ADSP-BF70x SMPU Register List (Continued)

Name	Description
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_SECURECTL	SMPU Control Secure Accesses Register
SMPU_SECURERCTL[n]	Region n Control Secure Accesses Register
SMPU_STAT	SMPU Status Register

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.

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 section). 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 the "ID Comparison" section).

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].RPROTENbit 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.

Table 5-2: System Master IDs

System Master	ID
SPORT0 A	10'b000000x000
SPORT0 B	10'b000000x001
SPORT1 A	10'b000000x010
SPORT1 B	10'b000000x011
SPI0 TX	10'b000000x100
SPI0 RX	10'b000000x101
SPI1 TX	10'b000100x101
SPI1 RX	10'b000100x110
SPI2 TX	10'b000100x000
SPI2 RX	10'b000100x001
UART0 TX	10'b000000x110
UART0 RX	10'b000000x111
UART1 TX	10'b000100x010
UART1 RX	10'b000100x011
EPPI Ch0	10'b001000x000
EPPI Ch1	10'b001000x001
MDMA0 SRC	10'b001000x010
MDMA0 DST	10'b001000x011
MDMA1 SRC	10'b001100x000
MDMA1 DST	10'b001100x001
MDMA2 SRC	10'b010000x000
MDMA2 DST	10'b010000x001
DSPMEM	10'b0101xxx000
DSPMMR	10'b0110000001
USB	10'b001000x100
MSI	10'b000100x100
PKTE	10'b011100x000
DAP	10'b1000000000
ETR	10'b0111000001
SPIH	10'b0010000101

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 5-3: 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
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.

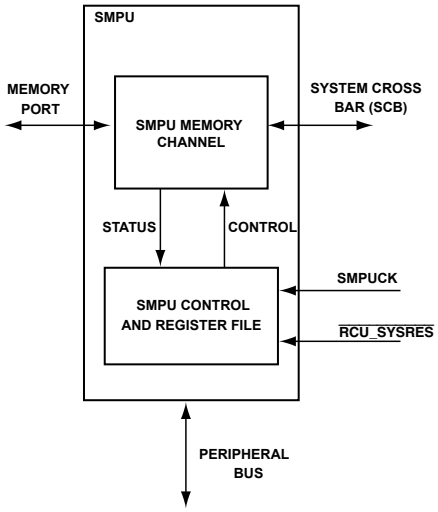

Figure 5-1: SMPU Top-Level Block Diagram

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. 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.

ADSP-BF70x SMPU Register Descriptions

The System Memory Protection Unit (SMPU) contains the following registers.

Table 5-4: ADSP-BF70x 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

Table 5-4: ADSP-BF70x SMPU Register List (Continued)

Name	Description
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_SECURECTL	SMPU Control Secure Accesses Register
SMPU_SECURERCTL[n]	Region n Control Secure Accesses 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.

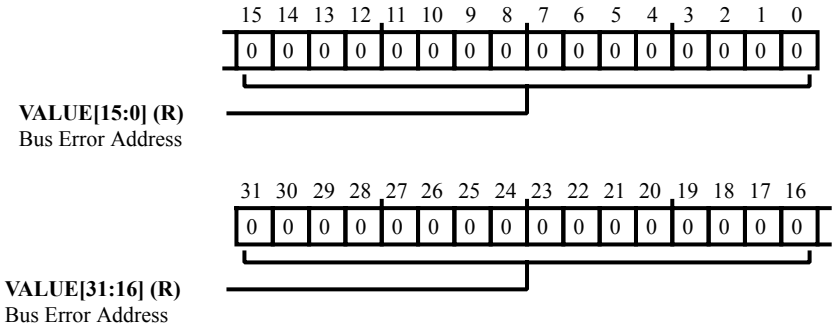

Figure 5-2: SMPU_BADDR Register Diagram

Table 5-5: 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.

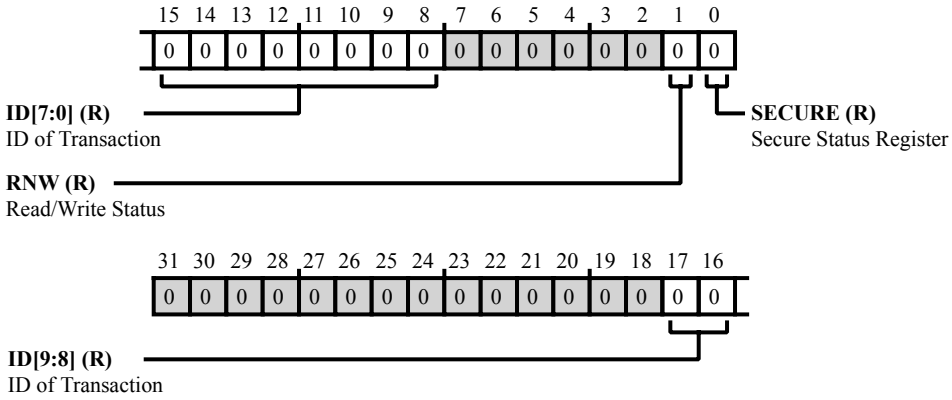

Figure 5-3: SMPU_BDTLS Register Diagram

Table 5-6: SMPU_BDTLS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17:8 (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
0 (R/NW)	SECURE	Secure Status Register. The SMPU_BDTLS.SECURE bit indicates whether the last transaction that caused the bad address error was secure or non-secure.	
		0	Transaction that caused last bus error was non-secure
		1	Transaction that caused last bus error was secure

SMPU Control Register

The SMPU_CTL register provides access to the locking control, error interrupts and SMPU violations.

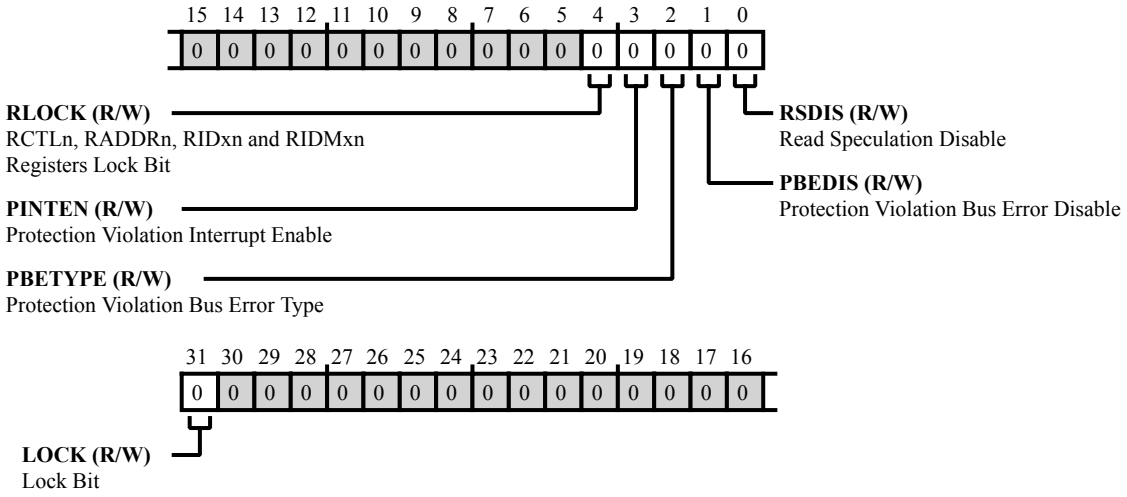

Figure 5-4: SMPU_CTL Register Diagram

Table 5-7: 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 5-7: 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.

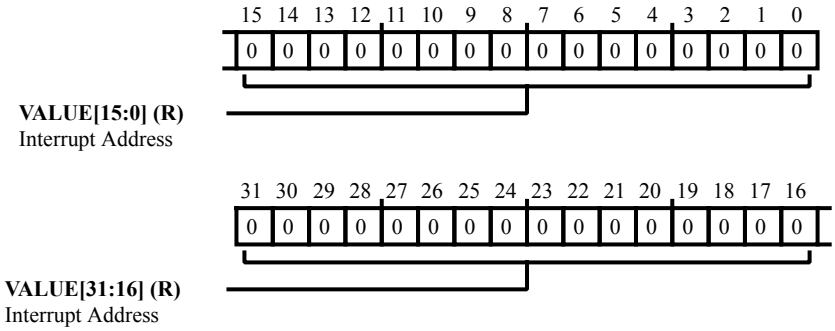

Figure 5-5: SMPU_IADDR Register Diagram

Table 5-8: 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.

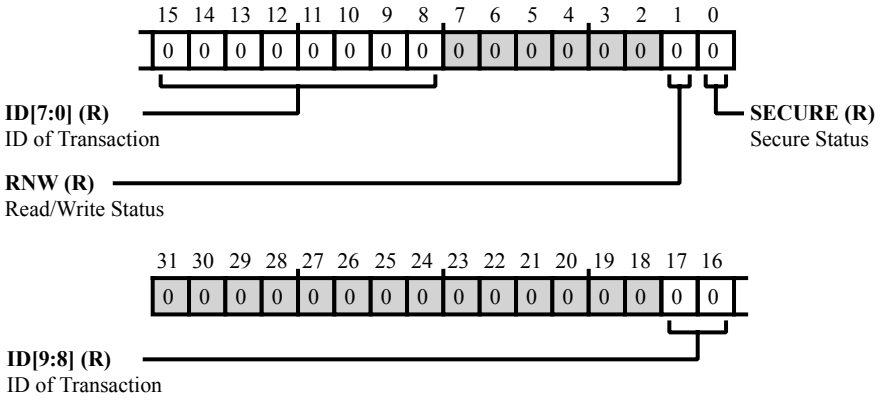

Figure 5-6: SMPU_IDTLS Register Diagram

Table 5-9: SMPU_IDTLS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17:8 (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
0 (R/NW)	SECURE	Secure Status. The SMPU_IDTLS.SECURE bit indicates whether the last transaction that caused the interrupt was secure or non-secure.	
		0	Transaction that caused last signaled interrupt was non-secure
		1	Transaction that caused last signaled interrupt was secure

Region n Address Register

The SMPU_RADDR[n] register is used to define the base address for a memory region to be protected.

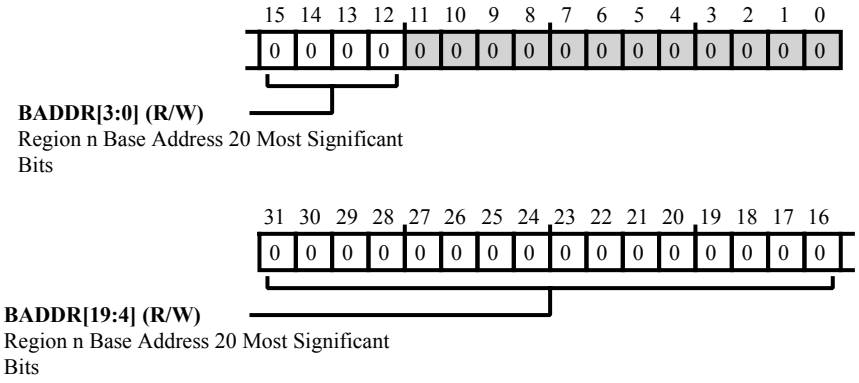

Figure 5-7: SMPU_RADDR[n] Register Diagram

Table 5-10: 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.

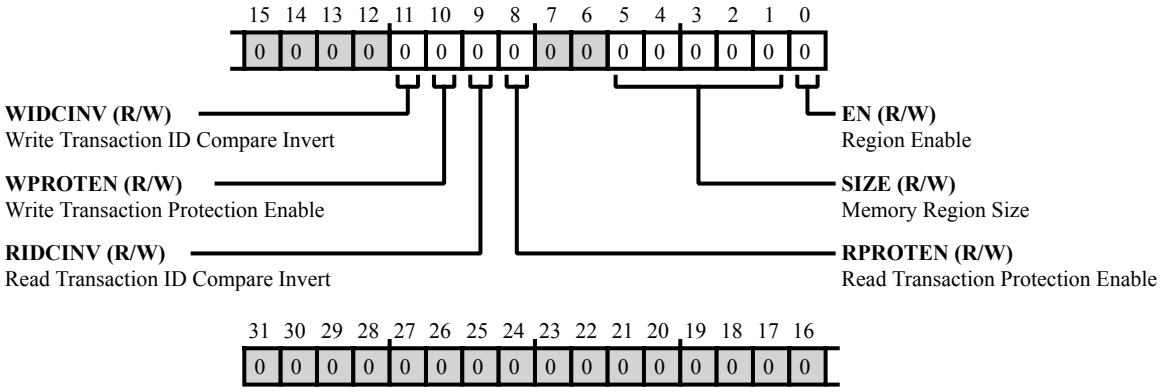

Figure 5-8: SMPU_RCTL[n] Register Diagram

Table 5-11: 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 5-11: 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 5-11: 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.

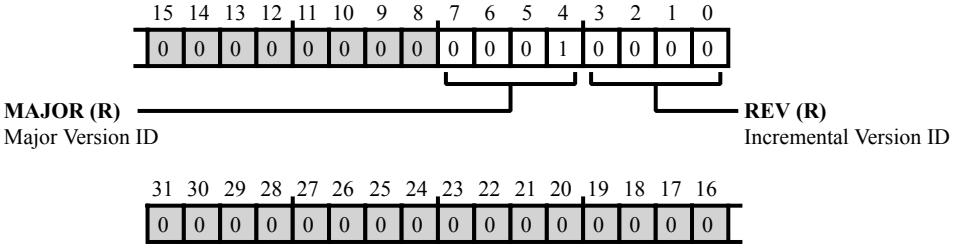

Figure 5-9: SMPU_REVID Register Diagram

Table 5-12: 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.

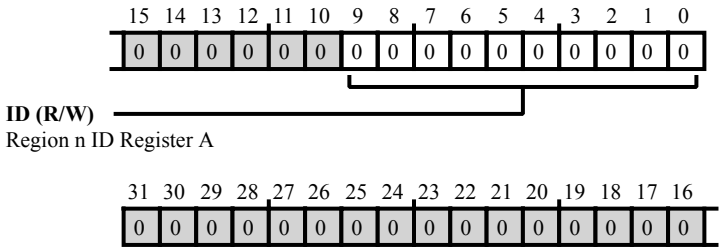

Figure 5-10: SMPU_RIDA[n] Register Diagram

Table 5-13: SMPU_RIDA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (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.

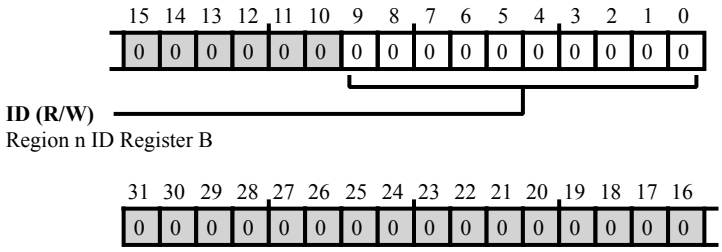

Figure 5-11: SMPU_RIDB[n] Register Diagram

Table 5-14: SMPU_RIDB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (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.

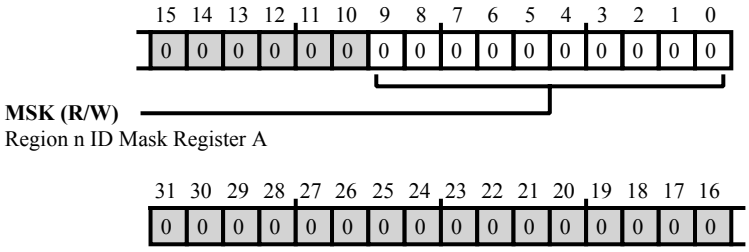

Figure 5-12: SMPU_RIDMSKA[n] Register Diagram

Table 5-15: SMPU_RIDMSKA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (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.

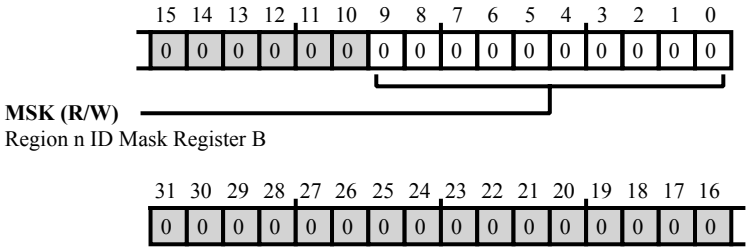

Figure 5-13: SMPU_RIDMSKB[n] Register Diagram

Table 5-16: SMPU_RIDMSKB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
9:0 (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 Control Secure Accesses Register

The SMPU_SECURECTL register provides the bits required to set up the security settings for the processor. These settings includes error generation and read/write security.

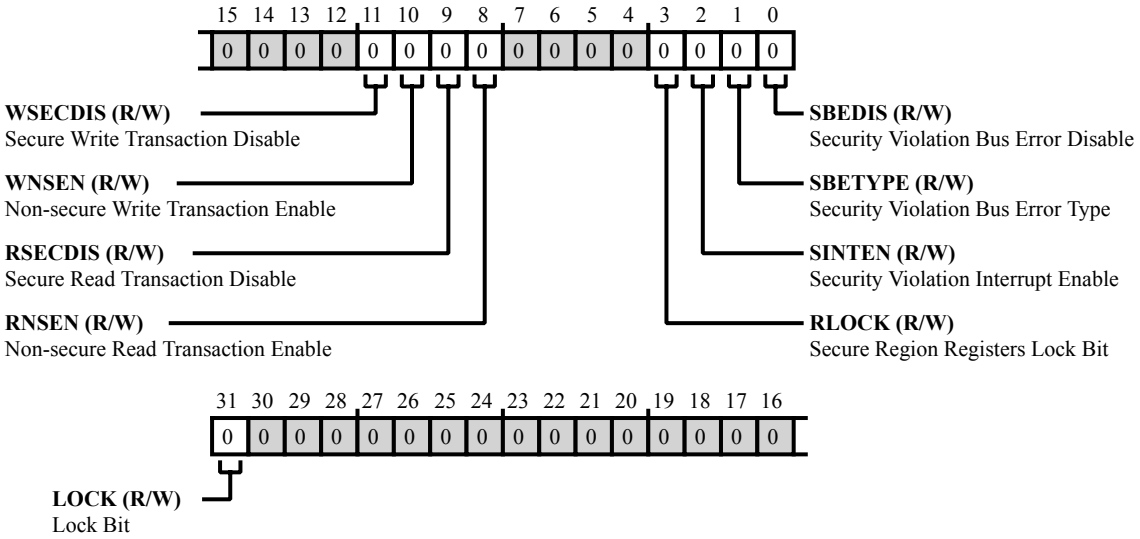

Figure 5-14: SMPU_SECURECTL Register Diagram

Table 5-17: SMPU_SECURECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	LOCK	Lock Bit. When the SMPU_SECURECTL.LOCK bit is set and the global lock signal is asserted from the SPU, the SMPU_SECURECTL register is write-protected. Write-protection is disabled only when the global lock signal becomes deasserted again.	
		0	SMPU_SECURECTL is not write-protected
		1	SMPU_SECURECTL is write-protected
11 (R/W)	WSECDIS	Secure Write Transaction Disable. The SMPU_SECURECTL.WSECDIS bit disables secure write transactions.	
		0	Enable secure write transactions
		1	Disable secure write transactions
10 (R/W)	WNSEN	Non-secure Write Transaction Enable. The SMPU_SECURECTL.WNSEN bit enables non-secure write transactions.	
		0	Disable non-secure writes
		1	Enable non-secure writes

Table 5-17: SMPU_SECURECTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	RSECDIS	Secure Read Transaction Disable. The SMPU_SECURECTL.RSECDIS bit disables secure read transactions.	
		0	Enable secure read transactions
		1	Disable secure read transactions
8 (R/W)	RNSEN	Non-secure Read Transaction Enable. The SMPU_SECURECTL.RNSEN bit enables non-secure read transactions.	
		0	Disable non-secure read transactions
		1	Enable non-secure read transactions
3 (R/W)	RLOCK	Secure Region Registers Lock Bit. When the SMPU_SECURECTL.RLOCK bit is set, the secure region control registers, SMPU_SECURERCTL[n], are write-protected when the global lock signal is active from the SPU. When the global lock signal is deasserted, write access is allowed again.	
		0	Disable write-protection on secure region registers
		1	Enable write-protection on secure region registers
2 (R/W)	SINTEN	Security Violation Interrupt Enable. The SMPU_SECURECTL.SINTEN bit enables interrupt generation when a security violation occurs.	
		0	Disable security settings violation interrupt
		1	Enable security settings violation interrupt
1 (R/W)	SBETYPE	Security Violation Bus Error Type. The SMPU_SECURECTL.SBETYPE bit controls whether a decode error or a slave error is returned when a security violation occurs.	
		0	Return a decode error error which violates the security settings
		1	Return a slave error which violates the security settings
0 (R/W)	SBEDIS	Security Violation Bus Error Disable. The SMPU_SECURECTL.SBEDIS bit controls whether or not a bus error is caused when a security violation occurs.	
		0	Enable bus error
		1	Disable bus error

Region n Control Secure Accesses Register

The SMPU_SECURERCTL[n] register contains bits that configure read/write security for a specific region.

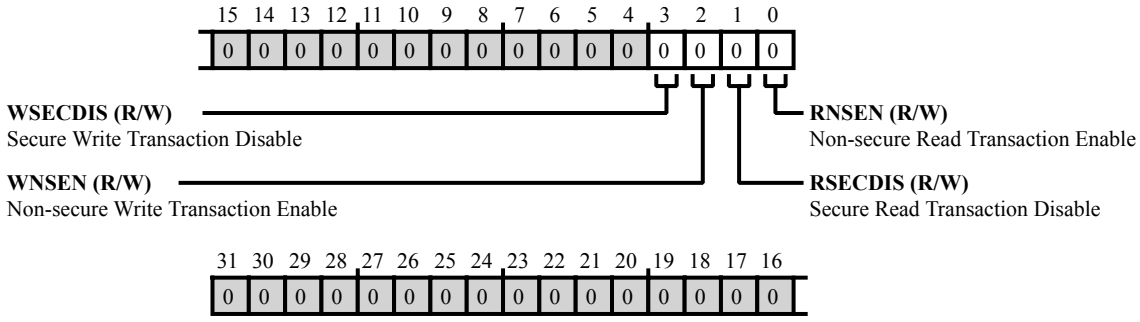

Figure 5-15: SMPU_SECURERCTL[n] Register Diagram

Table 5-18: SMPU_SECURERCTL[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	WSECDIS	Secure Write Transaction Disable. The SMPU_SECURERCTL[n].WSECDIS bit disables secure write transactions for the memory region.	
		0	Enable secure write transactions to this region
		1	Disable secure write transactions to this region
2 (R/W)	WNSEN	Non-secure Write Transaction Enable. This SMPU_SECURERCTL[n].WNSEN bit enables non-secure write transactions for the memory region.	
		0	Disable non-secure write transactions to this region
		1	Enable non-secure write transactions to this region
1 (R/W)	RSECDIS	Secure Read Transaction Disable. The SMPU_SECURERCTL[n].RSECDIS bit disables secure read transactions for the memory region.	
		0	Enable secure read transactions to this region
		1	Disable secure read transactions to this region
0 (R/W)	RNSEN	Non-secure Read Transaction Enable. The SMPU_SECURERCTL[n].RNSEN bit enables non-secure read transactions for the memory region.	
		0	Disable non-secure read transactions to this region
		1	Enable non-secure read transactions to this 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.

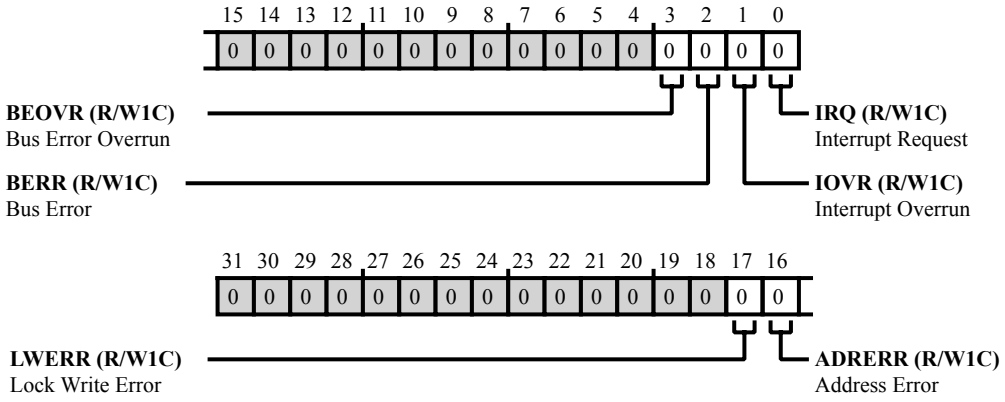

Figure 5-16: SMPU_STAT Register Diagram

Table 5-19: 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 unaligned 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 5-19: 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

6 System Security

The requirement to protect content, keys, IP and other sensitive information have become increasingly prevalent. The processor contains several modules and system elements that contribute to creating a secure operating environment for trusted code to execute. The modules and system elements that can be used are:

- Boot Kernel
- Secure Core
- System Protection Unit (SPU)
- System Memory Protection Unit (SMPU)
- DEBUG through the Test Access Port Controller (TAPC)
- One-Time Programmable (OTP) memory
- Cryptographic Accelerators (optionally)

NOTE: This product includes security features that can be used to protect embedded non-volatile memory contents and prevent execution of non-authorized code. When security is enabled on this device (either by the ordering party or the subsequent receiving parties), Analog Devices' ability to conduct Failure Analysis on returned devices will be limited. Please contact Analog Devices, Inc. for details on the Failure Analysis limitations for this device.

Security Features

The security infrastructure items in the system provide the following features:

- Secure operating environment for secure code execution
- Protect sensitive IP from theft by malicious users or competitors
- Protect sensitive data (for example cipher keys)
- Allow debugging while still maintaining security

Security Functional Description

In order to provide secure operating environment in a system, it requires the involvement of multiple elements.

Boot Kernel

The Boot Kernel is the root of trust for a secure system. Since the boot kernel is developed by Analog Devices, Inc and is stored in ROM and can't be changed, it can be trusted. The boot kernel validates the authenticity of the application binary image that needs to be booted in. It also handles decrypting the binary image if it's encrypted.

Verifying the authenticity of the application binary asserts that

1. The image hasn't been tampered with or altered
2. The image came from a trusted developer

If the image is encrypted, it ensures confidentiality since the boot image is stored on an external storage device and can be more easily read or stolen than in the part.

Once the boot kernel can verify and optionally decrypt the boot image, it can be loaded into the processor for execution. At this point, the *chain of trust* is continued with the verified application.

Secure Booting

Secure booting is when the boot kernel uses cryptographic algorithms to perform checks on the application binary and to decrypt it. Secure booting is done when security is enabled. If security is not enabled the boot kernel does not verify any signatures nor does it perform any decryption on the application binary. See Security Mode Configuration.

NOTE: On the ADSP-BF70x, secure boot uses SHA-224 with Elliptic Curve Digital Signature Algorithm (ECDSA) 224-bit curves. If confidentiality is required, AES-128 CBC is used as well.

Secure Core

In a system with multiple master and slave resources, not everything is considered secure. There are secure and non-secure peripherals and secure and non-secure segments of memory.

For a system to be considerd secure, a secure core that can access and execute instructions (verified application) from secure memory must be configured. Typically, in a single core processor, either the core is hardwired to be secure or the core can switch between secure and non-secure modes.

System Protection Unit (SPU)

The SPU in the system serves two functions. First, it acts as a gatekeeper, guarding against non-secure accesses to secure resources (peripherals). Second, it is used to define which resources in the system are secure or non-secure masters and which resources in the system are secure or non-secure slaves.

NOTE: Even thought the SPU can be configured by secure or non-secure master, the first steps that the verified secure application should perform are:

1. Configure the SPU as a secure slave itself so only other secure masters can configure it
2. Define and configure the secure masters and slaves using the SPU

This way, once the SPU is secure and other secure masters and slaves are configured, non-secure masters cannot tamper with the security privileges of secure masters and slaves nor can non-secure resources be changed to a secure resource.

System Memory Protection Unit (SMPU)

Similar to the SPU protecting the MMR address range for peripherals, the SMPU can guard memory ranges or pages. Memory pages can be configured as secure or non-secure. Again, like in the case of the SPU, the SMPU can guard against non-secure transaction attempts to secure memory.

NOTE: A verified application should reside in memory configured as secure. By default the SMPU has all memory configured as secure. If the program needs to update the memory protections via the SMPU, the application and it's data should remain in secure memory.

Debug

The typical way of accessing a system is through the debug port via a JTAG or serial wire interface. If these access points are not secure sensitive IP such as cipher keys can be exposed and code can be changed to disable other security settings. To guard against this type of attack or security hole and still provide debugging capabilities for a developer, a debug unit with security features is used.

When the developer first receives the part, they can define a JTAG/DEBUG key that is programmed into OTP memory. Once security is enabled, the debug unit compares the key sent from the host debugger with the key inside the system. If a match occurs, debug access to secure resources are allowed.

NOTE: On the ADSP-BF70x, the Secure JTAG debug key is 128-bits stored at USERKEY[3:0] in OTP memory.

One-Time-Programmable (OTP) Memory

Customer programmable OTP memory is used to safely and securely store sensitive information such as cipher keys.

In a public key algorithm (for example ECDSA which is used in secure boot), the public key is used to verify the digital signature that accompanies the application binary. The private key is used by the developer on the host development machine to create the digital signature. The public and private key pair is unique and the public key needs to be stored in non-volatile, one-time-programmable memory. If the public key is allowed to be changed than users can generate their own public/private key pairs and successfully boot in malicious code. This can either re-purpose the part or change security configurations to allow easier access to sensitive information stored elsewhere in OTP memory.

Cryptographic Accelerators

Cryptographic algorithms are mathematical tools to help provide security. Hardware engines provide some advantages but are not necessarily required. For computationally expensive operations like those used in Elliptic Curve Cryptography, like ECDSA used in secure boot, the operations can be accelerated while the core performs other tasks. Also, it's less likely that the hardware engine can be hacked to change the results.

Security Mode Configuration

Because the processor is unsecure by default, if security is not required then no steps need to be done.

To enable security features, use the following procedure.

- Generate the public/private key pair on the host development machine.[1]
- Program OTP memory with the public key.[2]
- Program OTP memory with the decryption key if the application binary needs to be encrypted.[2]
- Program OTP memory with the debug/JTAG key.[2]
- Develop the application and sign it, creating the digital signature with the private key.[1]
- If confidentiality is required, encrypt the application binary before signing it.
- Set the LOCK bit in OTP memory to enable security. After this, subsequent boots are secure boots.[2, 3]

[1] Software tools are provided with developments tools to generate keys, sign boot streams and also perform encryption. Refer to the development tools manuals for information on usage.

[2]Refer to the OTP Chapter for programming the OTP and other related information.

[3]Refer to the Booting Chapter for more information on Secure Booting and other related information.

Status and Error Signals

In a fully functional secure system, non-secure resources should not even attempt to access a secure resource. If this does occur, then either the code has been altered or replaced with malicious code or the system contains a bug.

Errors or error events are dependant on the configuration of the SPU and SMPU, the protection units which guard against security violations. In the case of the SMPU, the error can simply be captured, captured and interrupt generated, or the access prevented without capturing any error. It is the developer's responsibility to:

1. Determine if there was an error due to a blocked access.
2. Determine how to handle a blocked access (for example fix the bug (offline), or try to use a different resource (runtime).

7 Dynamic Power Management (DPM)

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

DPM provides facility for shutting down clocks for various peripherals. Shutting down peripheral's clocks is considered a power saving feature.

NOTE: The DPM peripheral disable feature is intended to be used only in a static configuration. There is no software register support for dynamically enabling and disabling a peripheral.

DPM Features

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

- Aids power savings through hibernate mode, which allows the VDD_INT supply to be shut off
- Permits operation of multiple, external wake-up sources

DPM Functional Description

The DPM can be programmed to transition between power modes.

ADSP-BF70x DPM Register List

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

Table 7-1: ADSP-BF70x DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_HIB_DIS	Hibernate Disable Register
DPM_PGCNTR	Power Good Counter Register
DPM_RESTORE[n]	Restore Registers

Table 7-1: ADSP-BF70x DPM Register List (Continued)

Name	Description
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.

Hibernate mode

A power-saving mode in which the VDD_INT supply can be shut off, and the contents of on-chip memory are not retained.

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
- HIBERNATE

The ***Operating Modes and Transitions*** figure shows the relationships between DPM modes for the ADSP-BF70x processor.

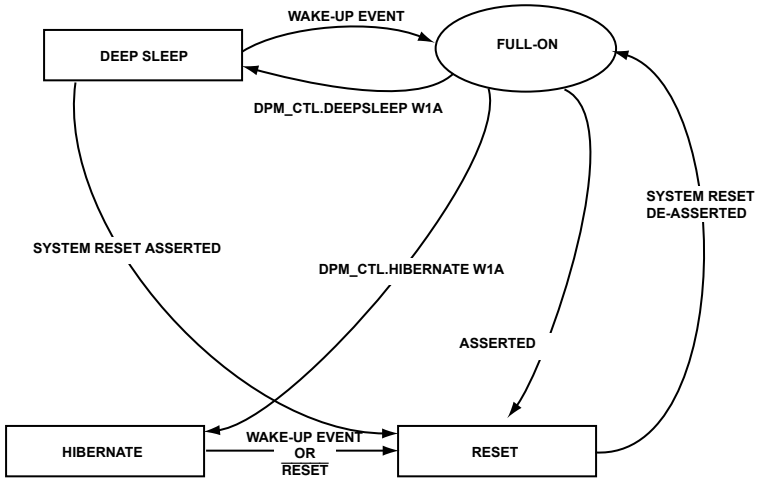

Figure 7-1: Operating Modes and Transitions

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.

If the DPM returns from hibernate mode, certain registers (see HIBERNATE) are exceptions to this return-to-default state and are preserved.

From RESET, the DPM always transitions to FULL-ON State.

Full-on Mode

Full-on mode is the default state of the DPM after RESET.

In full-on mode, the processor can reach its maximum clock rate and power dissipation can be at its highest. The DPM transitions from full-on mode to:

- Deep sleep mode if the `DPM_CTL.DEEPSLEEP` bit is set
- Hibernate mode if the `DPM_CTL.HIBERNATE` bit is set

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 SEC
- 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.

Hibernate Mode (ADSP-BF70x)

In hibernate mode, there is no power dissipation on the VDD_INT power domain as long as the voltage regulator powering VDD_INT is shut off. If state information, data, or code is preserved, it can be stored in the DPM_RESTORE[n] registers or in external memory. If dynamic memory is used, the memory device is placed into self-refresh mode before going to hibernate.

Hibernate mode is entered by setting the DPM_CTL.HIBERNATE bit. It is the programs responsibility in software to guarantee that system transfers including DMA are stopped before entering hibernate mode.

After the DPM_CTL.HIBERNATE bit has been set, the DPM preserves the state of the DPM_WAKE_EN, DPM_WAKE_POL, DPM_HIB_DIS, DPM_PGCNTR, and all of the DPM_RESTORE[n] registers in the DPM-HV. The VDD_EXT power domain powers the DPM-HV. All other registers (except the registers mentioned in the RCU chapter) and on-chip memory are erased when the VDD_INT supply is powered down. After the contents of the registers listed are moved to storage on the VDD_EXT power domain, the SYS_EXTWAKE signal is deasserted. This deassertion indicates that it is safe to shut off the regulator providing the VDD_INT supply.

Enabled hardware wake-ups or a hardware reset can begin the process of exiting hibernate mode. The DPM_WAKE_EN.WS[n] bits and the DPM_WAKE_POL.WS[n] bit determine which hardware wake ups are enabled and select their signal polarity. Wake-up assertions are latched only if they are enabled. The enabled wake-up assertion that occurs first is recorded in the DPM_WAKE_STAT register. After the processor latches a wake-up source

assertion, the SYS_EXTWAKE signal is asserted to indicate that the regulator for the VDD_INT supply is ramping up. The DPM waits for:

- the expiration of the DPM_PGCNTR.CNT counter

These conditions indicate that the VDD_INT supply has been restored within the operating conditions specified in the data sheet. For more information, see the section "Ensuring Internal Logic Supply is Restored Before Booting."

After the VDD_INT supply is restored, and the actions from the list occur, the processor exits hibernate mode, goes into reset state, and the PLL begins locking. While the PLL is locking, the DPM_WAKE_EN, DPM_WAKE_POL, DPM_HIB_DIS, DPM_PGCNTR, and all of the DPM_RESTORE[n] registers are restored from the DPM-HV. After coming out of reset mode and locking the PLL, the processor begins executing the boot code. Some processors support memory boot when returning from hibernate (instead of the boot mode selected by the SYS_BMODE0 signals). For more information, see the Booting chapter.

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_WAKE_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_WAKE_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

Ensuring Internal Logic Supply is Restored Before Booting

Before the processor boots after returning from hibernate mode, the internal logic supply (VDD_INT power domain) must be restored. This topic discusses how to signal the processor that the VDD_INT power domain is restored.

Using the PG Counter to Check that the Internal Logic Supply is Restored

The program must select the appropriate setting for DPM_PGCNTR.CNT values such that when the counter expires, the VDD_INT power domain is restored to within the operating conditions specified in the data sheet. The amount

of time that the counter takes to expire depends on both the DPM_PGCNTR.CNT value and the SYS_CLKIN frequency according to this equation:

Expiration Time [ms] = Cycle Count × 16 × SYS_CLKIN period [ms]

The text in brackets indicates that both the expiration time and SYS_CLKIN period are expressed in milliseconds. To determine the cycle count, refer to the ***Cycle Counts for Various DPM_PGCNTR.CNT Bit Field Settings*** table which shows cycle counts for the different DPM_PGCNTR.CNT value selections.

Table 7-2: Cycle Counts for Various DPM_PGCNTR.CNT Bit Field Settings

DPM_PGCNTR.CNT Values	**Cycle Count**
0x0004	128 (Default)
0x0080	4,096
0x0100	8,192
0x0200	16,384
0x0400	32,768
0x0800	65,536
0x1000	131,072
0x2000	262,144
0x4000	524,288
0x8000	1,048,576
All other values not listed	Reserved

The ***Example DPM Counter-Expiration Time Calculations*** table shows examples of different SYS_CLKIN frequencies and DPM_PGCNTR.CNT values and the resulting power good counter-expiration times. These examples are reference calculations only. The SYS_CLKIN frequency selected must be within the range allowed by the processor-specific data sheet.

Table 7-3: Example DPM Counter-Expiration Time Calculations

DPM_PGCNTR.CNT Setting	**Cycle Count (Cycles)**	**SYS_CLKIN Frequency (MHz)**	**SYS_CLKIN Period (ms)**	**Expiration Time (ms)**
0x0400	32,768	50	0.00002	10.45696
0x0800	65,536	40	0.000025	26.2144
0x4000	524,288	25	0.00004	335.54432

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.

Configuring Hibernate Mode

The hibernate mode permits the VDD_INT supply to be shut off to reduce power dissipation.

The DPM enters hibernate mode from any DPM state that can run software. Reading the DPM_STAT.CURMODE reveals the current power mode. Make sure to save any data that must be preserved to the DPM_RESTORE[n] registers or to external memory before placing the processor into hibernate mode.

Perform the following steps in code running from L1 memory on Blackfin processors.

1. Set the polarity of wake-up sources as needed with the DPM_WAKE_POL.WS[n] bits.
2. Enable the wake-up sources as needed with the DPM_WAKE_EN.WS[n] bits.
3. Write to the DPM_HIB_DIS register to disable any blocks not on the VDD_INT domain.

   ADDITIONAL INFORMATION: If there are no blocks on domains other than VDD_INT that can be disabled during hibernate, the DPM_HIB_DIS register does not exist on your processor.
4. Write the power good count value to the DPM_PGCNTR.CNT field.
5. Clear all pending core transactions (this action is done with an SSYNC instruction in Blackfin), DMA transactions, and interrupts.
6. Set the DPM_CTL.HIBERNATE bit

The DPM is now configured to hibernate mode.

ADSP-BF70x Wake-Up Sources

The *ADSP-BF70x HIBERNATE and DEEP SLEEP Wake-up Sources* table shows the hibernate mode and deep sleep mode wake-up sources for the ADSP-BF70x processor.

The first column shows which wake-up source bit (*WSn*) the processor uses in the DPM_WAKE_EN, DPM_WAKE_POL, and the DPM_WAKE_STAT registers. The *Assigned Source* column shows which peripheral or pin source is assigned to the *WSn* bit. Peripherals in parentheses indicate that the source can either be used as a general-purpose I/O wake-up or used as the specific peripheral wake-up listed. The *Deep Sleep Mode* column indicates whether the wake-up source can wake the processor from deep sleep mode. The *Hibernate Mode* column indicates whether the wake-up source can wake the processor from hibernate mode.

Table 7-4: ADSP-BF70x HIBERNATE and DEEP SLEEP Wake-up Sources

DPM_WAKE_EN.WS[n], DPM_WAKE_POL.WS[n], and DPM_WAKE_STAT.WS[n] bits	**Assigned Source**	**Deep Sleep Mode?**	**Hibernate Mode?**
WS0	PB_07	Yes	Yes
WS1	PB_08	Yes	Yes
WS2	PB_12	Yes	Yes
WS3	PC_02 (CAN0_RX)	Yes	Yes
WS4	PA_12 (CAN1_RX)	Yes	Yes
WS5	USB (DPM_WAKE_POL.WS5 must =1 for USB)	Yes	Yes
WS6	RTC (DPM_WAKE_POL.WS6 must =1 for RTC)	Yes	Yes
WS30:WS7	RESERVED	N/A	N/A

ADSP-BF70x Hibernate Disable Bit Assignments

The *ADSP-BF70x Bit Assignments* table shows the functional units that can be disabled during hibernate on the ADSP-BF70x processors.

The first column shows the hibernate disable bits (DPM_HIB_DIS.HD[n]). The *Functional Unit* column shows which functional unit is assigned to each DPM_HIB_DIS.HD[n] bit, permitting disable on entering hibernate mode.

Table 7-5: ADSP-BF70x Bit Assignments for DPM_HIB_DIS.HD[n]

DPM_HIB_DIS.HD[n] Bits	**Functional Unit**
HD0	System crystal oscillator
HD30:HD1	Reserved

ADSP-BF70x DPM Register Descriptions

Dynamic Power Management (DPM) contains the following registers.

Table 7-6: ADSP-BF70x DPM Register List

Name	Description
DPM_CTL	Control Register
DPM_HIB_DIS	Hibernate Disable Register
DPM_PGCNTR	Power Good Counter Register
DPM_RESTORE[n]	Restore Registers
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.

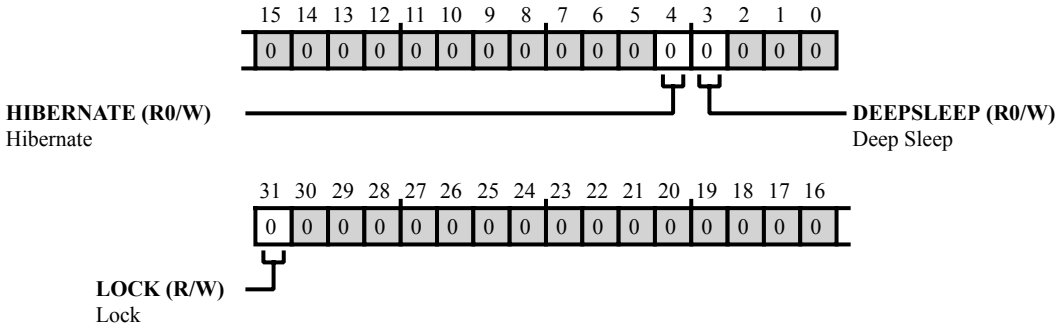

Figure 7-2: DPM_CTL Register Diagram

Table 7-7: 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
4 (R0/W)	HIBERNATE	Hibernate. The DPM_CTL.HIBERNATE bit puts the processor into hibernate mode. The processor stays in this mode until a wakeup event or reset occurs. For more information about DPM modes, see the operating modes.	
		0	No Action
		1	Hibernate
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

Hibernate Disable Register

Bits in the DPM_HIB_DIS register can written before going to HIBERNATE to disable hardware in the VDD_EXT power domain during HIBERNATE.

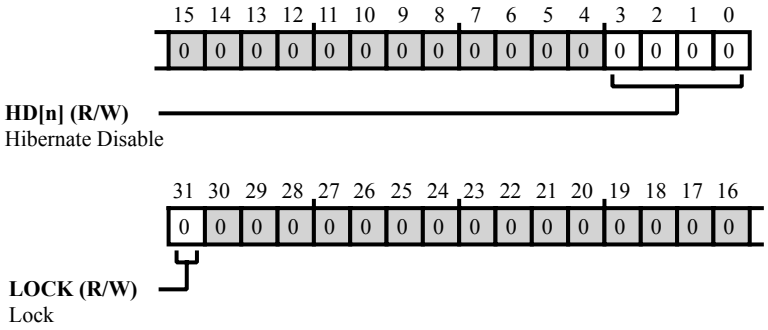

Figure 7-3: DPM_HIB_DIS Register Diagram

Table 7-8: DPM_HIB_DIS 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_HIB_DIS.LOCK bit is set, the DPM_HIB_DIS register is read only (locked).	
		0	Unlock
		1	Lock
3:0 (R/W)	HD[n]	Hibernate Disable. The DPM_HIB_DIS register controls which functional units on the V_{DD_EXT} domain are disabled during hibernate. For the bit assignments of this processor, see the DPM Hibernate Disable Bit Assignments topic.	

Power Good Counter Register

The DPM_PGCNTR register selects the count of CLKIN cycles the DPM waits for the V_{DDINT} power supply to reach the specified value. This register includes a write protection lock.

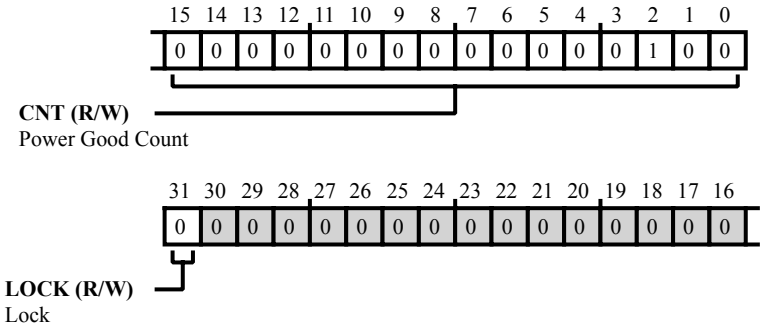

Figure 7-4: DPM_PGCNTR Register Diagram

Table 7-9: DPM_PGCNTR 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_PGCNTR.LOCK bit is set, the DPM_PGCNTR register is read only (locked).	
		0	Unlock
		1	Lock
15:0 (R/W)	CNT	Power Good Count. The DPM_PGCNTR.CNT bits select the count that the DPM uses to determine that the V_{DDINT} supply has reached the specified value. All values other than those shown are reserved.	
		4	128 cycles
		128	4K cycles
		256	8K cycles
		512	16K cycles
		1024	32K cycles
		2048	64K cycles
		4096	128K cycles
		8192	256K cycles
		16384	512K cycles
		32768	1M cycles

Restore Registers

The DPM_RESTORE[n] registers are general-purpose registers that the DPM preserves during hibernate mode and restores on exit from hibernate. The usage of these registers is application specific and does not affect DPM operations. For more information about using the DPM_RESTORE[n] registers, see the programming model.

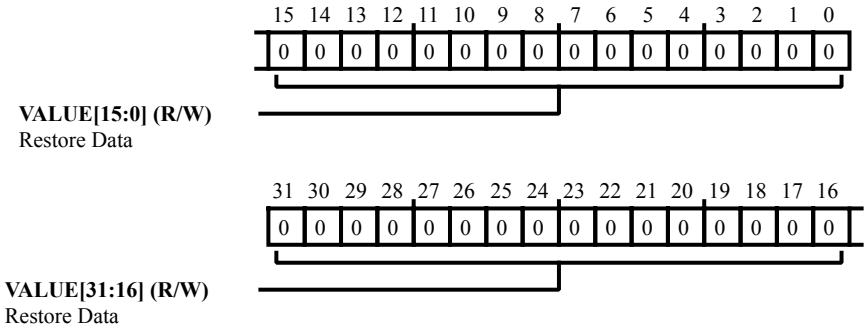

Figure 7-5: DPM_RESTORE[n] Register Diagram

Table 7-10: DPM_RESTORE[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Restore Data. The DPM_RESTORE[n].VALUE data is stored during hibernate mode and restored by the DPM on exit from hibernate.

Revision ID

The DPM_REVID register provides the revision of the DPM module.

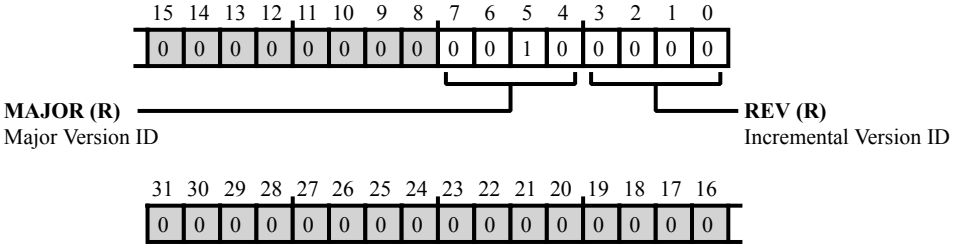

Figure 7-6: DPM_REVID Register Diagram

Table 7-11: 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

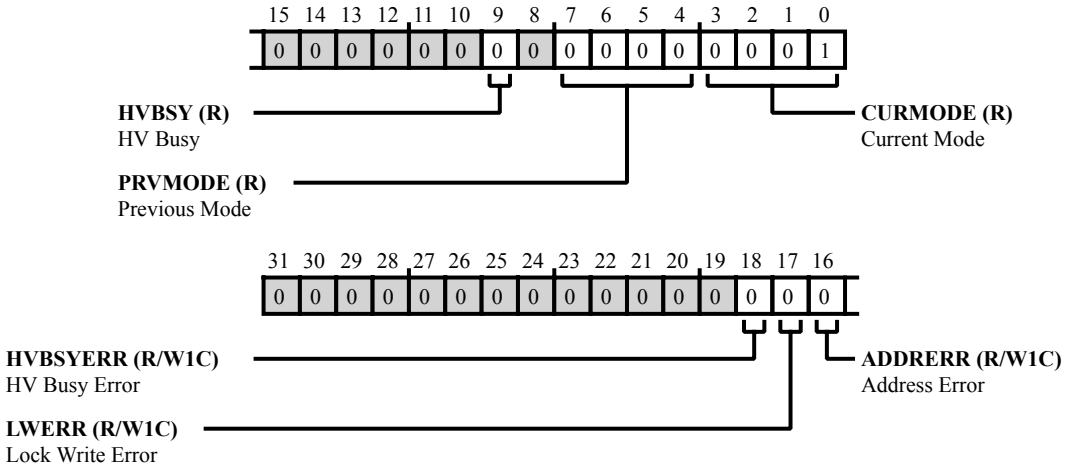

Figure 7-7: DPM_STAT Register Diagram

Table 7-12: DPM_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W1C)	HVBSYERR	HV Busy Error. The `DPM_STAT.HVBSYERR` bit indicates that a read access of `DPM_WAKE_STAT` or `DPM_RESTORE[n]` occurred during restore of DPM LV from DPM HV	
		0	Inactive
		1	Active
17 (R/W1C)	LWERR	Lock Write Error. The `DPM_STAT.LWERR` bit indicates an attempt to write to a locked DPM register while the global lock bit is set (`SPU_CTL.GLCK` bit =1).	
		0	Inactive
		1	Active
16 (R/W1C)	ADDRERR	Address Error. The `DPM_STAT.ADDRERR` bit indicates either an attempted read or write access to an address in the DPM MMR space that are not implemented addresses, an attempted write access to a DPM read only register, or a read or write access to a non aligned address in the DPM MMR space.	
		0	No address error
		1	Read or write transactions try to access unimplemented addresses or write transactions try to access a read only register or accesses are non aligned.

Table 7-12: DPM_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/NW)	HVBSY	HV Busy. The DPM_STAT.HVBSY bit indicates whether the DPM HV unit is ready (has completed all transactions) or is busy (transfers from HV have not yet completed).	
		0	Not Busy (ready)
		1	Busy
7:4 (R/NW)	PRVMODE	Previous Mode. The DPM_STAT.PRVMODE bits indicate the previous DPM mode of operation. All values not shown are reserved.	
		0	Reset
		1	Full-On
		2	Reserved
		3	Reserved
		4	Deep Sleep
		5	Hibernate
		6-15	Reserved
3:0 (R/NW)	CURMODE	Current Mode. The DPM_STAT.CURMODE bits indicate the current DPM mode of operation. All values not shown are reserved.	
		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 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.

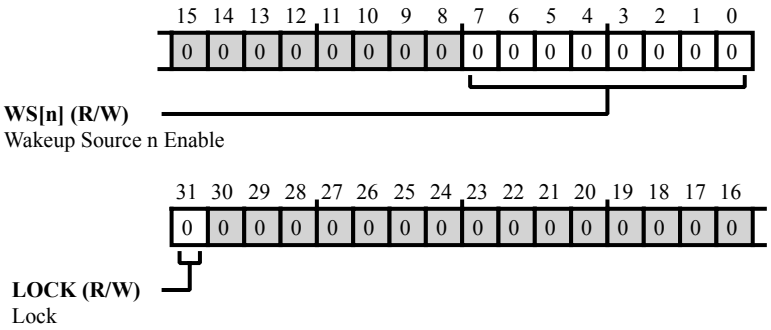

Figure 7-8: DPM_WAKE_EN Register Diagram

Table 7-13: 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
7:0 (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.

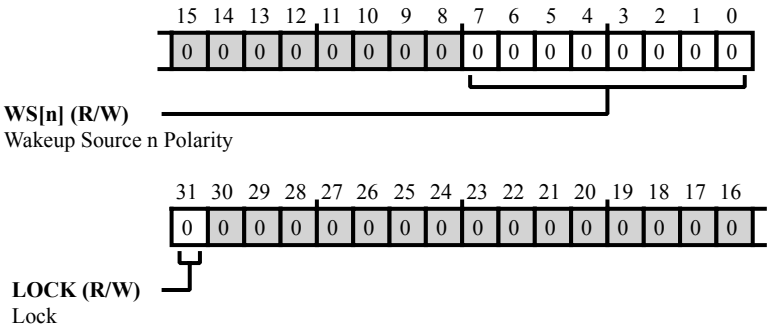

Figure 7-9: DPM_WAKE_POL Register Diagram

Table 7-14: 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
7:0 (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
		255	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.

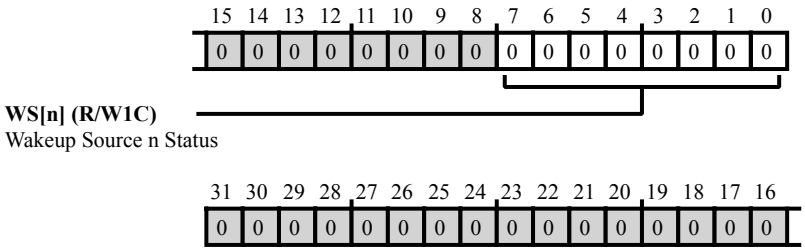

Figure 7-10: DPM_WAKE_STAT Register Diagram

Table 7-15: DPM_WAKE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:0 (R/W1C)	WS[n]	Wakeup Source n Status. The DPM_WAKE_STAT.WS[n] bits indicate the enabled and active status 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	No Status
		255	Enabled and Active

8 System Event Controller (SEC)

System event management is the responsibility of the system event controller (SEC). The SEC manages the configuration of all system event sources. The SEC also manages the propagation of system events to all connected cores and the system fault interface.

SEC Features

The following list describes the system event controller features.

- Comprehensive system event source management including interrupt enable, fault enable, priority, core mapping, and source grouping.
- Fault management including fault action configuration, timeout, external indication, and system reset.
- Determinism where all system events have the same propagation delay and provide unique identification of a specific system event source.
- Distributed programming model where each system event source control and all status fields are independent of all others.
- Slave control port which provides access to all SEC registers for configuration, status, and interrupt or fault service model.
- Global locking supports a register level protection model to prevent writes to “locked” registers.

SEC Functional Description

The following sections provide a functional description of the SEC.

ADSP-BF70x SEC Register List

The System Event Controller (SEC) manages the system interrupt and/or system fault sources, including control features such as enable/disable, prioritization, and active/pending source status. For more information on SEC functionality, see the SEC register descriptions.

Table 8-1: ADSP-BF70x SEC Register List

Name	Description
SEC_CACT[n]	SCI Active Register n
SEC_CCTL[n]	SCI Control Register n
SEC_CGMSK[n]	SCI Group Mask Register n
SEC_CPLVL[n]	SCI Priority Level Register n
SEC_CPMSK[n]	SCI Priority Mask Register n
SEC_CPND[n]	Core Pending Register n
SEC_CSID[n]	SCI Source ID Register n
SEC_CSTAT[n]	SCI Status Register n
SEC_END	Global End Register
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

ADSP-BF70x Interrupt List

Table 8-2: ADSP-BF70x Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
0	Reserved	Reserved	Reserved	Reserved
1	SEC0_ERR	SEC0 Fault Interrupt	Level	
2	CGU0_EVT	CGU0 Event	Edge	

Table 8-2: ADSP-BF70x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
3	WDOG0_EXP	WDOG0 Expiration	Level	
4	L2CTL0_ECC_ERR	L2CTL0 ECC Error	Level	
5	Reserved	Reserved	Reserved	Reserved
6	SYS_C0_DBL_FAULT	Core 0 Double Fault	Edge	
7	SYS_C0_HW_ERR	Core 0 Hardware Error	Edge	
8	SYS_C0_NMI_L1_PARITY_ERR	Core 0 Unhandled NMI or L1 Memory Parity Error	Edge	
9	L2CTL0_EVT	L2CTL0 L2 Event (Scrub or Initialization)	Level	
10	Reserved	Reserved	Reserved	Reserved
11	TIMER0_TMR0	TIMER0 Timer 0	Level	
12	TIMER0_TMR1	TIMER0 Timer 1	Level	
13	TIMER0_TMR2	TIMER0 Timer 2	Level	
14	TIMER0_TMR3	TIMER0 Timer 3	Level	
15	TIMER0_TMR4	TIMER0 Timer 4	Level	
16	TIMER0_TMR5	TIMER0 Timer 5	Level	
17	TIMER0_TMR6	TIMER0 Timer 6	Level	
18	TIMER0_TMR7	TIMER0 Timer 7	Level	
19	TIMER0_STAT	TIMER0 Status	Level	
20	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level	
21	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level	
22	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level	
23	EPPI0_STAT	EPPI0 Status	Level	
24	EPPI0_CH0_DMA	EPPI0 Channel 0 DMA	Level	14
25	EPPI0_CH1_DMA	EPPI0 Channel 1 DMA	Level	15
26	SYS_DMAC_ERR	DDE Aggregated Error	Level	
27	CNT0_STAT	CNT0 Status	Level	
28	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
29	SPORT0_A_DMA	SPORT0 Channel A DMA	Level	0
30	SPORT0_B_STAT	SPORT0 Channel B Status	Level	
31	SPORT0_B_DMA	SPORT0 Channel B DMA	Level	1

Table 8-2: ADSP-BF70x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
32	SPORT1_A_STAT	SPORT1 Channel A Status	Level	
33	SPORT1_A_DMA	SPORT1 Channel A DMA	Level	2
34	SPORT1_B_STAT	SPORT1 Channel B Status	Level	
35	SPORT1_B_DMA	SPORT1 Channel B DMA	Level	3
36	SPI0_ERR	SPI0 Error	Level	
37	SPI0_STAT	SPI0 Status	Level	
38	SPI0_TXDMA	SPI0 TX DMA Channel	Level	4
39	SPI0_RXDMA	SPI0 RX DMA Channel	Level	5
40	SPI1_ERR	SPI1 Error	Level	
41	SPI1_STAT	SPI1 Status	Level	
42	SPI1_TXDMA	SPI1 TX DMA Channel	Level	6
43	SPI1_RXDMA	SPI1 RX DMA Channel	Level	7
44	SPI2_ERR	SPI2 Error	Level	
45	SPI2_STAT	SPI2 Status	Level	
46	SPI2_TXDMA	SPI2 TX Channel	Level	8
47	SPI2_RXDMA	SPI2 RX Channel	Level	9
48	UART0_STAT	UART0 Status	Level	
49	UART0_TXDMA	UART0 Transmit DMA	Level	10
50	UART0_RXDMA	UART0 Receive DMA	Level	11
51	UART1_STAT	UART1 Status	Level	
52	UART1_TXDMA	UART1 Transmit DMA	Level	12
53	UART1_RXDMA	UART1 Receive DMA	Level	13
54	SYS_MDMA0_SRC	Memory DMA Stream 0 Source	None	16
55	SYS_MDMA0_DST	Memory DMA Stream 0 Destination	None	17
56	SYS_MDMA1_SRC	Memory DMA Stream 1 Source / CRC0 Input Channel	None	18
57	SYS_MDMA1_DST	Memory DMA Stream 1 Destination / CRC0 Output Channel	None	19
58	SYS_MDMA2_SRC	Memory DMA Stream 2 Source / CRC1 Input Channel	None	20
59	SYS_MDMA2_DST	Memory DMA Stream 2 Destination / CRC1 Output Channel	None	21

Table 8-2: ADSP-BF70x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
60	HADC0_EVT	HADC0 Event	Edge	
61	RTC0_EVT	RTC0 Event	Level	
62	TWI0_DATA	TWI0 Data Interrupt	Level	
63	CRC0_DCNTEXP	CRC0 Datacount expiration	Level	
64	CRC0_ERR	CRC0 Error	Level	
65	CRC1_DCNTEXP	CRC1 Data Count Expiration	Level	
66	CRC1_ERR	CRC1 Error	Level	
67	PKTE0_IRQ	PKTE0 Security Packet Engine Interrupt	Level	
68	PKIC0_IRQ	PKIC0 Public Key Interrupt	Level	
69	Reserved	Reserved	Reserved	Reserved
70	Reserved	Reserved	Reserved	Reserved
71	OTPC0_ERR	OTPC0 Dual-bit error	Level	
72	MSI0_STAT	MSI0 Status	Level	
73	SMPU0_ERR	SMPU0 Error (L2)	Level	
74	SMPU1_ERR	SMPU1 Error (DMC)	Level	
75	Reserved	Reserved	Reserved	Reserved
76	SPU0_INT	SPU0 Interrupt	Level	
77	USB0_STAT	USB0 Status/FIFO Data Ready	Level	
78	USB0_DATA	USB0 DMA Status/Transfer Complete	Level	
79	TRU0_SLV0	TRU0 Interrupt 0	Edge	
80	TRU0_SLV1	TRU0 Interrupt 1	Edge	
81	TRU0_SLV2	TRU0 Interrupt 2	Edge	
82	TRU0_SLV3	TRU0 Interrupt 3	Edge	
83	CGU0_ERR	CGU0 Error	Level	
84	DPM0_EVT	DPM0 Event	Level	
85	SPIHP0_ERR	SPIHP0 Error	Level	
86	SYS_SOFT0_INT	Software-Driven Interrupt 0	Edge	
87	SYS_SOFT1_INT	Software-Driven Interrupt 1	Edge	
88	SYS_SOFT2_INT	Software-Driven Interrupt 2	Edge	
89	SYS_SOFT3_INT	Software-Driven Interrupt 3	Edge	
90	CAN0_RX	CAN0 Receive	Level	

Table 8-2: ADSP-BF70x Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
91	CAN0_TX	CAN0 Transmit	Level	
92	CAN0_STAT	CAN0 Status	Level	
93	CAN1_RX	CAN1 Recieve	Level	
94	CAN1_TX	CAN1 Transmit	Level	
95	CAN1_STAT	CAN1 Status	Level	
96	CTI_C0_EVT	Core 0 CTI Event (CTI0)		
97	SWU0_EVT	SWU0 Event (L1)	None	
98	SWU1_EVT	SWU1 Event (Core L2)	None	
99	SWU2_EVT	SWU2 Event (DMA L2)	None	
100	SWU3_EVT	SWU3 Event (MMR)	None	
101	SWU4_EVT	SWU4 Event (DMC)	None	
102	SWU5_EVT	SWU5 Event (SMC)	None	
103	SWU6_EVT	SWU6 Event (SPIF)	None	
104	SWU7_EVT	SWU7 Event (OTP)	None	
105	TAPC0_KEYFAIL	TAPC0 User Key Fail Interrupt	Edge	

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.

SCI

SEC core interface, core interface subblock of the SEC

SPR

SEC prioritizer determines the highest priority pending interrupt and the highest priority active interrupt. The SPR provides these interrupts in the appropriate registers of the SCI for the priority and nesting model of the SCI.

SFI

SEC Fault Interface, fault management subblock of the SEC.

SEC Block Diagram

The ***SEC Block Diagram*** shows the event management architecture.

System sources connect to the SEC through the SSI. Each core has a dedicated SCI. The SFI provides fault action connections to the rest of the system.

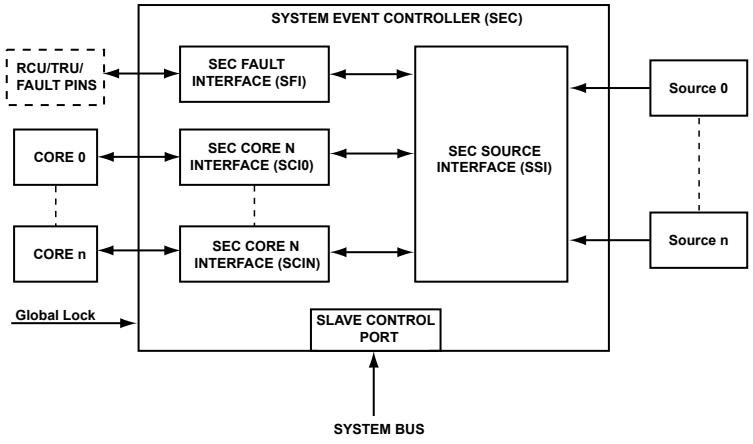

Figure 8-1: SEC Block Diagram

SFI Block Diagram

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.

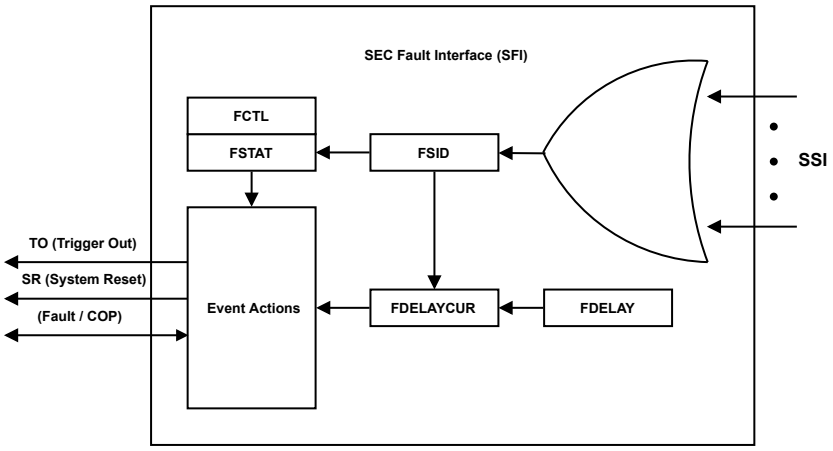

Figure 8-2: SFI Block Diagram

SCI Block Diagram

The SCI manages communication between the corresponding core and the SEC. The SPR of the SCI receives pending, active, and priority information from the SSI for each system event source assigned to this SCI. The SPR determines the highest-priority pending system event and the SCI determines whether it propagates to the core. The SCI maintains the coherency for the system event service model implemented on the connected core.

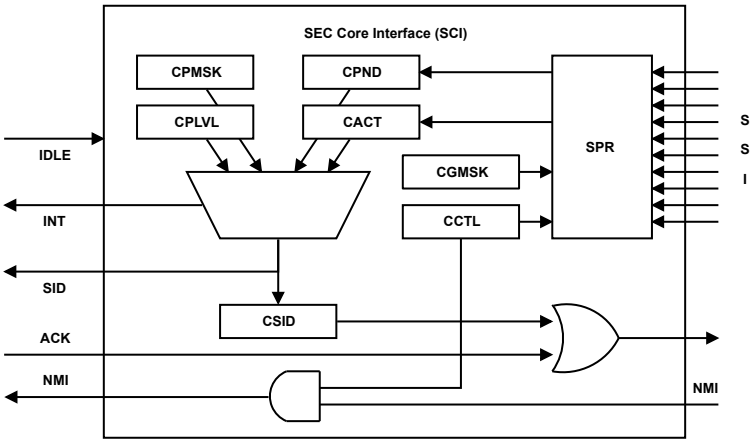

Figure 8-3: SCI Overview Block Diagram

SSI Block Diagram

The SSI manages all of the system event sources. It maintains the status of each source in the corresponding `SEC_SSTAT[n]` register. The corresponding `SEC_SCTL[n]` register manages the control of each source. A pending and enabled event passes its indication and priority to the SCI to which it is assigned for further processing.

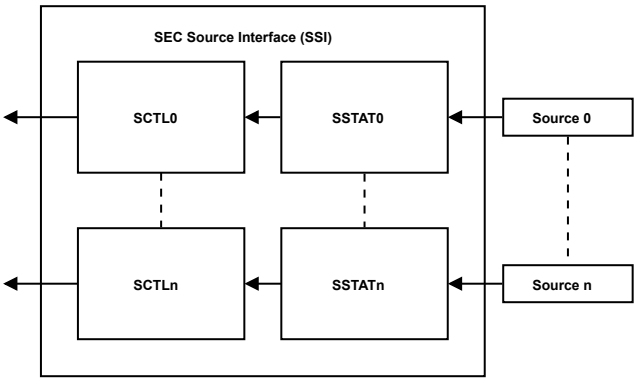

Figure 8-4: SSI Overview Block Diagram

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:

- A slave port write to the `SEC_CSID[n]` register.
- The assertion of an input acknowledge signal (the connected core generates the signal).

System Interrupt Groups

System sources can be assigned to groups using the `SEC_SCTL[n].GRP` bit field. Source groups allow fast context switching for system interrupts at each SCI. The `SEC_CGMSK[n]` register allows quick masking of interrupt groups of unlimited size with a single write operation.

System Interrupt Flow

An enabled and asserted system interrupt source is latched at the SSI and routed to the appropriate SCI based on the core target select (`SEC_SCTL[n].CTG`) bit field setting. The SEC priority ordering determines the highest priority pending system interrupt and the SCI updates the `SEC_CPND[n].SID` and `SEC_CACT[n].PRIO` bit field values. The SCI compares the `SEC_CPND[n]` register value against the highest priority active source in the `SEC_CACT[n]` register).

The priority level register (`SEC_CPLVL[n]`) determines how many of the MSBs the SEC uses in the comparison. The priority mask register (`SEC_CPMSK[n]`) and the group mask register (`SEC_CGMSK[n]`) determines which pending interrupt sources participate. If the `SEC_CPND[n]` register value is a higher priority (lower value) than the priority of the `SEC_CACT[n]` register from the comparison based on the `SEC_CPLVL[n]` register, the system interrupt output is asserted. The source ID register (`SEC_CSID[n]`) is updated with the `SEC_CPND[n].SID` bit field value and forwarded to the connected core.

After the core provides an interrupt acknowledgment, the interrupt source is active, until the SEC completes interrupt service with a write to the SEC_END.SID bit field with the same value. Note the following:

- Interrupt acknowledgement occurs with an MMR write of the SEC_CSID[n] register or the core version of the SEC_CSID[n] register.
- Interrupt active status indication is SEC_SSTAT[n].ACT==1.

The following sequence shows the example flow for a single interrupt.

1. The SEC compares the SEC_CPND[n] register value to the SEC_CACT[n] register value. If the interrupt in the SEC_CPND[n] register is higher priority, continue.
2. The SEC copies the SEC_CPND[n] register value to the SEC_CSID[n] register and asserts the interrupt signal.
3. The core reads the SEC_CSID[n] register (or core version).
4. The core writes to the SEC_CSID[n] register (or core version, asserts the acknowledge signal).
5. The SEC deasserts the interrupt signal and clears the SEC_SSTAT[n].PND bit and sets the SEC_SSTAT[n].ACT bit of the source going active.
6. The core writes the SEC_CSID[n] of the active interrupt to the SEC_END register.
7. The SEC clears theSEC_SSTAT[n].ACT bit of the source being ended.

The following sequence shows the example flow for interrupt nesting where interrupt A is a lower priority and occurs earlier than interrupt B.

1. The SEC compares the SEC_CPND[n] (A) register value to the SEC_CACT[n] register and if the interrupt in the SEC_CPND[n] register is a higher priority, continue.
2. The SEC copies SEC_CPND[n] (A) register to the SEC_CSID[n] register and asserts the interrupt signal.
3. The core reads the SEC_CSID[n] (A) register (or core version).
4. The core writes to the SEC_CSID[n] register (or core version, asserts the acknowledge signal).
5. The SEC deasserts the INT signal and clears the SEC_SSTAT[n].PND bit and sets the SEC_SSTAT[n].ACT bit of the source (A) going active.
6. The SEC compares the SEC_CPND[n] (B) register value to the SEC_CACT[n] (A) register value. If the SEC_CACT[n] (A) register value is a higher priority, continue.
7. The SEC copies the SEC_CPND[n] (B) register value to SEC_CSID[n] register and asserts the interrupt signal.
8. The core reads the SEC_CSID[n] (B) register (or core version).
9. The core writes to the SEC_CSID[n] register (or core version, asserts the acknowledge signal).

10. The SEC deasserts the INT signal and clears the SEC_SSTAT[n].PND bit and sets the SEC_SSTAT[n].ACT bit of the source (B) going active.
11. The core writes the SEC_CSID[n] of the active interrupt (B) to the SEC_END register.
12. The SEC clears the SEC_SSTAT[n].ACT bit of the source (B) being ended.
13. The core writes the SEC_CSID[n] of the active interrupt (A) to the SEC_END register.
14. The SEC clears the SEC_SSTAT[n].ACT bit of the source (A) being ended.

System Interrupt Priorities

Each system interrupt source has its own programmable priority level which is configured using the SEC_SCTL[n].PRIO bit field. The SCI evaluates the priority of all pending sources to determine the source of the highest-priority pending system interrupt for forwarding to the attached core. If more than one source of the pending system interrupt has the same priority setting, the SCI chooses the one with the lowest SID. For example, if SID 0, SID 1, and SID 2 are all pending and have the same priority setting, the SCI chooses SID 0 as the highest-priority source.

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.

SEC Programming Model

Implementing a system interrupt service model using the SEC requires, at a minimum:

- Proper configuration of a system interrupt source (for example a peripheral or DMA)
- A core interrupt or event service model

The core must be configured for response to system interrupts from the SEC. The SEC must be configured to enable and map the system interrupt source to the correct SCI and to forward interrupts to the connected core.

The system interrupt source must be configured to generate interrupt assertions. Alternatively, the processor can use software triggering for interrupt assertion. Software driven interrupts are generated by writing the source ID of the interrupt to be triggered to the SEC_RAISE register.

Programming Concepts

The following list provides the basic programming concepts necessary for configuring the SEC.

- Configuring an SSI as a system interrupt for a specific core.
- Configuring an SCI to provide system interrupts to the connected core. (See Configuring a System Source to Interrupt a Core.)
- Configuring an SSI as a system fault. (See Configuring a System Source as a Fault.)

- Configuring the SFI to manage system faults.

Programming Examples

This section provides example programming tasks that are typical for SEC usage.

Configuring a System Source to Interrupt a Core

1. Write to the SEC_GCTL register to enable the SEC.
2. Write to the SEC_CCTL[n] register of the specific SCI to enable interrupts to be sent to core.
3. Write to the SEC_SCTL[n] register of the specific source to enable the source as an interrupt and to set the core target field to map the source to the appropriate SCI.

Configuring a System Source as a Fault

1. Write to the SEC_GCTL register to enable the SEC.
2. Write to the SEC_FCTL register to configure specific fault actions.
3. Write to the SEC_FDLY bit field to specify fault delay.
4. Write to the control register of a specific source to enable the source as a fault.

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.

ADSP-BF70x SEC Register Descriptions

System Event Controller (SEC) contains the following registers.

Table 8-3: ADSP-BF70x SEC Register List

Name	Description
SEC_CACT[n]	SCI Active Register n
SEC_CCTL[n]	SCI Control Register n
SEC_CGMSK[n]	SCI Group Mask Register n
SEC_CPLVL[n]	SCI Priority Level Register n
SEC_CPMSK[n]	SCI Priority Mask Register n
SEC_CPND[n]	Core Pending Register n
SEC_CSID[n]	SCI Source ID Register n
SEC_CSTAT[n]	SCI Status Register n
SEC_END	Global End Register
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

SCI Active Register n

The SEC SCI active interrupt register (SEC_CACT[n]) contains the source ID and priority of the highest priority active interrupt detected by the SEC prioritizer.

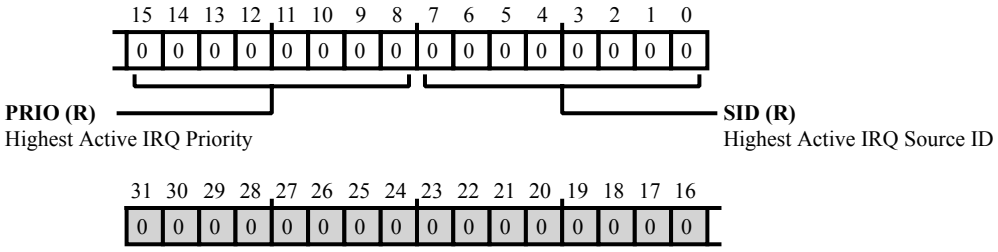

Figure 8-5: SEC_CACT[n] Register Diagram

Table 8-4: SEC_CACT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	PRIO	Highest Active IRQ Priority. The SEC_CACT[n].PRIO indicates the priority value of the highest priority active interrupt for core n.
7:0 (R/NW)	SID	Highest Active IRQ Source ID. The SEC_CACT[n].SID identifies the source ID value of the highest priority active interrupt for core n.

SCI Control Register n

The SEC control register (SEC_CCTL[n]) contains SCI control bits for all system sources.

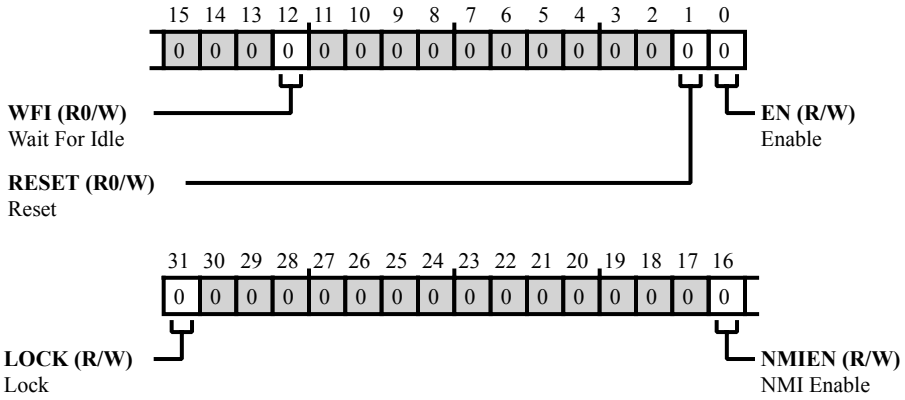

Figure 8-6: SEC_CCTL[n] Register Diagram

Table 8-5: SEC_CCTL[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_CCTL[n].LOCK bit is enabled, the SEC_CCTL[n] register is read only.	
		0	Unlock
		1	Lock
16 (R/W)	NMIEN	NMI Enable. The SEC_CCTL[n].NMIEN bit controls NMI propagation to the core. When the SEC_CCTL[n].NMIEN bit is enabled, the SCI allows NMIs to propagate to the core for servicing.	
		0	Disable
		1	Enable
12 (R0/W)	WFI	Wait For Idle. When set, the SEC_CCTL[n].WFI bit forces the SCI to wait for indication of core idle before the SCI resumes activity.	
		0	No Action
		1	Wait for Idle

Table 8-5: SEC_CCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R0/W)	RESET	Reset. When set, the `SEC_CCTL[n].RESET` bit resets all SCI registers to their default values.	
		0	No Action
		1	Reset
0 (R/W)	EN	Enable. The `SEC_CCTL[n].EN` bit controls operation of the SCI. Clearing the `SEC_CCTL[n].EN` bit halts the execution of the SCI without resetting status registers. (The INT signal to a core is not affected.) Setting the `SEC_CCTL[n].EN` bit enables the SCI to begin or resume operation with the current configuration and status.	
		0	Disable
		1	Enable

SCI Group Mask Register n

The SEC SCI group mask register (SEC_CGMSK[n]) contains selections for a group mask, an ungroup mask, and a register lock. This register contains the system interrupt group masks for the connected core. The core uses the SEC_CGMSK[n].UGRP and SEC_CGMSK[n].GRP fields to mask (disable) interrupts from the specified groups.

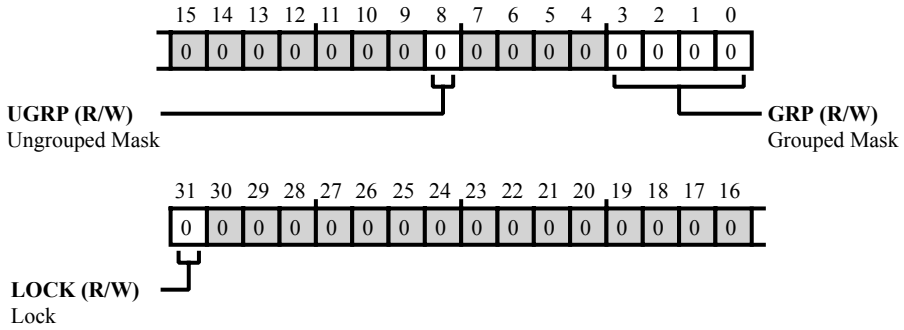

Figure 8-7: SEC_CGMSK[n] Register Diagram

Table 8-6: SEC_CGMSK[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_CGMSK[n].LOCK bit is enabled, the SEC_CGMSK[n] register is read only.	
		0	Unlock
		1	Lock
8 (R/W)	UGRP	Ungrouped Mask. The SEC_CGMSK[n].UGRP bit masks interrupts (if set) for the ungrouped interrupt sources for core n.	
		0	Unmask Ungrouped Sources
		1	Mask Ungrouped Sources
3:0 (R/W)	GRP	Grouped Mask. The SEC_CGMSK[n].GRP field selects a group of interrupt sources to mask for core n. (For more information about interrupt source groups, see the SEC_SCTL[n] register description.)	
		0	No groups masked
		1	Mask group 0
		2	Mask group 1
		3	Mask groups 0, 1
		4	Mask group 2

Table 8-6: SEC_CGMSK[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	Mask groups 0, 2
		6	Mask groups 1, 2
		7	Mask groups 0, 1, 2
		8	Mask group 3
		9	Mask groups 0, 3
		10	Mask groups 1, 3
		11	Mask groups 0, 1, 3
		12	Mask groups 2, 3
		13	Mask groups 0, 2, 3
		14	Mask groups 1, 2, 3
		15	Mask groups 0, 1, 2, 3

SCI Priority Level Register n

The SEC SCI priority level register (SEC_CPLVL[n]) contains selections for priority levels and a register lock. This register is used to divide the total number of priority levels into sub-levels. The sub-level priority resolution provides the tie breaker for simultaneously pending interrupts assigned to the same level.

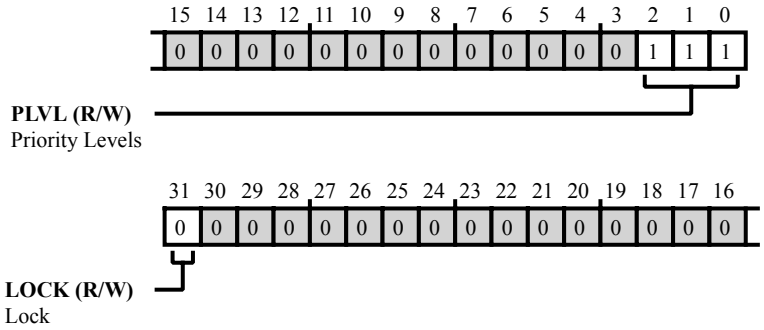

Figure 8-8: SEC_CPLVL[n] Register Diagram

Table 8-7: SEC_CPLVL[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_CPLVL[n].LOCK bit is enabled, the SEC_CPLVL[n] register is read only.	
		0	Unlock
		1	Lock
2:0 (R/W)	PLVL	Priority Levels. The SEC_CPLVL[n].PLVL field serves to divide the total number of interrupt priority levels into sub-levels. The sub-level priority resolution provides the tie breaker for simultaneously pending interrupts assigned to the same interrupt level. The sub-level priority value specifies the number of MSBs (minus 1) designated to interrupt levels, while the remaining LSBs are designated for sub-level specification. For example, if the SEC_CPLVL[n].PLVL field is set to two, the result is four priority levels are specified, because only the two MSBs are used for preemption evaluation. The remaining bits of the priority setting are used for sub-level prioritization.	
		0	1 MSBs (2 priority levels)
		1	2 MSBs (4 priority levels)
		2	3 MSBs (8 priority levels)
		3	4 MSBs (16 priority levels)
		4	5 MSBs (32 priority levels)
		5	6 MSBs (64 priority levels)

Table 8-7: SEC_CPLVL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		6	7 MSBs (128 priority levels)
		7	8 MSBs (256 priority levels)

SCI Priority Mask Register n

The SEC SCI priority mask register (SEC_CPMSK[n]) contains the SCI priority mask for core n and includes a register lock.

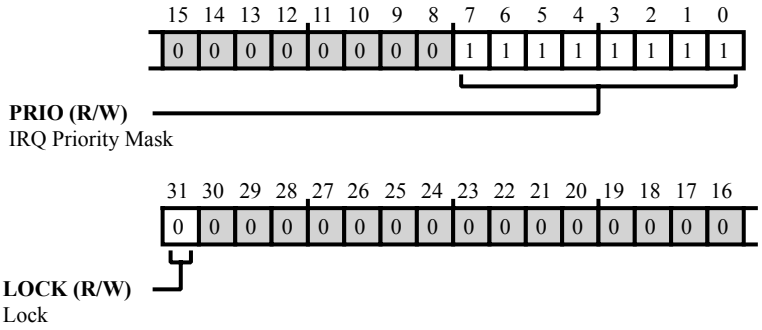

Figure 8-9: SEC_CPMSK[n] Register Diagram

Table 8-8: SEC_CPMSK[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_CPMSK[n].LOCK bit is enabled, the SEC_CPMSK[n] register is read only.	
		0	Unlock
		1	Lock
7:0 (R/W)	PRIO	IRQ Priority Mask. The SEC_CPMSK[n].PRIO contains the system interrupt priority mask for core n. The core uses the SEC_CPMSK[n].PRIO field to mask (block) interrupts below the specified level.	
		0	Priority level 0 (highest)
		1-254	
		255	Priority level 255 (lowest)

Core Pending Register n

The SCI pending interrupt register (SEC_CPND[n]) contains the source ID and priority of the highest priority pending interrupt detected by the SEC prioritizer.

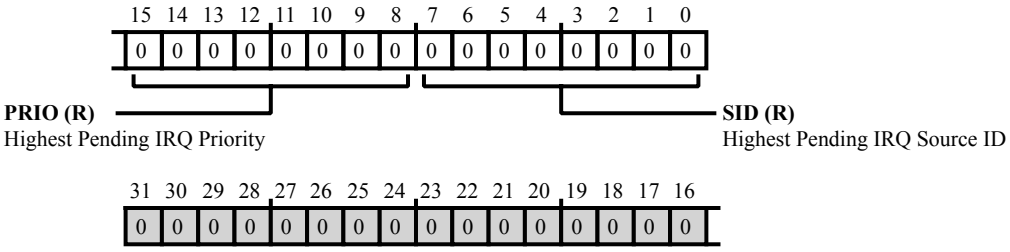

Figure 8-10: SEC_CPND[n] Register Diagram

Table 8-9: SEC_CPND[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/NW)	PRIO	Highest Pending IRQ Priority. The SEC_CPND[n].PRIO indicates the priority value of the highest priority pending interrupt for core n.
7:0 (R/NW)	SID	Highest Pending IRQ Source ID. The SEC_CPND[n].SID identifies the source ID value of the highest priority pending interrupt for core n.

SCI Source ID Register n

The SCI source ID register (SEC_CSID[n]) contains the source ID of the interrupt last issued to core n. The SEC_CSID[n] register value is loaded by the SCI when a system interrupt indication is sent to core n. The SCI does not change the SEC_CSID[n] until after the interface receives an interrupt acknowledge from core n. Writing to the SEC_CSID[n] register generates an interrupt acknowledge, but does not update the value in the register.

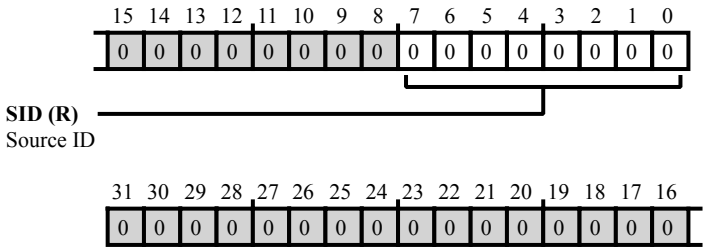

Figure 8-11: SEC_CSID[n] Register Diagram

Table 8-10: SEC_CSID[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	SID	Source ID. The SEC_CSID[n].SID bit is the source ID of the interrupt last issued to core n.

SCI Status Register n

The SCI status register (SEC_CSTAT[n]) contains status bits, indicating the operational status of the SCI.

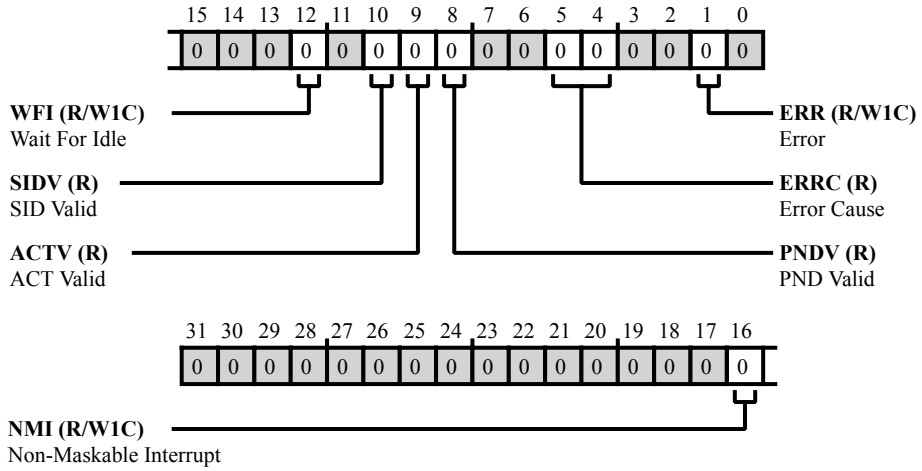

Figure 8-12: SEC_CSTAT[n] Register Diagram

Table 8-11: SEC_CSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	NMI	Non-Maskable Interrupt. The SEC_CSTAT[n].NMI bit indicates whether an NMI has occurred since the bit was last cleared.	
		0	No NMI Occurred
		1	NMI Occurred
12 (R/W1C)	WFI	Wait For Idle. The SEC_CSTAT[n].WFI bit indicates (if set) that the SCI is temporarily disabled, pending a core idle indication. This bit is set when SEC_CCTL[n].WFI is set.	
		0	Not Waiting
		1	Waiting
10 (R/NW)	SIDV	SID Valid. The SEC_CSTAT[n].SIDV bit indicates (if set) that the current value in the SEC_CSID[n] register is valid. The SCI sets the SEC_CSTAT[n].SIDV bit when the updating the SEC_CSID[n] register with a new value. The SEC_CSTAT[n].SIDV bit is cleared when the SEC_CSID[n] register is written. This status indication may be used to extract all pending interrupts in a single interrupt service routine.	
		0	Invalid
		1	Valid

Table 8-11: SEC_CSTAT[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/NW)	ACTV	ACT Valid. The SEC_CSTAT[n].ACTV bit indicates (if set) that the current value in the SEC_CACT[n] register is valid. The SCI sets the SEC_CSTAT[n].ACTV bit when updating the SEC_CACT[n] registers with a new value. The SEC_CSTAT[n].ACTV bit is cleared when the SEC_CSID[n] register is written.	
		0	Invalid
		1	Valid
8 (R/NW)	PNDV	PND Valid. The SEC_CSTAT[n].PNDV bit indicates (if set) that the current value in the SEC_CPND[n] register is valid. The SCI sets the SEC_CSTAT[n].PNDV bit when updating the SEC_CPND[n] register with a new value. The SEC_CSTAT[n].PNDV bit is cleared when the SEC_CSID[n] register is written.	
		0	Invalid
		1	Valid
5:4 (R/NW)	ERRC	Error Cause. The SEC_CSTAT[n].ERRC bits are updated on assertion of the SEC_CSTAT[n].ERR bit to indicate the SCI error type. SEC_CSTAT[n].ERRC is only updated on the assertion of SEC_CSTAT[n].ERR. Subsequent errors while SEC_CSTAT[n].ERR is asserted do not update SEC_CSTAT[n].ERRC.	
		0	Reserved
		1	Acknowledge Error. SCI has received an acknowledge without a pending, unacknowledged interrupt present.
		2	Reserved
		3	Reserved
1 (R/W1C)	ERR	Error. The SEC_CSTAT[n].ERR bit indicates that an error has occurred in the SCI. When SEC_CSTAT[n].ERR is set, the SCI updates the SEC_CSTAT[n].ERRC field to the value of the corresponding error cause.	
		0	No Error
		1	Error Occurred

Global End Register

The SEC global end register (SEC_END) contains a source ID interrupt service end field (SEC_END.SID). When a core has finished servicing an interrupt, the core writes the SEC_END.SID field in the SEC_END register. This write causes the SEC to clear the SEC_SSTAT[n].ACT bit in the SEC_SSTAT[n] register of the corresponding interrupt.

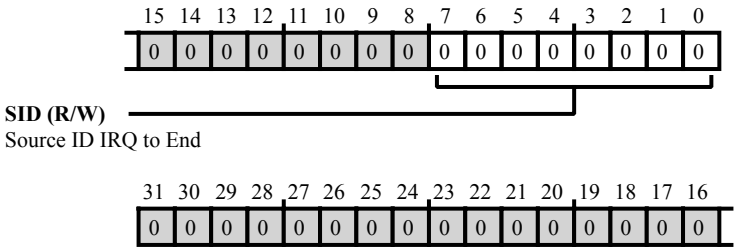

Figure 8-13: SEC_END Register Diagram

Table 8-12: SEC_END Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	SID	Source ID IRQ to End. The SEC_END.SID bit field contains the source ID interrupt service end value.

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.

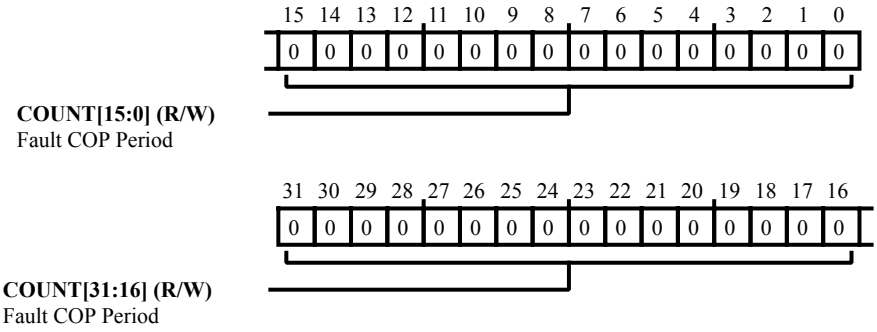

Figure 8-14: SEC_FCOPP Register Diagram

Table 8-13: 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.

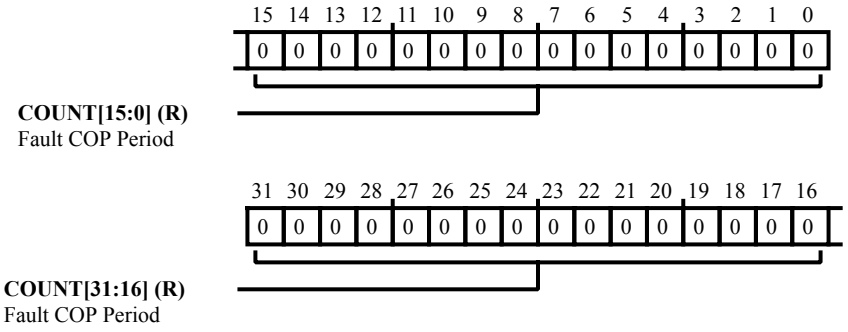

Figure 8-15: SEC_FCOPP_CUR Register Diagram

Table 8-14: 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).

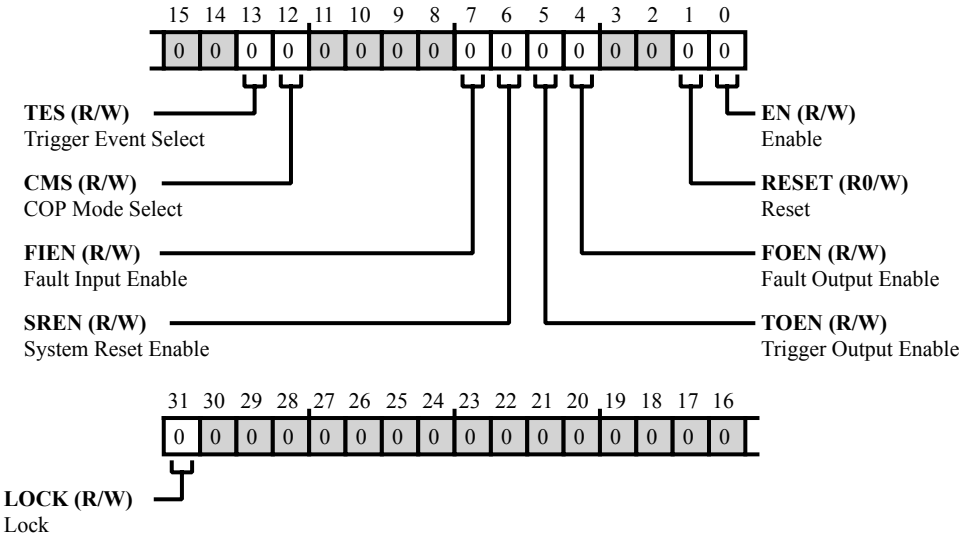

Figure 8-16: SEC_FCTL Register Diagram

Table 8-15: 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 8-15: 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 8-15: 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.

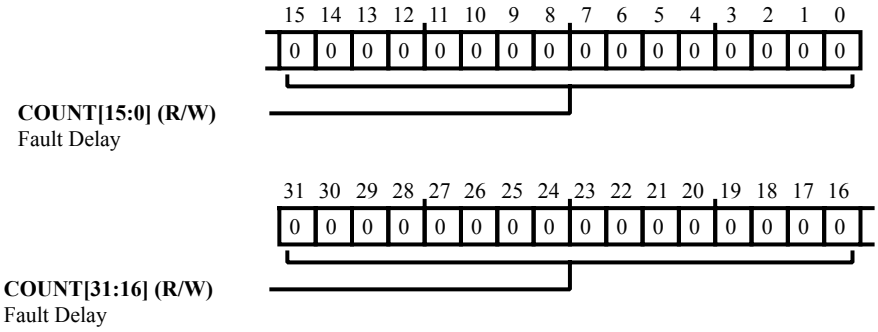

Figure 8-17: SEC_FDLY Register Diagram

Table 8-16: 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.

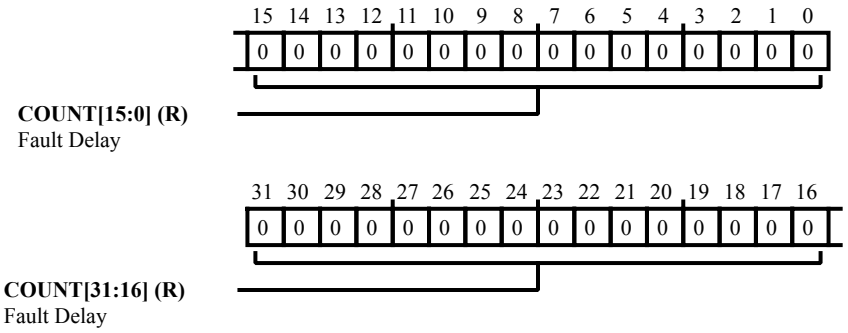

Figure 8-18: SEC_FDLY_CUR Register Diagram

Table 8-17: 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.

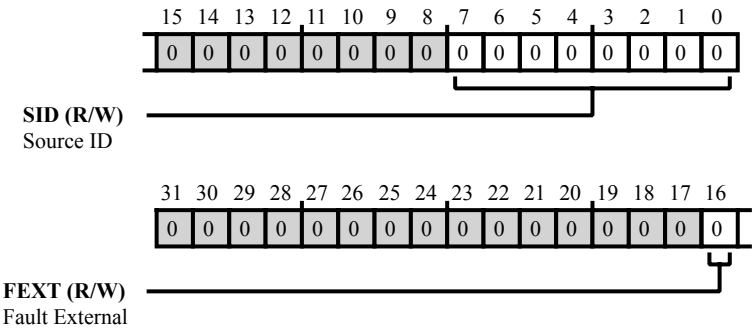

Figure 8-19: SEC_FEND Register Diagram

Table 8-18: 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.

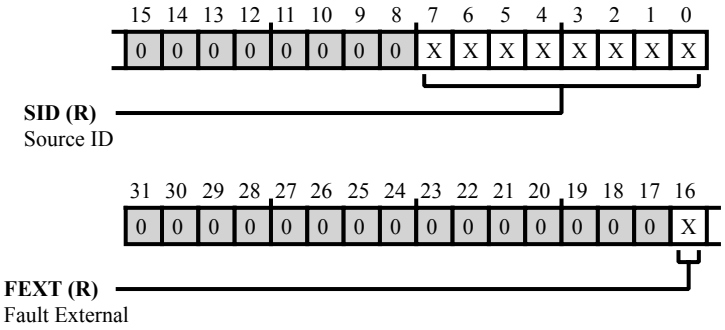

Figure 8-20: SEC_FSID Register Diagram

Table 8-19: 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.

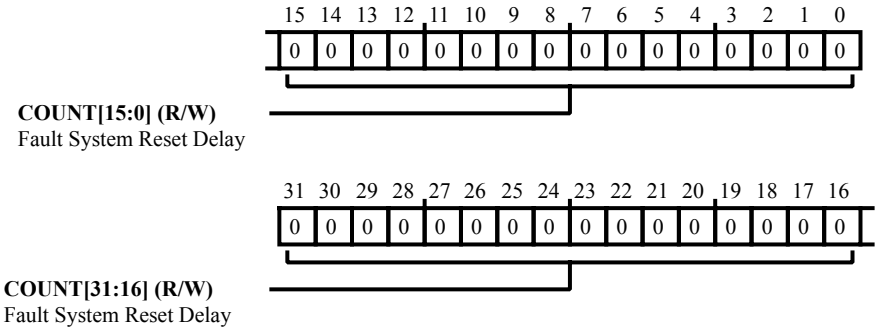

Figure 8-21: SEC_FSRDLY Register Diagram

Table 8-20: 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.

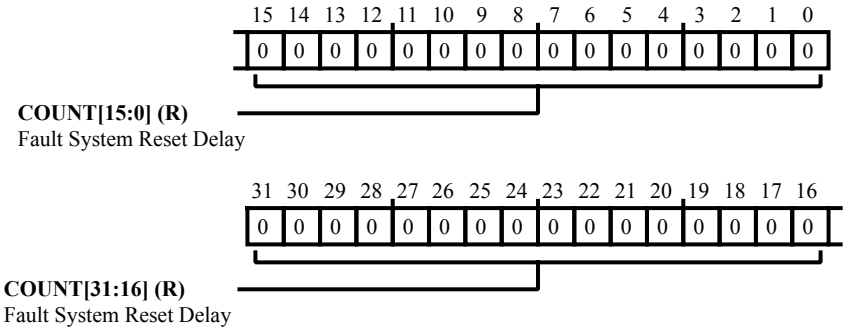

Figure 8-22: SEC_FSRDLY_CUR Register Diagram

Table 8-21: 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.

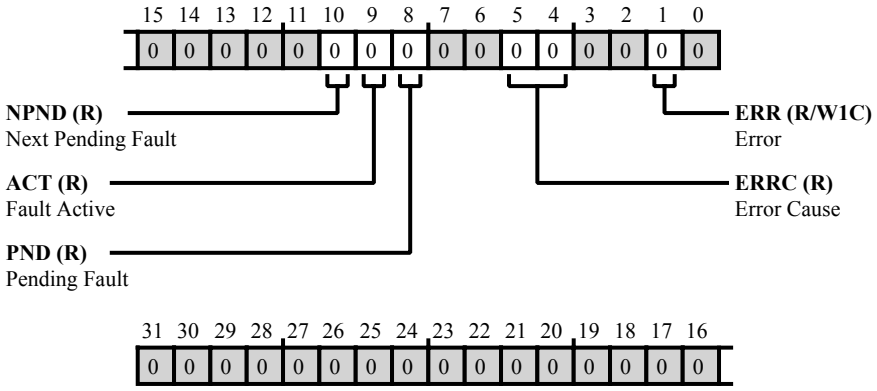

Figure 8-23: SEC_FSTAT Register Diagram

Table 8-22: 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 8-22: 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. When the error type is end error, the status indicates that an end was received for a source while the SEC_FSTAT.ACT bit was not set.	
		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.

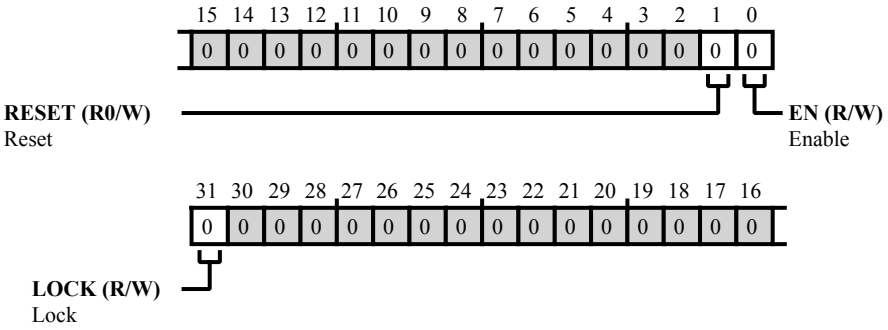

Figure 8-24: SEC_GCTL Register Diagram

Table 8-23: 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 and all SCIs. 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.

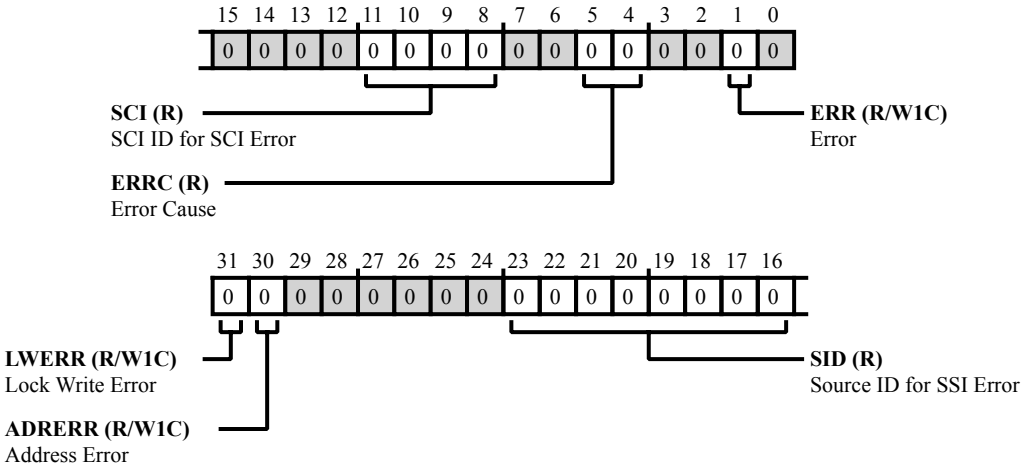

Figure 8-25: SEC_GSTAT Register Diagram

Table 8-24: 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
23:16 (R/NW)	SID	Source ID for SSI Error. The SEC_GSTAT.SID bits indicate the source ID that generated the last SSI error conveyed in the SEC_GSTAT.ERRC field.	
11:8 (R/NW)	SCI	SCI ID for SCI Error. The SEC_GSTAT.SCI bits indicate the number for the specific SCI that generated the last SCI error conveyed in the SEC_GSTAT.ERRC field.	

Table 8-24: SEC_GSTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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 SCI errors, the error status represents an OR of all the errors from each SCI. 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	SCI Error
		2	SSI Error
		3	Reserved
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.

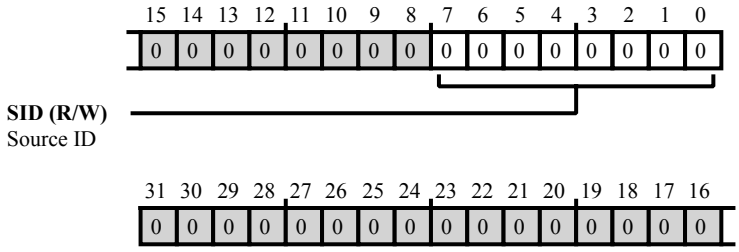

Figure 8-26: SEC_RAISE Register Diagram

Table 8-25: 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.

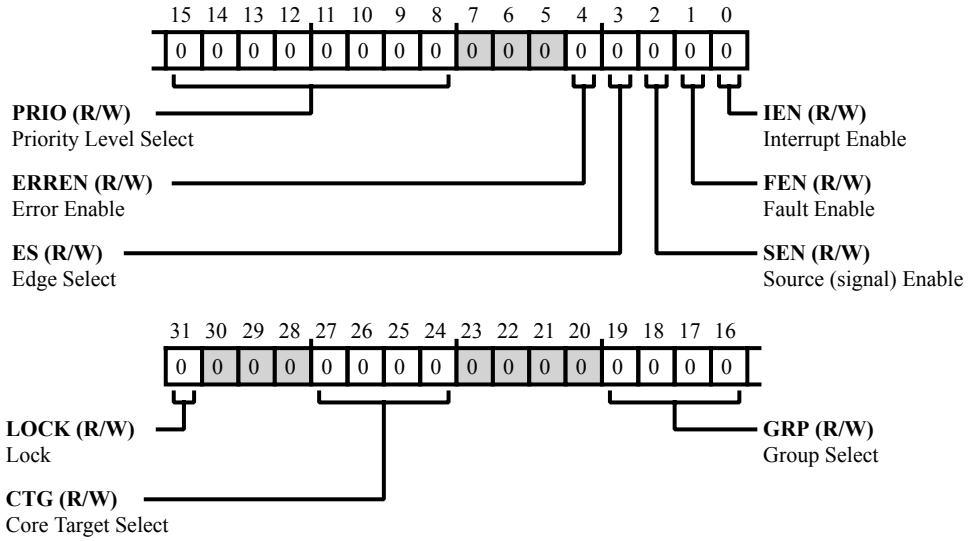

Figure 8-27: SEC_SCTL[n] Register Diagram

Table 8-26: 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
27:24 (R/W)	CTG	Core Target Select. The SEC_SCTL[n].CTG bits selects the specific SEC core interface to which the interrupt is mapped. Each system interrupt is mapped uniquely to one specific SEC core interface and (as a result) to a specific core.	
19:16 (R/W)	GRP	Group Select. The SEC_SCTL[n].GRP bits each select a specific group for the interrupt. Each system interrupt can be assigned to any combination of groups supported by the SEC_SCTL[n].GRP field. For example, consider the situation where SEC_SCTL[n].GRP0 represents interrupt group 0, SEC_SCTL[n].GRP1 represents interrupt group 1, and so on. One group might be used for all enabled interrupts (for example, group 0) and an additional	

Table 8-26: SEC_SCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		group might be used for all wakeup interrupts (for example, group 1). This approach supports a model of all interrupts and just the wakeup subset. Before going to idle or sleep, all non-wakeup interrupts can be masked off to allow only wakeup interrupts to be enabled for service. Selecting no group (all SEC_SCTL[n].GRP bits = 0) places the interrupt source in the category of "ungrouped".	
15:8 (R/W)	PRIO	Priority Level Select. The SEC_SCTL[n].PRIO bits sets the relative priority for an interrupt request. A pending interrupt request forwards its SEC_SCTL[n].PRIO value to the SEC core interface.	
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
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

Table 8-26: SEC_SCTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	IEN	Interrupt Enable. The `SEC_SCTL[n].IEN` bit controls whether the SEC may forward an interrupt request to a core for servicing. This bit does not affect the ability of an interrupt source to set an interrupt as pending.	
		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.

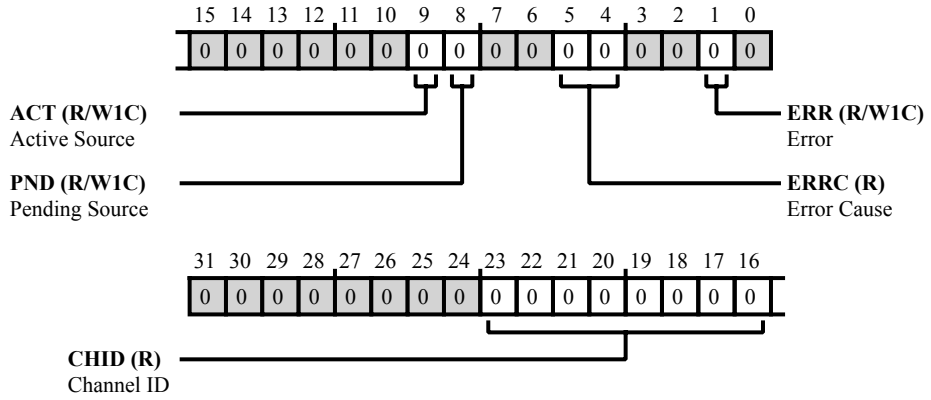

Figure 8-28: SEC_SSTAT[n] Register Diagram

Table 8-27: SEC_SSTAT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:16 (R/NW)	CHID	Channel ID. The SEC_SSTAT[n].CHID bits indicate the ID of the specific source (from a set of sources sharing one SEC source interface input) that asserted the SEC source interface input. An SEC source interface input may support multiple system sources, in which case the assertion must be qualified by an identifier to determine the channel that generated the assertion. The SEC_SSTAT[n].CHID field provides this value in the form of a numeric reference that is mapped to a specific interrupt source. The prioritization for simultaneously asserted sources is according to ID, with 0 being the highest priority. The SEC_SSTAT[n].CHID is captured when the SEC source interface input is acknowledged.	
9 (R/W1C)	ACT	Active Source. The SEC_SSTAT[n].ACT bit indicates the source has been accepted by a core for servicing, but the service is not yet complete. An SEC_SSTAT[n].ACT bit is set by the SEC when the specific system interrupt is acknowledged by the core through the SEC core interface. An SEC_SSTAT[n].ACT bit is cleared by the SEC when the core provides interrupt service end indication for the specific system interrupt through the SEC core interface.	
		0	Not Active
		1	Active

Table 8-27: SEC_SSTAT[n] Register Fields (Continued)

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 the SEC when the specific system event is acknowledged by the core through the SEC core interface or 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. When the error type is end error, the status indicates that an end was received for a source while the SEC_SSTAT[n].ACT bit was not set.	
		0	Source Overflow Error
		1	Reserved
		2	End Error
		3	Reserved
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

9 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.

- Trigger routing of any trigger master to any trigger slave
- Software generation of any trigger master ID
- Configuration protection through register-level lock bits and global lock indication

TRU Functional Description

The following sections provide a description of the TRU.

ADSP-BF70x 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 9-1: ADSP-BF70x 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

ADSP-BF70x TRU Interrupt List

Table 9-2: ADSP-BF70x TRU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
79	TRU0_SLV0	TRU0 Interrupt 0	Edge	
80	TRU0_SLV1	TRU0 Interrupt 1	Edge	
81	TRU0_SLV2	TRU0 Interrupt 2	Edge	
82	TRU0_SLV3	TRU0 Interrupt 3	Edge	

ADSP-BF70x Trigger List

Table 9-3: ADSP-BF70x Trigger List Masters

Trigger ID	Name	Description	Sensitivity
0		Reserved	
1	CGU0_EVT	CGU0 Event	Edge
2	TIMER0_TMR0_MST	TIMER0 Timer 0	Edge
3	TIMER0_TMR1_MST	TIMER0 Timer 1	Edge
4	TIMER0_TMR2_MST	TIMER0 Timer 2	Edge
5	TIMER0_TMR3_MST	TIMER0 Timer 3	Edge
6	TIMER0_TMR4_MST	TIMER0 Timer 4	Edge
7	TIMER0_TMR5_MST	TIMER0 Timer 5	Edge
8	TIMER0_TMR6_MST	TIMER0 Timer 6	Edge
9	TIMER0_TMR7_MST	TIMER0 Timer 7	Edge
10	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level
11	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level
12	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level

Table 9-3: ADSP-BF70x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
13	CNT0_STAT	CNT0 Status	Level
14	SPORT0_A_DMA	SPORT0 Channel A DMA	Edge
15	SPORT0_B_DMA	SPORT0 Channel B DMA	Edge
16	SPORT1_A_DMA	SPORT1 Channel A DMA	Edge
17	SPORT1_B_DMA	SPORT1 Channel B DMA	Edge
18	SPI0_TXDMA	SPI0 TX DMA Channel	Edge
19	SPI0_RXDMA	SPI0 RX DMA Channel	Edge
20	SPI1_TXDMA	SPI1 TX DMA Channel	Edge
21	SPI1_RXDMA	SPI1 RX DMA Channel	Edge
22	SPI2_TXDMA	SPI2 TX Channel	Edge
23	SPI2_RXDMA	SPI2 RX Channel	Edge
24	UART0_TXDMA	UART0 Transmit DMA	Edge
25	UART0_RXDMA	UART0 Receive DMA	Edge
26	UART1_TXDMA	UART1 Transmit DMA	Edge
27	UART1_RXDMA	UART1 Receive DMA	Edge
28	SYS_MDMA0_SRC	Memory DMA Stream 0 Source Channel	None
29	SYS_MDMA0_DST	Memory DMA Stream0 Destination Channel	None
30	SYS_MDMA1_SRC	Memory DMA Stream 1 Source/CRC0 Input Channel	None
31	SYS_MDMA1_DST	Memory DMA Stream 1 Destination/CRC0 Output Channel	None
32	SYS_MDMA2_SRC	Memory DMA Stream 2 Source/CRC1 Input Channel	None
33	SYS_MDMA2_DST	Memory DMA Stream 2 Destination/CRC1 Output Channel	None
34	EPPI0_CH0_DMA	EPPI0 Channel 0 DMA	Edge
35	EPPI0_CH1_DMA	EPPI0 Channel 1 DMA	Edge
36	USB0_DATA	USB0 DMA Status/Transfer Complete	Level
37	SEC0_FAULT	SEC0 Fault	Edge
38	SYS_SOFT0_MST	Software-driven Trigger 0	Edge
39	SYS_SOFT1_MST	Software-driven Trigger 1	Edge
40	SYS_SOFT2_MST	Software-driven Trigger 2	Edge
41	SYS_SOFT3_MST	Software-driven Trigger 3	Edge

Table 9-3: ADSP-BF70x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
42	SYS_SOFT4_MST	Software-driven Trigger 4	Edge
43	SYS_SOFT5_MST	Software-driven Trigger 5	Edge
44	HADC0_EOC	HADC0 End of Conversion Trigger	Edge
45	RTC0_EVT	RTC0 Event	Level
46	L2CTL0_EVT	L2CTL0 L2 Event	Level
47	MSI0_DONE	MSI0 Transfer Done	Level
48	SPIHP0_EVT	SPIHP0 SPI Host Port Ready	Level
49	SYS_C0_SI_DIS_ACK	Core 0 System Interface Disable Acknowledge	Level
50	SWU0_EVT	SWU0 Event (L1)	None
51	SWU1_EVT	SWU1 Event (Core L2)	None
52	SWU2_EVT	SWU2 Event (DMA L2)	None
53	SWU3_EVT	SWU3 Event (MMR)	None
54	SWU4_EVT	SWU4 Event (DMC)	None
55	SWU5_EVT	SWU5 Event (SMC)	None
56	SWU6_EVT	SWU6 Event (SPIF)	None
57	SWU7_EVT	SWU7 Event (OTP)	None
58	SWU0_DBG	SWU0 Debug (L1)	Edge
59	SWU1_DBG	SWU1 Debug (Core L2)	Edge
60	SWU2_DBG	SWU2 Debug (DMA L2)	Edge
61	SWU3_DBG	SWU3 Debug (MMR)	Edge
62	SWU4_DBG	SWU4 Debug (DMC)	Edge
63	SWU5_DBG	SWU5 Debug (SMC)	Edge
64	SWU6_DBG	SWU6 Debug (SPIF)	Edge
65	SWU7_DBG	SWU7 Debug (OTP)	Edge
66	CTI2_MST0	CTI2 SYSCTI System Halt 0	Edge
67	CTI2_MST1	CTI2 SYSCTI System Halt 1	Edge
68	CTI2_MST2	CTI2 SYSCTI System Halt 2	Edge
69	CTI2_MST3	CTI2 SYSCTI System Halt 3	Edge
70	CTI2_MST4	CTI2 SYSCTI System Halt 4	Edge
71	CTI2_MST5	CTI2 SYSCTI System Halt 5	Edge
72	CTI2_MST6	CTI2 SYSCTI System Halt 6	Edge

Table 9-3: ADSP-BF70x Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
73	CTI2_MST7	CTI2 SYSCTI System Halt 7	Edge

Table 9-4: ADSP-BF70x 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	TIMER0_TMR0_SLV0	TIMER0 Timer 0 Slave 0	Pulse
3	TIMER0_TMR1_SLV0	TIMER0 Timer 1 Slave 0	Pulse
4	TIMER0_TMR2_SLV0	TIMER0 Timer 2 Slave 0	Pulse
5	TIMER0_TMR3_SLV0	TIMER0 Timer 3 Slave 0	Pulse
6	TIMER0_TMR4_SLV0	TIMER0 Timer 4 Slave 0	Pulse
7	TIMER0_TMR5_SLV0	TIMER0 Timer 5 Slave 0	Pulse
8	TIMER0_TMR6_SLV0	TIMER0 Timer 6 Slave 0	Pulse
9	TIMER0_TMR7_SLV0	TIMER0 Timer 7 Slave 0	Pulse
10	TIMER0_TMR0_SLV1	TIMER0 Timer 0 Slave 1	Pulse
11	TIMER0_TMR1_SLV1	TIMER0 Timer 1 Slave 1	Pulse
12	TIMER0_TMR2_SLV1	TIMER0 Timer 2 Slave 1	Pulse
13	TIMER0_TMR3_SLV1	TIMER0 Timer 3 Slave 1	Pulse
14	TIMER0_TMR4_SLV1	TIMER0 Timer 4 Slave 1	Pulse
15	TIMER0_TMR5_SLV1	TIMER0 Timer 5 Slave 1	Pulse
16	TIMER0_TMR6_SLV1	TIMER0 Timer 6 Slave 1	Pulse
17	TIMER0_TMR7_SLV1	TIMER0 Timer 7 Slave 1	Pulse
18	SYS_C0_NMI_S0	NMI (Core 0) Slave 0	Pulse
19	SYS_C0_NMI_S1	NMI (Core 0) Slave 1	Pulse
20	TRU0_SLV0	TRU0 Interrupt Request 0	Pulse
21	TRU0_SLV1	TRU0 Interrupt Request 1	Pulse
22	TRU0_SLV2	TRU0 Interrupt Request 2	Pulse
23	TRU0_SLV3	TRU0 Interrupt Request 3	Pulse
24	SPORT0_A_DMA	SPORT0 Channel A DMA	Pulse
25	SPORT0_B_DMA	SPORT0 Channel B DMA	Pulse
26	SPORT1_A_DMA	SPORT1 Channel A DMA	Pulse

Table 9-4: ADSP-BF70x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
27	SPORT1_B_DMA	SPORT1 Channel B DMA	Pulse
28	SPI0_TXDMA	SPI0 TX DMA Channel	Pulse
29	SPI0_RXDMA	SPI0 RX DMA Channel	Pulse
30	SPI1_TXDMA	SPI1 TX DMA Channel	Pulse
31	SPI1_RXDMA	SPI1 RX DMA Channel	Pulse
32	SPI2_TXDMA	SPI2 TX Channel	Pulse
33	SPI2_RXDMA	SPI2 RX Channel	Pulse
34	UART0_TXDMA	UART0 Transmit DMA	Pulse
35	UART0_RXDMA	UART0 Receive DMA	Pulse
36	UART1_TXDMA	UART1 Transmit DMA	Pulse
37	UART1_RXDMA	UART1 Receive DMA	Pulse
38	SYS_MDMA0_SRC	Memory DMA Stream 0 Source Channel	Pulse
39	SYS_MDMA0_DST	Memory DMA Stream 0 Destination Channel	Pulse
40	SYS_MDMA1_SRC	Memory DMA Stream 1 Source/CRC0 Input Channel	Pulse
41	SYS_MDMA1_DST	Memory DMA Stream 1 Destination/CRC0 Output Channel	Pulse
42	SYS_MDMA2_SRC	Memory DMA Stream 2 Source/CRC1 Input Channel	Pulse
43	SYS_MDMA2_DST	Memory DMA Stream 2 Destination/CRC0 Output Channel	Pulse
44	EPPI0_CH0_DMA	EPPI0 Channel 0 DMA	Pulse
45	EPPI0_CH1_DMA	EPPI0 Channel 1 DMA	Pulse
46	RCU0_C0RST0	RCU0 Core reset	Pulse
47	SYS_C0_WAKE0	Core Wakeup Input 0	Pulse
48	SYS_C0_WAKE1	Core Wakeup Input 1	Pulse
49	SYS_C0_WAKE2	Core Wakeup Input 2	Pulse
50	SYS_C0_WAKE3	Core Wakeup Input 3	Pulse
51	SWU0_EN	SWU0 Enable	Pulse
52	SWU1_EN	SWU1 Enable	Pulse
53	SWU2_EN	SWU2 Enable	Pulse
54	SWU3_EN	SWU3 Enable	Pulse
55	SWU4_EN	SWU4 Enable	Pulse

Table 9-4: ADSP-BF70x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
56	SWU5_EN	SWU5 Enable	Pulse
57	SWU6_EN	SWU6 Enable	Pulse
58	SWU7_EN	SWU7 Enable	Pulse
59	PORTA_TOGGLE	PORTA Port A Toggle	Pulse
60	PORTB_TOGGLE	PORTB Port B Toggle	Pulse
61	PORTC_TOGGLE	PORTC Port C Toggle	Pulse
62	CTI2_SLV0	CTI2 SYSCTI System Halt 0	Pulse
63	CTI2_SLV1	CTI2 SYSCTI System Halt 1	Pulse
64	CTI2_SLV2	CTI2 SYSCTI System Halt 2	Pulse
65	CTI2_SLV3	CTI2 SYSCTI System Halt 3	Pulse
66	CTI2_SLV4	CTI2 SYSCTI System Halt 4	Pulse
67	CTI2_SLV5	CTI2 SYSCTI System Halt 5	Pulse
68	CTI2_SLV6	CTI2 SYSCTI System Halt 6	Pulse
69	CTI2_SLV7	CTI2 SYSCTI System Halt 7	Pulse
70	STM0_EVT0	STM0 Event 0	Pulse
71	STM0_EVT1	STM0 Event 1	Pulse
72	STM0_EVT2	STM0 Event 2	Pulse
73	STM0_EVT3	STM0 Event 3	Pulse
74	STM0_EVT4	STM0 Event 4	Pulse
75	STM0_EVT5	STM0 Event 5	Pulse
76	STM0_EVT6	STM0 Event 6	Pulse
77	STM0_EVT7	STM0 Event 7	Pulse
78	STM0_EVT8	STM0 Event 8	Pulse
79	STM0_EVT9	STM0 Event 9	Pulse
80	STM0_EVT10	STM0 Event 10	Pulse
81	STM0_EVT11	STM0 Event 11	Pulse
82	STM0_EVT12	STM0 Event 12	Pulse
83	STM0_EVT13	STM0 Event 13	Pulse
84	STM0_EVT14	STM0 Event 14	Pulse
85	STM0_EVT15	STM0 Event 15	Pulse
86	STM0_EVT16	STM0 Event 16	Pulse

Table 9-4: ADSP-BF70x Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
87	STM0_EVT17	STM0 Event 17	Pulse
88	STM0_EVT18	STM0 Event 18	Pulse
89	STM0_EVT19	STM0 Event 19	Pulse
90	STM0_EVT20	STM0 Event 20	Pulse
91	STM0_EVT21	STM0 Event 21	Pulse
92	STM0_EVT22	STM0 Event 22	Pulse
93	STM0_EVT23	STM0 Event 23	Pulse
94	STM0_EVT24	STM0 Event 24	Pulse
95	STM0_EVT25	STM0 Event 25	Pulse
96	STM0_EVT26	STM0 Event 26	Pulse
97	STM0_EVT27	STM0 Event 27	Pulse
98	STM0_EVT28	STM0 Event 28	Pulse
99	STM0_EVT29	STM0 Event 29	Pulse
100	STM0_EVT30	STM0 Event 30	Pulse
101	STM0_EVT31	STM0 Event 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.

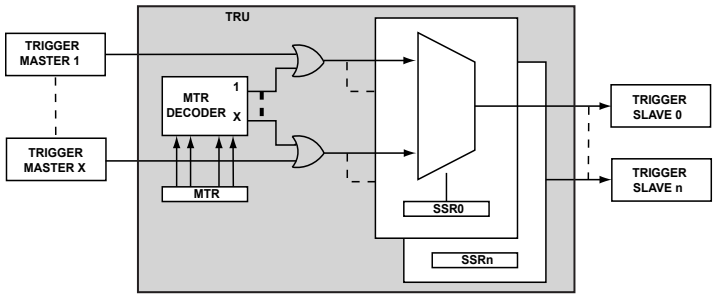

Figure 9-1: TRU Block Diagram

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 Example

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.

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 bus protocol.

ADSP-BF70x TRU Register Descriptions

Trigger Routing Unit (TRU) contains the following registers.

Table 9-5: ADSP-BF70x TRU Register List

Name	Description
`TRU_ERRADDR`	Error Address Register

Table 9-5: ADSP-BF70x TRU Register List (Continued)

Name	Description
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.

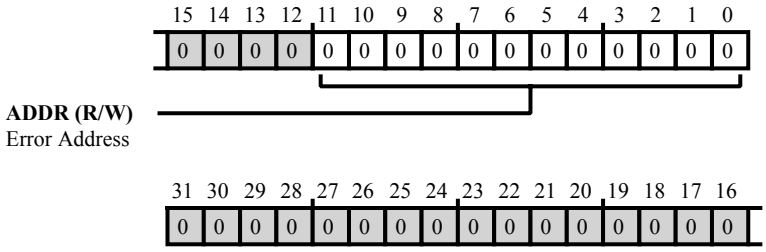

Figure 9-2: TRU_ERRADDR Register Diagram

Table 9-6: 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.

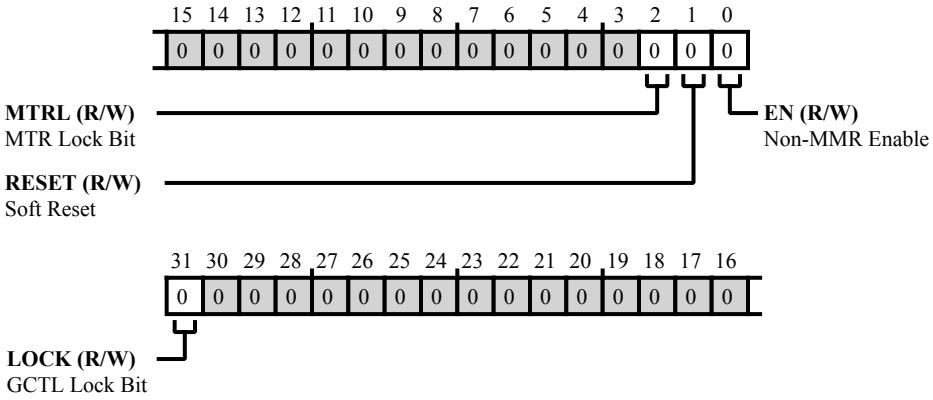

Figure 9-3: TRU_GCTL Register Diagram

Table 9-7: 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 9-7: 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.

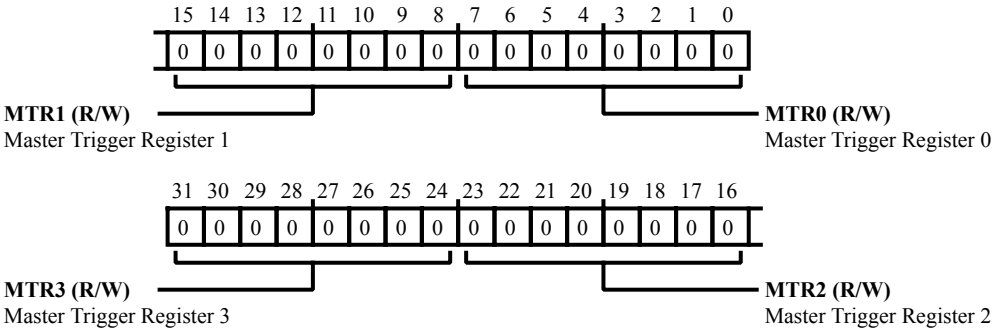

Figure 9-4: TRU_MTR Register Diagram

Table 9-8: TRU_MTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (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
		1-255	Range of valid masters
23:16 (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
		1-255	Range of valid masters
15:8 (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
		1-255	Range of valid masters
7:0 (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
		1-255	Range of valid masters

Slave Select Register

The TRU slave select registers (TRU_SSR[n]) each provide slave selection and register locking.

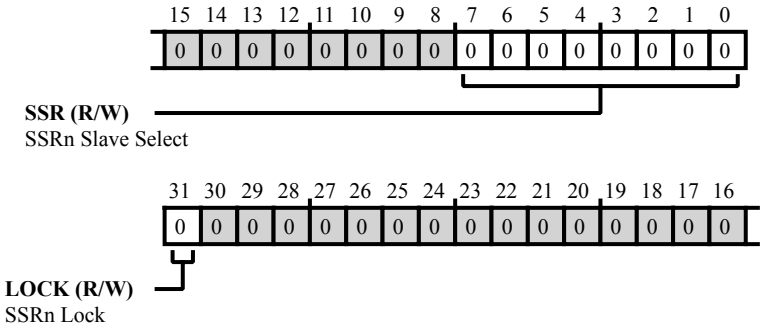

Figure 9-5: TRU_SSR[n] Register Diagram

Table 9-9: 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
7:0 (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
		1-255	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.

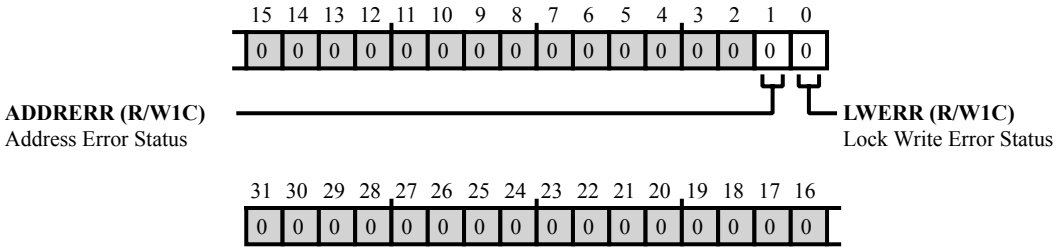

Figure 9-6: TRU_STAT Register Diagram

Table 9-10: 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

10 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
- NOR flash memory
- 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
- Supports asynchronous access extension (`SMC_ARDY` pin)
- Supports 8-bit data masking writes

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 SCLK0 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 (SMC_AOE) pin must be connected to the OR of the SMC_AMS0 and the SMC_ARE. Similarly, the write case requires an OR between SMC_AMS0 and 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.

ADSP-BF70x 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 10-1: ADSP-BF70x 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:

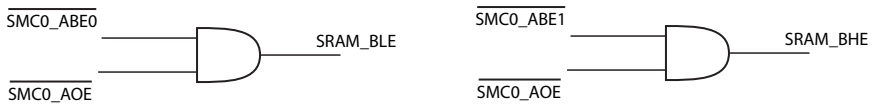

Figure 10-1: External Logic

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 SCLK0 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 SCLK0 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 accesses, such as cache line fills (16 words), 64-bit instruction reads (4 words), and 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 SCLK0 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.

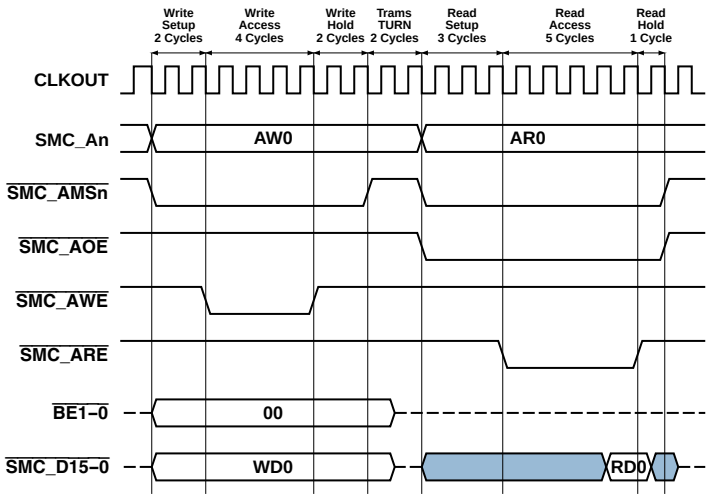

Figure 10-2: Basic Asynchronous SRAM Write Followed by Read

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 asserts. 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 ($\overline{\texttt{SMC\_AWE}}$) asserts.
3. At the end of the programmed write access, the $\overline{\texttt{SMC\_AWE}}$ signal deasserts. The target device is assumed to have captured the write data before $\overline{\texttt{SMC\_AWE}}$ deasserts.
4. At the end of the write hold period, the $\overline{\texttt{SMC\_AWE}}$ signal deasserts 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) presented on the address bus.
6. At the start of the read access period, the read enable signal, $\overline{\texttt{SMC\_ARE}}$ asserts.
7. At the end of the read access period, the $\overline{\texttt{SMC\_ARE}}$ signal deasserts 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.

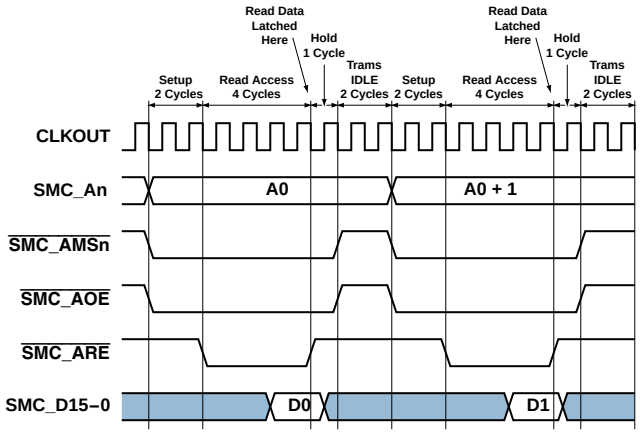

Figure 10-3: Asynchronous SRAM Read with IDLE Transition

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 $\overline{\texttt{SMC\_AMS[n]}}$ and $\overline{\texttt{SMC\_AOE}}$ signals. The setup period of the second read starts at the end of the IDLE transition cycle with the assertion of the `SMC_AMS[n]` and `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.

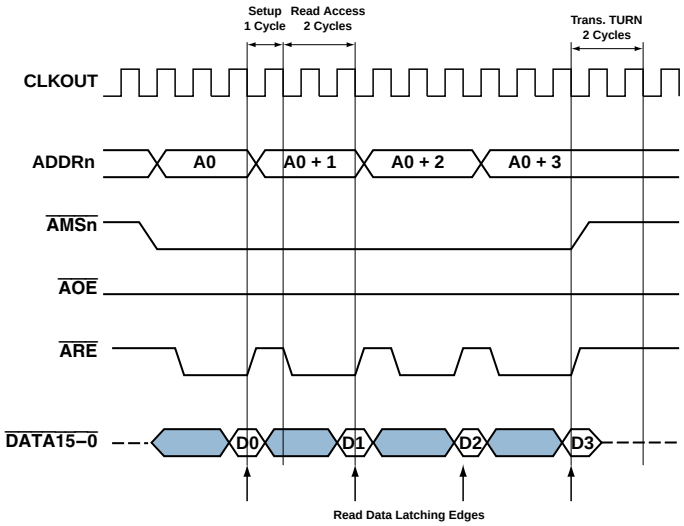

Figure 10-4: Fast Asynchronous SRAM Reads, Burst of Four Word

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 asserts 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.

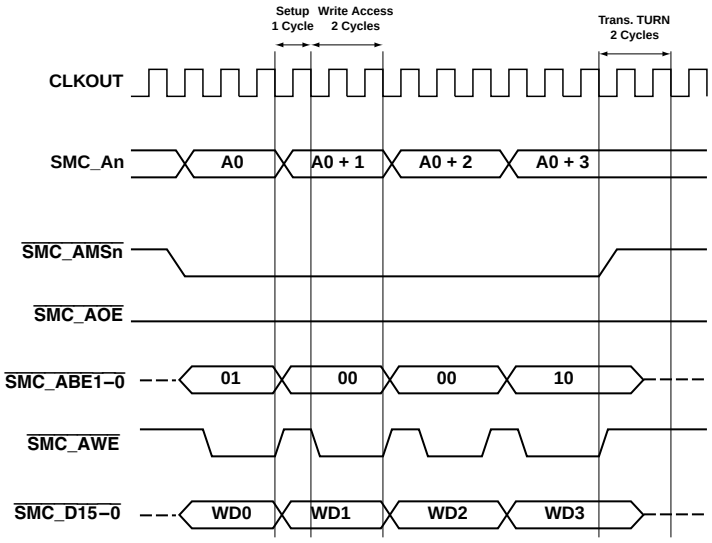

Figure 10-5: Fast Asynchronous SRAM Writes

Asynchronous SRAM Reads with ARDY

The ***Asynchronous SRAM Read with ARDY*** figure shows an extended asynchronous SRAM read bus cycle with `SMC_ARDY` enabled.

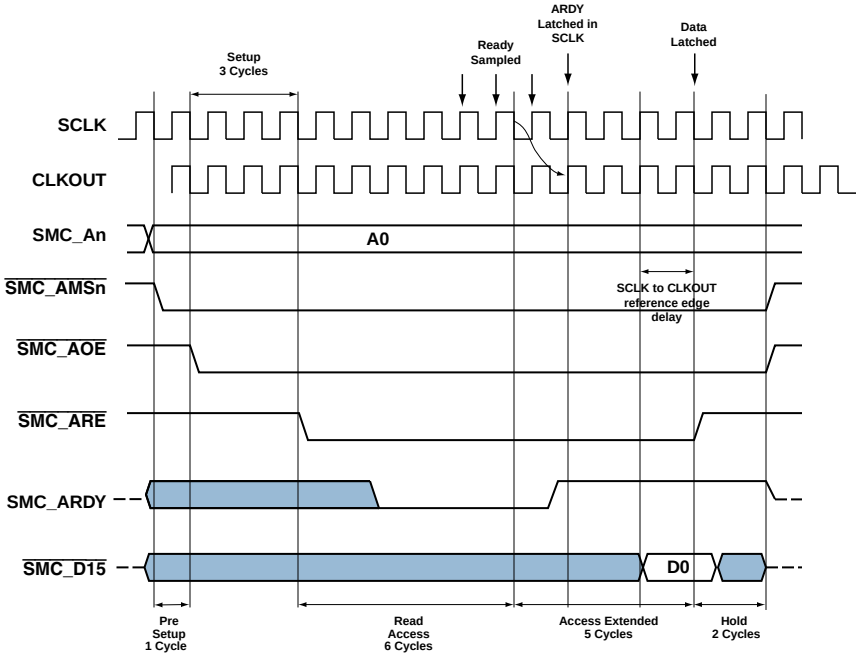

Figure 10-6: Asynchronous SRAM Read with ARDY

NOTE: SCLK in the ***Asynchronous SRAM Read with ARDY*** figure is SCLK0.

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, SMC_AMS[n] asserts, and read address SMC_A01 is presented on the address bus.
- At the start of the setup period, SMC_AOE asserts.
- At the start of the read access, SMC_ARE asserts.
- The CLKOUT signal is SCLK0 which is driven out of the pads. The CLKOUT signal falling edge can be delayed from the internal SCLK0 falling edge. See the data sheet for the specification related to this delay. All output signals out of the pads, for example SMC_ARE and 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 SCLK0 two cycles before the programmed number of read access cycles expires. The read access is extended (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 SCLK0, the read signal is pulled off internally in the SCLK0 domain. The total delay between the SMC_ARDY signal going high at the pads and the deassertion of the SMC_ARE signal at the pads can be a maximum of five SCLK0 cycles.
- Read data is latched at the falling edge of CLKOUT on the same edge where the SMC deasserts SMC_ARE.
- Hold bus cycles start after the SMC deasserts the SMC_ARE signal.
- At the end of the hold period, the SMC_AMS[n] and 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.

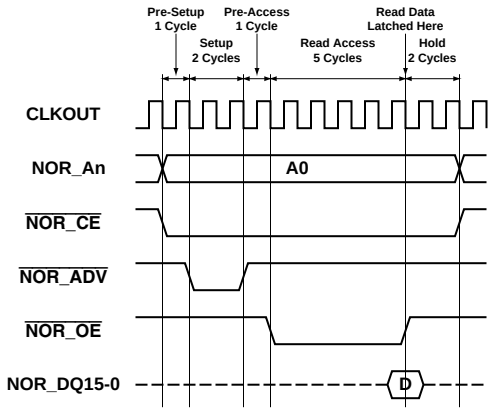

Figure 10-7: Asynchronous Flash Read with Pre-Setup and Pre-Access Cycles

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.

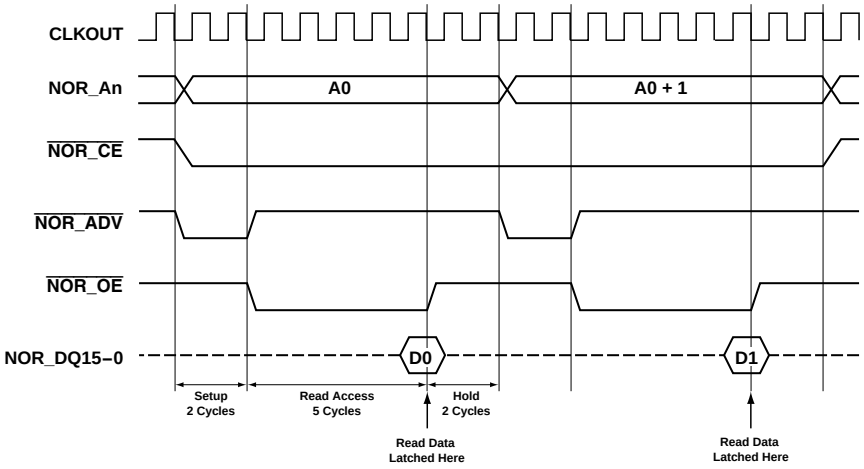

Figure 10-8: 32-bit Asynchronous Flash Read

Asynchronous Flash Writes

The ***Asynchronous Flash Write Operation*** figure shows a single asynchronous flash write bus cycle.

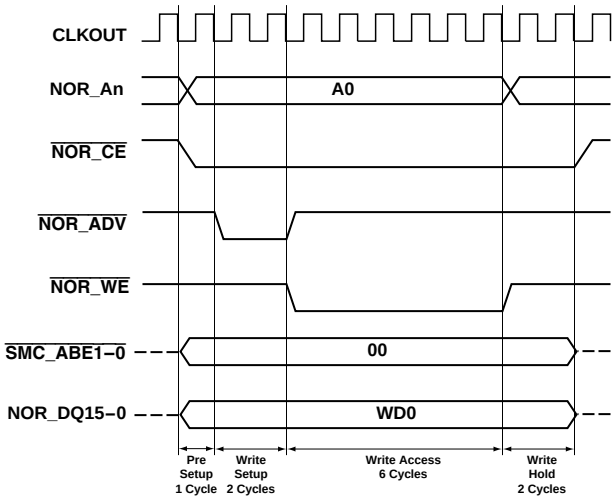

Figure 10-9: Asynchronous Flash Write Operation

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.

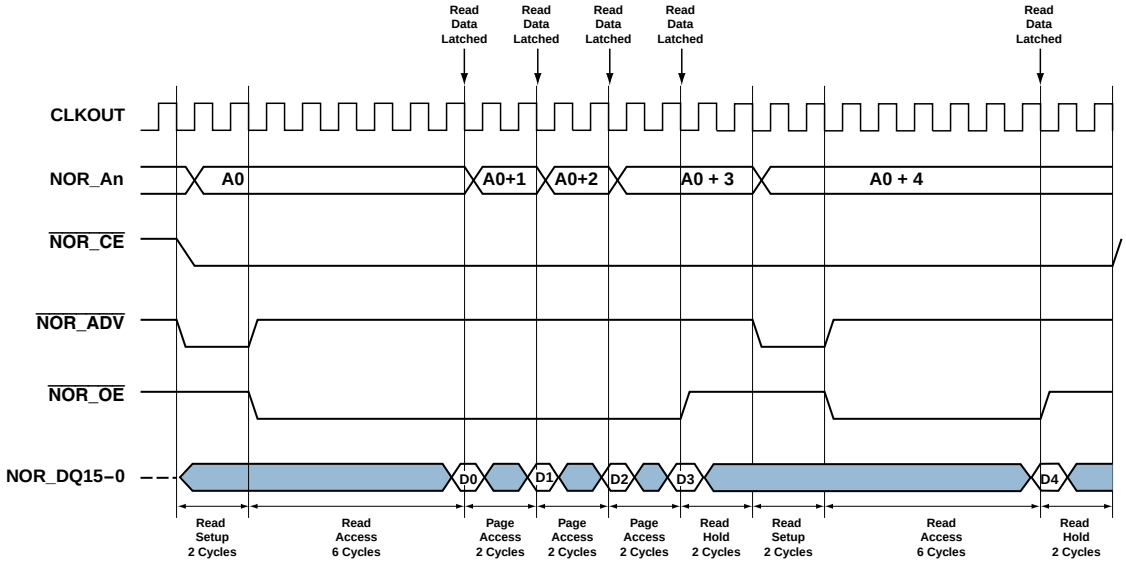

Figure 10-10: Asynchronous Page Mode Read Bus Cycle

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).

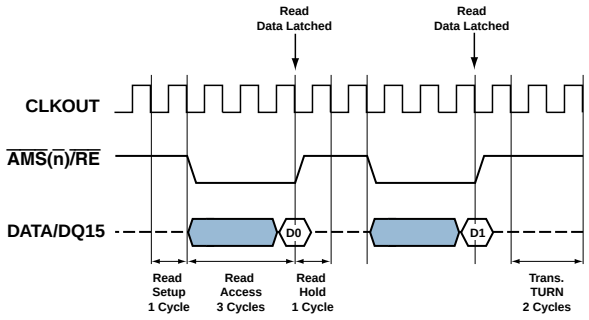

Figure 10-11: Asynchronous FIFO Read Bus Cycles

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 ($\overline{WE}$) 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.

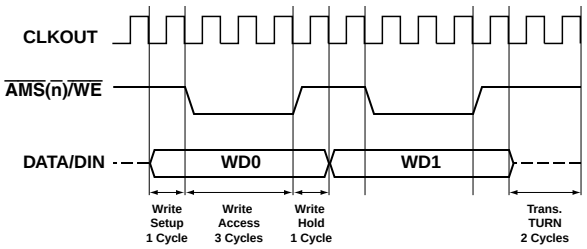

Figure 10-12: Asynchronous FIFO Write Bus Cycles

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

ADSP-BF70x SMC Register Descriptions

Static Memory Controller (SMC) contains the following registers.

Table 10-2: ADSP-BF70x 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.

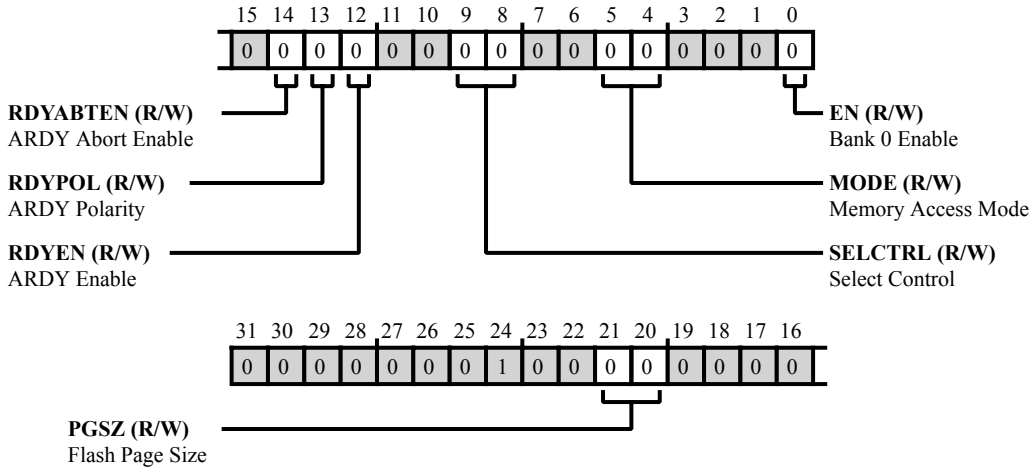

Figure 10-13: SMC_B0CTL Register Diagram

Table 10-3: 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
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 SCLK0. 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

Table 10-3: SMC_B0CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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.

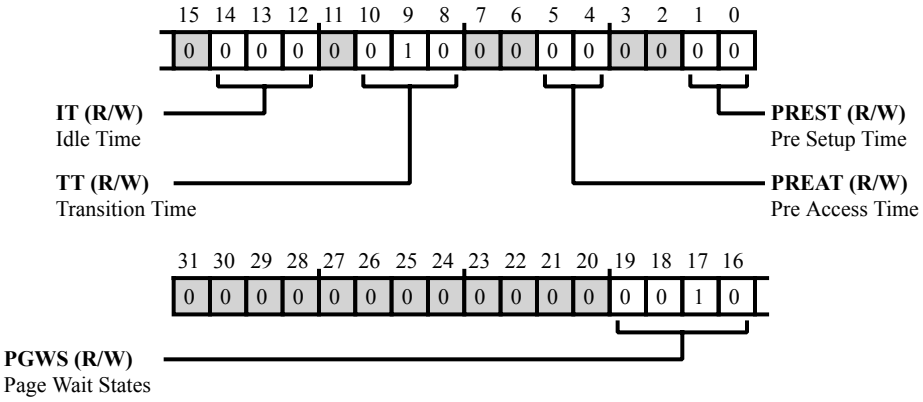

Figure 10-14: SMC_B0ETIM Register Diagram

Table 10-4: 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 SCLK0 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 SCLK0 cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK0 clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B0ETIM.IT bits select a bus idle time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		7	7 SCLK0 clock cycles

Table 10-4: 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 SCLK0 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 SCLK0 cycles.	
		0	No bank transition
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B0ETIM.PREAT bits select the pre-access time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		3	3 SCLK0 clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B0ETIM.PREST bits select the pre-setup time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		3	3 SCLK0 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.

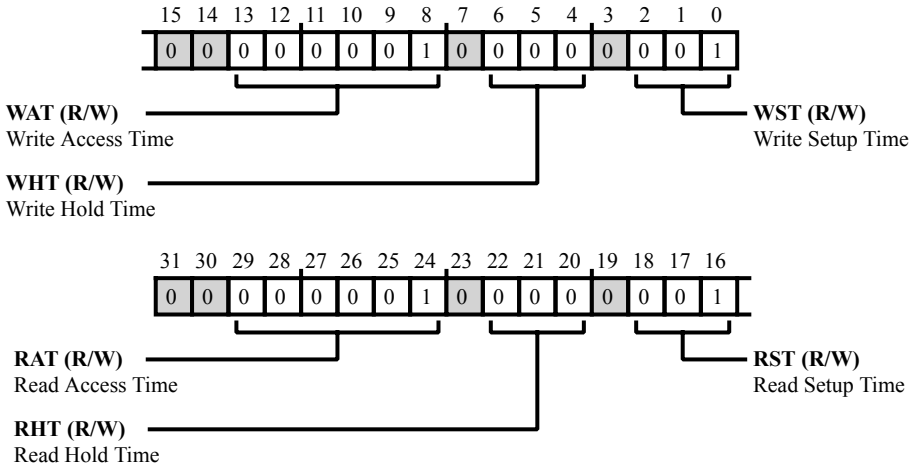

Figure 10-15: SMC_B0TIM Register Diagram

Table 10-5: 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 SCLK0 cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK0 cycles.	
		0	Not supported
		1	1 SCLK0 clock cycle
		63	63 SCLK0 clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B0TIM.RHT bits select the hold time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles

Table 10-5: 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 SCLK0 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 SCLK0 cycles.	
		0	8 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B0TIM.WAT bits select the access time (in SCLK0 cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK0 cycles.	
		0	Not supported
		1	1 SCLK0 clock cycle
		63	63 SCLK0 clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B0TIM.WHT bits select the hold time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B0TIM.WST bits select the setup time (in SCLK0 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 SCLK0 cycles.	
		0	8 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles

Bank 1 Control Register

The SMC_B1CTL register enables bank 1 accesses and configures the memory access features for this bank.

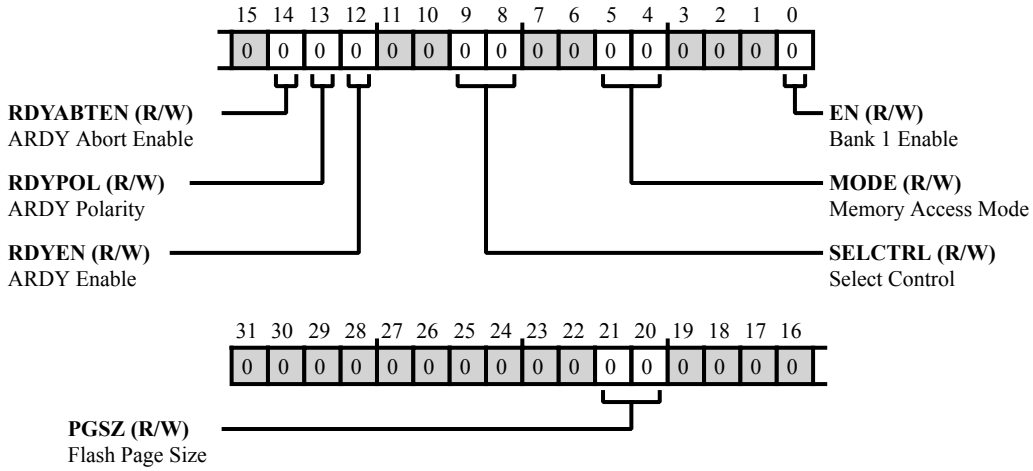

Figure 10-16: SMC_B1CTL Register Diagram

Table 10-6: 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
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 SCLK0. 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

Table 10-6: SMC_B1CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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.

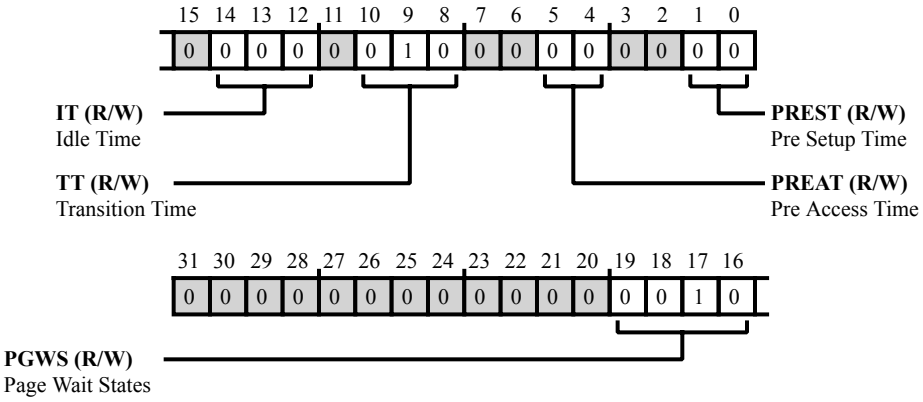

Figure 10-17: SMC_B1ETIM Register Diagram

Table 10-7: 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 SCLK0 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 SCLK0 cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK0 clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B1ETIM.IT bits select a bus idle time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		7	7 SCLK0 clock cycles

Table 10-7: 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 SCLK0 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 SCLK0 cycles.	
		0	No bank transition
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B1ETIM.PREAT bits select the pre-access time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		3	3 SCLK0 clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B1ETIM.PREST bits select the pre-setup time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		3	3 SCLK0 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.

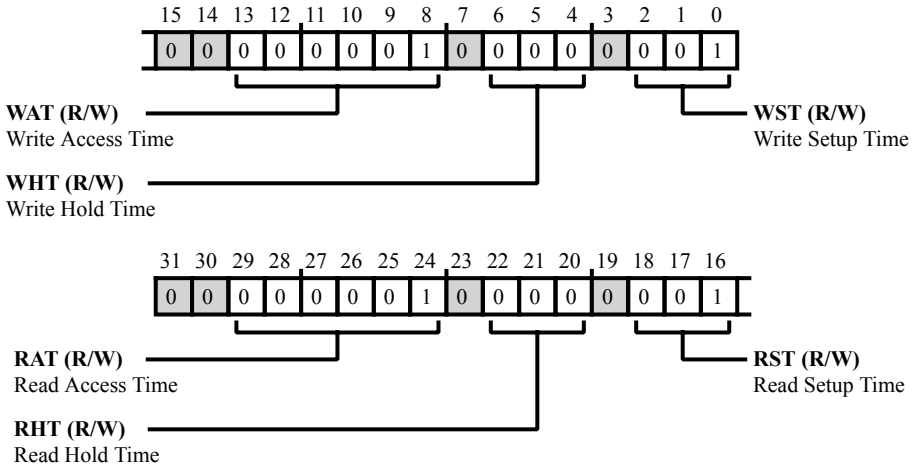

Figure 10-18: SMC_B1TIM Register Diagram

Table 10-8: 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 SCLK0 cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK0 cycles.	
		0	Not supported
		1	1 SCLK0 clock cycle
		63	63 SCLK0 clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B1TIM.RHT bits select the hold time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles

Table 10-8: 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 SCLK0 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 SCLK0 cycles.	
		0	8 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B1TIM.WAT bits select the access time (in SCLK0 cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK0 cycles.	
		0	Not supported
		1	1 SCLK0 clock cycle
		63	63 SCLK0 clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B1TIM.WHT bits select the hold time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B1TIM.WST bits select the setup time (in SCLK0 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 SCLK0 cycles.	
		0	8 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles

Bank 2 Control Register

The SMC_B2CTL register enables bank 2 accesses and configures the memory access features for this bank.

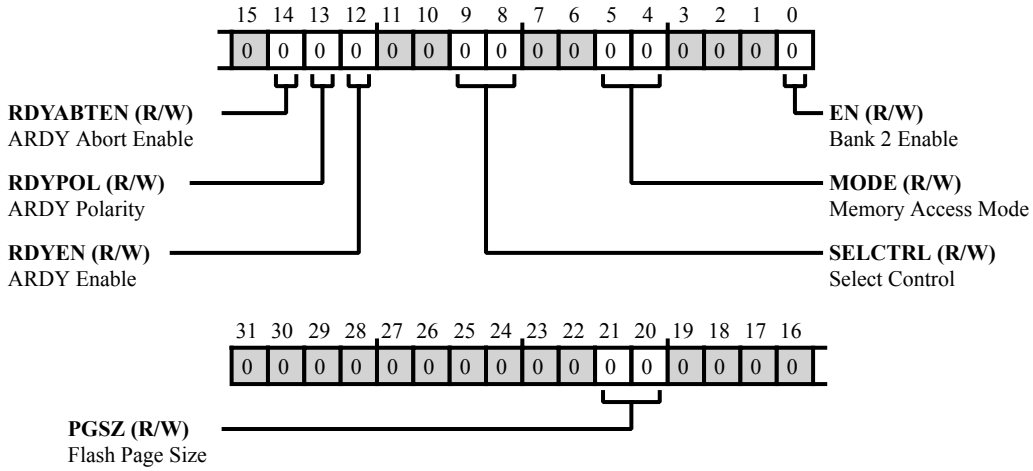

Figure 10-19: SMC_B2CTL Register Diagram

Table 10-9: 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
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 SCLK0. 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

Table 10-9: SMC_B2CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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.

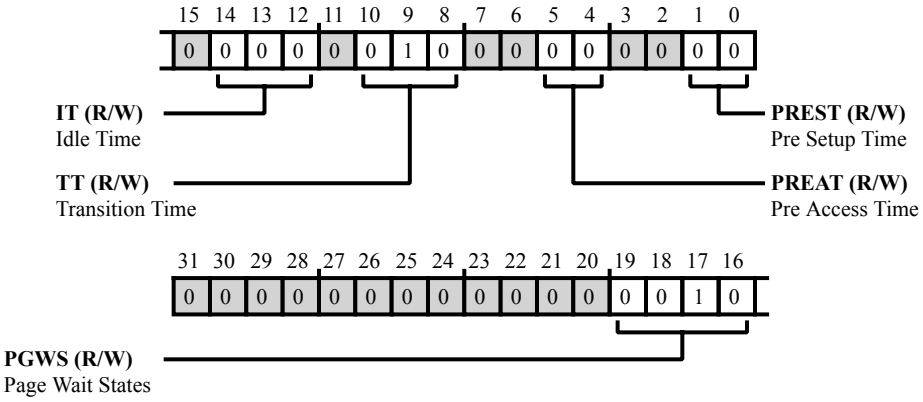

Figure 10-20: SMC_B2ETIM Register Diagram

Table 10-10: 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 SCLK0 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 SCLK0 cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK0 clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B2ETIM.IT bits select a bus idle time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		7	7 SCLK0 clock cycles

Table 10-10: 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 SCLK0 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 SCLK0 cycles.	
		0	No bank transition
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B2ETIM.PREAT bits select the pre-access time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		3	3 SCLK0 clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B2ETIM.PREST bits select the pre-setup time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		3	3 SCLK0 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.

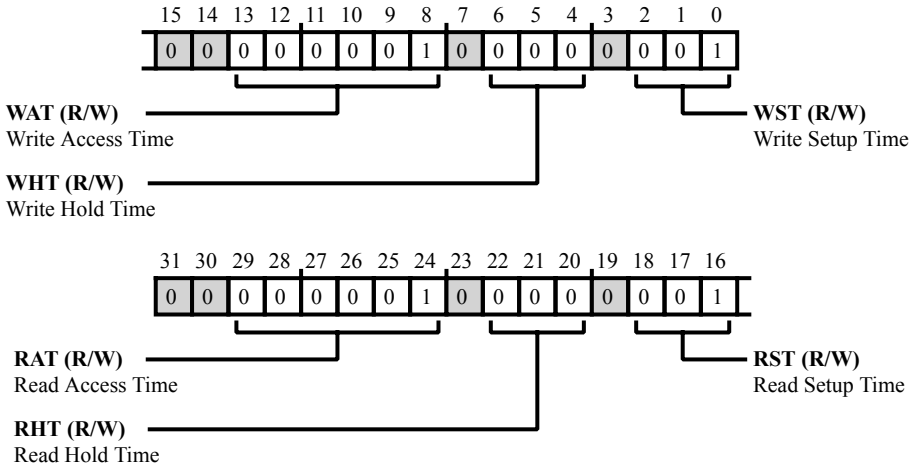

Figure 10-21: SMC_B2TIM Register Diagram

Table 10-11: 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 SCLK0 cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK0 cycles.	
		0	Not supported
		1	1 SCLK0 clock cycle
		63	63 SCLK0 clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B2TIM.RHT bits select the hold time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles

Table 10-11: 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 SCLK0 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 SCLK0 cycles.	
		0	8 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B2TIM.WAT bits select the access time (in SCLK0 cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK0 cycles.	
		0	Not supported
		1	1 SCLK0 clock cycle
		63	63 SCLK0 clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B2TIM.WHT bits select the hold time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B2TIM.WST bits select the setup time (in SCLK0 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 SCLK0 cycles.	
		0	8 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles

Bank 3 Control Register

The SMC_B3CTL register enables bank 3 accesses and configures the memory access features for this bank.

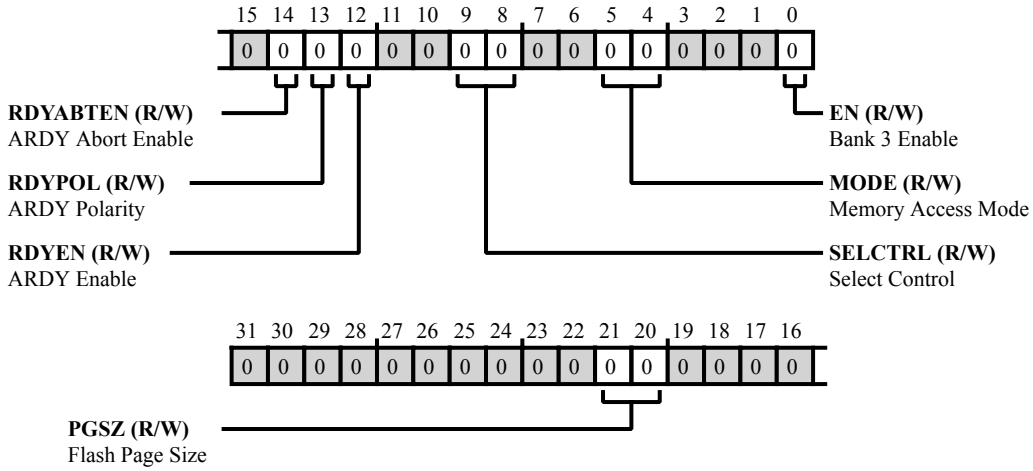

Figure 10-22: SMC_B3CTL Register Diagram

Table 10-12: 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
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 SCLK0. 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

Table 10-12: SMC_B3CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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.

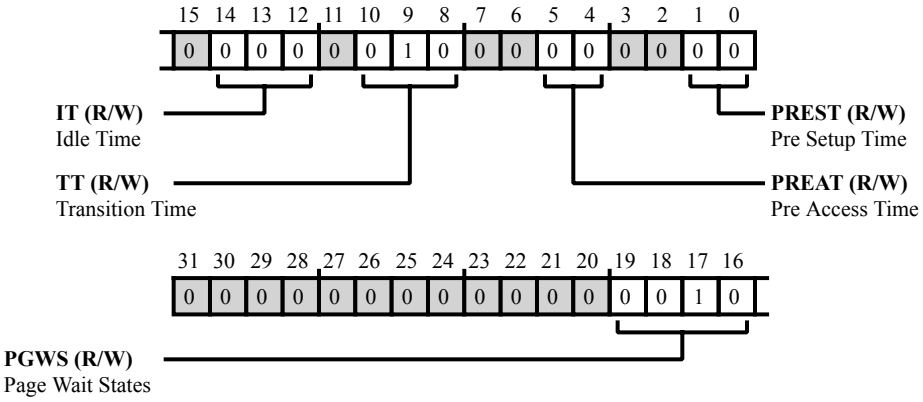

Figure 10-23: SMC_B3ETIM Register Diagram

Table 10-13: 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 SCLK0 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 SCLK0 cycles.	
		0	Not supported
		1	Not supported
		2-15	2-15 SCLK0 clock cycles
14:12 (R/W)	IT	Idle Time. The SMC_B3ETIM.IT bits select a bus idle time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		7	7 SCLK0 clock cycles

Table 10-13: 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 SCLK0 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 SCLK0 cycles.	
		0	No bank transition
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
5:4 (R/W)	PREAT	Pre Access Time. The SMC_B3ETIM.PREAT bits select the pre-access time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		3	3 SCLK0 clock cycles
1:0 (R/W)	PREST	Pre Setup Time. The SMC_B3ETIM.PREST bits select the pre-setup time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		3	3 SCLK0 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.

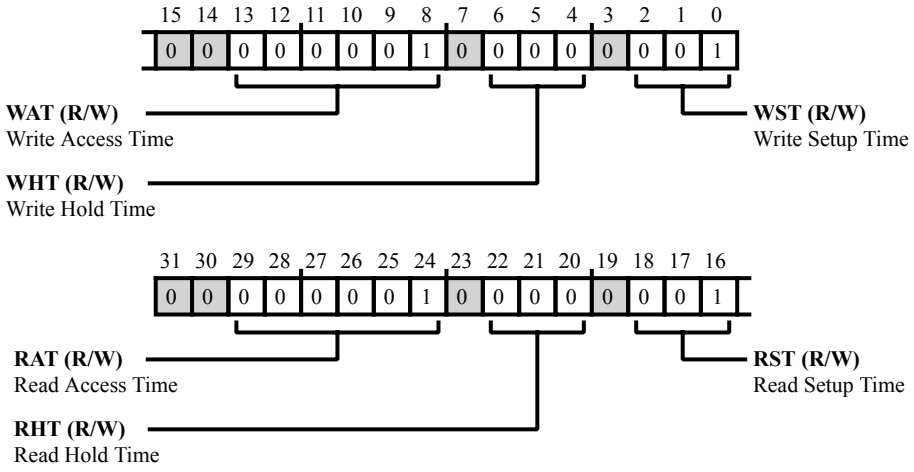

Figure 10-24: SMC_B3TIM Register Diagram

Table 10-14: 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 SCLK0 cycles) that the SMC asserts the SMC_ARE pin for a read access. The access time is from 1 to 63 SCLK0 cycles.	
		0	Not supported
		1	1 SCLK0 clock cycle
		63	63 SCLK0 clock cycles
22:20 (R/W)	RHT	Read Hold Time. The SMC_B3TIM.RHT bits select the hold time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles

Table 10-14: 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 SCLK0 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 SCLK0 cycles.	
		0	8 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
13:8 (R/W)	WAT	Write Access Time. The SMC_B3TIM.WAT bits select the access time (in SCLK0 cycles) that the SMC asserts the SMC_AWE pin for a write access. The access time is from 1 to 63 SCLK0 cycles.	
		0	Not supported
		1	1 SCLK0 clock cycle
		63	63 SCLK0 clock cycles
6:4 (R/W)	WHT	Write Hold Time. The SMC_B3TIM.WHT bits select the hold time (in SCLK0 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 SCLK0 cycles.	
		0	0 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles
2:0 (R/W)	WST	Write Setup Time. The SMC_B3TIM.WST bits select the setup time (in SCLK0 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 SCLK0 cycles.	
		0	8 SCLK0 clock cycles
		1	1 SCLK0 clock cycle
		7	7 SCLK0 clock cycles

11 L2 System Memory

The L2 system memory manages L2 SRAM and ROM memories and provides the interface between these memories and the system fabric. The L2 system memory is a shared resource. For example, on multi-core processors, processor cores and the DMA controllers can access L2 memories.

L2 system memories have significant bandwidth for core accesses, but it is important to note that L2 responds slower to core accesses than L1 memories. L2 SRAM is the ideal storage for multiple processor cores to share data and instruction resources, such as semaphores, shared buffers, and code libraries. Due to sophisticated data integrity protection and write protection, L2 SRAM is also ideal for data and instructions critical for safe operation of the application.

L2 System Memory Features

The L2 system memory features include:

- Operation at SYSCLK frequency
- Write protection of SRAM banks
- ECC protection of SRAM area
- ECC memory refresh
- 1M bytes of SRAM grouped into eight banks, 128K bytes each
- 512K bytes of ROM featuring the boot code
- Hardware support for L2 RAM Initialization
- Support for automatic refresh (or scrub) support
- Support for deep sleep and power down modes

L2 System Memory Functional Description

The L2 system memory manages all of the L2 SRAM and ROM memory banks. The system memory interface arbitrates competing accesses, write protection, and ensures SRAM data integrity. The L2 system memory domain is a unified instruction and data memory. It can hold any mixture of code and data required by the system design.

The *ADSP-BF70x Processor - L2 Memory Address Mapping* table shows the L2 memory map for the ADSP-BF70x processor.

Table 11-1: ADSP-BF70x Processor - L2 Memory Address Mapping

Start Address	End Address	Description
0x04000000	0x407FFFFF	L2 Bank 0 ROM (512 KB)
0x08000000	0x801FFFFF	L2 Bank 0 RAM (128 KB)
0x08020000	0x803FFFFF	L2 Bank 1 RAM (128 KB)
0x08040000	0x805FFFFF	L2 Bank 2 RAM (128 KB)
0x08060000	0x807FFFFF	L2 Bank 3 RAM (128 KB)
0x08080000	0x809FFFFF	L2 Bank 4 RAM (128 KB)
0x080A0000	0x80BFFFFF	L2 Bank 5 RAM (128 KB)
0x080C0000	0x80DFFFFF	L2 Bank 6 RAM (128 KB)
0x080E0000	0x80FFFFFF	L2 Bank 7 RAM (128 KB)

The following sections provide a functional description of the L2 system memory.

ADSP-BF70x L2CTL Trigger List

Table 11-2: ADSP-BF70x L2CTL Trigger List Masters

Trigger ID	Name	Description	Sensitivity
46	`L2CTL0_EVT`	L2CTL0 L2 Event	Level

Table 11-3: ADSP-BF70x L2CTL Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

ADSP-BF70x L2CTL Interrupt List

Table 11-4: ADSP-BF70x L2CTL Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
4	L2CTL0_ECC_ERR	L2CTL0 ECC Error	Level	
5	Reserved	Reserved	Reserved	Reserved
9	L2CTL0_EVT	L2CTL0 L2 Event (Scrub or Initialization)	Level	

ADSP-BF70x L2CTL Register List

The L2 memory controller (L2CTL) includes the controls to manage each L2 memory bank independently. A set of registers governs L2CTL operations. For more information on L2CTL functionality, see the L2CTL register descriptions.

Table 11-5: ADSP-BF70x L2CTL Register List

Name	Description
L2CTL_ACTL_C0	Access Control Core 0 Register
L2CTL_ACTL_SYS	Access Control System Register
L2CTL_CTL	Control Register
L2CTL_EADDR0	Error Type 0 Address Register
L2CTL_EADDR1	Error Type 1 Address Register
L2CTL_ERRADDR0	ECC Error Address 0 Register
L2CTL_ERRADDR1	ECC Error Address 1 Register
L2CTL_ERRADDR2	ECC Error Address 2 Register
L2CTL_ERRADDR3	ECC Error Address 3 Register
L2CTL_ERRADDR4	ECC Error Address 4 Register
L2CTL_ERRADDR5	ECC Error Address 5 Register
L2CTL_ERRADDR6	ECC Error Address 6 Register
L2CTL_ERRADDR7	ECC Error Address 7 Register
L2CTL_ERRADDR8	ECC Error Address 8 Register
L2CTL_ET0	Error Type 0 Register
L2CTL_ET1	Error Type 1 Register
L2CTL_INIT	Initialization Register
L2CTL_ISTAT	Initialization Status Register
L2CTL_PCTL	Power Control Register
L2CTL_REVID	Revision ID Register
L2CTL_RPCR	Read Priority Count Register
L2CTL_SADR	Scrub Start Address Register
L2CTL_SCNT	Scrub Count Register
L2CTL_SCTL	Scrub Control Register
L2CTL_STAT	Status Register
L2CTL_WPCR	Write Priority Count Register

L2 System Memory Block Diagram

As shown in the ***L2 System Memory Block Diagram***, the L2 system memory organizes data the SRAM in multiple banks, and each bank has 128K bytes of data. Within each bank, the L2 system memory organizes data into 16384 words, with each word comprising 64 bits of data and 14 bits of ECC checksum. ROM has 65536 words with each word comprising 64 bits of data and 14 bits of ECC checksum. When the L2 system memory accesses RAM and ROM cells, it always reads and writes whole 64-bit words. Despite this access word width, the L2 system memory supports 8-bit, 16-bit, and 32-bit reads and writes from cores and system by applying respective data masks.

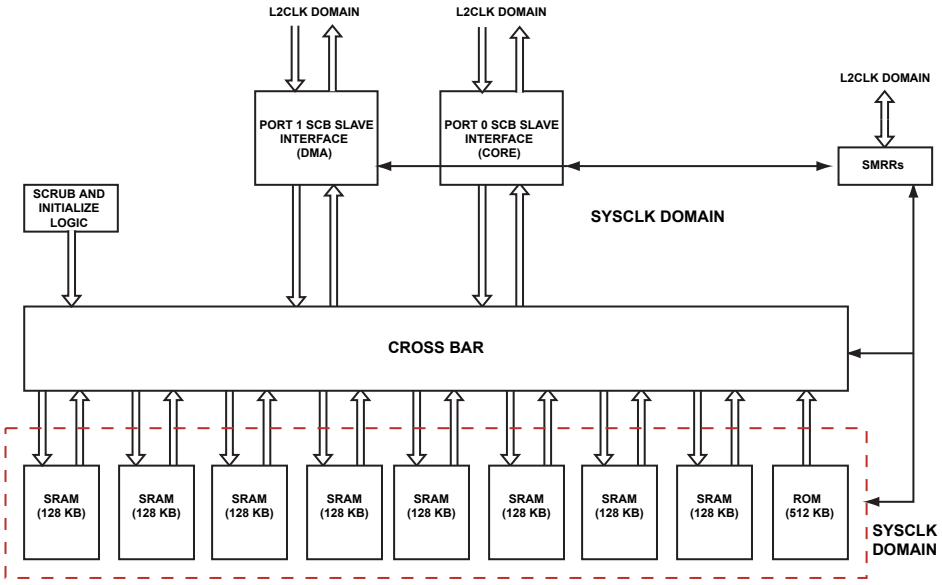

Figure 11-1: L2 System Memory Block Diagram

L2 System Memory Architectural Concepts

The following sections describe architecture features of the L2 system memory.

- Read/Write Latency and Throughput
- Arbitration and Priority

Power Modes

This section discusses the low-power modes available in L2 system memory module.

If the bank is not in use, put each L2 system memory bank into the following low-power modes.

Deep Sleep Mode

Set the `L2CTL_PCTL.BK7DS` through `L2CTL_PCTL.BK0DS` control bits to enter this mode. This mode shuts down power to the peripheral of the memory cells, but maintains the memory contents.

Shut Down Mode

Set the L2CTL_PCTL.BK8SD through L2CTL_PCTL.BK0SD control bits to enter this mode. This mode shuts down power to the peripheral and to the memory core of each memory bank. The memory banks do not retain the data (existing data is lost).

Both deep sleep and shut down modes provide considerable power savings. In the worst corner case for power, deep sleep reduces the idle power by approximately 33%. This mode shuts down by approximately 76% on a per bank basis. Access to a memory bank with deep sleep or shut down activated can result in unpredictable behavior. The access can be made to the particular bank only 30 SYSCLK cycles after deep sleep or after shut down to the particular bank is deactivated. The L2 system memory does not restrict accesses when deep sleep or shut down is set. The program must ensure that accesses are not made, and that initialization or refresh is disabled to the particular bank which is put into low-power mode.

Access Characteristics

The L2 system memory interface converts all 8-bit, 16-bit, and 32-bit accesses to 64-bit accesses. Additionally, it converts 8-bit, 16-bit, and 32-bit bursts to an equivalent internal 64-bit access. For example, the L2 system memory interface converts a 64-bit address-aligned burst of 8-bit accesses of burst length 8 to a single 64-bit access.

The L2 system memory supports restriction of both core and DMA write accesses to a particular memory bank. Set the appropriate bank write disable bit in the L2CTL_ACTL_C0 registers of the port to disable the write access by a port to a particular RAM bank. Illegal access attempts generate an error response.

Read/Write Latency and Throughput

The L2 memory design is optimized for burst accesses at the crossbar interface. The L2 system memory buffers and converts write data of 8/16/32-bit to an equivalent 64-bit access. This conversion creates modulo-32-bit writes if the starting addresses are 32-bit aligned. A single 8-bit or 16-bit access, or a non-32-bit address-aligned 8-bit or 16-bit burst access to an ECC-enabled bank creates an extra latency of two SYSCLK cycles. No extra latency is seen if the ECC is disabled.

NOTE: Continuous 8/16-bit core access to an ECC-enabled L2 bank is not recommended from a throughput perspective.

Arbitration and Priority

Each bank of L2 RAM or ROM has an arbiter which receives requests from the two crossbar ports.

Each arbiter follows a fixed priority scheme for giving grants when more than one channel requests the same bank. The arbiter also supports priority elevation through urgent priority requests.

NOTE: Attempting a write access to both L2 ROM spaces returns an error.

The ***Fixed Priority*** table shows the priority for fixed priority mode (with urgent priority disabled) for each SCB channel.

If two cores simultaneously try to access L2 for the same instance (both read or both write), even to different banks, software allows only one core access at a time. One access port can support one read and write at the same time. However, if one core issues a write and the other issues a read, then access can proceed simultaneously. There is no extra latency inside L2, as long as the accesses are to different banks (assuming pending DMA traffic is also to a non-conflicting bank).

When a core and DMA both access the same bank via the same port (both read or both writes), the best access rate that DMA can achieve is one 64-bit access in every three SYSCLK cycles during the conflict period. This access rate is achieved by programming the read priority count register (`L2CTL_RPCR.RPC0`) bit and the write priority count register (`L2CTL_WPCR.WPC0`) bit to 0, while programming the `L2CTL_RPCR.RPC1` and the `L2CTL_WPCR.WPC1` bits to 1.

Table 11-6: Fixed Priority

Channel	Priority Level
L2 Refresh Request	5 (highest)
Port 0 Read Channel	4
Port 0 Write Channel	3
Port 1 Read Channel	2
Port 1 Write Channel	1 (lowest)

Table 11-7: Fixed Priority

Channel	Priority Level
L2 Refresh Request	7 (highest)
Port 0 Read Channel	6
Port 0 Write Channel	5
Port 1 Read Channel	4
Port 1 Write Channel	3
Port 2 Read Channel	2
Port 2 Write Channel	1 (lowest)

The arbiters also support priority elevation for a particular channel that has been starved of grants for many SYSCLK cycles. If a channel does not get a grant for *N* cycles after its request, then that channel can elevate the priority of its request by issuing an urgent priority request. This request causes that particular channel to become the highest priority master for the next grant cycle (pipelined arbitration for urgent priority). The number of cycles *N*, after which the priority is elevated, can be programmed for each channel separately using the `L2CTL_RPCR` and `L2CTL_WPCR` registers.

Programming the bits in the `L2CTL_RPCR` and `L2CTL_WPCR` registers appropriately achieves the best grant rate for DMA. This grant rate of one in three SYSCLK cycles during the conflict period is achievable under the following conflict conditions:

- An access conflict between the core and DMA to the same memory bank in the fixed priority arbitration scheme with core activity always prioritized over DMA activity
- An access conflict within the pipelined implementation of urgent priority

To disable urgent priority requests, set the `L2CTL_CTL.DISURP` bit. This bit disables the urgent priority requests for all port channels. Each channel can also be prevented from raising the urgent priority request through the priority count register for the specific channel. However, there is no support for disabling urgent priority for a specific memory bank arbiter.

The ***Fixed Priority With Priority Elevation*** table provides the various priority levels for the L2 system memory.

Table 11-8: Fixed Priority With Priority Elevation

Channel	Priority Level
L2 Refresh Request	9 (highest)
Port 0 Read Channel Urgent Request	8
Port 0 Write Channel Urgent Request	7
Port 1 Read Channel Urgent Request	6
Port 1 Write Channel Urgent Request	5
Port 0 Read Channel Normal Request	4
Port 0 Write Channel Normal Request	3
Port 1 Read Channel Normal Request	2
Port 1 Write Channel Normal Request	1 (lowest)

Data Integrity

To ensure data integrity, the L2 system memory protects all L2 SRAM and ROM with an error-correcting code (ECC). A 7-bit ECC checksum protects each 32-bit SRAM/ROM entity. If the L2 system memory detects a single bit error at read time, the system memory identifies the failing bit and auto-corrects the value outputted to the system. Also, the L2 system memory can detect 2-bit errors safely and can detect a large range of multi-bit errors. This scheme is often referred to as (39,32) single-error correction, dual-error detection (SECDED) checksum protection.

The following sections provide information on how the L2 system memory ensures data integrity.

ECC Algorithm

Hsaio encoding calculates the ECC syndrome. A 7-bit syndrome is generated during write operation and stored as a 7-bit parity field along with the 32 data bits. Each data bit contributes to three parity bits according. Each parity bit represents the XOR value of 13 or 14 data bits according to the following mapping:

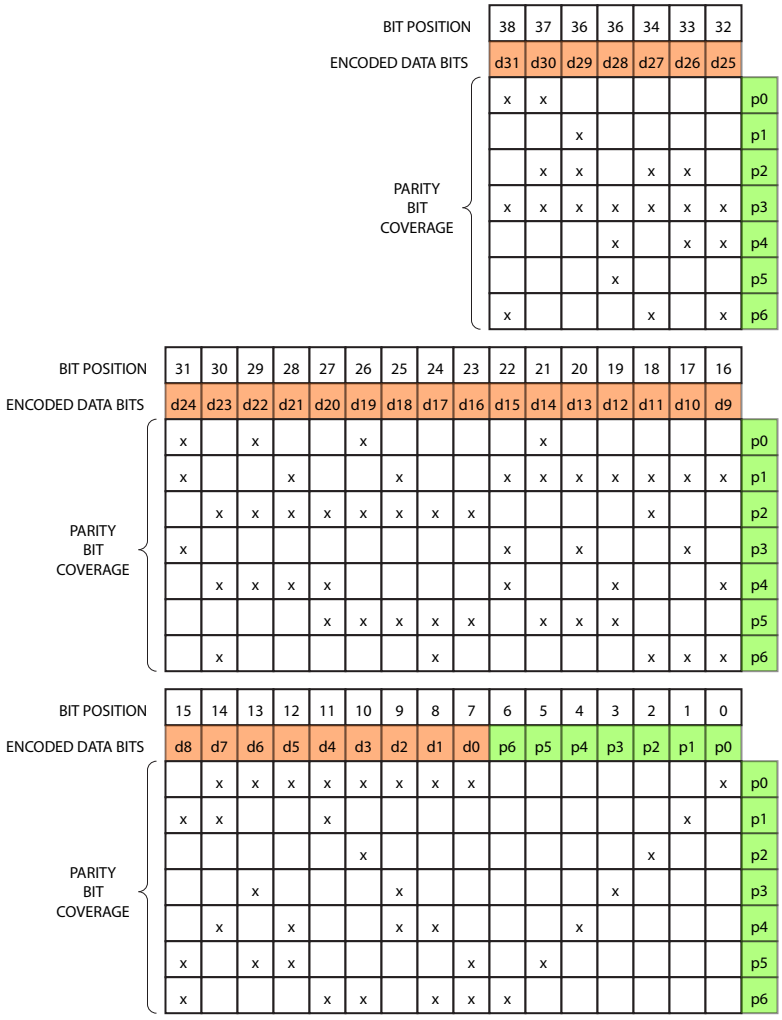

Figure 11-2: Hsaio Parity Bit Mapping

During read operation, the parity bits become part of the syndrome equation. The new syndrome bits are now the XOR values of the 13 or 14 data bits plus the respective stored parity bit. If any of the seven syndrome bits is set, an error situation is detected. An OR gate cross of the 7 bits reports the error, without specifying the type of the error.

If a single parity bit failed, the new 7-bit syndrome has 1 bit that is set. If a single data bit failed, the new syndrome has 3 bits that are set, because all three related parity bits fail. So, an XOR gate cross of all seven syndrome bits detects a single-bit error, indicating that an odd number of syndrome bits is set.

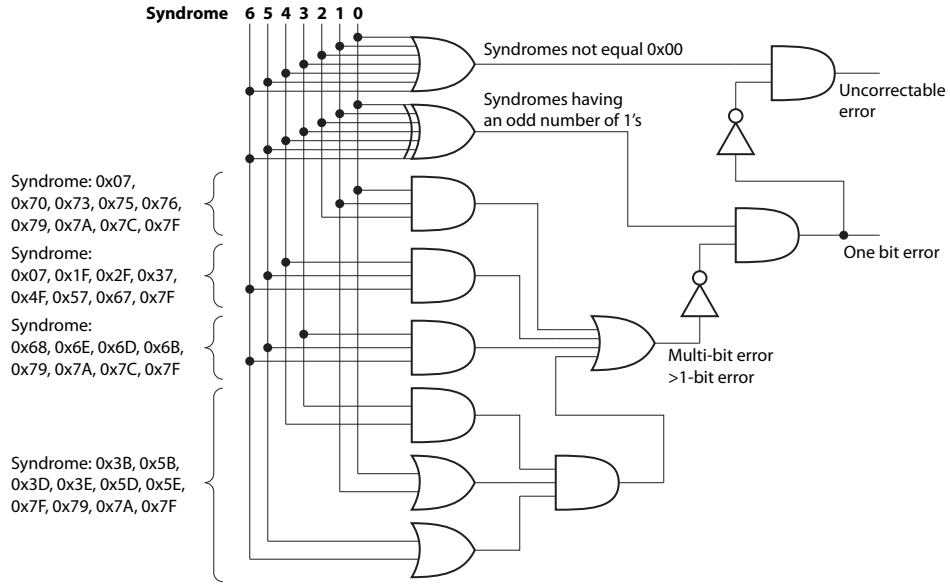

Figure 11-3: Hsiao Error Reports

The XOR gate detects single-bit errors and does not flag any dual-bit error. But, the gate does flag 50% of the other multi-bit errors undesirably. Extra logic is implemented to increase the detection rate of multi-bit errors to 68.7% as shown in the figure.

If a single-bit error is detected, the failing bit can be localized and corrected. If all three syndrome bits corresponding to a specific data bit are 1, a data error is assumed. The respective data bit is toggled on its way to the system bus.

ECC Hardware Control

After reset, ECC protection is enabled. The boot code initializes all L2 SRAM data and checksum cells. ECC protection adds some cycle penalty when 8-bit and 16-bit values write L2 memory. Disable ECC protection for individual SRAM banks by setting the `L2CTL_CTL.BK0EDIS` through `L2CTL_CTL.BK7EDIS` disable bits. Due to caching mechanisms of the processor cores and data bursting of the DMA channels, 8-bit and 16-bit write accesses are rather uncommon. Typically, only two-dimensional DMA operations or uncached 8-bit and 16-bit store instruction can trigger these writes.

For system integrity testing, the L2 system memory also provides a method for accessing the ECC checksum area directly. The `L2CTL_CTL.ECCMAP0` through `L2CTL_CTL.ECCMAP7` bits map the ECC checksum values into the address space of the data bits. This feature can be activated per SRAM bank. In this mode, only 32-bit accesses are allowed. 32-bit reads return the checksum value in the lower 7 bits while the upper bits read zero. Any 32-bit write overwrites the checksum. The upper bits are ignored.

Using this checksum mapping feature, safety critical applications can verify the ECC hardware during boot up sequence or even at run time. It is not required to set the `L2CTL_CTL.BK0EDIS` through `L2CTL_CTL.BK7EDIS` disable bits explicitly. To test the ECC hardware, use the following steps:

1. Write data values to L2 SRAM destination (preferable an even number of 32-bit words).
2. If data cache enabled, make sure that it flushes out data.

3. Execute SSYNC instruction.
4. Set L2CTL_CTL.ECCMAP7-L2CTL_CTL.ECCMAP0 bits of interest.
5. Execute SSYNC instruction.
6. Write checksum values using 32-bit store instructions.
7. If data cache enabled, make sure that it flushes out checksum values.
8. Execute SSYNC instruction.
9. Clear L2CTL_CTL.ECCMAP7-L2CTL_CTL.ECCMAP0 bits.
10. Execute SSYNC instruction.
11. Read data values back.

ECC Error Management

The L2 system memory flags 2-bit and multi-bit errors to the system by:

- Raising the ECC_ERR interrupt
- Reporting a read error to the system bus
- Setting the sticky L2CTL_STAT.ECCERR7-L2CTL_STAT.ECCERR0 status flag
- Latching the address of the failing operation into the respective L2CTL_ERRADDR7-L2CTL_ERRADDR0 register.

There is one error status bit and one error address register per L2 SRAM bank.

Typically, ECC_ERR events are declared as system faults in the system event controller (SEC). Whether these faults are reported, the interrupt service routine can consult the L2CTL_STAT register and the L2CTL_ERRADDR0 through L2CTL_ERRADDR7 registers to determine whether:

- The data at the failing L2 address was critical enough to require an immediate reboot of the system
- The data at the failing L2 address was less critical or can be restored

The L2CTL_STAT.ECCERR0 through L2CTL_STAT.ECCERR7 flags are cleared with a W1C operation.

Memory Refresh

If data in L2 SRAM contains single-bit errors, the data is corrected on its way to the system buses. The corrected value is not written back to the SRAM location. To prevent any risk of accumulation of single-bit errors over time and to minimize likelihood of multi-bit errors, the L2 system memory provides a special memory refresh mechanism.

If there are dual-bit or multi-bit errors, the ECC_ERR interrupt is raised, and data is not written back to memory.

Program the memory refresh or scrub using:

- TheL2CTL_SADR register with the start address of the scrub
- The L2CTL_SCNT register with the total number of 64-bit addresses to be scrubbed starting from the L2CTL_SADR address

Program the number of cycles between each scrub using the L2CTL_SCTL register. During the scrub, the L2 system memory issues a read, followed by a write-back operation if there is a single-bit ECC error. Once the L2 system memory completes scrubbing the programmed memory region, it generates an interrupt and starts again from the L2CTL_SADR address unless the L2CTL_SCTL.SEN bit is disabled. If the L2CTL_SCTL.SEN bit is cleared before completing the programmed address range, the scrub stops after completing any already issued scrub access. The scrub read/writes always start from the full 64-bit equivalent of the address written into the L2CTL_SADR register. A 64-bit value is always read, and the 64-bit value is written back. The scrub access has the highest priority. Programs can configure 8-bit, 16-bit, or 32-bit addresses in the L2CTL_SADR register. The lower 3 bits of this register are treated as *do-not-care* values because the internal memory array is always accessed when using 64 bits.

Memory refresh operation is meaningless when the L2CTL_CTL.BK0EDIS through L2CTL_CTL.BK7EDIS disable bits are set.

Access Control

The L2 system memory provides write protection, which prevents unauthorized data sources from (over) writing individual SRAM banks. By default, write protection is disabled, permitting the processor cores and DMA controller write-access to every SRAM bank. Using L2 system memory features, programs can selectively disable write privileges for the processor core or the DMA controller to any number of L2 SRAM banks.

- The L2CTL_ACTL_C0.BK0WDIS - L2CTL_ACTL_C0.BK7WDIS bits disable core 0 write privileges to L2 SRAM banks 0–7.
- The L2CTL_ACTL_SYS.BK0WDIS - L2CTL_ACTL_SYS.BK7WDIS bits disable DMA controller write privileges to L2 SRAM banks 0–7.

When an unauthorized write is detected, the L2 system memory generates an error response to the system buses. At any time, any master can read all SRAM locations, because read privileges are not controlled.

Another level of protection is available with the LOCK bit in the following registers:

- L2CTL_CTL
- L2CTL_ACTL_C0
- L2CTL_ACTL_C1
- L2CTL_ACTL_SYS

When this bit is set and global locking is enabled with the system protection unit (SPU), the control registers become write-protected (locked).

Systems can ensure that multiple steps are required before any data source can accidentally overwrite protected data by using the combination of:

- The write disable bits for banks, and
- The lock bits for control registers

L2 System Memory Event Control

The following sections describe event control features of the L2 system memory, such as error response.

ECC Error Interrupt

A bus error is signaled under any of the following conditions.

- A write access to ROM address space
- A read/write access to reserved address space
- A write access (including a TESTSET) to a restricted memory bank
- An ECC multi-bit error in an ECC-enabled bank. A non-modulo, 32-bit write to an ECC-enabled bank can also potentially create a bus error response due to an ECC multi-bit error. This response is because the L2 system memory implements a 32-bit ECC, and therefore a non-modulo, 32-bit write results in a read. This read can create multi-bit errors even if the memory was initialized.

Bus error notifications are stored in the L2CTL_STAT register, and the addresses that generated the error on a given port are stored in the L2CTL_EADDR0/L2CTL_EADDR1 register of that port. The details of the error are stored in the L2CTL_ET0/L2CTL_ET1 register of the port.

Scrub/Init Interrupt

This interrupt is set when the scrub or initialization is complete and is cleared by writing to the corresponding W1C status bits (L2CTL_STAT.SCRBDN/L2CTL_STAT.INITDN).

ADSP-BF70x L2CTL Register Descriptions

L2 Memory Controller (L2CTL) contains the following registers.

Table 11-9: ADSP-BF70x L2CTL Register List

Name	Description
L2CTL_ACTL_C0	Access Control Core 0 Register
L2CTL_ACTL_SYS	Access Control System Register
L2CTL_CTL	Control Register
L2CTL_EADDR0	Error Type 0 Address Register
L2CTL_EADDR1	Error Type 1 Address Register
L2CTL_ERRADDR0	ECC Error Address 0 Register

Table 11-9: ADSP-BF70x L2CTL Register List (Continued)

Name	Description
L2CTL_ERRADDR1	ECC Error Address 1 Register
L2CTL_ERRADDR2	ECC Error Address 2 Register
L2CTL_ERRADDR3	ECC Error Address 3 Register
L2CTL_ERRADDR4	ECC Error Address 4 Register
L2CTL_ERRADDR5	ECC Error Address 5 Register
L2CTL_ERRADDR6	ECC Error Address 6 Register
L2CTL_ERRADDR7	ECC Error Address 7 Register
L2CTL_ERRADDR8	ECC Error Address 8 Register
L2CTL_ET0	Error Type 0 Register
L2CTL_ET1	Error Type 1 Register
L2CTL_INIT	Initialization Register
L2CTL_ISTAT	Initialization Status Register
L2CTL_PCTL	Power Control Register
L2CTL_REVID	Revision ID Register
L2CTL_RPCR	Read Priority Count Register
L2CTL_SADR	Scrub Start Address Register
L2CTL_SCNT	Scrub Count Register
L2CTL_SCTL	Scrub Control Register
L2CTL_STAT	Status Register
L2CTL_WPCR	Write Priority Count Register

Access Control Core 0 Register

The L2CTL_ACTL_C0 register includes a write protection bit and enables core 0 write access for L2 banks.

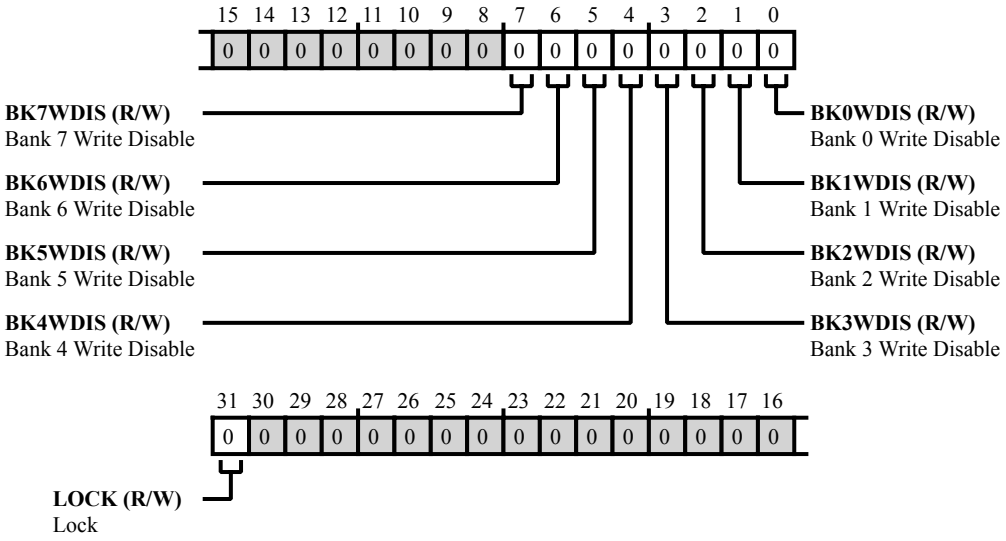

Figure 11-4: L2CTL_ACTL_C0 Register Diagram

Table 11-10: L2CTL_ACTL_C0 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 L2CTL_ACTL_C0.LOCK bit is set, the L2CTL_ACTL_C0 register is read only (locked).	
		0	Unlock
		1	Lock
7 (R/W)	BK7WDIS	Bank 7 Write Disable. The L2CTL_ACTL_C0.BK7WDIS bit disables core 0 writes to L2 bank 7 RAM.	
		0	Enable Write
		1	Disable Write
6 (R/W)	BK6WDIS	Bank 6 Write Disable. The L2CTL_ACTL_C0.BK6WDIS bit disables core 0 writes to L2 bank 6 RAM.	
		0	Enable Write
		1	Disable Write

Table 11-10: L2CTL_ACTL_C0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	BK5WDIS	Bank 5 Write Disable. The `L2CTL_ACTL_C0.BK5WDIS` bit disables core 0 writes to L2 bank 5 RAM.	
		0	Enable Write
		1	Disable Write
4 (R/W)	BK4WDIS	Bank 4 Write Disable. The `L2CTL_ACTL_C0.BK4WDIS` bit disables core 0 writes to L2 bank 4 RAM.	
		0	Enable Write
		1	Disable Write
3 (R/W)	BK3WDIS	Bank 3 Write Disable. The `L2CTL_ACTL_C0.BK3WDIS` bit disables core 0 writes to L2 bank 3 RAM.	
		0	Enable Write
		1	Disable Write
2 (R/W)	BK2WDIS	Bank 2 Write Disable. The `L2CTL_ACTL_C0.BK2WDIS` bit disables core 0 writes to L2 bank 2 RAM.	
		0	Enable Write
		1	Disable Write
1 (R/W)	BK1WDIS	Bank 1 Write Disable. The `L2CTL_ACTL_C0.BK1WDIS` bit disables core 0 writes to L2 bank 1 RAM.	
		0	Enable Write
		1	Disable Write
0 (R/W)	BK0WDIS	Bank 0 Write Disable. The `L2CTL_ACTL_C0.BK0WDIS` bit disables core 0 writes to L2 bank 0 RAM.	
		0	Enable Write
		1	Disable Write

Access Control System Register

The L2CTL_ACTL_SYS register includes a write protection bit and enables system and DMA write access for L2 banks.

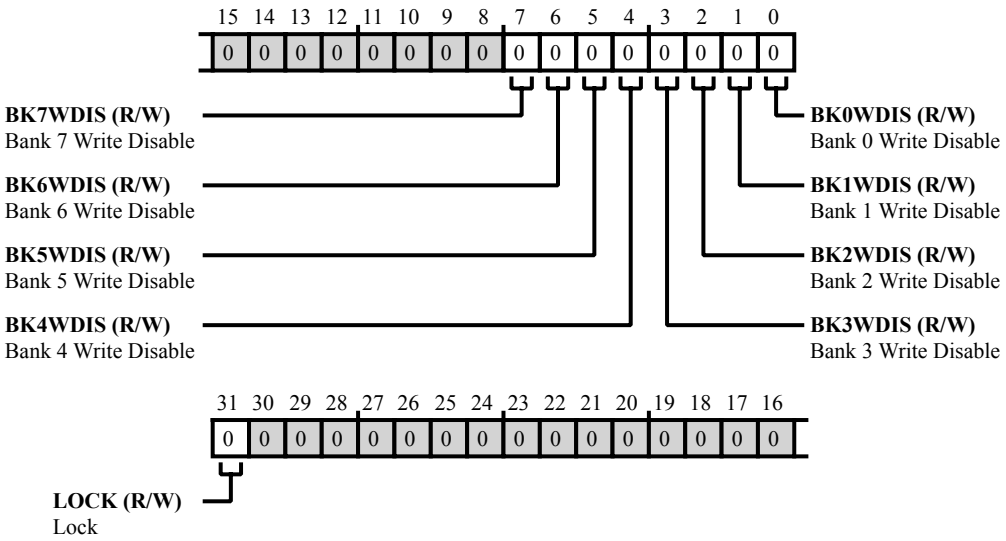

Figure 11-5: L2CTL_ACTL_SYS Register Diagram

Table 11-11: L2CTL_ACTL_SYS 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 L2CTL_ACTL_SYS.LOCK bit is set, the L2CTL_ACTL_SYS register is read only (locked).	
		0	Unlock
		1	Lock
7 (R/W)	BK7WDIS	Bank 7 Write Disable. The L2CTL_ACTL_SYS.BK7WDIS bit disables system or DMA writes to L2 bank 7 RAM.	
		0	Enable Write
		1	Disable Write
6 (R/W)	BK6WDIS	Bank 6 Write Disable. The L2CTL_ACTL_SYS.BK6WDIS bit disables system or DMA writes to L2 bank 6 RAM.	
		0	Enable Write
		1	Disable Write

Table 11-11: L2CTL_ACTL_SYS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	BK5WDIS	Bank 5 Write Disable. The `L2CTL_ACTL_SYS.BK5WDIS` bit disables system or DMA writes to L2 bank 5 RAM.	
		0	Enable Write
		1	Disable Write
4 (R/W)	BK4WDIS	Bank 4 Write Disable. The `L2CTL_ACTL_SYS.BK4WDIS` bit disables system or DMA writes to L2 bank 4 RAM.	
		0	Enable Write
		1	Disable Write
3 (R/W)	BK3WDIS	Bank 3 Write Disable. The `L2CTL_ACTL_SYS.BK3WDIS` bit disables system or DMA writes to L2 bank 3 RAM.	
		0	Enable Write
		1	Disable Write
2 (R/W)	BK2WDIS	Bank 2 Write Disable. The `L2CTL_ACTL_SYS.BK2WDIS` bit disables system or DMA writes to L2 bank 2 RAM.	
		0	Enable Write
		1	Disable Write
1 (R/W)	BK1WDIS	Bank 1 Write Disable. The `L2CTL_ACTL_SYS.BK1WDIS` bit disables system or DMA writes to L2 bank 1 RAM.	
		0	Enable Write
		1	Disable Write
0 (R/W)	BK0WDIS	Bank 0 Write Disable. The `L2CTL_ACTL_SYS.BK0WDIS` bit disables system or DMA writes to L2 bank 0 RAM.	
		0	Enable Write
		1	Disable Write

Control Register

The L2CTL_CTL register includes a write protection bit, enables L2 banks, and selects mapping of banks (as ECC RAM or data RAM).

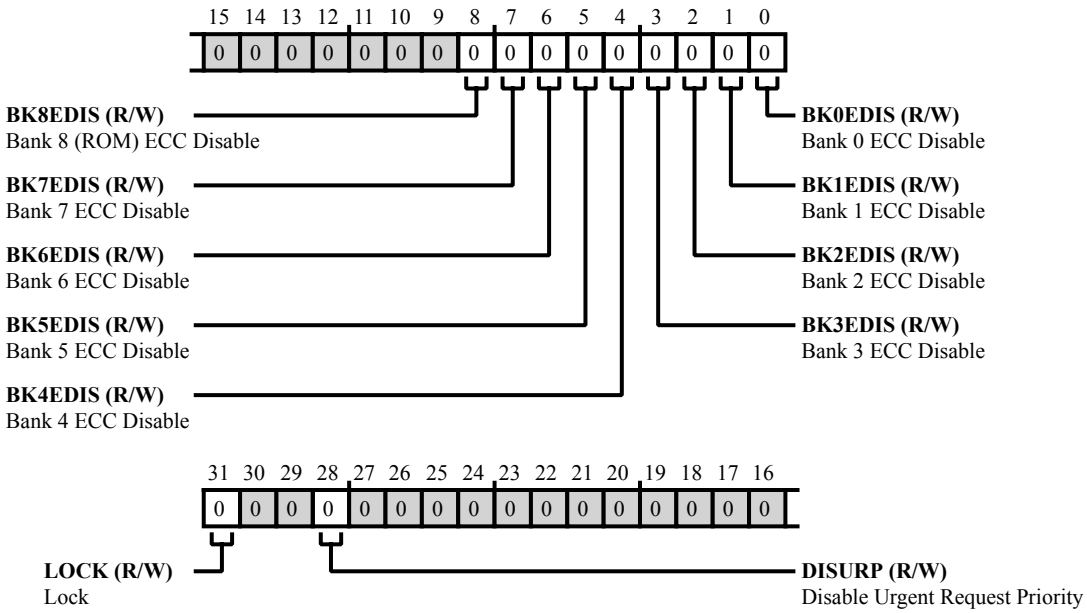

Figure 11-6: L2CTL_CTL Register Diagram

Table 11-12: L2CTL_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 L2CTL_CTL.LOCK bit is set, the L2CTL_CTL register is read only (locked).	
		0	Unlock
		1	Lock
28 (R/W)	DISURP	Disable Urgent Request Priority. The L2CTL_CTL.DISURP disables urgent request priority mode for all L2 banks.	
		0	Enable URP
		1	Disable URP
8 (R/W)	BK8EDIS	Bank 8 (ROM) ECC Disable. The L2CTL_CTL.BK8EDIS bit disables L2 bank 8 (ROM) ECC operation.	
		0	Enable ECC
		1	Disable ECC

Table 11-12: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	BK7EDIS	Bank 7 ECC Disable. The L2CTL_CTL.BK7EDIS bit disables L2 bank 7 ECC operation.	
		0	Enable ECC
		1	Disable ECC
6 (R/W)	BK6EDIS	Bank 6 ECC Disable. The L2CTL_CTL.BK6EDIS bit disables L2 bank 6 ECC operation.	
		0	Enable ECC
		1	Disable ECC
5 (R/W)	BK5EDIS	Bank 5 ECC Disable. The L2CTL_CTL.BK5EDIS bit disables L2 bank 5 ECC operation.	
		0	Enable ECC
		1	Disable ECC
4 (R/W)	BK4EDIS	Bank 4 ECC Disable. The L2CTL_CTL.BK4EDIS bit disables L2 bank 4 ECC operation.	
		0	Enable ECC
		1	Disable ECC
3 (R/W)	BK3EDIS	Bank 3 ECC Disable. The L2CTL_CTL.BK3EDIS bit disables L2 bank 3 ECC operation.	
		0	Enable ECC
		1	Disable ECC
2 (R/W)	BK2EDIS	Bank 2 ECC Disable. The L2CTL_CTL.BK2EDIS bit disables L2 bank 2 ECC operation.	
		0	Enable ECC
		1	Disable ECC
1 (R/W)	BK1EDIS	Bank 1 ECC Disable. The L2CTL_CTL.BK1EDIS bit disables L2 bank 1 ECC operation.	
		0	Enable ECC
		1	Disable ECC

Table 11-12: L2CTL_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	BK0EDIS	Bank 0 ECC Disable. The `L2CTL_CTL.BK0EDIS` bit disables L2 bank 0 ECC operation.	
		0	Enable ECC
		1	Disable ECC

Error Type 0 Address Register

The L2CTL_EADDR0 register holds the address that created an access error on the L2 port 0 bus interface. This register is be updated only if the corresponding error status bit (L2CTL_STAT.ERR0) is cleared. After the status bit is set for an error, further errors do not update the L2CTL_EADDR0 register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the register captures the write access error address.

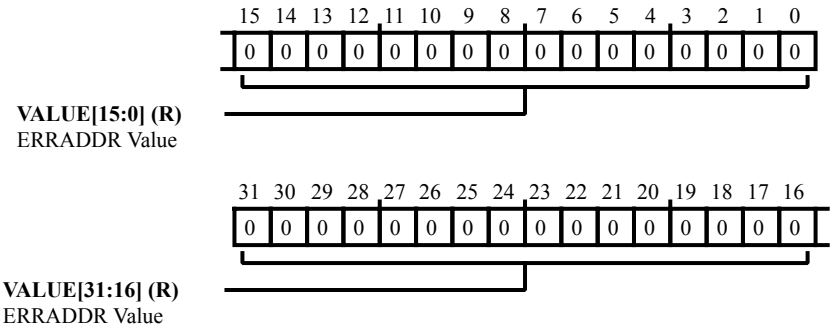

Figure 11-7: L2CTL_EADDR0 Register Diagram

Table 11-13: L2CTL_EADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_EADDR0.VALUE bits hold the address causing the bus error.

Error Type 1 Address Register

The L2CTL_EADDR1 register holds the address that created an access error on the L2 port 0 bus interface. This register is be updated only if the corresponding error status bit (L2CTL_STAT.ERR1) is cleared. After the status bit is set for an error, further errors do not update the L2CTL_EADDR1 register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the register captures the write access error address.

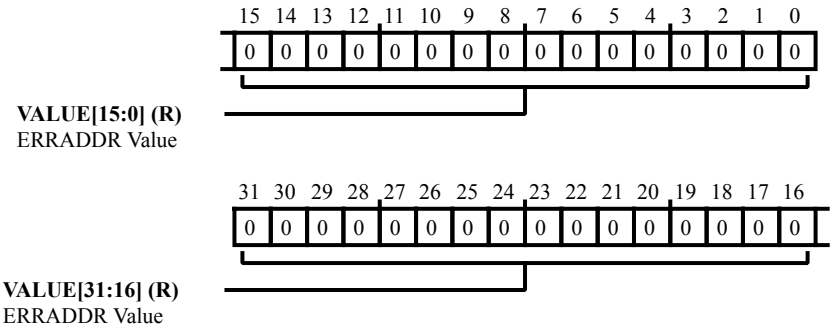

Figure 11-8: L2CTL_EADDR1 Register Diagram

Table 11-14: L2CTL_EADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_EADDR1.VALUE bits hold the address causing the bus error.

ECC Error Address 0 Register

The L2CTL_ERRADDR0 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR0) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

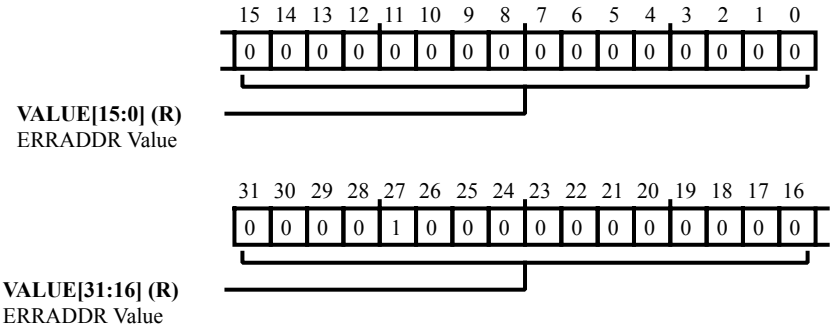

Figure 11-9: L2CTL_ERRADDR0 Register Diagram

Table 11-15: L2CTL_ERRADDR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR0.VALUE bits hold the address containing the ECC double-bit error.

ECC Error Address 1 Register

The L2CTL_ERRADDR1 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR1) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

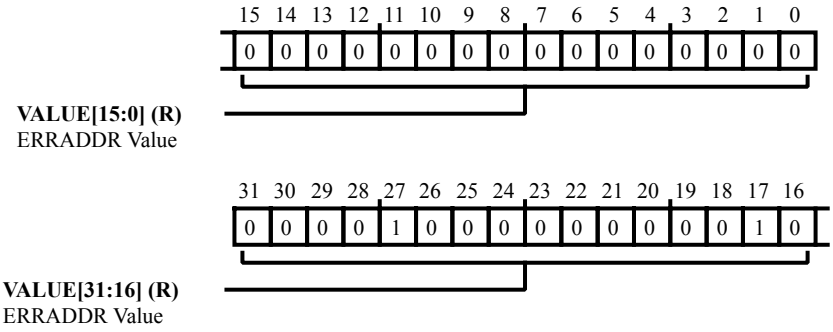

Figure 11-10: L2CTL_ERRADDR1 Register Diagram

Table 11-16: L2CTL_ERRADDR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR1.VALUE bits hold the address containing the ECC double-bit error.

ECC Error Address 2 Register

The L2CTL_ERRADDR2 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR2) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

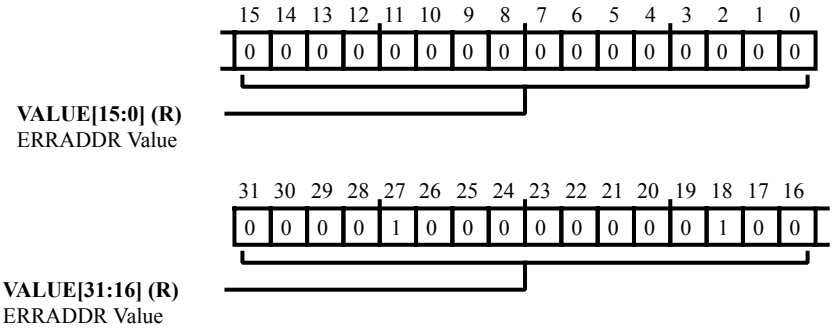

Figure 11-11: L2CTL_ERRADDR2 Register Diagram

Table 11-17: L2CTL_ERRADDR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR2.VALUE bits hold the address containing the ECC double-bit error.

ECC Error Address 3 Register

The L2CTL_ERRADDR3 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR3) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

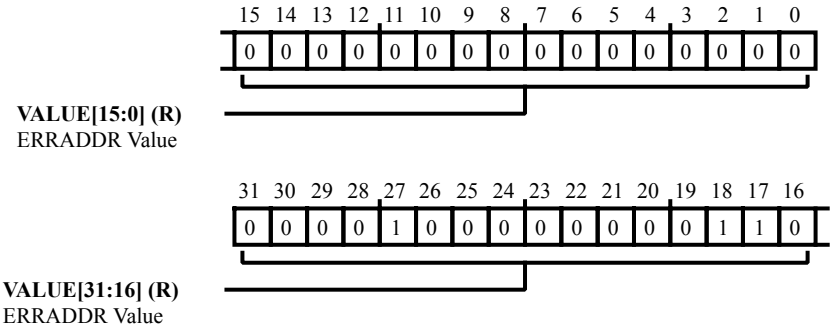

Figure 11-12: L2CTL_ERRADDR3 Register Diagram

Table 11-18: L2CTL_ERRADDR3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR3.VALUE bits hold the address containing the ECC double-bit error.

ECC Error Address 4 Register

The L2CTL_ERRADDR4 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR4) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

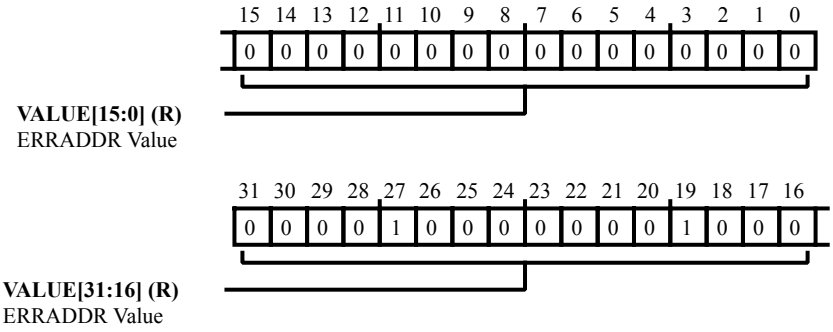

Figure 11-13: L2CTL_ERRADDR4 Register Diagram

Table 11-19: L2CTL_ERRADDR4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR4.VALUE bits hold the address containing the ECC double-bit error.

ECC Error Address 5 Register

The L2CTL_ERRADDR5 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR5) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

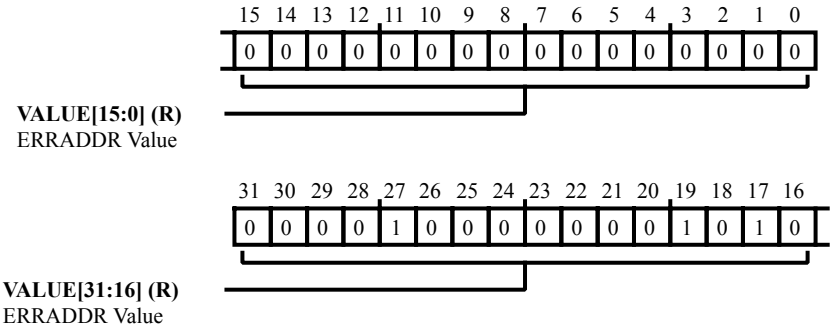

Figure 11-14: L2CTL_ERRADDR5 Register Diagram

Table 11-20: L2CTL_ERRADDR5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR5.VALUE bits hold the address containing the ECC double-bit error.

ECC Error Address 6 Register

The L2CTL_ERRADDR6 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR6) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

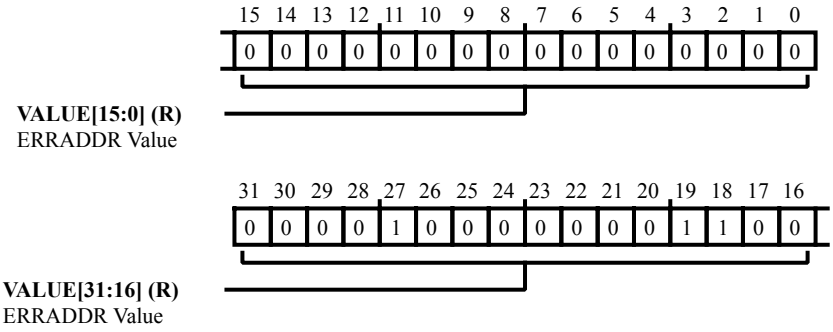

Figure 11-15: L2CTL_ERRADDR6 Register Diagram

Table 11-21: L2CTL_ERRADDR6 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR6.VALUE bits hold the address containing the ECC double-bit error.

ECC Error Address 7 Register

The L2CTL_ERRADDR7 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR7) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

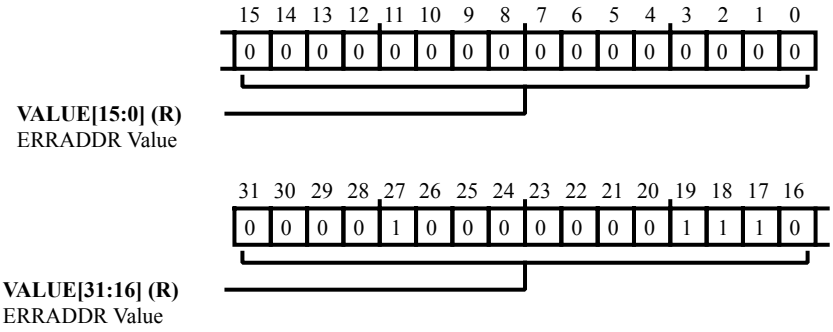

Figure 11-16: L2CTL_ERRADDR7 Register Diagram

Table 11-22: L2CTL_ERRADDR7 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR7.VALUE bits hold the address containing the ECC double-bit error.

ECC Error Address 8 Register

The L2CTL_ERRADDR8 register holds the address containing an ECC multi-bit error for the corresponding bank. The L2CTL updates this register only if the bank's status bit (L2CTL_STAT.ECCERR8) is cleared. After the bank's status bit is set for an error, further errors in the same bank are not detected until a W1C clears the status bit.

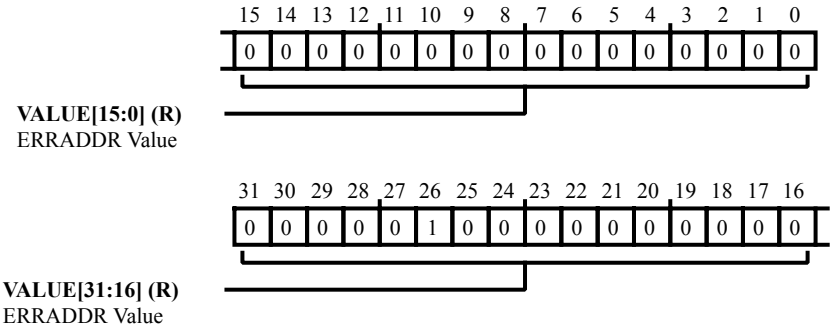

Figure 11-17: L2CTL_ERRADDR8 Register Diagram

Table 11-23: L2CTL_ERRADDR8 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	ERRADDR Value. The L2CTL_ERRADDR8.VALUE bits hold the address containing the ECC double-bit error.

Error Type 0 Register

The L2CTL_ET0 register holds information about the error transaction that has occurred on the bus for the corresponding L2 bus port. This register is updated only if the corresponding error status bit L2CTL_STAT.ERR0 is cleared. After the status bit is set for an error, further errors do not update the L2CTL_ET0 register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the L2CTL_ET0 captures the write access error, keeping in sync with the error address register (L2CTL_EADDR0).

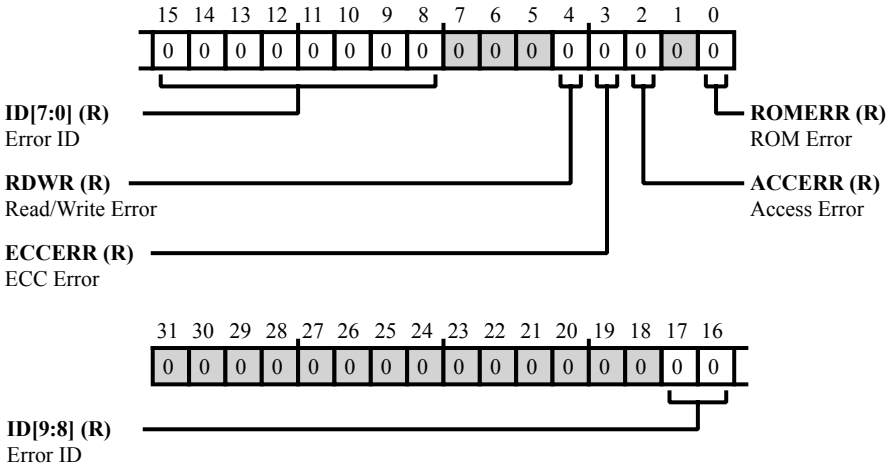

Figure 11-18: L2CTL_ET0 Register Diagram

Table 11-24: L2CTL_ET0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17:8 (R/NW)	ID	Error ID. The L2CTL_ET0.ID bits hold the bus master ID of the access that caused an error.	
4 (R/NW)	RDWR	Read/Write Error. The L2CTL_ET0.RDWR bit indicates whether a read or write access caused an error.	
		0	Read Access created Error
		1	Write Access created Error
3 (R/NW)	ECCERR	ECC Error. The L2CTL_ET0.ECCERR bit indicates whether the access had an ECC double-bit error.	
2 (R/NW)	ACCERR	Access Error. The L2CTL_ET0.ACCERR bit indicates whether the access went to a restricted bank.	
0 (R/NW)	ROMERR	ROM Error. The L2CTL_ET0.ROMERR bit indicates whether a write access went to a ROM area.	

Error Type 1 Register

The L2CTL_ET1 register holds information about the error transaction that has occurred on the bus for the corresponding L2 bus port. This register is updated only if the corresponding error status bit L2CTL_STAT.ERR1 is cleared. After the status bit is set for an error, further errors do not update the L2CTL_ET1 register until a W1C clears the corresponding status bit. If read and write access errors occur simultaneously, the L2CTL_ET1 captures the write access error, keeping in sync with the error address register (L2CTL_EADDR1).

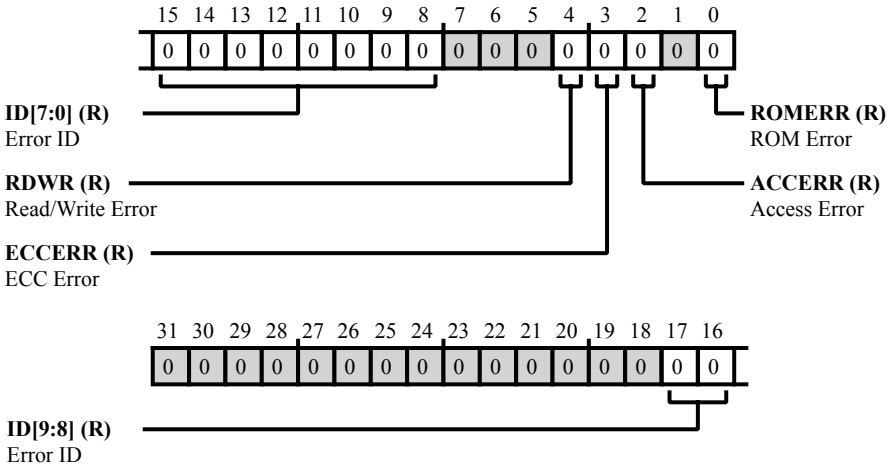

Figure 11-19: L2CTL_ET1 Register Diagram

Table 11-25: L2CTL_ET1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17:8 (R/NW)	ID	Error ID. The L2CTL_ET1.ID bits hold the bus master ID of the access that caused an error.	
4 (R/NW)	RDWR	Read/Write Error. The L2CTL_ET1.RDWR bit indicates whether a read or write access caused an error.	
		0	Read access created error
		1	Write access created error
3 (R/NW)	ECCERR	ECC Error. If the L2CTL_ET1.ECCERR bit =1, the access had an ECC double-bit error.	
2 (R/NW)	ACCERR	Access Error. If the L2CTL_ET1.ACCERR bit =1, the access went to a restricted bank.	
0 (R/NW)	ROMERR	ROM Error. If the L2CTL_ET1.ROMERR bit =1, a write access went to a ROM area.	

Initialization Register

The L2CTL_INIT register initializes memory banks with 64'b0 and ECC bits corresponding to 64'b0. Any writes to the bits in this register while initialization is occurring to any of the banks is ignored.

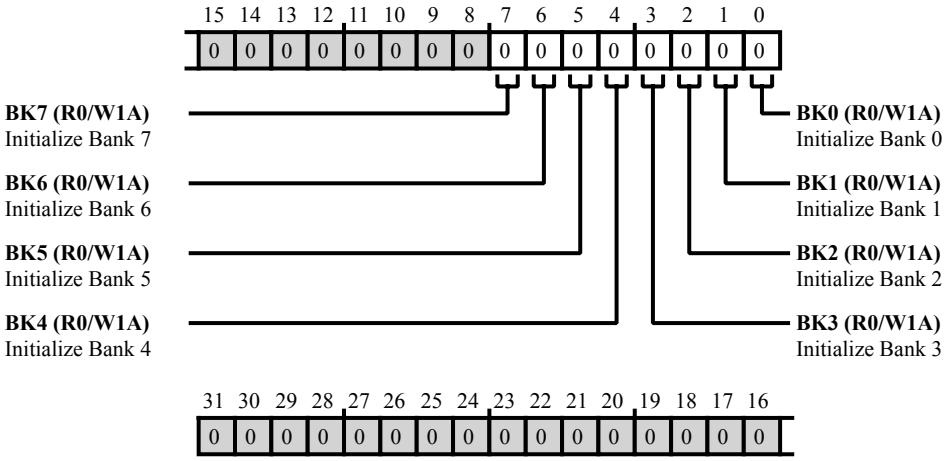

Figure 11-20: L2CTL_INIT Register Diagram

Table 11-26: L2CTL_INIT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R0/W1A)	BK7	Initialize Bank 7. The Write-1 to the L2CTL_INIT.BK7 bit will initialize bank 7 with 64b0 and ECC bits corresponding to 64b0. Any writes to this bit while initialization is happening to any of the banks are ignored.
6 (R0/W1A)	BK6	Initialize Bank 6. The Write-1 to the L2CTL_INIT.BK6 bit will initialize bank 6 with 64b0 and ECC bits corresponding to 64b0. Any writes to this bit while initialization is happening to any of the banks are ignored.
5 (R0/W1A)	BK5	Initialize Bank 5. The Write-1 to the L2CTL_INIT.BK5 bit will initialize bank 5 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit while initialization is happening to any of the banks is ignored.
4 (R0/W1A)	BK4	Initialize Bank 4. The Write-1 to the L2CTL_INIT.BK4 bit will initialize bank 4 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit while initialization is happening to any of the banks is ignored.
3 (R0/W1A)	BK3	Initialize Bank 3.

Table 11-26: L2CTL_INIT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The Write-1 to the L2CTL_INIT.BK3 bit will initialize bank 3 with 64b0 and ECC bits corresponding to 64b0. Any writes to this bit while initialization is happening to any of the banks are ignored.
2 (R0/W1A)	BK2	Initialize Bank 2. The Write-1 to the L2CTL_INIT.BK2 bit will initialize bank 2 with 64b0 and ECC bits corresponding to 64b0. Any writes to this bit while initialization is happening to any of the banks are ignored.
1 (R0/W1A)	BK1	Initialize Bank 1. The Write-1 to the L2CTL_INIT.BK1 bit will initialize bank 1 with 64b0 and ECC bits corresponding to 64b0. Any write to this bit while initialization is happening to any of the banks is ignored.
0 (R0/W1A)	BK0	Initialize Bank 0. The Write-1 to the L2CTL_INIT.BK0 bit will initialize bank 0 with 64b0 and ECC bits corresponding to 64b0. Any writes to this bit while initialization is happening to any of the banks are ignored.

Initialization Status Register

The L2CTL_ISTAT register holds the status of the RAM bank initialization. If set, the corresponding bank is initialized.

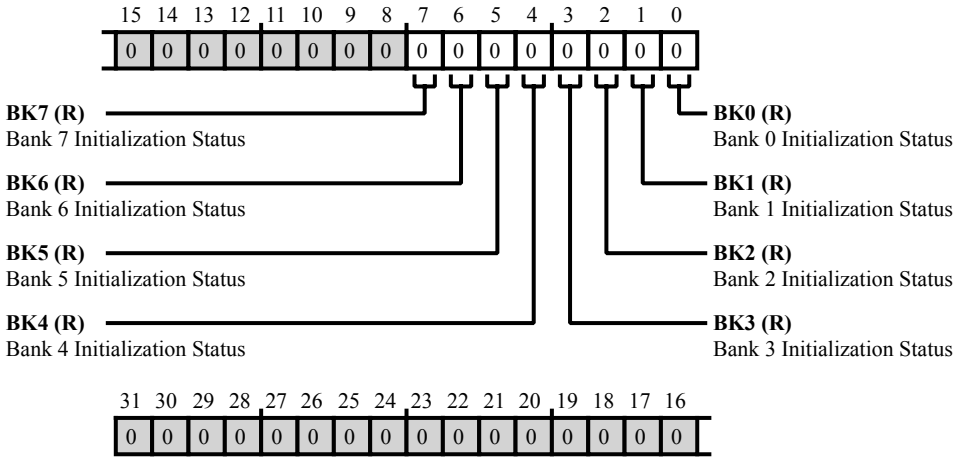

Figure 11-21: L2CTL_ISTAT Register Diagram

Table 11-27: L2CTL_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	BK7	Bank 7 Initialization Status. The L2CTL_ISTAT.BK7 bits hold the bank 7 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.
6 (R/NW)	BK6	Bank 6 Initialization Status. The L2CTL_ISTAT.BK6 bits hold the bank 6 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.
5 (R/NW)	BK5	Bank 5 Initialization Status. The L2CTL_ISTAT.BK5 bits hold the bank 5 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.
4 (R/NW)	BK4	Bank 4 Initialization Status. The L2CTL_ISTAT.BK4 bits hold the bank 4 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.
3 (R/NW)	BK3	Bank 3 Initialization Status. The L2CTL_ISTAT.BK3 bits hold the bank 3 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.
2 (R/NW)	BK2	Bank 2 Initialization Status. The L2CTL_ISTAT.BK2 bits hold the bank 2 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.

Table 11-27: L2CTL_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/NW)	BK1	Bank 1 Initialization Status. The `L2CTL_ISTAT.BK1` bits hold the bank 1 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.
0 (R/NW)	BK0	Bank 0 Initialization Status. The `L2CTL_ISTAT.BK0` bits hold the bank 0 initialization status. A W1A on a BKxINIT bit clears this bit and the bit is set when initialization completes.

Power Control Register

The L2CTL_PCTL register has the various control settings for selectively enabling the deep sleep and shut down power saving features.

NOTE: The corresponding L2 bank should not be accessed if the power control feature is enabled for that bank. An access to a bank in deep sleep/shut down may create unpredictable behavior.

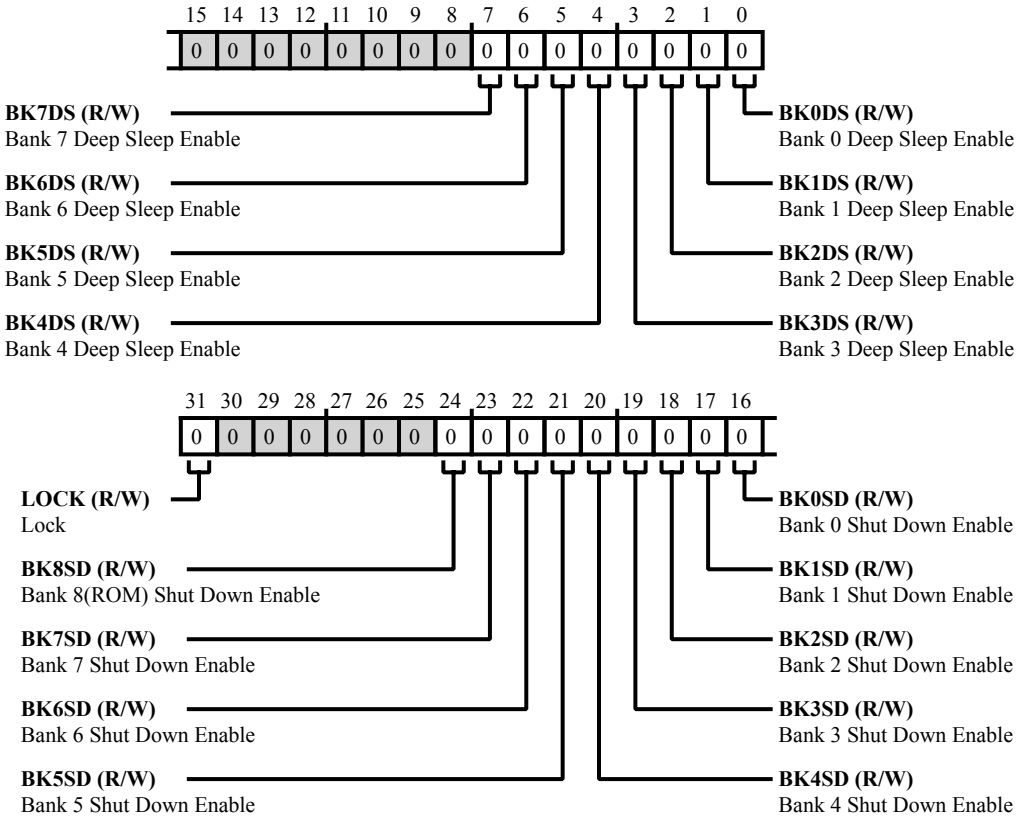

Figure 11-22: L2CTL_PCTL Register Diagram

Table 11-28: L2CTL_PCTL 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 L2CTL_PCTL.LOCK bit is set, the L2CTL_PCTL register is read only (locked).	
		0	Unlock
		1	Lock
24 (R/W)	BK8SD	Bank 8(ROM) Shut Down Enable. The L2CTL_PCTL.BK8SD bits enables bank 8 (ROM) shut down.	

Table 11-28: L2CTL_PCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
23 (R/W)	BK7SD	Bank 7 Shut Down Enable. The L2CTL_PCTL.BK7SD bits enables bank 7 shut down.
22 (R/W)	BK6SD	Bank 6 Shut Down Enable. The L2CTL_PCTL.BK6SD bits enables bank 6 shut down.
21 (R/W)	BK5SD	Bank 5 Shut Down Enable. The L2CTL_PCTL.BK5SD bits enables bank 5 shut down.
20 (R/W)	BK4SD	Bank 4 Shut Down Enable. The L2CTL_PCTL.BK4SD bits enables bank 4 shut down.
19 (R/W)	BK3SD	Bank 3 Shut Down Enable. The L2CTL_PCTL.BK3SD bits enables bank 3 shut down.
18 (R/W)	BK2SD	Bank 2 Shut Down Enable. The L2CTL_PCTL.BK2SD bits enables bank 2 shut down.
17 (R/W)	BK1SD	Bank 1 Shut Down Enable. The L2CTL_PCTL.BK1SD bits enables bank 1 shut down.
16 (R/W)	BK0SD	Bank 0 Shut Down Enable. The L2CTL_PCTL.BK0SD bits enables bank 0 shut down.
7 (R/W)	BK7DS	Bank 7 Deep Sleep Enable. The L2CTL_PCTL.BK7DS bits enables bank 7 deep sleep.
6 (R/W)	BK6DS	Bank 6 Deep Sleep Enable. The L2CTL_PCTL.BK6DS bits enables bank 6 deep sleep.
5 (R/W)	BK5DS	Bank 5 Deep Sleep Enable. The L2CTL_PCTL.BK5DS bits enables bank 5 deep sleep.
4 (R/W)	BK4DS	Bank 4 Deep Sleep Enable. The L2CTL_PCTL.BK4DS bits enables bank 4 deep sleep.
3 (R/W)	BK3DS	Bank 3 Deep Sleep Enable. The L2CTL_PCTL.BK3DS bits enables bank 3 deep sleep.
2 (R/W)	BK2DS	Bank 2 Deep Sleep Enable. The L2CTL_PCTL.BK2DS bits enables bank 2 deep sleep.
1 (R/W)	BK1DS	Bank 1 Deep Sleep Enable. The L2CTL_PCTL.BK1DS bits enables bank 1 deep sleep.
0 (R/W)	BK0DS	Bank 0 Deep Sleep Enable. The L2CTL_PCTL.BK0DS bits enables bank 0 deep sleep.

Revision ID Register

The L2CTL_REVID register provides the L2 Revision ID.

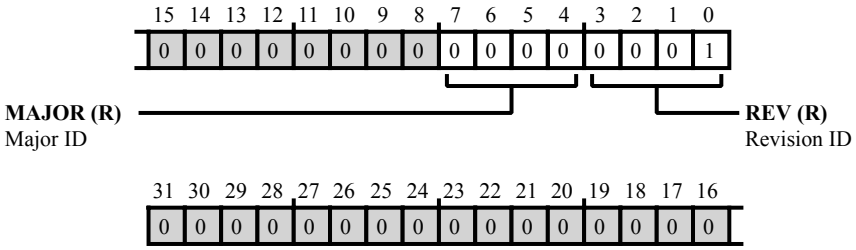

Figure 11-23: L2CTL_REVID Register Diagram

Table 11-29: L2CTL_REVID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	MAJOR	Major ID. The L2CTL_REVID.MAJOR bit indicates L2 Major ID.
3:0 (R/NW)	REV	Revision ID. The L2CTL_REVID.REV bit indicates the L2 Revision ID.

Read Priority Count Register

The L2CTL_RPCR register stores the count value to be used for priority elevation for bus read channels. If a bus channel is not granted access from the bank arbiter, the channel waits for the programmed number of SYSCLK cycles, before the request is elevated to a high priority request. If a priority count value is programmed as zero for a channel, that channel does not raise the urgent priority request.

This is a read/write register, but a new value in the corresponding field must be written only when there are no outstanding transactions on the corresponding bus read channel. A best practice is to program this register before initiating an L2 access.

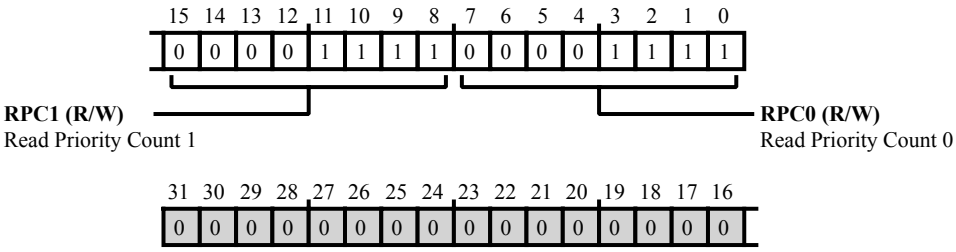

Figure 11-24: L2CTL_RPCR Register Diagram

Table 11-30: L2CTL_RPCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	RPC1	Read Priority Count 1. The L2CTL_RPCR.RPC1 bits hold the priority count for L2 bus read channel 1.
7:0 (R/W)	RPC0	Read Priority Count 0. The L2CTL_RPCR.RPC0 bits hold the priority count for L2 bus read channel 0.

Scrub Start Address Register

The L2CTL_SADR register stores the scrub start address value. Writes to this register can be prevented by setting the L2CTL_SCTL.LOCK bit and the global lock.

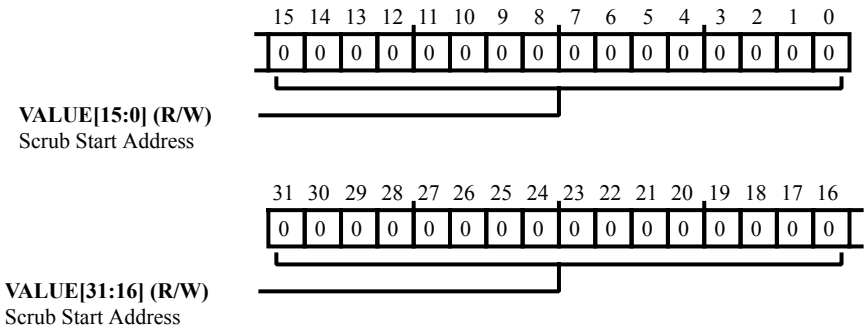

Figure 11-25: L2CTL_SADR Register Diagram

Table 11-31: L2CTL_SADR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Scrub Start Address. The L2CTL_SADR.VALUE bits hold the scrub start address. The writes to this register can be prevented by setting the L2_SCTL.LOCK and the global lock.

Scrub Count Register

The L2CTL_SCNT register determines the number of 64-bit locations scrubbed starting from the start address (L2CTL_SADR register). Writes to the L2CTL_SCNT register can be prevented by setting the L2CTL_CTL.LOCK bit and the global lock.

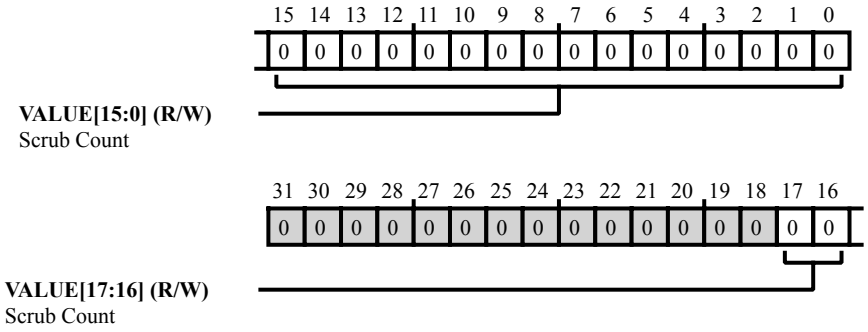

Figure 11-26: L2CTL_SCNT Register Diagram

Table 11-32: L2CTL_SCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
17:0 (R/W)	VALUE	Scrub Count. The L2CTL_SCNT.VALUE bits determines the number of 64-bit locations scrubbed starting from the start address. Ensure value programmed is less than the total addressable 64-bit L2 RAM locations available.

Scrub Control Register

The L2CTL_SCTL register holds the automatic scrub related controls. This is a read/write register. Memory scrub can be performed by programming the L2CTL_SADR (scrub start address) register with the start address of the scrub and the L2CTL_SCNT register with the total number of 64-bit addresses to be scrubbed starting from the L2CTL_SADR register. The number of cycles between each scrub can be programmed with the L2CTL_SCTL.SRT bit.

During the scrub the controller issues a read followed by write back if there is a single bit ECC error. Once the L2 controller completes scrubbing the memory region mentioned using the L2CTL_SADR and the L2CTL_SCNT registers, an interrupt is generated and scrubbing re-starts from the start unless the scrub enable bit is disabled in the control register. If scrub enable is cleared before completing the address range used in the L2CTL_SADR and L2CTL_SCNT registers, the scrub stops after completing any already issued scrub access.

The scrub read/writes always start from the full 64-bit equivalent of the address written into the L2CTL_SADR register. A 64-bit value is always read and the 64-bit value is written back. The scrub access has the highest priority. Programs can configure 8-, 16-, or 32-bit addresses in the L2CTL_SADR register but the lower 3 bits are treated as don't care because the internal memory array is always accessed in a 64-bit mode.

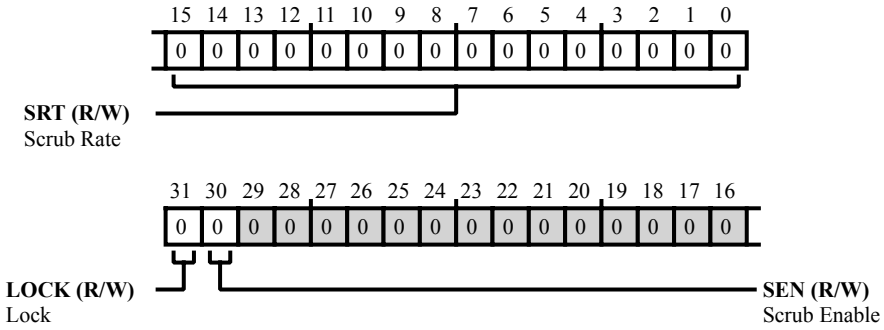

Figure 11-27: L2CTL_SCTL Register Diagram

Table 11-33: L2CTL_SCTL 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 L2CTL_SCTL.LOCK bit is set, the L2CTL_SCTL register is read only (locked).	
		0	Unlock
		1	Lock
30 (R/W)	SEN	Scrub Enable. The L2CTL_SCTL.SEN bits enable automatic scrub.	
15:0	SRT	Scrub Rate.	

Table 11-33: L2CTL_SCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		The `L2CTL_SCTL.SRT` bits determines the number of clock cycles that elapsed between two automatic scrubs.

Status Register

The L2CTL_STAT register indicates ECC error status, refresh register status, and bus error status.

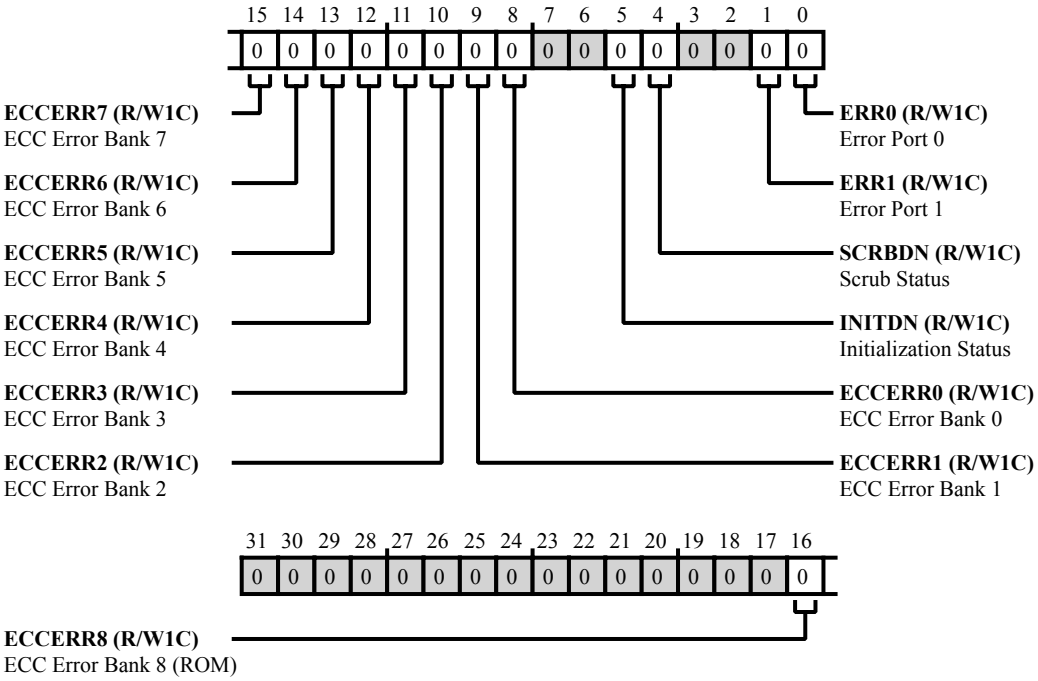

Figure 11-28: L2CTL_STAT Register Diagram

Table 11-34: L2CTL_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	ECCERR8	ECC Error Bank 8 (ROM). The L2CTL_STAT.ECCERR8 bit indicates that an ECC double-bit error occurred inside L2 bank 8 (ROM).	
		0	No Status
		1	ECC Double Bit Error
15 (R/W1C)	ECCERR7	ECC Error Bank 7. The L2CTL_STAT.ECCERR7 bit indicates that an ECC double-bit error occurred inside L2 bank 7.	
		0	No Status
		1	ECC Double Bit Error

Table 11-34: L2CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W1C)	ECCERR6	ECC Error Bank 6. The `L2CTL_STAT.ECCERR6` bit indicates that an ECC double-bit error occurred inside L2 bank 6.	
		0	No Status
		1	ECC Double Bit Error
13 (R/W1C)	ECCERR5	ECC Error Bank 5. The `L2CTL_STAT.ECCERR5` bit indicates that an ECC double-bit error occurred inside L2 bank 5.	
		0	No Status
		1	ECC Double Bit Error
12 (R/W1C)	ECCERR4	ECC Error Bank 4. The `L2CTL_STAT.ECCERR4` bit indicates that an ECC double-bit error occurred inside L2 bank 4.	
		0	No Status
		1	ECC Double Bit Error
11 (R/W1C)	ECCERR3	ECC Error Bank 3. The `L2CTL_STAT.ECCERR3` bit indicates that an ECC double-bit error occurred inside L2 bank 3.	
		0	No Status
		1	ECC Double Bit Error
10 (R/W1C)	ECCERR2	ECC Error Bank 2. The `L2CTL_STAT.ECCERR2` bit indicates that an ECC double-bit error occurred inside L2 bank 2.	
		0	No Status
		1	ECC Double Bit Error
9 (R/W1C)	ECCERR1	ECC Error Bank 1. The `L2CTL_STAT.ECCERR1` bit indicates that an ECC double-bit error occurred inside L2 bank 1.	
		0	No Status
		1	ECC Double Bit Error

Table 11-34: L2CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W1C)	ECCERR0	ECC Error Bank 0. The `L2CTL_STAT.ECCERR0` bit indicates that an ECC double-bit error occurred inside L2 bank 0.	
		0	No Status
		1	ECC Double Bit Error
5 (R/W1C)	INITDN	Initialization Status. The `L2CTL_STAT.INITDN` bit indicates whether initialization has been completed.	
		0	Initialization Not Complete
		1	Initialization Completed
4 (R/W1C)	SCRBDN	Scrub Status. The `L2CTL_STAT.SCRBDN` bit indicates whether a round of memory scrub has completed.	
		0	Scrub Not Complete
		1	Scrub Completed
1 (R/W1C)	ERR1	Error Port 1. The `L2CTL_STAT.ERR1` indicates whether the L2CTL has detected a bus access error on L2s bus port 1.	
		0	No Error
		1	Bus Access Error
0 (R/W1C)	ERR0	Error Port 0. The `L2CTL_STAT.ERR0` indicates whether the L2CTL has detected a bus access error on L2s bus port 0.	
		0	No Error
		1	Bus Access Error

Write Priority Count Register

The L2CTL_WPCR register stores the count value to be used for priority elevation for bus write channels. If a bus channel is not granted access from the bank arbiter, the channel waits for the programmed number of SYSCLK cycles, before the request is elevated to a high priority request. If a priority count value is programmed as zero for a channel, that channel does not raise the urgent priority request.

This is a read/write register, but a new value in the corresponding field must be written only when there are no outstanding transactions on the corresponding bus write channel. A best practice is to program this register before initiating an L2 access.

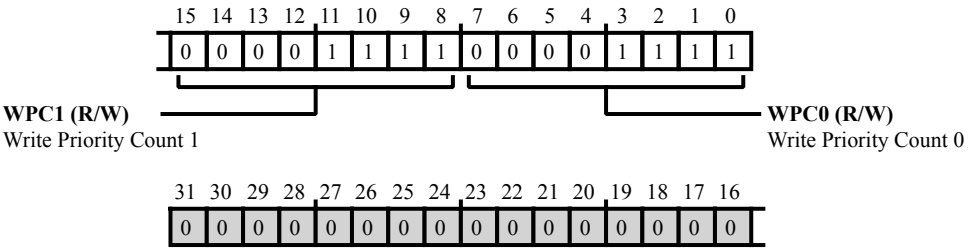

Figure 11-29: L2CTL_WPCR Register Diagram

Table 11-35: L2CTL_WPCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W)	WPC1	Write Priority Count 1. The L2CTL_WPCR.WPC1 bits hold the priority count for L2 bus write channel 0.
7:0 (R/W)	WPC0	Write Priority Count 0. The L2CTL_WPCR.WPC0 bits hold the priority count for L2 bus write channel 1.

ADSP-BF70x Processor-Specific Information

The L2CLK runs at the same rate as SYSCLK.

The L2 system memory has two SCB ports accessing 1024 KB of RAM in eight banks and 512 KB of ROM in a single bank.

12 Dynamic Memory Controller (DMC)

NOTE: The terms DDR2 and LPDDR SDRAM are referred to generically as DDR SDRAM in the rest of this chapter unless otherwise noted.

The DMC is partitioned in a manner that allows reconfiguration and maintainability. The memory access protocol state machine along with JEDEC standard specific logic is embedded in the *protocol controller*. An access and operation reordering mechanism is incorporated as an *efficiency controller*. An SCB slave interface is provided to interface with the on-chip interconnect. This interface results in an efficient slave implementation owing to its out-of-order transaction capabilities. The control and status registers present in the DMC can be accessed using the MMR access bus.

The DMC supports access to the external memory by core and DMA accesses.

DMC Features

The DMC includes a protocol controller that supports:

- JESD79-2E compatible DDR2 SDRAM devices
- JESD209A low-power DDR (LPDDR) SDRAM devices

The features of the dynamic memory controller are:

- Provides 16-bit data only interface to SDRAM devices
- Supports a single external rank (one chip select)
- Supports various burst lengths
- Provides page hit detection that supports multiple column accesses to the same row
- User-specified active, precharge, and refresh commands.
- Programmable SDRAM access timing parameters
- Enables automatic refresh generation with programmable refresh intervals
- Self-refresh mode to reduce system power consumption
- Efficient transaction processing to improve throughput and bandwidth using:

- Software programmable SCB IDs to allow SCB ID-based priority
- The ability to postpone up to eight auto-refresh commands
- Software selectable closed page scheme on a per bank basis
- Simple transaction scheduling mechanism to reduce read write turnaround frequency on the memory bus
- Accesses with the same SCB ID are scheduled back-to-back to take advantage of same page access in SDRAM
- Caching of SDRAM read data burst for specific masters to reduce the latency for same burst accesses.

The DDR2 features are:

- 512M bit to 2G-bit device sizes
- Burst length BL = 4 or 8
- Support for additive latency
- Support for programmable ODT and drive impedance from memory end
- Support for programmable (and ZQ calibration) ODT and drive impedance from the processor end

The LPDDR features are:

- 64M bit to 2G-bit device sizes
- Burst length BL = 4 or 8
- Support for deep power-down mode

Feature Exclusions

The DMC exclusions are as follows:

For DDR2:

- 4-bit and 8-bit wide DDR2 DRAM memories are not supported
- OCD is not supported
- Burst interleaved accesses are not supported

For LPDDR:

- 32-bit wide LPDDR memory devices are not supported
- Status register read (SRR) is not supported
- Sampling the optional temperature output (TQ) signal is not supported
- Clock stop mode is not supported
- Bursts of 2 and 16 words are not supported

- No support for BURST_TERMINATE command
- Dual-die, two CS# and two CKE packages are not supported

DMC Functional Description

The dynamic memory controller consists of master and slave interfaces, a protocol controller, and an efficiency controller. The following sections describe the function of these interfaces and controllers.

ADSP-BF70x DMC Register List

The Dynamic Memory Controller module (DMC) provides an interface to external double-data-rate SDRAM. This interface supports various DDR standards (see chapter descriptions). A set of registers governs DMC controller operations. For more information on DMC controller functionality, see the DMC Controller Register Descriptions.

Table 12-1: ADSP-BF70x DMC Register List

Name	Description
DMC_CFG	Configuration Register
DMC_CPHY_CTL	Controller to PHY Interface Register
DMC_CTL	Control Register
DMC_DLLCTL	DLL Control Register
DMC_EFFCTL	Efficiency Control Register
DMC_EMR1	Shadow EMR1 Register
DMC_EMR2	Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)
DMC_MR	Shadow MR Register
DMC_MSK	Mask (Mode Register Shadow) Register
DMC_PRIO	Priority ID Register 1
DMC_PRIO2	Priority ID Register 2
DMC_PRIOMSK	Priority ID Mask Register 1
DMC_PRIOMSK2	Priority ID Mask Register 2
DMC_RDDATABUFID1	DMC Read Data Buffer ID Register 1
DMC_RDDATABUFID2	DMC Read Data Buffer ID Register 2
DMC_RDDATABUFMSK1	DMC Read Data Buffer Mask Register 1
DMC_RDDATABUFMSK2	DMC Read Data Buffer Mask Register 2
DMC_STAT	Status Register
DMC_TR0	Timing 0 Register
DMC_TR1	Timing 1 Register
DMC_TR2	Timing 2 Register

ADSP-BF70x DMC Register List

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

Table 12-2: ADSP-BF70x DMC Register List

Name	Description
DMC_CAL_PADCTL0	Calibration PAD Control 0 Register
DMC_CAL_PADCTL1	Calibration PAD Control 1 Register
DMC_CAL_PADCTL2	Calibration PAD Control 2 Register
DMC_PHY_CTL0	PHY Control 0 Register
DMC_PHY_CTL1	PHY Control 1 Register
DMC_PHY_CTL2	PHY Control 2 Register
DMC_PHY_CTL3	PHY Control 3 Register
DMC_PHY_CTL4	PHY Control 4 Register
DMC_PHY_CTL5	PHY Control 5 Register
DMC_PHY_STAT0	PHY Status 0 Register
DMC_PHY_STAT3	PHY Status 3 Register
DMC_PHY_STAT4	PHY Status 4 Register
DMC_PHY_STAT5	PHY Status 5 Register

Protocol Controller

The DDR SDRAM protocol controller translates memory access requests from the SCB (system crossbar) interface to JEDEC protocol-specific transactions used by DDR SDRAM devices.

The protocol controller ensures that the various timing parameters are met before reading and writing the SDRAM. The controller also performs the SDRAM initialization sequence as mandated by the standard. The protocol controller can:

- Issue reads and writes
- Precharge a row in a bank
- Activate a row in a bank
- Put the SDRAM devices in self-refresh and power-down modes

The protocol controller takes mode register writes from the MMR interface and translates them into mode register writes to SDRAM. Writing into the mode register is restricted through a mask register.

Efficiency Controller

The efficiency controller controls the ordering of transfers buffered in the read and write command buffers. It attempts to order transfers to optimize the available memory bandwidth. The DMC uses a number of schemes, described in the following sections, to increase the throughput.

Page-Based Scheduling

The DMC parses each write and read transaction that it buffered and gets the information of the row (page) and bank address. The protocol controller maintains the information about the pages that are opened in each bank. The efficiency controller uses the information about the opened pages while scheduling the buffered transactions. The transactions to the opened pages are given higher priority than the other outstanding transactions.

Same Master Transaction Scheduling

The DMC also stores the ID of each transaction that it buffered. In most of the cases, the transactions related to a master result in page hits from the locality of reference rule. The efficiency controller uses the ID information of the transactions while scheduling. When the page-based scheduling of the buffered transactions is complete, same master transaction scheduling is triggered. If multiple transactions from a master are received, the efficiency controller schedules the transactions back-to-back.

DMC Read Data Buffer

The DMC read data buffer contains a data buffer and an address buffer. The depth of the data buffer is equal to the burst length that is programmed in SDRAM. The address buffer holds the corresponding SDRAM burst address. When an SDRAM write address from any master matches an address in the DMC read data buffer, the DMC invalidates the related data in the read buffer. When the DMC_RDDATABUFMSK1 or DMC_RDDATABUFMSK2 register is programmed with a value other than zero, the DMC read data buffer operation is enabled. The set of masters whose data is buffered and retrieved are programmed in the DMC_RDDATABUFID1 or DMC_RDDATABUFID2 registers. The DMC can use the DMC_RDDATABUFMSK1 and DMC_RDDATABUFMSK2 ID registers to select a set of masters similar to the programming of the DMC_PRIOMSK and DMC_PRIOMSK2 registers.

See the SCB ID-Based Priority section for details.

Closed Page Per Bank

The DMC_EFFCTL register provides per-bank granularity for closing pages. The software can determine that most accesses to a given bank in memory always result in a missed page. In this case, set the PREC_BANK bit corresponding to the required bank to close the row after every transfer. This proactive step can result in reduced thrashing and increases memory throughput.

SCB ID-Based Priority

The primary goal of the dynamic memory controller is to improve sustainable memory system bandwidth so that the service time for the average request can be reduced. However, to service critical requests from any master in the

system, the DMC provides a mechanism to elevate priority of a given access. The DMC priority ID registers (DMC_PRIO and DMC_PRIO2) can be programmed with up to two SCB IDs with elevated priority.

After every access in a snapshot, the command buffers are searched to determine whether an ID of a command matches with the ID programmed in the DMC_PRIO and DMC_PRIO2 registers. The priority SCB ID access is sent before the subsequent access in the snapshot if:

- A match occurs, and
- The direction of the access (for example write) is the same as the direction of the snapshot (write)

There is an alternative to providing priority to a specific SCB ID. If a number of IDs from the same master require priority, program the DMC priority mask ID registers (DMC_PRIOMSK and DMC_PRIOMSK2) so that the corresponding bits are 0. The DMC uses a combination of the DMC_PRIO and DMC_PRIO2 registers and the DMC_PRIOMSK/DMC_PRIOMSK2 registers to elevate the priority of a select few or all IDs that belong to a master. By default, none of the IDs are prioritized. The following are a few possibilities:

- The DMC_PRIOMSK field is set to 0x00000000. If a single ID (7234) needs priority, set the DMC_PRIOMSK field to 0xFFFFFFFF and set the DMC_PRIO field to 7234.
- If the DMC_PRIOMSK field is set to 0xFFFFFFFE, the SCB IDs 7234 and 7235 are given priority.
- If the DMC_PRIOMSK field is set to 0xFFFFFFFC, the SCB IDs 7234, 7235, 7236, and 7237 are given priority.
- If two transactions with priority, one read and the other a write, are outstanding, the priority transaction that does not change the direction of the DMC access gets priority. The other priority transaction is handled at the beginning of the next snapshot. For example, if a write snapshot is in-progress, the write priority transaction is sent. The read priority transaction is sent at the beginning of the next read snapshot.

NOTE: Use SCB ID-based priority judiciously because it can significantly reduce the throughput of the DMC.

Delaying up to Eight Auto-Refresh Commands

The DMC uses this method to ensure that auto-refresh does not interfere with any critical data transfers. Up to eight auto-refresh commands can accumulate in the DMC. The exact number of auto-refresh commands can be programmed using the DMC_EFFCTL.NUMREF bit.

After the first refresh command is accumulated, the DMC constantly looks for an opportunity to schedule a refresh command. When the SCB read and write command buffers become empty for the programmed number of clock cycles (DMC_EFFCTL.IDLECYC bit field), the accumulated number of refresh commands are sent back-to-back to the DRAM. (The empty state of the SCB command buffers implies that no access is outstanding.)

After every refresh, the SCB command buffers are checked to ensure that they stay empty. If the SCB command buffers are always full, once the programmed number of refresh commands accumulates, the refresh operation is elevated to urgent priority. One refresh command is sent immediately. After this process, the DMC continues to wait for an opportunity to send out refresh commands. If self-refresh mode is enabled, all pending refresh commands are given out only after that DMC enters into self-refresh mode.

Page and Bank Interleaving

Page and bank interleaving allow consecutive row accesses to fall into the same bank (bank interleaving) or into a different bank (page interleaving). The DMC uses bank interleaving by default (`DMC_CTL.ADDRMODE` bit =0). If the `DMC_CTL.ADDRMODE` bit =1, the DMC uses page interleaving. Page misses in one addressing mode result in hits in the other addressing mode.

System Crossbar Slave Interface

The DMC uses the system crossbar slave interface to move all data. The system crossbar interface accepts interleaved write transactions and sends out-of-order responses. The read and write interfaces consist of buffers for address, data, and control information transferred to or from the system crossbar bus.

The system crossbar interface transactions are sent to the SDRAM only after the SDRAM has been initialized. However, if transactions arrive before or during initialization, they accumulate in the system crossbar interface and are sent out to the protocol controller once the initialization completes.

To increase throughput, the system crossbar write-response is sent out as soon as the final DDR burst is scheduled for transfer into the SDRAM. However, if an auto-refresh is needed, the scheduled write data is sent only after the auto-refresh. A delay can occur. The delay is a maximum of 64 clock cycles from the moment the write response is sent on the SCB to the write operation of the data into SDRAM.

The system crossbar interface performs the following operations:

- Buffers read and write command requests from the system crossbar bus
- Processes the requests by converting them to protocol controller user-interface transfers
- Sends and receives data to or from the protocol controller
- Creates a suitable read/write response and sends read data back to the system crossbar bus

The system crossbar slave interface supports the following:

- All burst lengths (1–16)
- Incremental and wrap bursts
- Data transfer sizes of 8-bit, 16-bit, or 32-bit
- Arrival of write data before write address
- Generation of error responses which include:
    - Any access to an unimplemented region of the external memory space
    - Any access when the SDRAM is in self-refresh, power-down, or deep power-down mode (in case of LPDDR)
    - Any access when the direct command interface is in operation

Read/Write Command and Data Buffers

The system crossbar interface consists of a four-deep read command buffer and a four-deep write command buffer. Up to four write commands and four read commands can be waiting for access to the SDRAM. The system crossbar write buffer is 32 deep. It can support write data interleaving of two. The system crossbar read buffer is 32 deep.

Peripheral Bus Slave Interface

The peripheral bus slave interface connects the dynamic memory controller to the peripheral bus and provides a host controller with access to the registers. The peripheral bus slave interface supports the following features:

- Read and write word accesses
- 32-bit data bus

Architectural Concepts

The following sections provide information on the architecture of the interface.

Controller On Die Termination (ODT)

The controller ODT is enabled with the granularity of a byte lane. The description of this feature can be obtained in the description of the corresponding PHY registers. Controller ODT involves extra overhead in terms of power consumption during reads.

The DMC implements dynamic on die termination at processor pads. When controller ODT is enabled, the termination resistors in the pads are turned on when the controller reads data from the DRAM. These resistors are turned off when the controller writes to the DRAM.

Mode Register Set and Extended Mode Register Set Command

The load mode register command initializes the SDRAM operation parameters. The DMC supports the mode register set and extended mode register set commands. The controller automatically issues the mode register set command during power-on initialization and also when the DMC_MR register is written with the DMC_MSK.MR bit. The mode register set command is sent after the ongoing data transfer completes.

The DMC automatically issues the mode register set command when the shadow EMR1/EMR2 registers are written. The corresponding DMC_MSK.EMR2/DMC_MSK.EMR1 bits must be enabled.

DDR2 SDRAM Organization

The DMC supports DDR2 SDRAM memory modules ranging from 256Mb to 2Gb. The following tables list the address translation mechanism from the user interface to DDR2 memory interface. The controller also supports two types of addressing modes: bank interleaving (DMC_CTL.ADDRMODE=1) and page interleaving (DMC_CTL.ADDRMODE =0).

Bank Interleaving

The ***DDR2 Bank Interleaving*** table shows DDR2 bank interleaving.

Table 12-3: DDR2 Bank Interleaving

SDRAM size	Bank address bits	Row address bits	Column address bits
256 Mb	24:23	22:10	9:1
512 Mb	25:24	23:11	10:1
1 Gb	26:24	23:11	10:1
2 Gb	27:25	24:11	10:1

Page Interleaving

The ***DDR2 Page Interleaving*** table shows DDR2 page interleaving.

Table 12-4: DDR2 Page Interleaving

SDRAM size	Row address bits	Bank address bits	Column address bits
256 Mb	24:12	11:10	9.1
512 Mb	25:13	12:11	10:1
1 Gb	26:14	13:11	10:1
2 Gb	27:14	13:11	10:1

LPDDR SDRAM Organization

The DMC supports LPDDR SDRAM memory modules ranging from 64M bit to 2G bit. The following tables list the address translation mechanism from the user interface to LPDDR memory interface.

The controller also supports two types of addressing modes: bank interleaving (`DMC_CTL.ADDRMODE`=1) and page interleaving (`DMC_CTL.ADDRMODE`=0).

Bank Interleaving

The ***LPDDR Bank Interleaving*** table shows LPDDR bank interleaving.

Table 12-5: LPDDR Bank Interleaving

SDRAM size	Bank address bits	Row address bits	Column address bits
64 Mb	22:21	20:9	8:1
128 Mb	23:22	21:10	9:1
256 Mb	24:23	22:10	9:1
512 Mb	25:24	23:11	10:1
1 Gb	26:24	23:11	10:1
2 Gb	27:26	25:12	11:1

Page Interleaving

The ***LPDDR Page Interleaving*** table shows LPDDR page interleaving.

Table 12-6: LPDDR Page Interleaving

SDRAM size	Row address bits	Bank address bits	Column address bits
64 Mb	22:11	10:9	8.1
128 Mb	23:12	11:10	9:1
256 Mb	24:12	11:10	9:1
512 Mb	25:13	12:11	10:1
1 Gb	26:14	13:11	10:1
2 Gb	27:14	13:11	10.1

DMC Clocking

The DMC uses a divided-down version of the *PLLCLK* (PLL clock) to generate an internal clock for clocking the DMC block and interface. The specific value of the *DCLK* frequency is programmed in the `CGU_DIV` register. The section Initializing the DMC (ADSP-BF70x) describes the procedure.

For information on the maximum clock frequency supported for specific modes, refer to the processor data sheet.

NOTE: For the processor variants that have two DMC blocks, both blocks run on the same DCLK frequency.

DMC DMA

The DMC supports DMA-based transfers to and from external DDR SDRAM memory and internal memory.

The DMC DMA controller, part of the distributed DMA engines (DDE) that are dispersed through the infrastructure, connects to the system crossbar fabric.

The DMC uses two DDEs for memory-to-memory DMA (MemDMA). One channel is the source channel, and the second, the destination channel.

DMA transfers on the processor are descriptor-based or register-based. Register-based DMA allows the processor to program DMA control registers directly to initiate a DMA transfer. On completion, the control registers can be automatically updated 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. This transfer allows the chaining together of multiple DMA sequences. In descriptor-based DMA operations, a DMA channel can be programmed to set up and start another DMA transfer automatically after the current sequence completes.

Refer to the DMA chapter for further details.

DMC Operating Modes

By default, the DMC is in DDR2 mode. To enable LPDDR mode, the corresponding bits in the DMC_CTL and DMC_PHY_CTL4 registers must be configured.

DDR2 Mode

This mode is the default mode of the DMC module and supports JESD79-2E compatible DDR2 SDRAM. In this mode, the DMC_CTL.LPDDR bit =0, and the DMC_PHY_CTL4.DDRMODE bit field is 0b'01.

LPDDR Mode

The DMC module supports JESD209A low-power DDR (LPDDR) SDRAM. To configure this mode of operation, set (=1) the DMC_CTL.LPDDR bit and set the DMC_PHY_CTL4.DDRMODE bit field to 0b'11.

Deep Power-Down Mode

The DMC module supports JESD209A low-power DDR (LPDDR) SDRAM. To configure this mode of operation, set (=1) the DMC_CTL.LPDDR bit and set the DMC_PHY_CTL4.DDRMODE bit field to 0b'11.

When the processor does not require the data stored in SDRAM (assume reset state of SDRAM), the DMC can put the SDRAM in deep power-down mode. When the DMC is in deep power-down mode, any data accesses cause the DMC to generate a bus error. To configure this mode, set (=1) the DMC_CTL.DPDREQ bit when low-power DMC operation is enabled (DMC_CTL.LPDDR =1).

The DMC_STAT.IDLE bit indicates the activity in the DMC. If this bit is set, there is no activity all through the DMC. The DMC_STAT.DPDACK bit is asserted when the SDRAM enters deep power-down mode. The DMC stays in deep power-down mode as long as the DMC_CTL.DPDREQ bit is asserted.

Clearing (=0) the DMC_CTL.DPDREQ bit causes the DMC to exit deep power-down mode. Also, when exiting deep power-down mode, the controller clears the DMC_STAT.DPDACK bit. The user must re-initialize the DMC after it comes out of deep power-down mode.

Self-Refresh Mode

For low-power consumption, the SDRAM can be put in self-refresh mode. During a processor hibernate, or when no data activity occurs, the DMC can put the SDRAM in self-refresh mode to save power. The DMC_STAT.IDLE bit indicates the activity on the DMC. If this bit is set, there is no activity in the DMC.

Enable self-refresh mode by writing the DMC_CTL.SRREQ bit. The DMC stays in a self-refresh state as long as this bit is asserted. The DMC_STAT.SRACK bit indicates when the SDRAM enters self-refresh mode.

When the DMC is in self-refresh mode, the DMC generates an SCB error when any data accesses (read or write requests) are requested.

The DMC can be brought out of self-refresh mode by clearing the DMC_CTL.SRREQ bit again. The controller clears the DMC_STAT.SRACK bit after the self-refresh operation completes.

DMC Event Control

The DMC has no related interrupt or trigger event information.

DMC Programming Model

The dynamic memory controller contains five groups of memory-mapped registers. The DMC uses the MMR access bus to connect to these registers.

- Control and status registers. These registers control the various operation modes of the dynamic memory controller and provide status.
- Timing parameter registers. The value programmed in these registers depends on the speed grade of the SDRAM device used.
- Mode register mirror registers. These shadow registers are copies of the mode registers residing in the SDRAM device.
- PHY control and status registers. The DMC uses these registers to control the operation of the PHY.
- PAD control registers. The DMC uses these registers to control the various aspects of the I/O pads.

The DMC control registers contain sensitive timing parameters and settings for the DDR SDRAM. These registers are programmed with values that are in the operating range of the DDR used.

Writing to reserved fields or writing any reserved values in register bits can cause the dynamic memory controller to function erroneously.

Programming Considerations for DDR2 and LPDDR Memory

The ***DDR2 and LPDDR Programming*** table shows important programming considerations and differences across DDR2 and LPDDR memory technologies. The table serves as a quick reference when configuring the DMC and PHY registers.

Table 12-7: DDR2 and LPDDR Programming

PHY/ Controller	Description	Registers and Bit Fields Involved	DDR2	LPDDR
PHY	Enabling DDR2/LPDDR modes	DMC_PHY_CTL4	Select DDR2 mode by setting the DMC_PHY_CTL4.DDRMODE bit field to 01.	Select LPDDR mode by setting the DMC_PHY_CTL4.DDRMODE bit field to 11.
PHY	ODT and drive impedance calibration	DMC_CAL_PADCTL0, DMC_CAL_PADCTL2	The DMC supports ODT and drive impedance calibration. Configure the DMC_CAL_PADCTL0 and DMC_CAL_PADCTL2 registers accordingly. For details, refer PAD Calibration for Driver	The DMC does not support ODT and drive impedance calibration. No need to program the DMC_CAL_PADCTL0and DMC_CAL_PADCTL2 registers. Must make sure that the

Table 12-7: DDR2 and LPDDR Programming (Continued)

PHY/ Controller	Description	Registers and Bit Fields Involved	DDR2	LPDDR
			Impedance and On Die Termination (ODT) section. Programming driver impedance is required. Programming ODT is optional. To disable ODT, set the DMC_PHY_CTL1.BYPODTEN bit.	DMC_PHY_CTL1.BYPODTEN bit is set to bypass the processor ODT settings.
Controller	Enabling DDR2/LPDDR modes	DMC_CTL.LPDDR	Default is DDR2 mode. Make sure that the DMC_CTL.LPDDR bit is cleared.	Select LPDDR mode by setting the DMC_CTL.LPDDR bit.
Controller	Configuring DMC_CTL.RDTOWR bit field	DMC_CTL.RDTOWR	Make sure that the DMC_CTL.RDTOWR bit field is always set to 010 (=2 in decimal).	
Controller	Configuring DMC_CFG register fields	DMC_CFG	Make sure that the DMC_CFG.IFWID and DMC_CFG.SDRWID bit fields are always set to 0010 (=2 in decimal) as the DMC only supports 16-bit wide interface and SDRAM widths. Make sure that the bit field DMC_CFG.EXTBANK is always set to 0000 as the DMC only supports one external bank. Select the DMC_CFG.SDRSIZE as per the SDRAM size. Supported sizes are 64 Mb to 2 Gb for LPDDR, 256 Mb to 4 Gb for DDR2.	
Controller	Configuring controller timing parameter registers	DMC_TR0, DMC_TR1, DMC_TR2	Configure the parameters t_{RCD}, t_{WTR}, t_{RP}, t_{RAS}, t_{MRD}, t_{REF}, t_{RFC}, t_{RRD}, t_{WR}, t_{XP}, t_{CKE} (DDR2/LPDDR), and t_{FAW}, t_{RTP} (DDR2) in terms of DCLK cycles.	
Controller	Configuring burst length	DMC_MR.BLEN (for DDR2 and LPDDR)	The DMC supports both burst length of 4 and 8. Configure DMC_MR.BLEN field to 10 for burst length of 4 and to 11 for burst length of 8 words.	The DMC supports both burst length of 4 and 8. Configure DMC_MR.BLEN field to 10 for burst length of 4 and to 11 for burst length of 8 words.
Controller	Configuring CAS latency	DMC_MR.CL (for DDR2 and LPDDR)	The DMC supports CAS latencies of 3 to 6. Refer to the DMC_MR register description for more details.	The DMC supports CAS latency of 3. Refer to the DMC_MR register description for more details.
Controller	Configuring the DMC_MR.DLLRST bit (DDR2/ LPDDR)	DMC_MR.DLLRST	Setting of this bit is optional for DDR2.	Reserved
Controller	Write recovery timing	DMC_MR.WRRECOV	The DMC supports WR values of 2 to 8. Refer to the DMC_MR register description for more details.	Reserved

Table 12-7: DDR2 and LPDDR Programming (Continued)

PHY/ Controller	Description	Registers and Bit Fields Involved	DDR2	LPDDR
Controller	Configuring DMC_EMR1 (DDR2) register	DMC_EMR1 (DDR2)	The DMC can use this register to configure memory drive impedance, ODT, and additive latency parameters. Refer to the corresponding register descriptions for more details.	This register is not used for LPDDR programming.
Controller	Configuring DMC_EMR2 (DDR2) or DMC_EMR (LPDDR) register	DMC_EMR2 (DDR2) or DMC_EMR (LPDDR)	The DMC can use this register to configure the Partial Array Self-Refresh (PASR) and High Temperature Self-Refresh Rate Enable (SRF) functionalities of DDR2 memory device. For more details on these functionalities, refer to the DDR2 memory device data sheet.	The DMC can use this register to configure the Partial Array Self-Refresh (PASR), Temperature compensated self-refresh (TCSR), and drive strength (DS) functionalities of the memory device. For more details on these functionalities, refer to the LPDDR memory device data sheet.
Controller	Configuring the DMC_DLLCTL register	DMC_DLLCTL	Always configure the DMC_DLLCTL.DLLCALRDCNT field to 0x4B and DMC_DLLCTL.DATACYC field to 0x5.	

PHY DLL Calibration

The PHY DLL calibration is performed as part of the SDRAM power-up initialization. It calibrates data against the DQS and CLK signal. However, running DLL calibration after self-refresh or at an arbitrary time is required in certain cases.

The DMC allows PHY DLL calibration to start by setting the DMC_CTL.DLLCAL bit. The DMC_STAT.DLLCALDONE bit can be used to monitor the progress of the calibration. Once calibration is over, this bit is set. Once the calibration procedure is started by writing to the DMC_CAL_PADCTL0.CALSTRT bit, the full calibration takes 300 DCLK cycles to complete.

NOTE: DLL calibration can be initiated only when the DMC is idle (DMC_STAT.IDLE= 1).

On Die Termination (ODT)

In addition to the driver impedance, the bidirectional pads (Data and DQS) also require the initialization sequence to program the termination impedance by writing to the DMC_CAL_PADCTL2.IMPRTT field. See the following table.

Termination Impedance	Member Pads
IMPRTT (Correction factor = 0.8, explanation)	DATA PADS, DQS PADS

The DMC pads use parallel termination; one branch goes from the pad to the I/O supply and the other to the I/O ground. The value programmed to this 8-bit field is the value to be used for each branch. There is a correction factor involved while programming this register. The value of this correction factor is 0.8.

For example, suppose that a termination of 50 Ω is required on the data pads to match with the board trace. Then, this value is programmed to 100 × 0.8 = 80, as the two parallel paths lead to an effective impedance of 50 Ω. The table lists the value to be programmed for common cases.

RTT (Effective termination at the pads) = "x"	Value to be programmed in IMPRTT = "2*x*0.8"
50	80
75	120
150	240

Once the calibration procedure is started by writing to the `DMC_CAL_PADCTL0.CALSTRT` bit, the full calibration takes 300 DCLK cycles to complete.

NOTE: Drive impedance and ODT calibration are supported for DDR2 mode only. For LPDDR mode, there is no need to program the `DMC_CAL_PADCTL0` and `DMC_CAL_PADCTL2` registers. However, ensure that the `DMC_PHY_CTL1.BYPODTEN` bit is set to bypass the processor ODT settings.

Configuring the DMC (ADSP-BF70x)

After a hardware or software reset of the processor, the DMC clocks are enabled. However, the DMC must be configured and initialized before any data transfer can take place.

NOTE: The DMC must be in an appropriate state before programming the DMC and executing the power-up (initialization) sequence. Ensure that the clock to the SDRAM is enabled after the power has stabilized. For the required stabilization time, see the corresponding SDRAM specification.

NOTE: Bits 6, 7, 25, and 27 of the `DMC_PHY_CTL3` register must be set for DDR2 and LPDDR modes. For example, set these bits during first-time DMC initialization. Then, software does not need to touch or clear these bits. Use the following C code to set these bits for the first time DMC initialization:

```
*pREG_DMC0_PHY_CTL3= 0xA0000C0;
```

For DDR2, when initialing the DMC for the first time after power-up, perform PAD calibration before initializing the DMC. The DMC PAD calibration procedure is as follows:

1. Set the device mode (DDR2/LPDDR) in the `DMC_PHY_CTL4` register. For LPDDR mode, skip steps 2, 3, and 4 and set the `DMC_PHY_CTL1.BYPODTEN` bit.
2. Configure the necessary values in the `DMC_CAL_PADCTL0`, and `DMC_CAL_PADCTL2` registers without setting the `DMC_CAL_PADCTL0.CALSTRT` bit.
3. Set the `DMC_CAL_PADCTL0.CALSTRT` bit.

4. Wait for 300 *DCLK* cycles for the PAD calibration to complete.

Initializing the DMC (ADSP-BF70x)

To initialize the DMC:

1. Poll the DMC_STAT.IDLE bit, waiting for it to set, to ensure that the DMC is idle.
2. Apply a clock change, if needed:
   a. Place the DMC in self-refresh mode by setting the DMC_CTL.SRREQ bit and polling for the DMC_STAT.SRACK bit to set.
   b. Program the PLL frequency to a new value (See the CGU chapter for details).
   c. Wait 4500 DCLK cycles to ensure that the DLL has locked.
   d. Bring the DMC out of self-refresh mode by clearing the DMC_CTL.SRREQ bit and polling for the DMC_STAT.SRACK bit to clear.
3. If not already done, wait 4500 DCLK cycles to ensure that the DLL is locked.
4. Program the DMC_CFG, DMC_CTL, DMC_TR0, DMC_TR1, and DMC_TR2 registers with the appropriate values, to properly set SDRAM cycle timingand configuration options.

   ADDITIONAL INFORMATION: For example, t_{RAS}, t_{RC}, t_{RP}, t_{RCD}, t_{WR}, and t_{FAW} are some of the parameters.
5. Program the shadow registers DMC_EMR1- DMC_EMR2 with the needed burst length, CAS latency, additive latency, and other parameters.
6. Set the DMC_CTL.INIT bit to begin the power-up initialization sequence.
7. Wait for the SDRAM initialization sequence to complete by polling for the DMC_STAT.INITDONE to set.

   ADDITIONAL INFORMATION: The DMC PHY requires recalibration after a DCLK change has been effected; therefore, the power-up initialization sequence must be performed whether or not the timing registers have been modified.

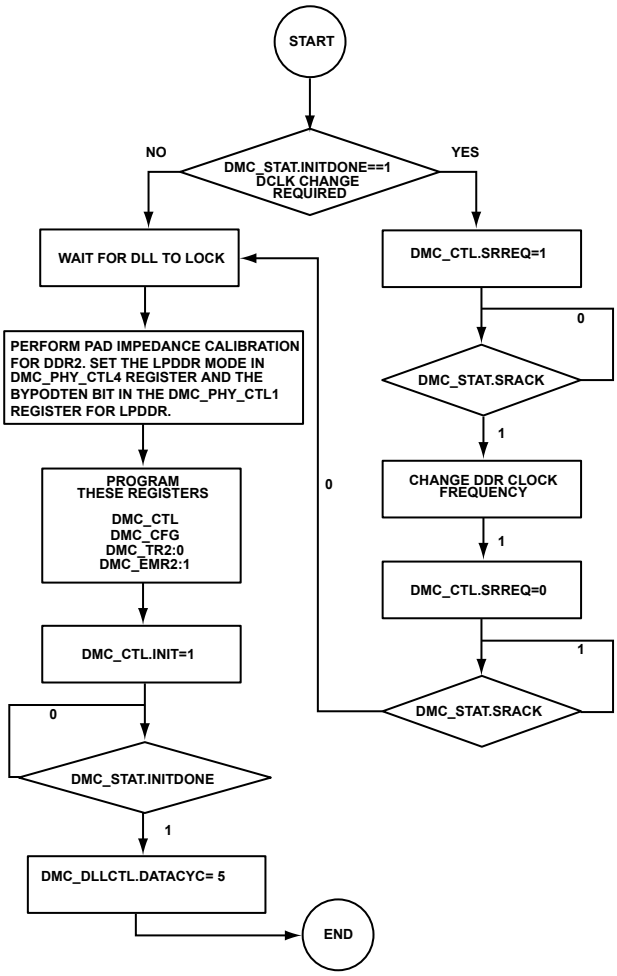

Figure 12-1: DMC Initialization Flow

The DMC accumulates system crossbar transactions that occur during or before initialization and sends them to SDRAM once the SDRAM initialization or DLL calibration is complete.

Saving Power with the DMC

This section discusses the suggested flow to enter and exit DDR SDRAM self-refresh or deep power-down (LPDDR only) mode before and after the processor enters the hibernate state.

1. To put the DDR device in self-refresh or deep power-down mode, set the `DMC_CTL.SRREQ` or `DMC_CTL.DPDREQ` bit, respectively.

   The DDR goes through the self-refresh or deep power-down entry sequence and enters the self-refresh or deep power-down mode. The `DMC_STAT.SRACK`/`DMC_STAT.DPDACK` bit is set.

2. Save the content of the following DMC registers in the DPM restore registers: `DMC_CFG`, `DMC_MR`, `DMC_EMR1`, `DMC_EMR2`, `DMC_TR0`, `DMC_TR1`, `DMC_TR2`, `DMC_DLLCTL`, `DMC_CTL`. The `DMC_EMR1` register is only specific to the DDR2 device and not necessarily saved when using an LPDDR device.

3. PAD recalibration: if calibration is needed after exiting hibernate mode, save the following bit fields of the DMC PHY registers: DMC_PHY_CTL0[31:0], DMC_PHY_CTL1[31:16] + DMC_PHY_CTL4[7:0] + DMC_PHY_CTL5[7:0], DMC_PHY_CTL2[31:0], DMC_PHY_CTL3[31:0], DMC_CAL_PADCTL0[31:12], (DMC_PHY_STAT3[2:0] and DMC_PHY_STAT0[8:0]) DMC_CAL_PADCTL2[31:0] OR
4. PAD calibration restore: if PAD recalibration is not needed after exiting hibernate mode, save the following bit fields of the DMC PHY registers. In this case, the PAD calibration data restores as is. DMC_PHY_CTL0[31:0], DMC_PHY_CTL1[31:16] + DMC_PHY_CTL4[7:0] + DMC_PHY_CTL5[7:0], DMC_PHY_CTL2[31:0], DMC_PHY_CTL3[31:0], (DMC_PHY_STAT4[31:12], and DMC_PHY_STAT3[2:0], DMC_PHY_STAT0[8:0],DMC_PHY_STAT5[31:0]).
5. Initialize the DPM_PGCNTR register to the appropriate values to count off the time for the core V_{DD} to reach a safe value when exiting the hibernate state.
6. Enter the hibernate state by following the procedure detailed in the DPM chapter. Hibernate occurs when the SYS_EXTWAKE signal goes low.
7. When SYS_EXTWAKE, the part enters the hibernate state and remains in this state until brought out by a wake-up event.
8. When a wake-up event occurs, a series of responses follow:
    a. The SYS_EXTWAKE signal goes high.
    b. The core domain logic resets when core V_{DD} power reaches a proper value.
    c. The DDR controller drives the input of the DMC_CKE pad low.
    d. A counter-expired signal occurs when the counter reaches 0.

    ADDITIONAL INFORMATION: The core V_{DD} reaches a safe value. After the DDR controller stops driving the DMC_CKE pin, the value present on the DDR controller input pin is driven out to the pads.
9. Upon wake-up from hibernate, wait for the DLL to lock (wait for 4500 DCLK cycles).
10. PAD recalibration: restore all the following PHY registers: DMC_PHY_CTL0[31:0], DMC_PHY_CTL1[31:16] + DMC_PHY_CTL4[7:0] + DMC_PHY_CTL5[7:0], DMC_PHY_CTL2[31:0], DMC_PHY_CTL3[31:0], DMC_CAL_PADCTL0[31:12], (DMC_PHY_STAT3[2:0], and DMC_PHY_STAT0[8:0] not necessary to restore), DMC_CAL_PADCTL2[31:0].
    a. Set the DMC_CAL_PADCTL0.CALSTRT bit.
    b. Wait for 300 *DCLK* cycles for the PAD calibration to complete.
    c. Reset the DMC_CAL_PADCTL0.CALSTRT bit OR
11. PAD calibration restore: restore all the following PHY registers: DMC_PHY_CTL0[31:0], DMC_PHY_CTL1[31:16] + DMC_PHY_CTL4[7:0] + DMC_PHY_CTL5[7:0], DMC_PHY_CTL2[31:0], DMC_PHY_CTL3[31:0], (DMC_PHY_STAT3[2:0], and DMC_PHY_STAT0[8:0] not necessary to restore). Copy

the following registers from DMC_PHY_STAT4[31:12] and DMC_PHY_STAT5[31:0] to the pad control registers as shown in the table.

ADDITIONAL INFORMATION:

Table 12-8: PAD Calibration Restore Source and Destination

Source (DPMHV)	Destination
DMC_PHY_STAT4[23:12]	DMC_CAL_PADCTL0[14:3]
DMC_PHY_STAT0[31:26]	DMC_CAL_PADCTL1[11:6]
DMC_PHY_STAT5[25:20]	DMC_CAL_PADCTL0[26:21]
DMC_PHY_STAT5[19:14]	DMC_CAL_PADCTL0[5:0]
DMC_PHY_STAT5[13:8]	DMC_CAL_PADCTL0[20:1]

a. Set the restore impedance setting DMC_CAL_PADCTL0[2:0] to binary 111.

b. Wait for 20 *DCLK* cycles.

12. Restore the following DMC registers: DMC_CFG, DMC_MR, DMC_EMR1, DMC_EMR2, DMC_TR0, DMC_TR1, DMC_TR2, DMC_DLLCTL, and DMC_CTL.

    ADDITIONAL INFORMATION: By restoring the DMC registers, the DDR goes into self-refresh or deep power-down mode without the initialization bit being set. Wait until the DMC_STAT.SRACK and DMC_STAT.DPDACK bits are set.

13. Bring the DDR controller out of self-refresh or deep power-down mode by clearing the DMC_CTL.SRREQ or DMC_CTL.DPDREQ bits.

    ADDITIONAL INFORMATION: Wait until the DMC_STAT.SRACK and DMC_STAT.DPDACK bits clear. On deep power-down mode exit, the DMC initializes and calibrates the DLL automatically while coming out of deep power-down mode.

14. Initiate the read calibration by setting the DMC_CTL.DLLCAL bit.

    ADDITIONAL INFORMATION: Deep power-down mode does not require this step.

15. Wait for the DMC_STAT.DLLCALDONE and DMC_STAT.INITDONE bits to get set.

ADSP-BF70x DMC Register Descriptions

Dynamic Memory Controller (DMC) contains the following registers.

Table 12-9: ADSP-BF70x DMC Register List

Name	Description
DMC_CFG	Configuration Register
DMC_CPHY_CTL	Controller to PHY Interface Register

Table 12-9: ADSP-BF70x DMC Register List (Continued)

Name	Description
DMC_CTL	Control Register
DMC_DLLCTL	DLL Control Register
DMC_EFFCTL	Efficiency Control Register
DMC_EMR1	Shadow EMR1 Register
DMC_EMR2	Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)
DMC_MR	Shadow MR Register
DMC_MSK	Mask (Mode Register Shadow) Register
DMC_PRIO	Priority ID Register 1
DMC_PRIO2	Priority ID Register 2
DMC_PRIOMSK	Priority ID Mask Register 1
DMC_PRIOMSK2	Priority ID Mask Register 2
DMC_RDDATABUFID1	DMC Read Data Buffer ID Register 1
DMC_RDDATABUFID2	DMC Read Data Buffer ID Register 2
DMC_RDDATABUFMSK1	DMC Read Data Buffer Mask Register 1
DMC_RDDATABUFMSK2	DMC Read Data Buffer Mask Register 2
DMC_STAT	Status Register
DMC_TR0	Timing 0 Register
DMC_TR1	Timing 1 Register
DMC_TR2	Timing 2 Register

Configuration Register

The DMC_CFG register selects SDRAM device specific parameters and selects the SDRAM interface width.

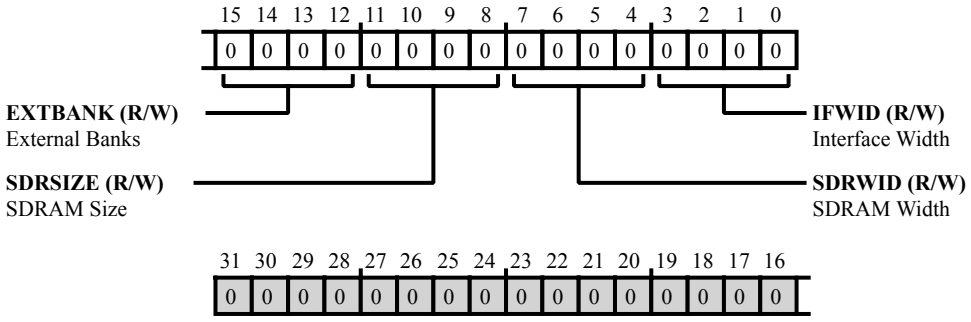

Figure 12-2: DMC_CFG Register Diagram

Table 12-10: DMC_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	EXTBANK	External Banks. The DMC_CFG.EXTBANK bits select the number of external banks connected to the DMC. Note that all values other than those shown are reserved.	
		0	1 External Bank
		1-15	Reserved
11:8 (R/W)	SDRSIZE	SDRAM Size. The DMC_CFG.SDRSIZE bits select the size of individual SDRAM connected to the DMC. Note that all values other than those shown are reserved.	
		0	64M Bit SDRAM (LPDDR Only)
		1	128M Bit SDRAM (LPDDR Only)
		2	256M Bit SDRAM
		3	512M Bit SDRAM
		4	1G Bit SDRAM
		5	2G Bit SDRAM
		6	4G Bit SDRAM
		7	8G Bit SDRAM

Table 12-10: DMC_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7:4 (R/W)	SDRWID	SDRAM Width. The DMC_CFG.SDRWID bits select the width of the individual SDRAM connected to the DMC. Note that all values other than those shown are reserved.	
		0-1	Reserved
		2	16-Bit Wide SDRAM
		3-15	Reserved
3:0 (R/W)	IFWID	Interface Width. The DMC_CFG.IFWID bits select the width of the interface between the DMC and SDRAM. Note that all values other than those shown are reserved.	
		0-1	Reserved
		2	16-Bit Wide Interface. All other values are reserved. This field specifies the interface width between the controller and the SDRAM.
		3-15	Reserved

Controller to PHY Interface Register

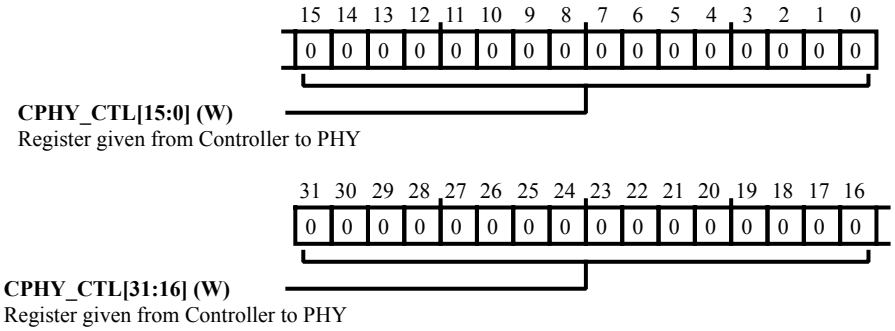

Figure 12-3: DMC_CPHY_CTL Register Diagram

Table 12-11: DMC_CPHY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	CPHY_CTL	Register given from Controller to PHY.

Control Register

The DMC_CTL register controls DMC modes, DLL calibration, and DRAM initialization.

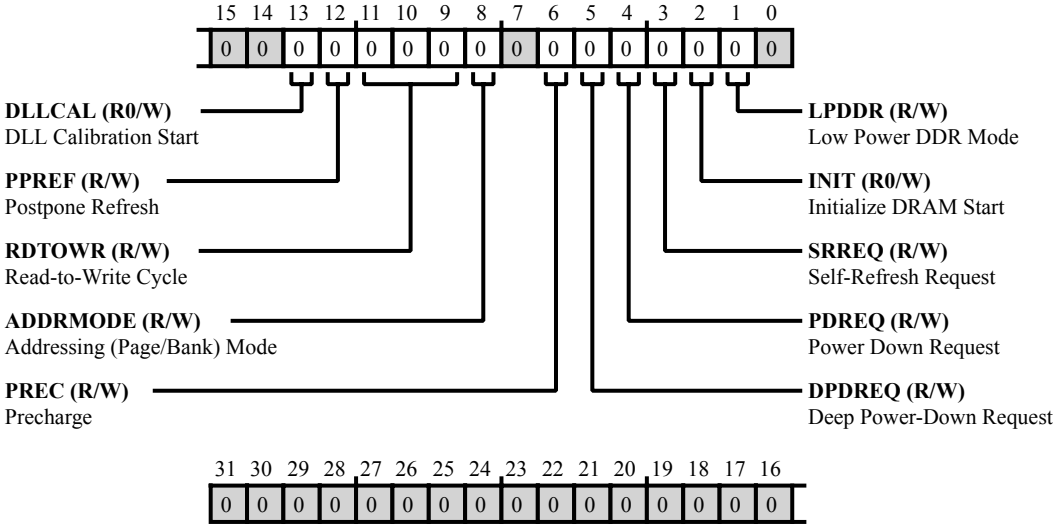

Figure 12-4: DMC_CTL Register Diagram

Table 12-12: DMC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R0/W)	DLLCAL	DLL Calibration Start. The DMC_CTL.DLLCAL bit starts the PHY DLL calibration sequence. Note that this bit always reads as 0.	
		0	No effect
		1	Start PHY DLL calibration
12 (R/W)	PPREF	Postpone Refresh. The DMC_CTL.PPREF bit enables postponing the DMCs sending of auto-refresh commands. When enabled, the DMC accumulates refresh commands. The DMC_EFFCTL.NUMREF field selects the number of refresh commands that the DMC can accumulate. When disabled, the DMC_TR1.TREF field selects the interval for auto-refresh command distribution. A maximum of eight auto-refresh commands can be accumulated in DDR2 mode and a maximum of four auto-refresh commands in low power DDR mode.	
		0	Disable Postpone Refresh
		1	Enable Postpone Refresh
11:9	RDTOWR	Read-to-Write Cycle.	

Table 12-12: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
(R/W)		The DMC_CTL.RDTOWR bits select the number of cycles that the DMC adds when a write operation follows a read operation. For proper operation, it should be programmed with the value of 010.	
		0	1 Cycle Added from JEDEC Spec Value
		1	2 Cycles Added from JEDEC Spec Value
		2	3 Cycles Added from JEDEC Spec Value
		3	4 Cycles Added from JEDEC Spec Value
		4	5 Cycles Added from JEDEC Spec Value
		5	6 Cycles Added from JEDEC Spec Value
		6	7 Cycles Added from JEDEC Spec Value
		7	8 Cycles Added from JEDEC Spec Value
8 (R/W)	ADDRMODE	Addressing (Page/Bank) Mode. The DMC_CTL.ADDRMODE bit selects whether the DMC uses page or bank interleaving for addressing. When using page interleaving, the bank address bits follow the most significant column address bits. When using bank interleaving, the bank address bits follow the most significant row address bits.	
		0	Bank Interleaving
		1	Page Interleaving
6 (R/W)	PREC	Precharge. The DMC_CTL.PREC bit enables precharge, which closes DRAM rows immediately after access. When disabled, all accesses result in the respective DRAM rows remaining open, until the DMC needs to close them.	
		0	No Effect
		1	Enable Precharge
5 (R/W)	DPDREQ	Deep Power-Down Request. The DMC_CTL.DPDREQ bit enables deep power-down mode if low power DMC operation is enabled (DMC_CTL.LPDDR =1). When the processor does not require the data stored in SDRAM (assume reset state of SDRAM), the DMC may put the SDRAM in deep power-down mode. When the DMC is in deep power-down mode, any data accesses cause the DMC to generate a bus error. The DRAM remains in deep power-down mode as along as this bit is 1.	
		0	Disable Deep Power-Down
		1	Enable Deep Power-Down

Table 12-12: DMC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	PDREQ	Power Down Request. The DMC_CTL.PDREQ bit enables power-down mode. When the DMC is in power-down mode, any data accesses cause the DMC to generate a bus error. The DRAM remains in power-down mode as along as this bit is 1.	
		0	Disable Power-Down
		1	Enable Power-Down
3 (R/W)	SRREQ	Self-Refresh Request. The DMC_CTL.SRREQ bit enables self-refresh mode. When the DMC is in self-refresh mode, any data accesses cause the DMC to generate a bus error. The DRAM remains in self-refresh mode as along as this bit is 1.	
		0	Disable Self-Refresh
		1	Enable Self-Refresh
2 (R0/W)	INIT	Initialize DRAM Start. The DMC_CTL.INIT bit starts the power up DRAM initialization sequence and DLL calibration sequence. Note that this bit always reads as 0.	
		0	No Effect
		1	Start DRAM Initialization
1 (R/W)	LPDDR	Low Power DDR Mode. The DMC_CTL.LPDDR bit selects whether the DMC operates in low power DDR mode or DDR2 mode.	
		0	DDR2 mode
		1	LPDDR mode

DLL Control Register

The DMC_DLLCTL register holds the programmable parameters associated with the DLLs within the DMC PHY.

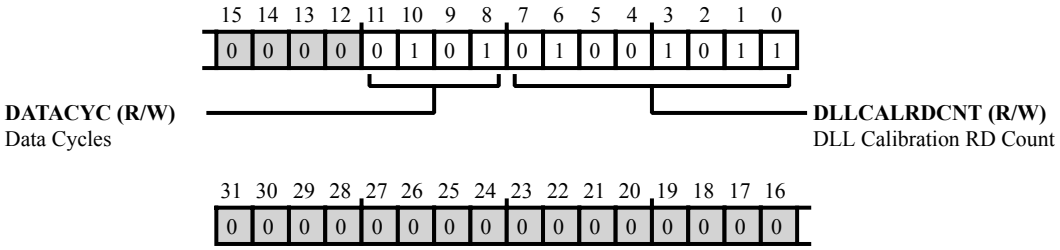

Figure 12-5: DMC_DLLCTL Register Diagram

Table 12-13: DMC_DLLCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:8 (R/W)	DATACYC	Data Cycles. The DMC_DLLCTL.DATACYC bits select the latency after which the DMC reads data from the PHY. This field must be written with the value (5). All other values are reserved. Taking round trip delay into account, the DLL indicates whether a latency of 2 cycles is supported by means of status bits.
7:0 (R/W)	DLLCALRDCNT	DLL Calibration RD Count. The DMC_DLLCTL.DLLCALRDCNT field selects the number of read operations that the PHY uses for DLL calibration.

Efficiency Control Register

The DMC_EFFCTL register control DMC features that improve throughput efficiency. These include features such as auto-refresh management, precharge options, and write data options.

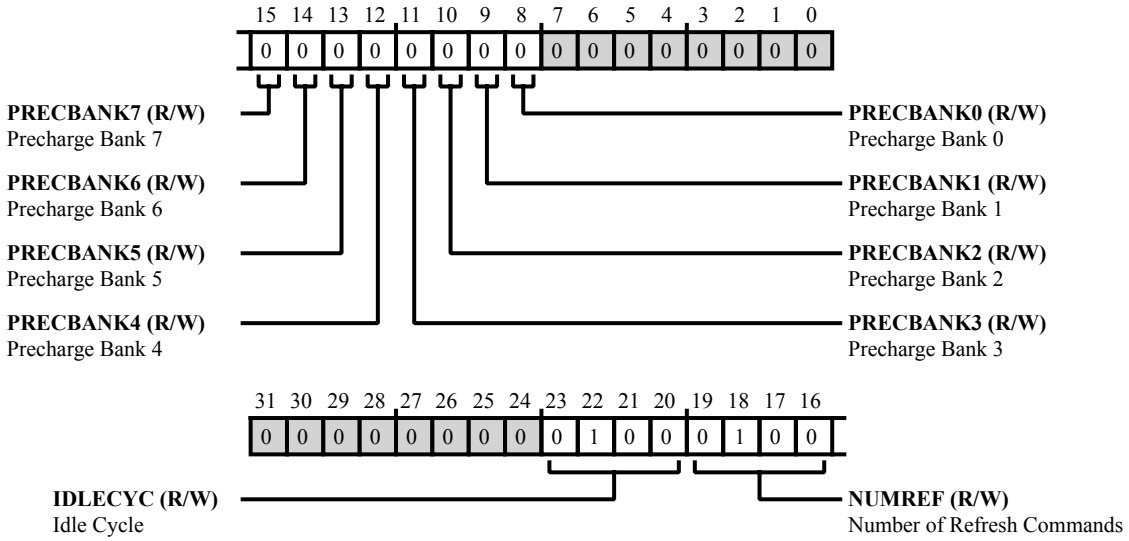

Figure 12-6: DMC_EFFCTL Register Diagram

Table 12-14: DMC_EFFCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:20 (R/W)	IDLECYC	Idle Cycle. The DMC_EFFCTL.IDLECYC bits select the number of cycles after which the DMC issues any accumulated auto-refresh commands if postpone refresh is enabled (DMC_CTL.PPREF =1). When DMC_EFFCTL.IDLECYC is set to 0, the DMC ignores the DMC_CTL.PPREF selection and does not accumulate/postpone periodic auto-refresh commands. Note 1: By default, accumulated auto-refresh commands are issued after counting four idle cycles. Note 2: This value is ignored if DMC_CTL.PPREF is not set. Note 3: Setting this value to 0000 overrides the "postpone refresh" feature and does not accumulate/postpone periodic auto refreshes.	
		0-15	0 to 15 Idle Cycles to Postpone Refresh Commands
19:16 (R/W)	NUMREF	Number of Refresh Commands. The DMC_EFFCTL.NUMREF bits select the number of auto-refresh commands that the DMC can accumulate if postpone refresh is enabled (DMC_CTL.PPREF =1). The number of auto-refresh commands that can accumulate depends on whether the DMC is in DDR2 or LPDDR mode as selected by the DMC_CTL.LPDDR bit. In LPDDR	

Table 12-14: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		mode, the DMC can accumulate up to four auto-refresh commands. In DDR2 mode, the DMC may accumulate up to eight auto-refresh commands. Note 1: By default, accumulated auto-refresh commands are issued after counting four idle cycles. Note 2: This value is ignored if DMC_CTL.PPREF is not set.	
		0	No Refresh Commands Accumulate
		1	1 Refresh Command May Accumulate
		2	2 Refresh Commands May Accumulate
		3	3 Refresh Commands May Accumulate
		4	4 Refresh Commands May Accumulate
		5	5 Refresh Commands May Accumulate
		6	6 Refresh Commands May Accumulate
		7	7 Refresh Commands May Accumulate
		8	8 Refresh Commands May Accumulate
15 (R/W)	PRECBANK7	Precharge Bank 7. The DMC_EFFCTL.PRECBANK7 bit enables precharge (closes the page) of bank 7 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 7
		1	Enable Precharge Bank 7
14 (R/W)	PRECBANK6	Precharge Bank 6. The DMC_EFFCTL.PRECBANK6 bit enables precharge (closes the page) of bank 6 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 6
		1	Enable Precharge Bank 6

Table 12-14: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	PRECBANK5	Precharge Bank 5. The DMC_EFFCTL.PRECBANK5 bit enables precharge (closes the page) of bank 5 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 5
		1	Enable Precharge Bank 5
12 (R/W)	PRECBANK4	Precharge Bank 4. The DMC_EFFCTL.PRECBANK4 bit enables precharge (closes the page) of bank 4 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 4
		1	Enable Precharge Bank 4
11 (R/W)	PRECBANK3	Precharge Bank 3. The DMC_EFFCTL.PRECBANK3 bit enables precharge (closes the page) of bank 3 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 3
		1	Enable Precharge Bank 3
10 (R/W)	PRECBANK2	Precharge Bank 2. The DMC_EFFCTL.PRECBANK2 bit enables precharge (closes the page) of bank 2 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 2
		1	Enable Precharge Bank 2
9 (R/W)	PRECBANK1	Precharge Bank 1. The DMC_EFFCTL.PRECBANK1 bit enables precharge (closes the page) of bank 1 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 1
		1	Enable Precharge Bank 1

Table 12-14: DMC_EFFCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	PRECBANK0	Precharge Bank 0. The DMC_EFFCTL.PRECBANK0 bit enables precharge (closes the page) of bank 0 after each transfer if the DMC precharge feature is enabled (DMC_CTL.PREC =1). Note: The (DMC_CTL.PREC) takes precedence over value in this register. If (DMC_CTL.PREC =1) then all banks are precharged.	
		0	Disable Precharge Bank 0
		1	Enable Precharge Bank 0

Shadow EMR1 Register

The DMC_EMR1 register in the DMC shadows the EMR1 register in the SDRAM when the DMC is in DDR2 mode (DMC_CTL.LPDDR =0). Note that this register must not be used when the DMC is in LPDDR mode (DMC_CTL.LPDDR =1).

If unmasked by the corresponding bit in the shadow mask register (DMC_MSK.EMR1 =1), a write to DMC_EMR1 triggers an extended "mode register set" command on the memory interface. If masked, a write to DMC_EMR1 only updates the register in the DMC, not the register in the SDRAM.

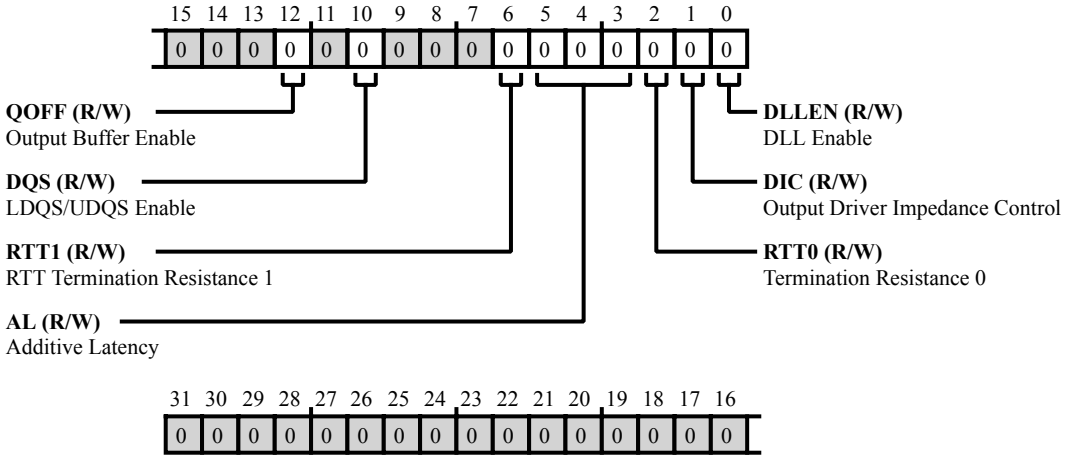

Figure 12-7: DMC_EMR1 Register Diagram

Table 12-15: DMC_EMR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	QOFF	Output Buffer Enable. The DMC_EMR1.QOFF bit enables the SDRAM output pins. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Enable
		1	Disable
10 (R/W)	DQS	LDQS/UDQS Enable. The DMC_EMR1.DQS bit enables operation of the DMC_LDQS/$\overline{\text{DMC\_LDQS}}$ or DMC_UDQS/$\overline{\text{DMC\_UDQS}}$ pin. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Enable
		1	Disable
6 (R/W)	RTT1	RTT Termination Resistance 1. The DMC_EMR1.RTT1 bit combines with the DMC_EMR1.RTT0 bit to set the termination resistance. See the DMC_EMR1.RTT0 bit description for more information.	

Table 12-15: DMC_EMR1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5:3 (R/W)	AL	Additive Latency. The DMC_EMR1.AL bits select a number of added latency time for CAS operations in terms of clock cycles (t_{CK}). For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	0 Clock Cycles Added
		1	1 Clock Cycle Added
		2	2 Clock Cycles Added
		3	3 Clock Cycles Added
		4	4 Clock Cycles Added
		5	5 Clock Cycles Added
2 (R/W)	RTT0	Termination Resistance 0. The DMC_EMR1.RTT0 bit and the DMC_EMR1.RTT1 bits select the SDRAM termination resistance. RTT1=0, RTT0=0: No ODT at memory device RTT1=0, RTT0=1: 75 Ohm ODT at memory device RTT1=1, RTT0=0: 150 Ohm ODT at memory device RTT1=1, RTT0=1: 50 Ohm ODT at memory device For more information about this operation, see the data sheet for the SDRAM being used in your system.	
1 (R/W)	DIC	Output Driver Impedance Control. The DMC_EMR1.DIC bit selects the drive strength mode for the SDRAM. For more information about this operation, see the data sheet for the SDRAM being used in your system. It must be kept at 0 if the SDRAM does not support this bit.	
		0	Full Strength
		1	Reduced Strength
0 (R/W)	DLLEN	DLL Enable. The DMC_EMR1.DLLEN bit enables the DLL in the SDRAM. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Enable DLL (Normal Operation)
		1	Disable DLL (Test/Debug Operation)

Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)

The DMC_EMR2 register in the DMC shadows the EMR2 register in the SDRAM when the DMC is in DDR2 mode (DMC_CTL.LPDDR =0) and shadows the EMR register in the SDRAM when the DMC is in LPDDR mode (DMC_CTL.LPDDR =1). If unmasked by the corresponding bit in the shadow mask register (DMC_MSK.EMR2 =1), a write to DMC_EMR2 triggers an extended "mode register set" command on the memory interface. If masked, a write to DMC_EMR2 only updates the register in the DMC, not the register in the SDRAM.

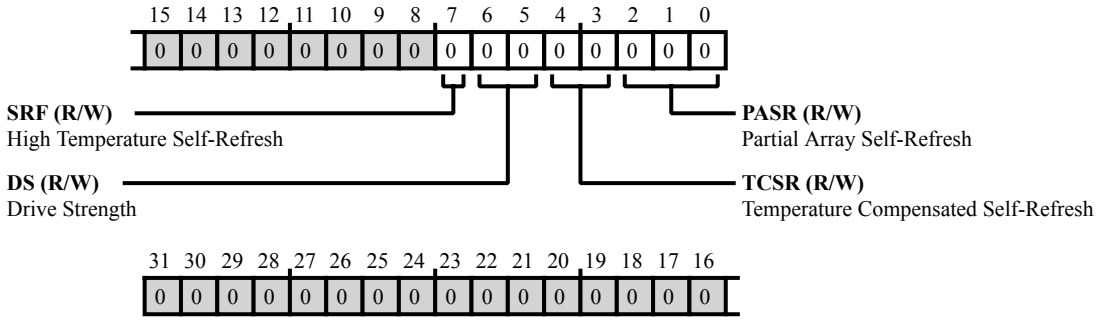

Figure 12-8: DMC_EMR2 Register Diagram

Table 12-16: DMC_EMR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SRF	High Temperature Self-Refresh. The DMC_EMR2.SRF bit enables the SDRAM's high temperature self-refresh rate feature when the DMC is in DDR2 mode. (This bit is reserved in LPDDR mode.) For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Disable
		1	Enable
6:5 (R/W)	DS	Drive Strength. The DMC_EMR2.DS bits select the drive strength value when the DMC is in LPDDR mode. (These bits are reserved when the DMC is in DDR2 mode.) Note that all values other than those shown are reserved. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		4	Octant Drive strength
		0	Full Drive Strength
		1	1/2 Drive Strength
		2	3/4 Drive Strength
		3	1/4 Drive Strength

Table 12-16: DMC_EMR2 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/W)	TCSR	Temperature Compensated Self-Refresh. The DMC_EMR2.TCSR bits select the temperature for applying temperature compensated self-refresh when the DMC is in LPDDR mode. (These bits are reserved when the DMC is in DDR2 mode.) For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	70 degree C (in LPDDR Mode)
		1	45 degree C
		2	15 degree C
		3	85 degree C
2:0 (R/W)	PASR	Partial Array Self-Refresh. The DMC_EMR2.PASR bits select the amount of memory to be refreshed during self-refresh. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Full Array for DDR2 and LPDDR Modes. Applies to both 4 and 8 bank devices.
		1	1/2 Array for DDR2 and LPDDR Modes. For 4 bank devices, BA[1:0] =00 and 01. For 8 bank devices, BA[2:0] = 000, 001, 010, 011
		2	1/4 Array for DDR2 and LPDDR Modes. For 4 bank devices, BA[1:0] = 00. For 8 bank devices, BA[2:0] = 000 and 001.
		3	1/8 Array for 8 DDR2 Banks Only. Reserved for LPDDR. For 4 bank devices, not defined. For 8 bank devices, BA[2:0] = 000.
		4	3/4 Array for DDR2 . Reserved for LPDDR. For 4 bank devices, BA[1:0]=01, 10 and 11. For 8 bank devices, BA[2:0] = 010, 011, 100, 101, 110, and 111.
		5	1/2 Array for DDR2 . 1/8 Array for LPDDR. For 4 bank devices, BA[1:0]=10 and 11. For 8 bank devices, BA[2:0] = 100, 101, 110, and 111.
		6	1/4 Array for DDR2 . 1/16 Array for LPDDR. For 4 bank devices, BA[1:0]=11. For 8 bank devices, BA[2:0] =110 and 111.
		7	1/8 array (for DDR2 Banks only); Reserved (LPDDR)

Shadow MR Register

The DMC_MR register in the DMC shadows the MR register in the SDRAM when the DMC is in DDR2 mode or LPDDR mode (DMC_CTL.LPDDR =0 or =1). If unmasked by the corresponding bit in the shadow mask register (DMC_MSK.MR =1), a write to DMC_MR triggers a “mode register set” command on the memory interface. If masked, a write to DMC_MR only updates the register in the DMC, not the register in the SDRAM.

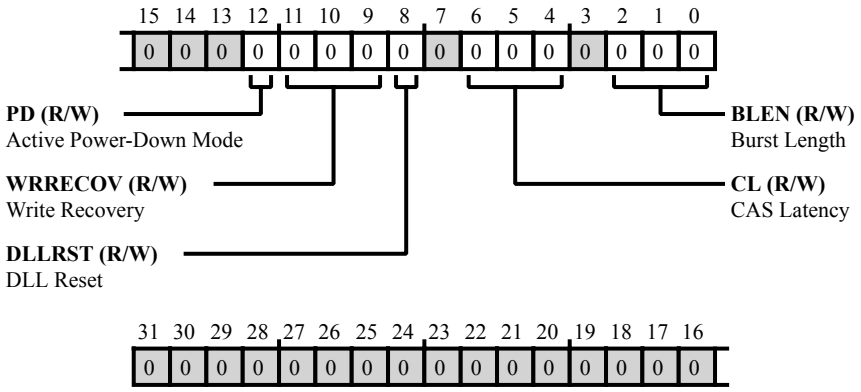

Figure 12-9: DMC_MR Register Diagram

Table 12-17: DMC_MR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	PD	Active Power-Down Mode. The DMC_MR.PD bit selects the active power-down mode. Note that this parameter applies only for DDR2 mode and is reserved for LPDDR mode. For more information about this mode, see the data sheet for the SDRAM being used in your system.	
		0	Fast Exit (normal)
		1	Slow Exit (low power)
11:9 (R/W)	WRRECOV	Write Recovery. The DMC_MR.WRRECOV bit selects the write recovery time in terms of clock cycles (t_{CK}). Note that this parameter applies only for DDR2 mode and is reserved for LPDDR mode. For more information about this mode, see the data sheet for the SDRAM being used in your system.	
		1	2 Clock Cycles
		2	3 Clock Cycles
		3	4 Clock Cycles
		4	5 Clock Cycles
		5	6 Clock Cycles
		6	7 Clock Cycles

Table 12-17: DMC_MR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		7	8 Clock Cycles
8 (R/W)	DLLRST	DLL Reset. The DMC_MR.DLLRST bit initiates a DLL reset on the SDRAM. Note that this parameter applies only for DDR2 mode and is reserved for LPDDR mode. For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		0	Normal Operation
		1	Reset DLL
6:4 (R/W)	CL	CAS Latency. The DMC_MR.CL bits select latency from the assertion of a read/write signal to the SDRAM until the first valid data on the output from the SDRAM in terms of clock cycles (t_{CK}). For more information about this operation, see the data sheet for the SDRAM being used in your system.	
		2	Reserved
		3	3 Clock Cycle Latency
		4	4 Clock Cycle Latency (DDR2)
		5	5 Clock Cycle Latency (DDR2)
		6	6 Clock Cycle Latency (DDR2)
2:0 (R/W)	BLEN	Burst Length. The DMC_MR.BLEN bits select burst length for transfers. For more information about this operation, see the data sheet for the SDRAM being used in your system. Note that values other than those shown are not supported.	
		2	4-Bit Burst Length
		3	8-Bit Burst Length

Mask (Mode Register Shadow) Register

The DMC_MSK register permits masking (disabling) writes to the MR and EMRn registers in the SDRAM. When masked, writes to these registers go instead to shadow copies of these registers (DMC_MR, DMC_EMR1, DMC_EMR2), which are maintained within the DMC. When a shadow register's corresponding bit is unmasked (enabled), the DMC generates the MRS or EMRS command to transfer the contents of the shadow register (in the DMC) to the actual register (in the SDRAM).

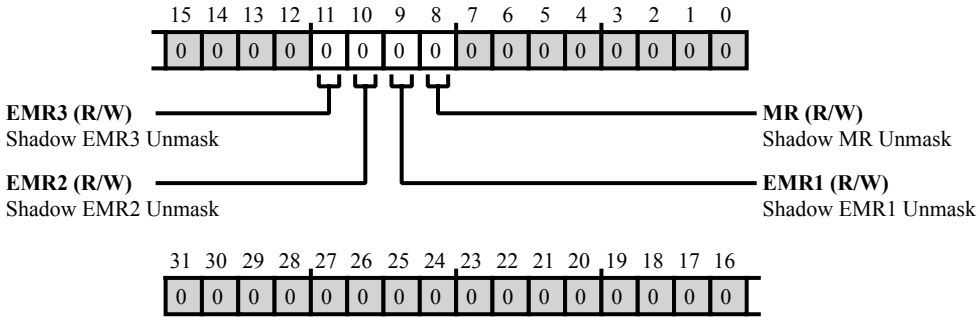

Figure 12-10: DMC_MSK Register Diagram

Table 12-18: DMC_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EMR3	Shadow EMR3 Unmask. The DMC_MSK.EMR3 bit masks or unmasks writes to the EMR3 register (in DDR2) in the SDRAM. When masked, writes to this register instead go to the EMR3 register. When unmasked, the DMC writes the EMR3 value to the EMR3 register (in DDR2) in the SDRAM. After completing the write, the DMC clears this bit. Note that this bit must not be enabled when in LPDDR mode (DMC_CTL.LPDDR =1).	
		0	Mask (Disable) Write to EMR3
		1	Unmask (Enable) Write to EMR3
10 (R/W)	EMR2	Shadow EMR2 Unmask. The DMC_MSK.EMR2 bit masks or unmasks writes to the EMR2 register (in DDR2) or the EMR register (in LPDDR) in the SDRAM. When masked, writes to this register instead go to the DMC_EMR2 register. When unmasked, the DMC writes the DMC_EMR2 value to the EMR2 register (in DDR2) or the EMR register (in LPDDR) in the SDRAM. After completing the write, the DMC clears this bit.	
		0	Mask (Disable) Write to EMR2
		1	Unmask (Enable) Write to EMR2

Table 12-18: DMC_MSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	EMR1	Shadow EMR1 Unmask. The DMC_MSK.EMR1 bit masks or unmasks writes to the EMR1 register in the SDRAM. When masked, writes to this register instead go to the DMC_EMR1 register. When unmasked, the DMC writes the DMC_EMR1 value to the EMR1 register in the SDRAM. After completing the write, the DMC clears this bit. Note that this bit must not be enabled when in LPDDR mode (DMC_CTL.LPDDR =1).	
		0	Mask (Disable) Write to EMR1
		1	Unmask (Enable) Write to EMR1
8 (R/W)	MR	Shadow MR Unmask. The DMC_MSK.MR bit masks or unmasks writes to the MR register in the SDRAM. When masked, writes to this register instead go to the DMC_MR register. When unmasked, the DMC writes the DMC_MR value to the MR register in the SDRAM. After completing the write, the DMC clears this bit.	
		0	Mask (Disable) Write to MR
		1	Unmask (Enable) Write to MR

Priority ID Register 1

The DMC_PRIO register allows transactions from selected masters that generate specific SCB IDs to obtain higher priority than the transactions proceeding in the usual fashion. The contents of the register are masked with the contents of the DMC_PRIOMSK register to obtain a single SCB ID or a range of IDs that get elevated priority.

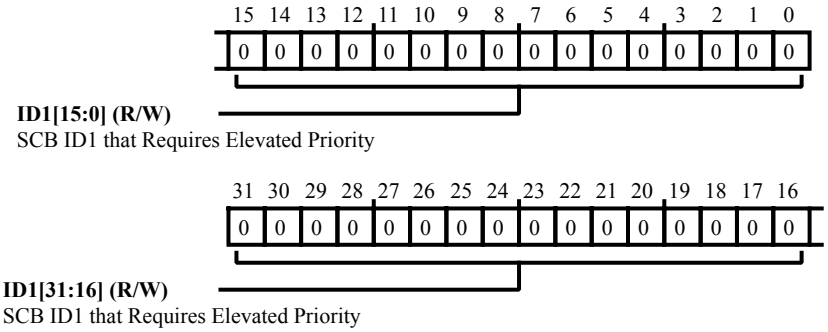

Figure 12-11: DMC_PRIO Register Diagram

Table 12-19: DMC_PRIO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID1	SCB ID1 that Requires Elevated Priority.

Priority ID Register 2

The DMC_PRIO2 register is another register which allows transactions from selected masters that generate specific SCB IDs to obtain higher priority than the transactions proceeding in the usual fashion. The contents of the register are masked with the contents of the DMC_PRIOMSK2 register to obtain a single SCB ID or a range of IDs that get elevated priority.

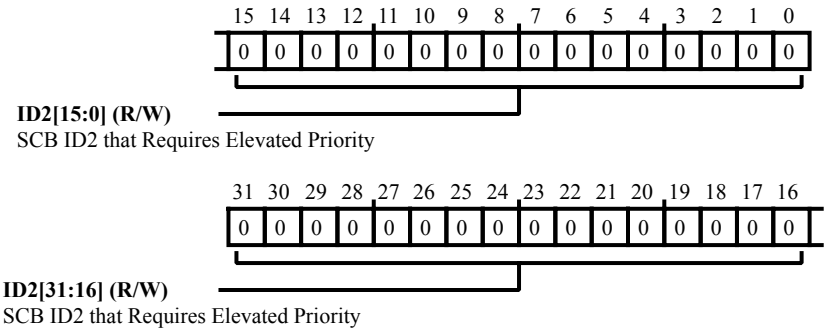

Figure 12-12: DMC_PRIO2 Register Diagram

Table 12-20: DMC_PRIO2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID2	SCB ID2 that Requires Elevated Priority.

Priority ID Mask Register 1

The DMC_PRIOMSK register masks the respective ID bits in the DMC_PRIOMSK register. This masking provides for elevating the access priority of either a single ID or a range of IDs.

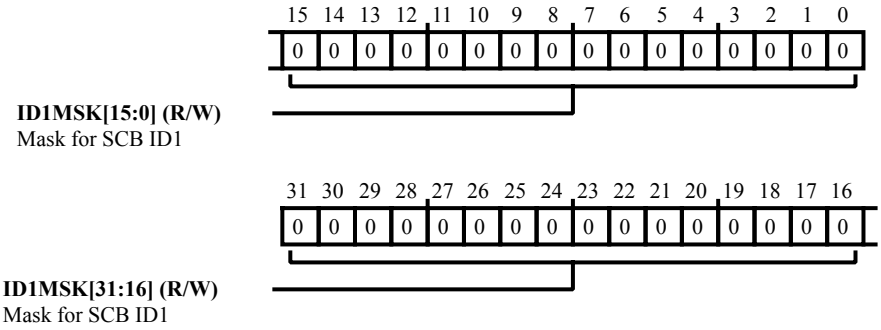

Figure 12-13: DMC_PRIOMSK Register Diagram

Table 12-21: DMC_PRIOMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID1MSK	Mask for SCB ID1.

Priority ID Mask Register 2

The DMC_PRIOMSK2 register bits mask the respective ID bits in the DMC_PRIO2 register. This masking provides for elevating the access priority of either a single ID or a range of IDs.

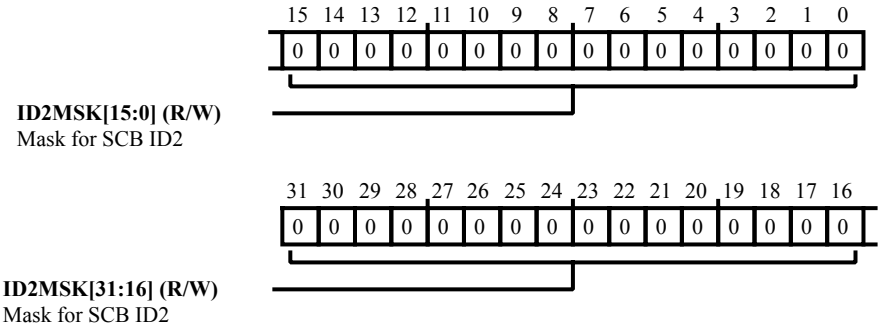

Figure 12-14: DMC_PRIOMSK2 Register Diagram

Table 12-22: DMC_PRIOMSK2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ID2MSK	Mask for SCB ID2.

DMC Read Data Buffer ID Register 1

The DMC_RDDATABUFID1 register allows read transactions from selected masters to make use of DMC read data buffer. The contents of the register are masked with the contents of the DMC_RDDATABUFMSK1 register to obtain a single SCB ID or a range of IDs that get elevated priority.

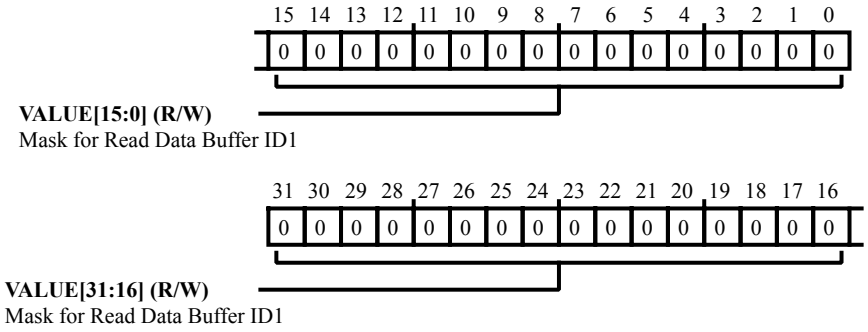

Figure 12-15: DMC_RDDATABUFID1 Register Diagram

Table 12-23: DMC_RDDATABUFID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID1.

DMC Read Data Buffer ID Register 2

The DMC_RDDATABUFID2 register allows read transactions from selected masters to make use of DMC read data buffer. The contents of the register are masked with the contents of the DMC_RDDATABUFMSK2 register to obtain a single SCB ID or a range of IDs that get elevated priority.

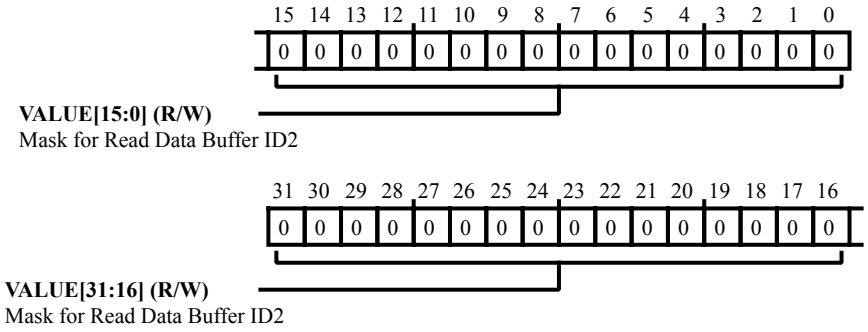

Figure 12-16: DMC_RDDATABUFID2 Register Diagram

Table 12-24: DMC_RDDATABUFID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID2.

DMC Read Data Buffer Mask Register 1

The DMC_RDDATABUFMSK1 register bits mask the respective ID bits in the DMC Priority Mask ID register.

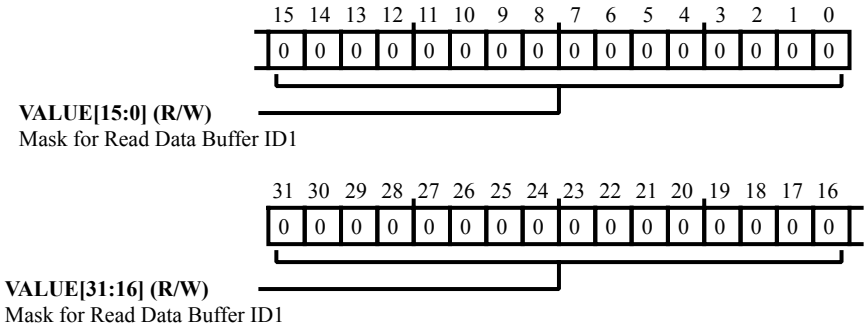

Figure 12-17: DMC_RDDATABUFMSK1 Register Diagram

Table 12-25: DMC_RDDATABUFMSK1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID1.

DMC Read Data Buffer Mask Register 2

The DMC_RDDATABUFMSK2 register bits mask the respective ID bits in the DMC Priority Mask ID register.

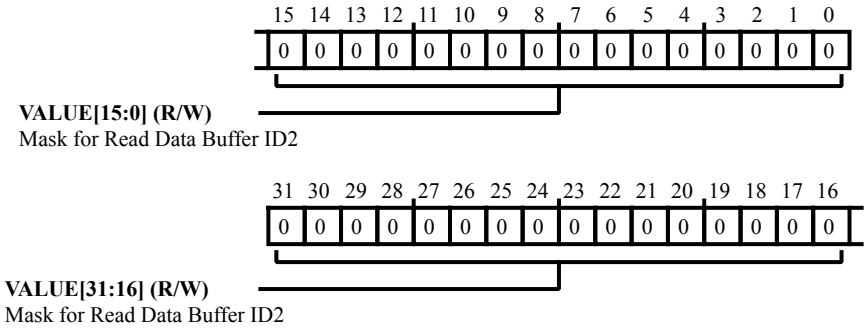

Figure 12-18: DMC_RDDATABUFMSK2 Register Diagram

Table 12-26: DMC_RDDATABUFMSK2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Mask for Read Data Buffer ID2.

Status Register

The DMC_STAT register indicates status for modes selected with the DMC_CTL register and indicates status DMC operations.

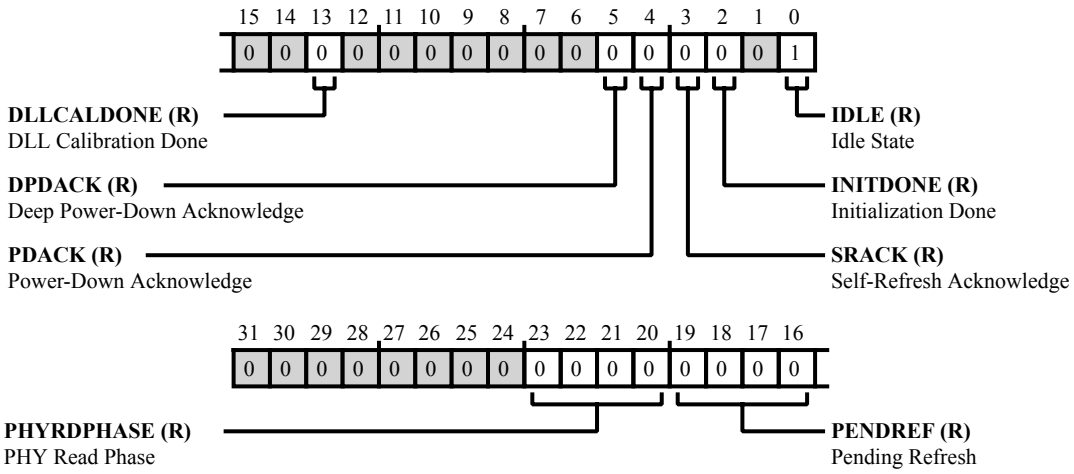

Figure 12-19: DMC_STAT Register Diagram

Table 12-27: DMC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
23:20 (R/NW)	PHYRDPHASE	PHY Read Phase. The DMC_STAT.PHYRDPHASE bits indicate the latency after which the DMC may read from the PHY. Taking round trip delay into account, the DLL indicates the exact number of clock cycles after which the controller needs to read data. Values other than those shown are reserved.	
		2	2 Clock Cycles Latency
		3	3 Clock Cycles Latency
		4	4 Clock Cycles Latency
		5	5 Clock Cycles Latency
		6	6 Clock Cycles Latency
		7	7Clock Cycles Latency
19:16 (R/NW)	PENDREF	Pending Refresh. The DMC_STAT.PENDREF bits indicate the number of pending auto-refresh commands whose value can be from "0000" to "0111". When the DMC is in low power DDR mode (DMC_CTL.LPDDR =1), the maximum value for DMC_STAT.PENDREF is 3.	

Table 12-27: DMC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/NW)	DLLCALDONE	DLL Calibration Done. The DMC_STAT.DLLCALDONE indicates that the PHY DLL calibration sequence is complete.	
		0	No Status
		1	Completed PHY DLL Calibration
5 (R/NW)	DPDACK	Deep Power-Down Acknowledge. The DMC_STAT.DPDACK bit indicates that deep power-down mode is active. Note that this status is available in low power DDR mode (DMC_CTL.LPDDR =1) only.	
		0	Not in Deep Power-Down Mode
		1	Deep Power-Down Mode Active
4 (R/NW)	PDACK	Power-Down Acknowledge. The DMC_STAT.PDACK bit indicates that power-down mode is active.	
		0	Not in Power-Down Mode
		1	Power-Down Mode Active
3 (R/NW)	SRACK	Self-Refresh Acknowledge. The DMC_STAT.SRACK bit indicates that self-refresh mode is active.	
		0	Not in Self-Refresh Mode
		1	Self-Refresh Mode Active
2 (R/NW)	INITDONE	Initialization Done. The DMC_STAT.INITDONE bit indicates that the initialization sequence is complete.	
		0	No Status
		1	Initialize Done
0 (R/NW)	IDLE	Idle State. The DMC_STAT.IDLE bit indicates whether the DMC is idle or busy.	
		0	Busy
		1	Idle

Timing 0 Register

The DMC_TR0 register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and must be programmed before initializing the SDRAM. Note that all values for bit fields in DMC_TR0 are in increments of clock cycle time (t_{CK}).

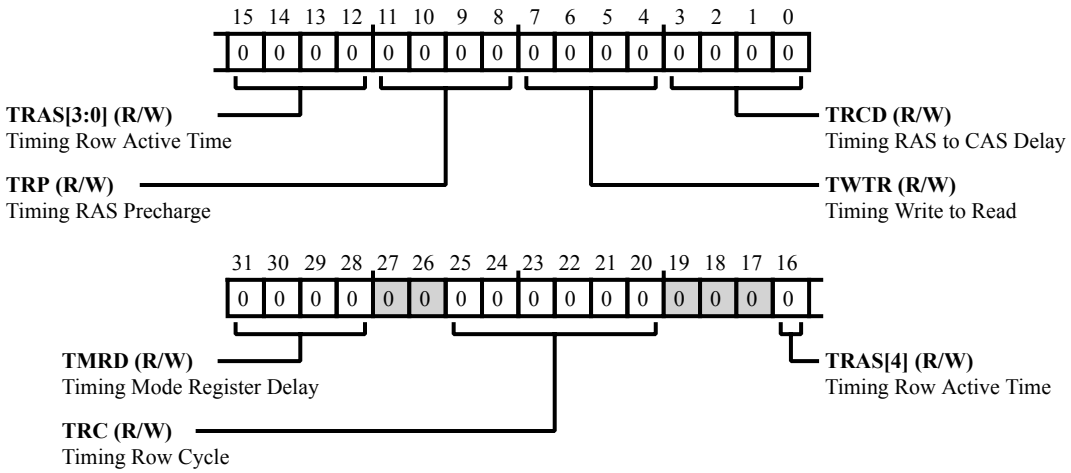

Figure 12-20: DMC_TR0 Register Diagram

Table 12-28: DMC_TR0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/W)	TMRD	Timing Mode Register Delay. The DMC_TR0.TMRD field selects the set-to-active timing parameter (t_{MRD}), which is the number of clock cycles that occur after the mode registers in the SDRAM are set and before the next command is issued.
25:20 (R/W)	TRC	Timing Row Cycle. The DMC_TR0.TRC field selects the active-to-active time (t_{RC}), which is the minimum number of clock cycles that occur from an active command to the next active command in the same bank.
16:12 (R/W)	TRAS	Timing Row Active Time. The DMC_TR0.TRAS field selects the active-to-precharge time (t_{RAS}), which is the number of clock cycles that occur from an active command until a precharge command is allowed.
11:8 (R/W)	TRP	Timing RAS Precharge. The DMC_TR0.TRP field selects the precharge-to-active time (t_{RP}), which is the number of clock cycles that occur while the SDRAM recovers from a precharge command and becomes ready to accept the next active command.
7:4	TWTR	Timing Write to Read.

Table 12-28: DMC_TR0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		The DMC_TR0.TWTR field selects the write-to-read delay time (t_{WTR}), which is the number of clock cycles that occur from the last write data to the next read command.
3:0 (R/W)	TRCD	Timing RAS to CAS Delay. The DMC_TR0.TRCD field selects the RAS to CAS delay time (t_{RCD}), which is the number of clock cycles that occur from an active command to a read/write assertion.

Timing 1 Register

The DMC_TR1 register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and must be programmed before initializing the SDRAM. Note that all values for bit fields in DMC_TR1 are in increments of clock cycle time (t_{CK}).

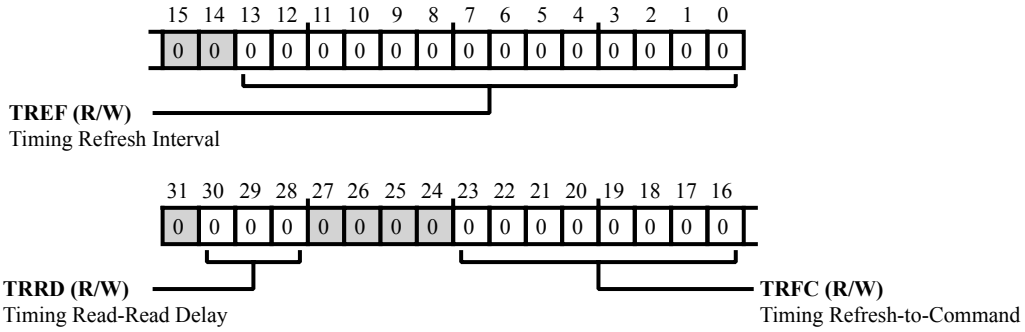

Figure 12-21: DMC_TR1 Register Diagram

Table 12-29: DMC_TR1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
30:28 (R/W)	TRRD	Timing Read-Read Delay. The DMC_TR1.TRRD field selects the active-to-active time (t_{RRD}), which is the minimum number of clock cycles occurring from a bank x active command to a bank y active command.
23:16 (R/W)	TRFC	Timing Refresh-to-Command. The DMC_TR1.TRFC field selects the refresh-to-active command delay (t_{RFC}), which is the number of clock cycles required for the SDRAM to recover from a refresh signal to be ready to take the next command. It is also the number of clock cycles needed for the SDRAM to recover from executing one active command and ready to accept the next active command.
13:0 (R/W)	TREF	Timing Refresh Interval. The DMC_TR1.TREF field selects the refresh interval time (t_{REF}), which is the number of clock cycles occurring from one refresh command to the next refresh command. The actual timing of issuing a precharge command may be delayed by if the SDRAM is processing a normal access. However, the delay is not accumulative so there is no need to shorten the refresh interval to account for the memory access time. The non-accumulative refresh delay typically increases memory bandwidth by a few percentage points.

Timing 2 Register

The DMC_TR2 register selects timing parameters for DMC operation to corresponding with parameters of the SDRAM device that is used in the system. The timing registers must be programmed to match the device for correct operation of the SDRAM and before initializing the SDRAM.

Note that all values for bit fields in DMC_TR2 are in increments of clock cycle time (t_{CK}).

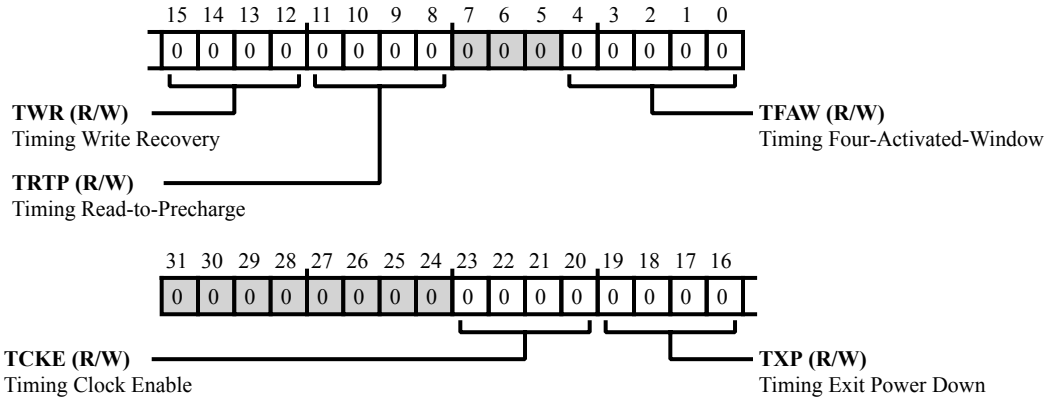

Figure 12-22: DMC_TR2 Register Diagram

Table 12-30: DMC_TR2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:20 (R/W)	TCKE	Timing Clock Enable. The DMC_TR2.TCKE field selects the CKE minimum pulsewidth (t_{CKE}).
19:16 (R/W)	TXP	Timing Exit Power Down. The DMC_TR2.TXP field selects the exit power down to next valid command time (t_{XP}).
15:12 (R/W)	TWR	Timing Write Recovery. The DMC_TR2.TWR field selects the write recovery time (t_{WR}). Note that this parameter applies to LPDDR only.
11:8 (R/W)	TRTP	Timing Read-to-Precharge. The DMC_TR2.TRTP field selects the internal read to precharge time (t_{RTP}) for DDR2 mode. If the resulting value is less than 2, the register must be programmed with 2. For LPDDR mode, the DMC_TR2.TRTP field should be programmed with zero. Note: The minimum t_{RTP} that must be programmed for DDR2 mode is 2.
4:0 (R/W)	TFAW	Timing Four-Activated-Window. The DMC_TR2.TFAW field selects the four-banks-activated window time (t_{FAW}). No more than four SDRAM banks should be activated within this window.

ADSP-BF70x DMC Register Descriptions

DMCPHY (DMC) contains the following registers.

Table 12-31: ADSP-BF70x DMC Register List

Name	Description
DMC_CAL_PADCTL0	Calibration PAD Control 0 Register
DMC_CAL_PADCTL1	Calibration PAD Control 1 Register
DMC_CAL_PADCTL2	Calibration PAD Control 2 Register
DMC_PHY_CTL0	PHY Control 0 Register
DMC_PHY_CTL1	PHY Control 1 Register
DMC_PHY_CTL2	PHY Control 2 Register
DMC_PHY_CTL3	PHY Control 3 Register
DMC_PHY_CTL4	PHY Control 4 Register
DMC_PHY_CTL5	PHY Control 5 Register
DMC_PHY_STAT0	PHY Status 0 Register
DMC_PHY_STAT3	PHY Status 3 Register
DMC_PHY_STAT4	PHY Status 4 Register
DMC_PHY_STAT5	PHY Status 5 Register

Calibration PAD Control 0 Register

The DMC_CAL_PADCTL0 register sets the pad calibration controls.

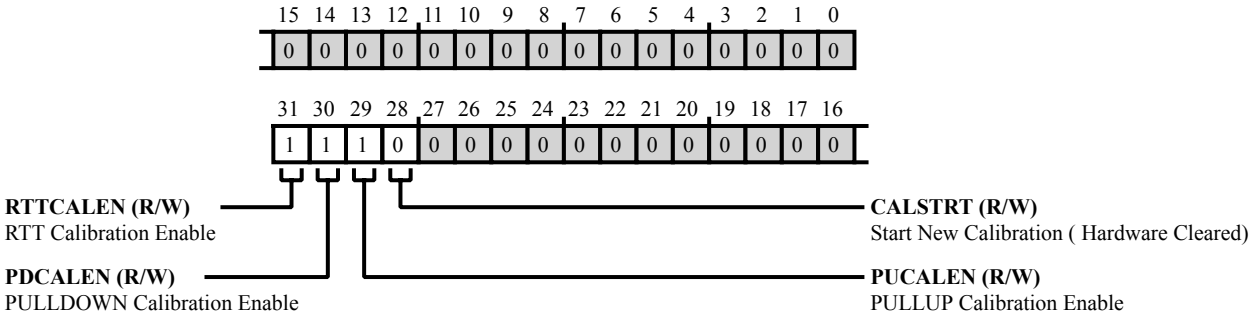

Figure 12-23: DMC_CAL_PADCTL0 Register Diagram

Table 12-32: DMC_CAL_PADCTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	RTTCALEN	RTT Calibration Enable. The DMC_CAL_PADCTL0.RTTCALEN bit is set to 1 at reset. Programming this bit to 0 is not allowed.
30 (R/W)	PDCALEN	PULLDOWN Calibration Enable. The DMC_CAL_PADCTL0.PDCALEN bit is set to 1 at reset. Programming this bit to 0 is not allowed.
29 (R/W)	PUCALEN	PULLUP Calibration Enable. The DMC_CAL_PADCTL0.PUCALEN bit is set to 1 at reset. Programming this bit to 0 is not allowed.
28 (R/W)	CALSTRT	Start New Calibration (Hardware Cleared).

Calibration PAD Control 1 Register

The DMC_CAL_PADCTL1 register sets the pad calibration controls.

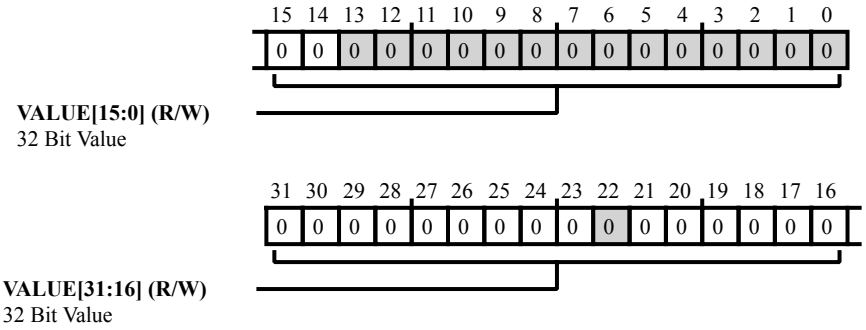

Figure 12-24: DMC_CAL_PADCTL1 Register Diagram

Table 12-33: DMC_CAL_PADCTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

Calibration PAD Control 2 Register

The DMC_CAL_PADCTL2 register sets the pad calibration controls. The DMC pads can be auto-calibrated to the required driver impedance and the On Die Termination (ODT) value using the corresponding bits in this register. These values are translated by the auto calibration logic into a corresponding drive strength control inside the PHY and then routed to the PADS. Auto-calibration starts as soon as the DMC_CAL_PADCTL0.CALSTRT bit is programmed. The DCLK needs to be set at the required frequency before setting the DMC_CAL_PADCTL0.CALSTRT bit.

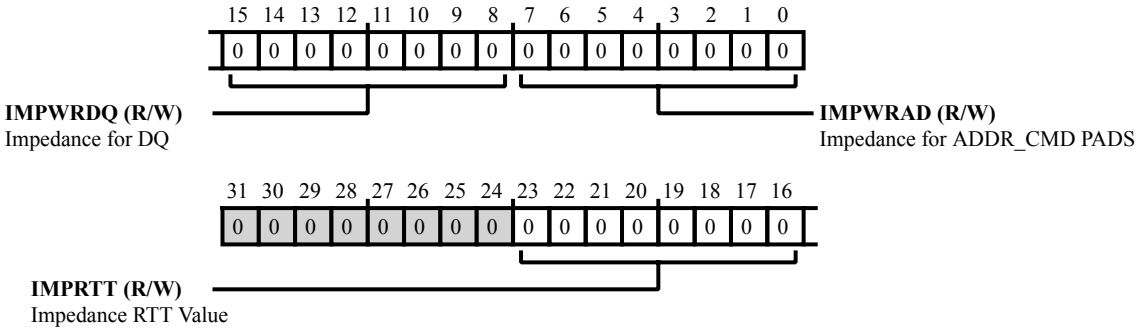

Figure 12-25: DMC_CAL_PADCTL2 Register Diagram

Table 12-34: DMC_CAL_PADCTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	IMPRTT	Impedance RTT Value. Writing to the DMC_CAL_PADCTL2.IMPRTT bit field sets the required initialization sequence to program the termination impedance for the data PADS and the DQS PADS.
15:8 (R/W)	IMPWRDQ	Impedance for DQ. The DMC_CAL_PADCTL2.IMPWRDQ bit field sets the drive impedance for DQ DQS CLK and DM pads. Data pads (DDR_DQ[NN]), DQS pads (DDR_LDQS, /DDR_LDQS, DDR_UDQS, /DDR_UDQS), Clock pads (DDR_CK, /DDR_CK), DM pads (DDR_UDM, DDR_LDM)
7:0 (R/W)	IMPWRAD	Impedance for ADDR_CMD PADS. The DMC_CAL_PADCTL2.IMPWRAD bit field sets the desired drive for address pads (DDR_A[NN]), Command pads (DDR_RAS, DDR_CAS, DDR_CKE, DDR_WE, DDR_CS[N], DDR_ODT).

PHY Control 0 Register

The DMC_PHY_CTL0 register controls programmable PHY features.

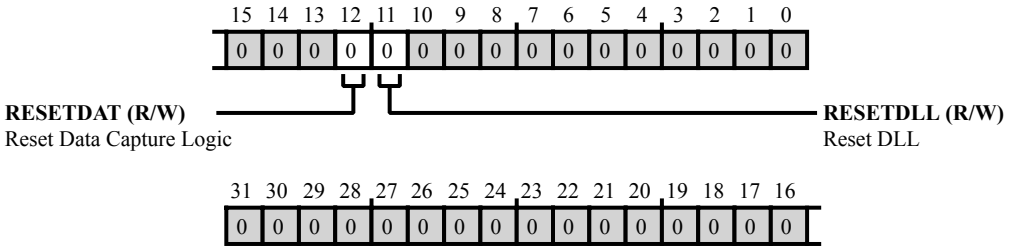

Figure 12-26: DMC_PHY_CTL0 Register Diagram

Table 12-35: DMC_PHY_CTL0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12 (R/W)	RESETDAT	Reset Data Capture Logic. The DMC_PHY_CTL0.RESETDAT bit resets the data capture logic only, including P and N buffers. If Quickboot is used, this bit does not have any effect. The DMC_PHY_CTL0.RESETDAT bit is reset by the hardware. A read of this bit returns zero.
11 (R/W)	RESETDLL	Reset DLL. The DMC_PHY_CTL0.RESETDLL bit resets DLL control logic only, including the 90 degree DQS shifters. If Quickboot is used, this bit does not have any effect. The DMC_PHY_CTL0.RESETDLL bit is reset by the hardware a read of this bit returns zero.

PHY Control 1 Register

The DMC_PHY_CTL1 register controls programmable PHY features.

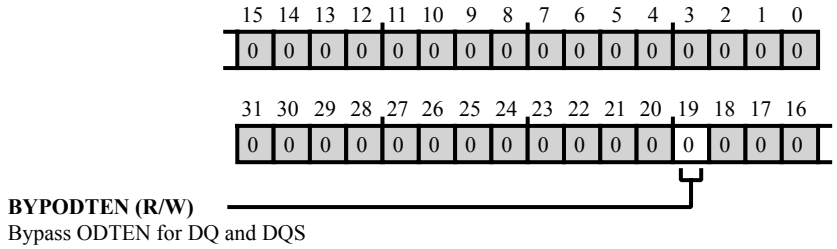

Figure 12-27: DMC_PHY_CTL1 Register Diagram

Table 12-36: DMC_PHY_CTL1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/W)	BYPODTEN	Bypass ODTEN for DQ and DQS.	
		0	Reserved
		1	Reserved

PHY Control 2 Register

The DMC_PHY_CTL2 register controls programmable PHY features. Program this register as per the programming guidelines for proper operation of the DMC.

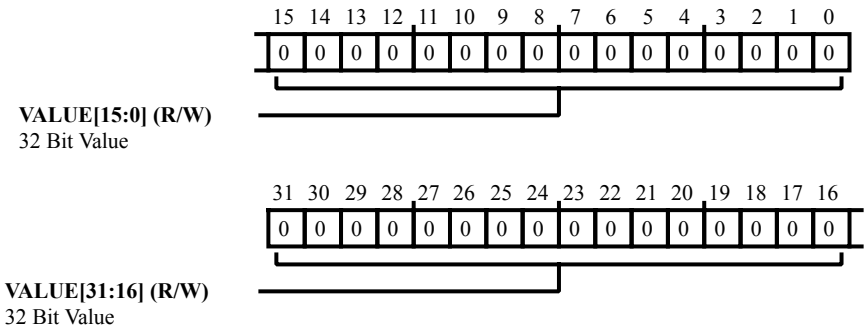

Figure 12-28: DMC_PHY_CTL2 Register Diagram

Table 12-37: DMC_PHY_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

PHY Control 3 Register

The DMC_PHY_CTL3 register controls programmable PHY features. Program this register as per the programming guidelines for proper operation of DMC.

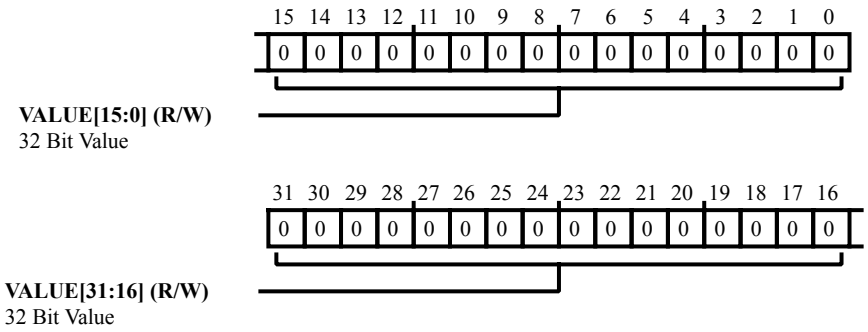

Figure 12-29: DMC_PHY_CTL3 Register Diagram

Table 12-38: DMC_PHY_CTL3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

PHY Control 4 Register

The DMC_PHY_CTL4 register controls programmable PHY features.

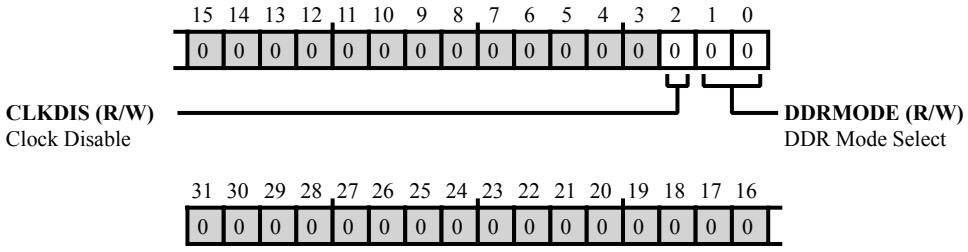

Figure 12-30: DMC_PHY_CTL4 Register Diagram

Table 12-39: DMC_PHY_CTL4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	CLKDIS	Clock Disable. The DMC_PHY_CTL4.CLKDIS bit enables and disables the DDR clock.	
		0	Enable Clock
		1	Disable Clock
1:0 (R/W)	DDRMODE	DDR Mode Select. The DMC_PHY_CTL4.DDRMODE bit field selects between the various DDR modes. Not all modes are available on all processors.	
		1	DDR2 Mode
		2	Reserved
		3	LPDDR Mode

PHY Control 5 Register

The DMC_PHY_CTL5 register controls programmable PHY features.

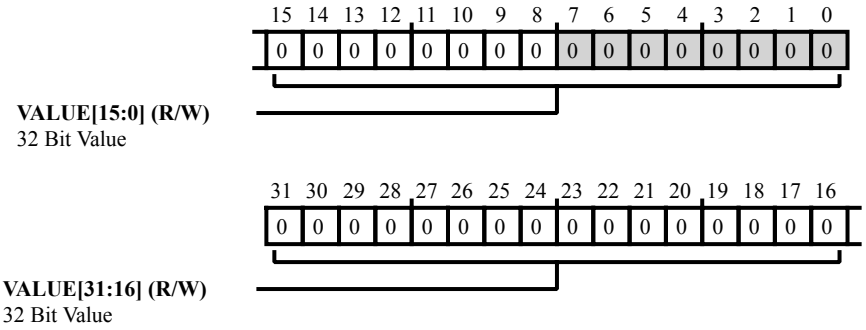

Figure 12-31: DMC_PHY_CTL5 Register Diagram

Table 12-40: DMC_PHY_CTL5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

PHY Status 0 Register

The DMC_PHY_STAT0 register indicates status of PHY operations.

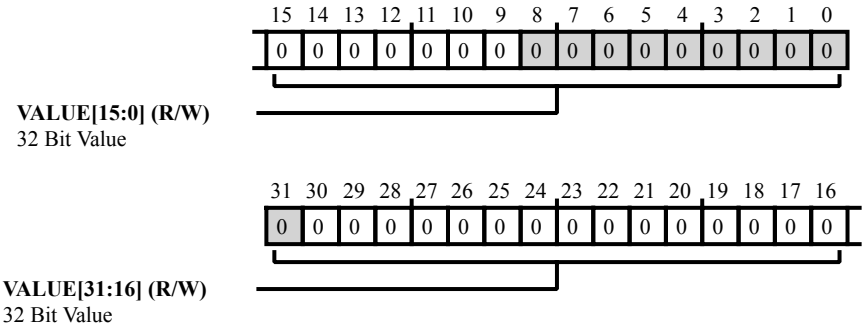

Figure 12-32: DMC_PHY_STAT0 Register Diagram

Table 12-41: DMC_PHY_STAT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

PHY Status 3 Register

The DMC_PHY_STAT3 register indicates the status of PHY operations.

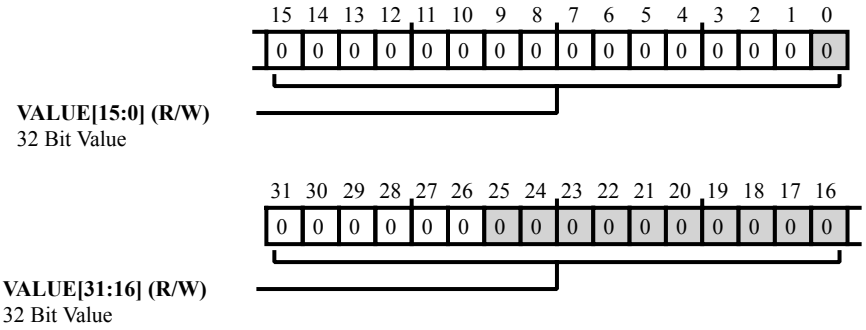

Figure 12-33: DMC_PHY_STAT3 Register Diagram

Table 12-42: DMC_PHY_STAT3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

PHY Status 4 Register

The DMC_PHY_STAT4 register provides status for debugging purposes.

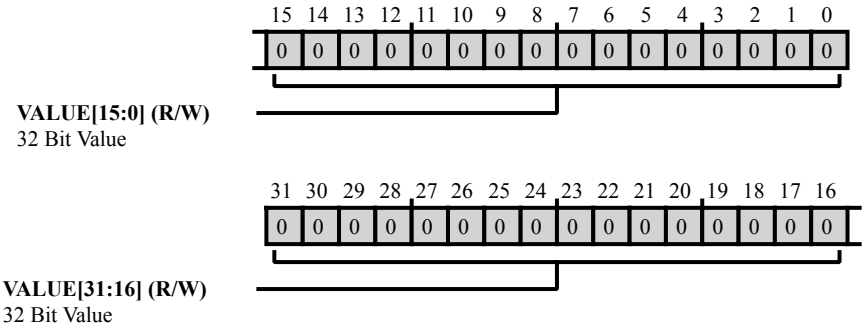

Figure 12-34: DMC_PHY_STAT4 Register Diagram

Table 12-43: DMC_PHY_STAT4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

PHY Status 5 Register

The DMC_PHY_STAT5 register provides status for debugging purposes.

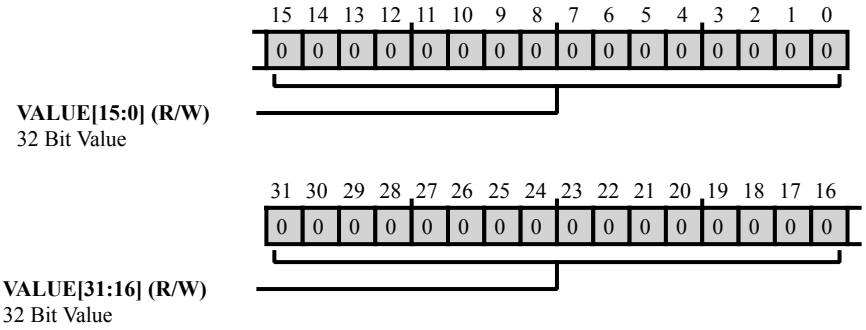

Figure 12-35: DMC_PHY_STAT5 Register Diagram

Table 12-44: DMC_PHY_STAT5 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	32 Bit Value.

13 One-Time Programmable Memory Controller (OTPC)

This chapter describes the operation of the OTP controller. The OTP module is a complete system integrating an OTP memory core with a programming controller, charge pump, and voltage regulator. A built-in Hamming Code Error Correction (ECC), and a fully implemented double-redundant program or read scheme protect the OTP data.

OTP memory access is through the OTP API Overview provided by the ROM.

CAUTION: OTP memory does not support burst transfers, which are required to support cache line fills. As such, OTP memory should not be covered by a cache-enabled DCPLB. If it is, the OTP controller returns an error when a read access is attempted.

OTPC Features

The OTP memory and controller have the following features:

- Built-in redundant read mode
- Built-in integrated power supply
- Built-in Hamming Code Error Correction (ECC)
- Full word serial (single bit at a time) programming with internal VPP

Error Correction

The OTP memory features a Hamming error correction implementation. Signal bit errors are automatically corrected, and dual-bit errors are detected. Refer to OTPC Interrupt Signals.

ECC is always enabled. ECC applies to each 16-bit segment. Because of this functionality, each 16-bit location can only be written to once. Writing to a 16-bit location a second time results in unexpected behavior.

OTP Layout

This section details the memory layout of the OTP memory.

Table 13-1: ADSP-BF70x OTP Layout

Name	Byte Address	Size (bits)	Description
Customer OTP Area - 896 Bytes			
huk	0x0 + 0 - 0x1c	256	Hardware Unique Key
gp0	0x20 + 0 - 0x164	2624	General purpose 0
pvt_128key0	0x168 + 0 - 0x174	128	Customer Privatekey 0 128bits (Encryption AES Key)
pvt_128key1	0x178 + 0 - 0x184	128	Customer Privatekey 1 128bits (Encryption AES Key)
pvt_128key2	0x188 + 0 - 0x194	128	Customer Privatekey 2 128bits (Encryption AES Key)
pvt_128key3	0x198 + 0 - 0x1a4	128	Customer Privatekey 3 128bits (Encryption AES Key)
ek	0x1a8 + 0 - 0x1c4	256	Endorsement Key
secure_emu_key	0x1c8 + 0 - 0x1d4	128	Secure Emulation Key
Reserved	0x1d8 + 0 - 0x1fc	320	Reserved
public_key0	0x200 + 0 - 0x23c	512	Customer Public Key0 (Authentication Key)
public_key1	0x240 + 0 - 0x27c	512	Customer Public Key1 (Authentication Key)
boot_info	0x280 + 0 - 0x2bc	512	Customer Programmable Boot Information
antiroll_nv_cntr	0x2c0 + 0 - 0x2fc	512	AntiRollback NV Counter
gp1	0x300 + 0 - 0x33c	512	General Purpose 1
Reserved	0x340 + 0 - 0x340	24	Reserved
bootModeDisable	0x30 + 24 - 0x343	8	Boot Mode Disable Bits
preboot_ddr_cfg	0x344 + 0 - 0x370	384	User PrebootDDR configuration
stageID	0x374 + 0 - 0x376	48	StageID
Reserved	0x37a + 16 - 0x37a	16	Reserved
Reserved	0x37c + 0 - 0x37c	32	Reserved
fsn	0x380 + 0 - 0x38c	128	Factory Serial Number
Customer BOOT - 116 Bytes			
lock	0x48c + 0 - 0x48c	1	lockbit
Reserved	0x48c + 1 - 0x4fc	927	Reserved

OTPC Event Control

The following sections provide information on OTP events and error management.

OTPC Interrupt Signals

When making 32-bit accesses to OTP memory, a double-bit error in any 16-bit segment triggers the OTPC_INT interrupt.

OTPC Status and Error Signals

The OTP controller does not produce error signals.

OTP API Overview

The ROM provides a set of functions to facilitate OTP field access. The OTP memory is broken up into a set of specialized fields that are described in this section. The API removes the requirement of understanding the details of the layout or OTP access procedures.

All OTP accesses are made through the provided API.

OTP Programming

The OTP programming API provides a simple access, abstracting particulars of the OTP controller.

Any fields that contain zero or null pointers are skipped.

All addresses are assumed to be byte addresses unless otherwise noted.

A list of APIs follows:

`bool adi_rom_otp_pgm(otp_data* data);`	OTP Program
`bool adi_rom_lock();`	Lock API

OTP Program

Program OTP memory using a struct containing the following predefined data fields.

Name	OTP Program	-
PP Define	`FUNC_ROM_OTPPGM`	
Prototype	`bool adi_rom_otp_pgm(otp_data* data);`	-
Argument	data	struct containing data to program OTP with
Return Value	bool	true for programming success
Stack Requirements	valid stack	-

```
bool res = adi_rom_otp_pgm(data);
```

The following type of struct is available for programming. Refer to the ROM header file for the exact definition

```
typedef struct {
    uint32_t (*huk)[8];
    uint32_t (*dtcp_ecc_key)[40];
    uint32_t (*dtcp_const_key)[19];
    uint32_t (*dtcp_dev_key)[32];
    uint32_t (*pvt_128key1)[4];
    uint32_t (*pvt_128key2)[4];
```

```
    uint32_t (*pvt_128key3)[4];
    uint32_t (*pvt_128key4)[4];
    uint32_t (*pvt_192key1)[6];
    uint32_t (*pvt_192key2)[6];
    uint32_t (*public_key0)[16];
    uint32_t (*public_key1)[16];
    uint32_t (*ek)[8];
    uint32_t (*secure_emu_key)[4];
    uint8_t  bootModeDisable;
    uint32_t (*boot_info)[16];
    uint32_t (*gp0)[16];
    uint32_t antiroll_nv_cntr;
    uint8_t  stageID;
    uint32_t (*preboot_ddr_cfg)[11];
} otp_data;
```

NOTE: Make OTP memory a non-cacheable region if the core needs access to it.

OTP Reading

This API provides a unified source for retrieving OTP data fields.

All addresses are assumed to be byte addresses, unless otherwise noted.

A list of APIs follow:

`bool adi_rom_otp_get( OTPCMD cmd, uint32_t* data);`	OTP Get Field

OTP Get Field

Retrieves indicated data from OTP memory.

Name	OTP Get Field	
Prototype	`bool adi_rom_otp_get( OTPCMD cmd, uint32_t* data);`	
Argument	cmd	Indicates what data to fetch, based on the`OTPCMD` enum.
Argument	data	memory location to write the data to
Return Value	bool	true for a successful read
Stack Requirements	valid stack	

```
bool res = adi_rom_otp_get(otpcmd_info,data);
```

The data specified by the `OTPCMD` enum parameter is fetched from OTP memory and placed in the location specified by data. The `OTPCMD` enum contains entries for each field defined in OTP memory, for the most current list please refer to the OTP header file.

An example of the enum style follows:

```
                /*  Field Name                Description                  No of
Bits */
                typedef enum {
                  /* add msi bits */
                  otpcmd_reserved0 = 0,
                  otpcmd_pvt_128key1,      /* Private 128-bit Key 1
[128] */
                  otpcmd_pvt_128key2,      /* Private 128-bit Key 2
[128] */
                  otpcmd_pvt_128key3,      /* Private 128-bit Key 3
[128] */
                  otpcmd_pvt_128key4,      /* Private 128-bit Key 4
[128] */
                  otpcmd_pvt_192key1,      /* Private 192-bit Key 1
[192] */
                  otpcmd_pvt_192key2,      /* Private 192-bit Key 2
[192] */
                  otpcmd_huk,              /* Hardware Unique Key
[256] */
                  otpcmd_ek,               /* Endorsement Key (EK)
[256] */
                  otpcmd_dtcp_ecc_key,     /* DTCP Key (ECC parameters)
[1280] */
                  otpcmd_dtcp_dev_key,     /* DTCP Key (device specific keys)
[1024] */
                  otpcmd_dtcp_const_key,   /* DTCP Key (constant for content key
[312] */
                  otpcmd_secure_emu_key,   /* Secure Emulation Key
[128] */
                  otpcmd_public_key1,      /* Customer Public Key 1
[512] */
                  otpcmd_public_key2,      /* Customer Public Key 2
[512] */
                  otpcmd_antiroll_nv_cntr, /* Anti-Rollback NV Counter
[32] */
                  otpcmd_nonvolatile_cntr, /* NV Counter
[32] */
                  otpcmd_bootModeDisable,  /* Boot Mode disable
field             [8] */
                  otpcmd_stageID,          /* Stage
ID                          [8] */
                  otpcmd_gp0,              /* General Purpose
[512] */
                  otpcmd_boot_info,        /* Customer Programmable Boot info
[384] */
                  otpcmd_preboot_ddr_cfg,  /* User Preboot DDR configuration
[352] */
```

```
    otpcmd_reserved1        /* invalid */
} OTPCMD;
```

OTP Counters

The OTPC module implements a counter API to allow easy reading or writing of the counter without dealing with the complexities of rewriting OTP memory sections that are ECC protected.

The OTPC module provides two functional APIs for counters. These APIs are not extra; the module uses the same `get` and `pgm` APIs. The APIs are functionally unique in the way that they set and retrieve data as counters in OTP memory.

The API uses a different method to count bits because each bit in OTP memory can only be set =1 once, and the ECC protects each 16-bit unit. This functionality essentially means that each 16-bit unit can only be written to once. Therefore, a counter that can count 0–31 requires 32 × 16 bits of memory.

The API receives and returns the value of the counter as a `uint8_t` binary number. Writing a value less than the current value of the counter or greater than the maximum value results in an error.

To implement this functionality, the driver counts by shifting 1's from the left, treating each block as 1 bit. A three-bit counter is encoded as follows.

bit 2	bit 1	bit 0	Value
0000	0000	0000	0
0001	0000	0000	1
0001	0001	0000	2
0001	0001	0001	3

Lock API

This API locks the device.

Name	Lock API	-
PP Define	`FUNC_ROM_LOCK`	-
Prototype	`bool adi_rom_lock();`	-
Return Value	bool	true for success
Stack Requirements	valid stack	-

```
bool res = adi_rom_lock();
```

Calling this function locks the device, making it a secure. Once locked, the `OTPC_SECU_STATE` register indicates that the part is locked, and access is limited. For more information, refer to the security documentation regarding a locked device.

NOTE: Locked Status. The OTPC_SECU_STATE register is updated only after the part is rebooted. After calling the lock function, the register still indicates that the part is open.

ADSP-BF70x OTPC Register Descriptions

OTP Memory Controller (OTPC) contains the following registers.

Table 13-2: ADSP-BF70x OTPC Register List

Name	Description
OTPC_SECU_STATE	OTP Security State Register
OTPC_STAT	OTP Status Register

OTP Security State Register

The OTPC_SECU_STATE register provides lock status information.

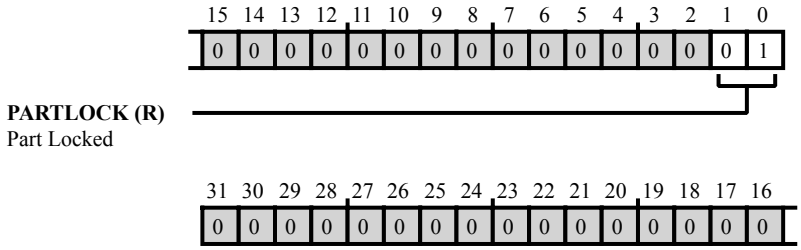

Figure 13-1: OTPC_SECU_STATE Register Diagram

Table 13-3: OTPC_SECU_STATE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/NW)	PARTLOCK	Part Locked. The OTPC_SECU_STATE.PARTLOCK bit indicates a locked part.	
		0	OPEN part
		1	Locked part
		2	Unlocked part

OTP Status Register

The OTPC_STAT register bits indicate errors and flag status and control the protection bits.

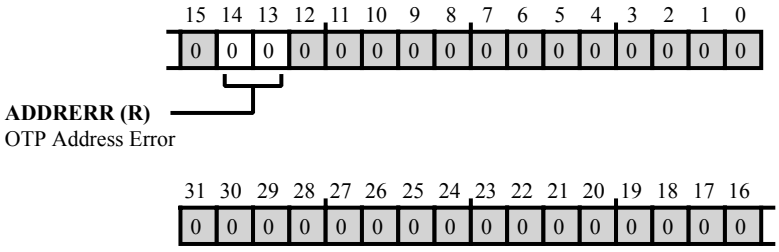

Figure 13-2: OTPC_STAT Register Diagram

Table 13-4: OTPC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
14:13 (R/NW)	ADDRERR	OTP Address Error. The OTPC_STAT.ADDRERR bit field indicates errors which occur when the OTP programming address is out of range or tries to access protected space.	
		0	No error - proper OTP address
		1	OTP address out of range
		2	8-bit OTP address
		3	Protected OTP address

14 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 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 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.

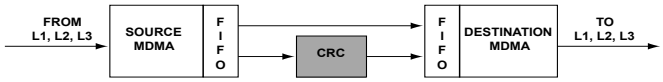

Figure 14-1: Memory Flow

ADSP-BF70x 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 14-1: ADSP-BF70x 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

ADSP-BF70x CRC Interrupt List

Table 14-2: ADSP-BF70x CRC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
63	CRC0_DCNTEXP	CRC0 Datacount expiration	Level	
64	CRC0_ERR	CRC0 Error	Level	
65	CRC1_DCNTEXP	CRC1 Data Count Expiration	Level	
66	CRC1_ERR	CRC1 Error	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.

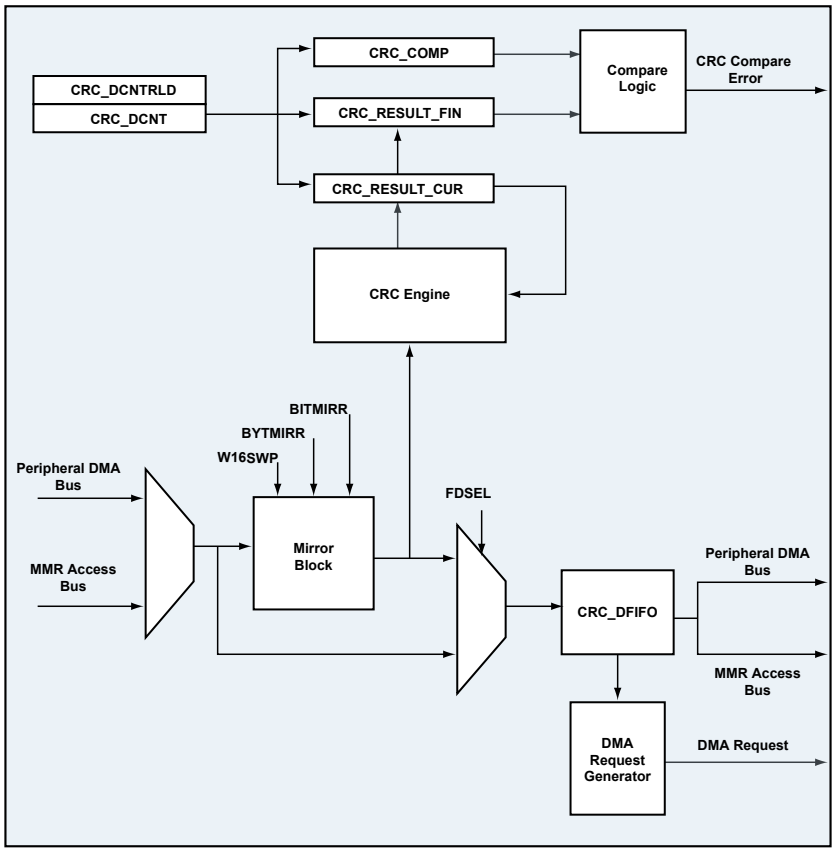

Figure 14-2: CRC Block Diagram

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 flag a system fault.

CRC Architectural Concepts

The CRC peripheral includes a 32-bit CRC engine. The engine implements the reduced table look-up scheme that operates on 32 bits of data per cycle. This functionality results in a single cycle implementation for every 32 bits of data written to the peripheral.

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 14-3: 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]

Table 14-3: Data Mirroring Options (Continued)

W16SWP	BYTMIRR	BITMIRR	Output Data
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 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 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 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 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 interrupts that are optionally enabled within the system event controller. 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 the CRC_STAT register to clear both 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 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

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

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 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

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

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.

ADSP-BF70x CRC Register Descriptions

Cyclic Redundancy Check Unit (CRC) contains the following registers.

Table 14-4: ADSP-BF70x CRC Register List

Name	Description
CRC_COMP	Data Compare Register
CRC_CTL	Control Register

Table 14-4: ADSP-BF70x CRC Register List (Continued)

Name	Description
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.

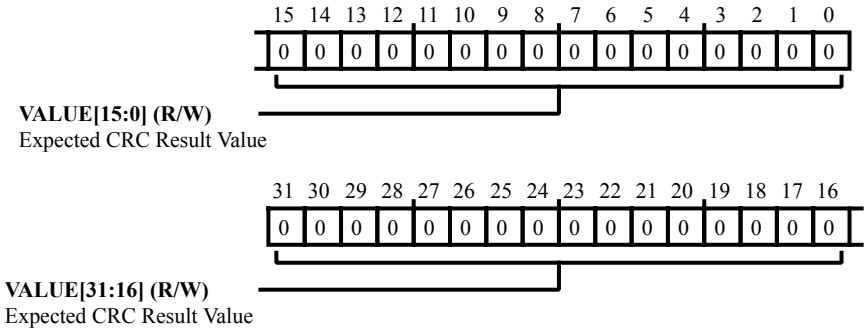

Figure 14-3: CRC_COMP Register Diagram

Table 14-5: 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.

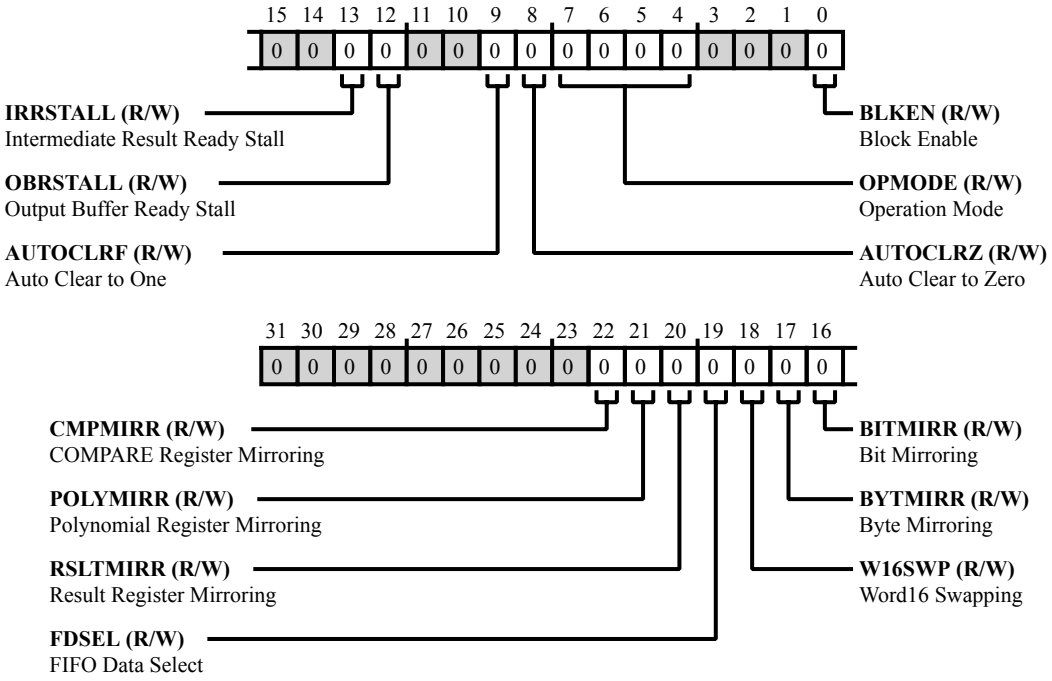

Figure 14-4: CRC_CTL Register Diagram

Table 14-6: 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 14-6: 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 14-6: 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.

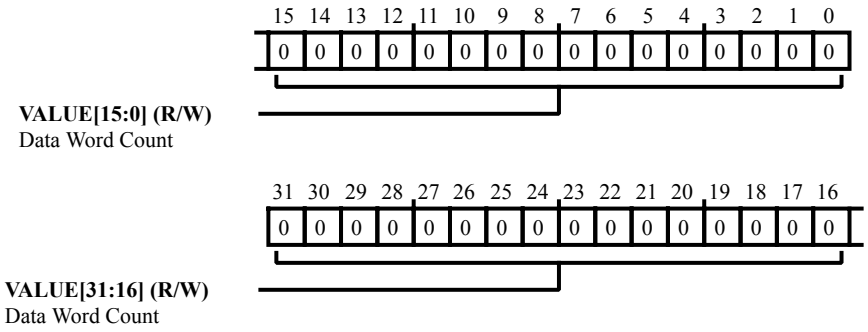

Figure 14-5: CRC_DCNT Register Diagram

Table 14-7: 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.

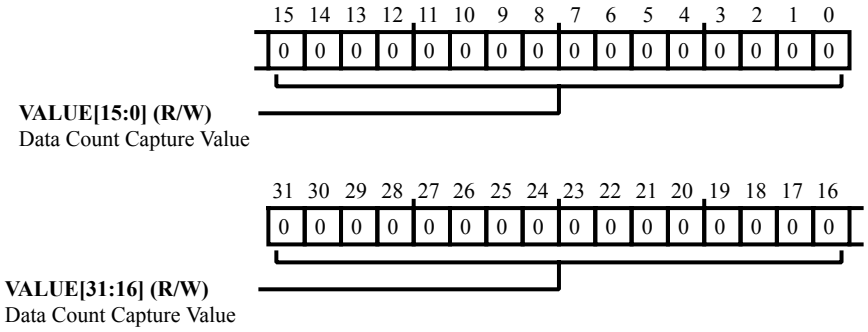

Figure 14-6: CRC_DCNTCAP Register Diagram

Table 14-8: 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.

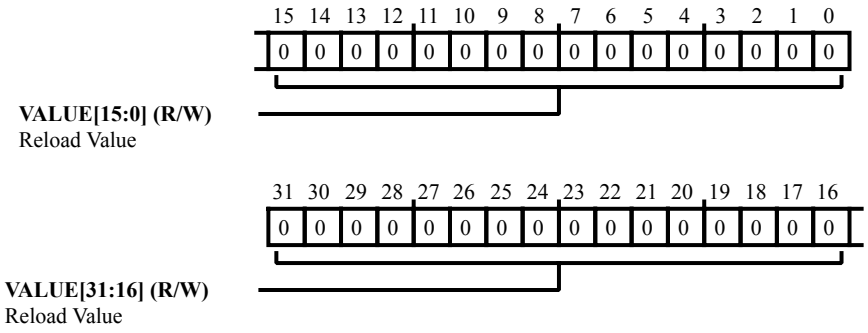

Figure 14-7: CRC_DCNTRLD Register Diagram

Table 14-9: 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.

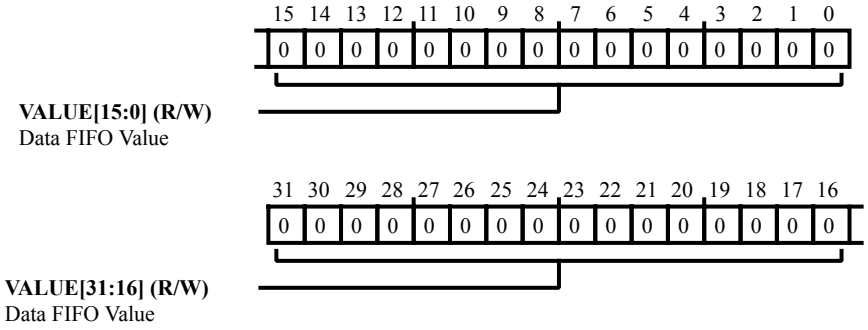

Figure 14-8: CRC_DFIFO Register Diagram

Table 14-10: 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.

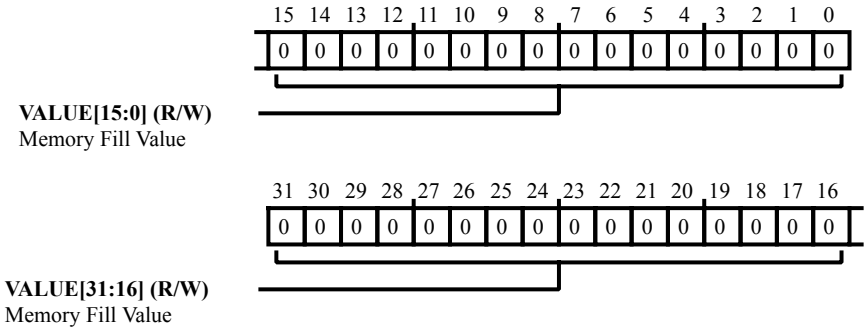

Figure 14-9: CRC_FILLVAL Register Diagram

Table 14-11: 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).

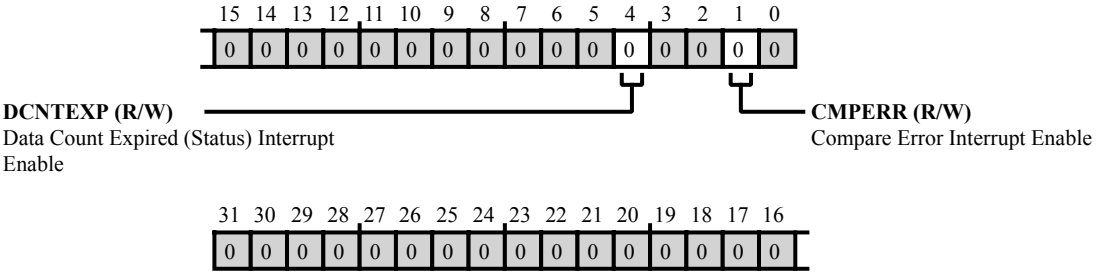

Figure 14-10: CRC_INEN Register Diagram

Table 14-12: 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.

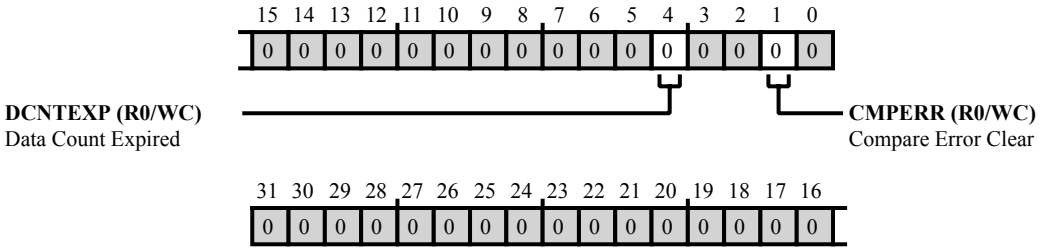

Figure 14-11: CRC_INEN_CLR Register Diagram

Table 14-13: 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.

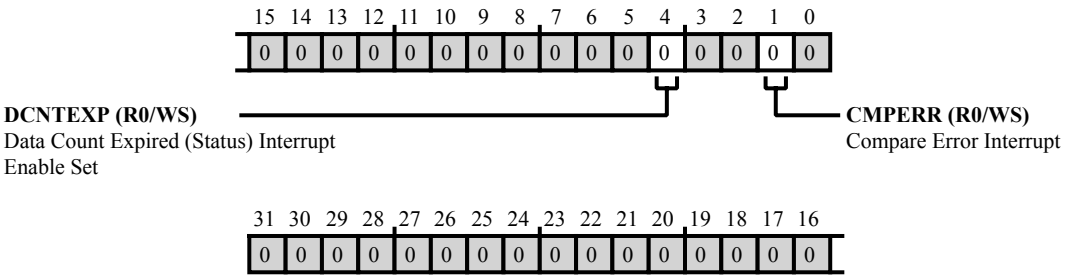

Figure 14-12: CRC_INEN_SET Register Diagram

Table 14-14: 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.

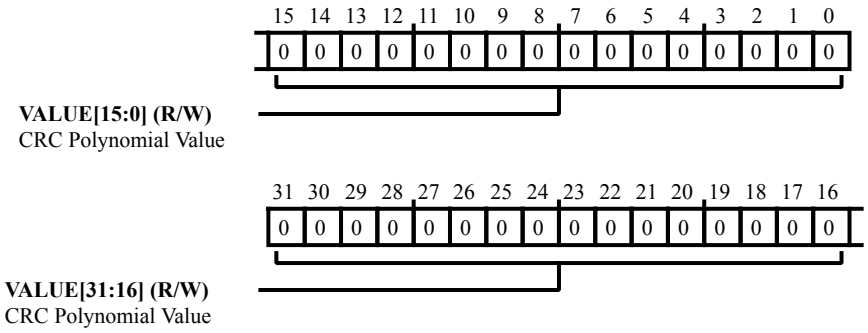

Figure 14-13: CRC_POLY Register Diagram

Table 14-15: 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.

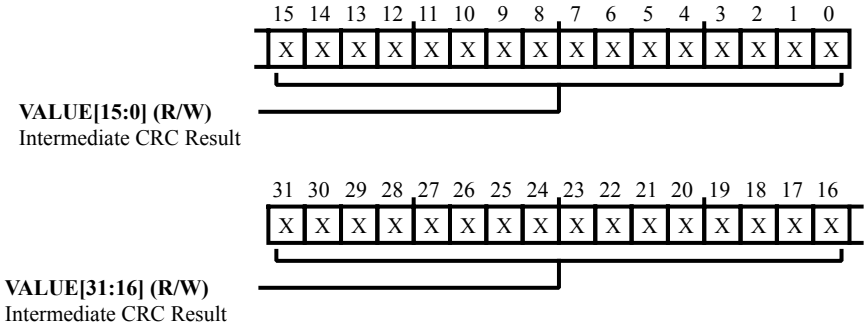

Figure 14-14: CRC_RESULT_CUR Register Diagram

Table 14-16: 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.

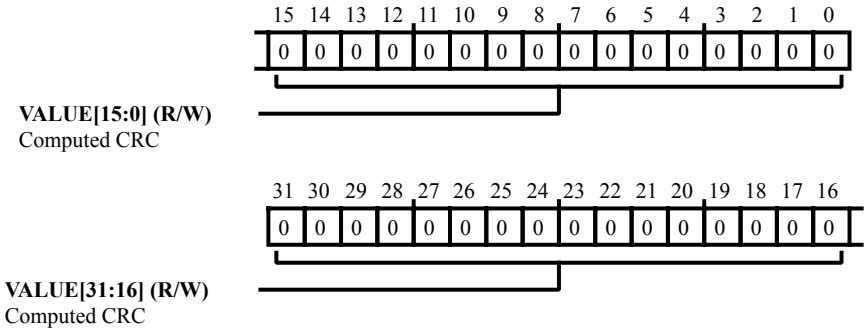

Figure 14-15: CRC_RESULT_FIN Register Diagram

Table 14-17: 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.

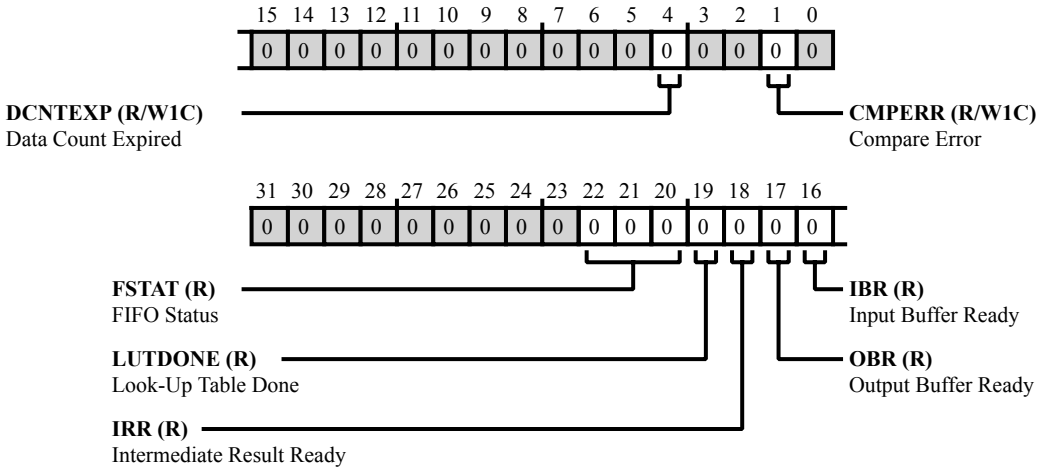

Figure 14-16: CRC_STAT Register Diagram

Table 14-18: 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 14-18: 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

15 Security Packet Engine (PKTE)

The PKTE is a security packet engine designed to off-load the host processor to improve the speed of applications requiring cryptographic processing.

PKTE Features

The PKTE has the following features.

- Hardware assisted processing for the cryptographic ciphers, hashes, and pseudo-random number generation
- Header and trailer processing for Internet security protocols
- DMA capability to move data in and out of the engine efficiently and to allow the engine to run autonomously while moving data
- Interrupt controller to signal module status and errors
- Clock manager for enabling or disabling different features to save power

NOTE: Not all algorithms, decrypt, and hash functions and extra features are available on all product models. For complete information on included features, see the product-specific data sheet.

NOTE: This packet engine provides support for various network security protocols by processing headers and trailers as well as accelerating cryptographic functions. Not all processors have direct support for Ethernet. As such, the packet engine can still be used if Ethernet is indirectly used.

PKTE Functional Description

The packet engine contains a set of modules for encryption and decryption, hashing, and pseudo-random number generation. The following sections describe these functional blocks.

ADSP-BF70x PKTE Register List

The Security Packet Engine (PKTE) provides security-related features. A set of registers governs PKTE operations. For more information on PKTE functionality, see the PKTE register descriptions.

Table 15-1: ADSP-BF70x PKTE Register List

Name	Description
PKTE_BUF_PTR	Packet Engine Buffer Pointer Register
PKTE_BUF_THRESH	Packet Engine Buffer Threshold Register
PKTE_CDRBASE_ADDR	Packet Engine Command Descriptor Ring Base Address
PKTE_CDSC_CNT	Packet Engine Command Descriptor Count Register
PKTE_CDSC_INCR	Packet Engine Command Descriptor Count Increment Register
PKTE_CFG	Packet Engine Configuration Register
PKTE_CLK_CTL	PE Clock Control Register
PKTE_CONT	PKTE Continue Register
PKTE_CTL_STAT	Packet Engine Control Register
PKTE_DATAIO_BUF	Starting Entry of 256-byte Data Input/Output Buffer
PKTE_DEST_ADDR	Packet Engine Destination Address
PKTE_DMA_CFG	Packet Engine DMA Configuration Register
PKTE_ENDIAN_CFG	Packet Engine Endian Configuration Register
PKTE_HLT_CTL	Packet Engine Halt Control Register
PKTE_HLT_STAT	Packet Engine Halt Status Register
PKTE_IMSK_DIS	Interrupt Mask Disable Register
PKTE_IMSK_EN	Interrupt Mask Enable Register
PKTE_IMSK_STAT	Interrupt Masked Status Register
PKTE_INBUF_CNT	Packet Engine Input Buffer Count Register
PKTE_INBUF_INCR	Packet Engine Input Buffer Count Increment Register
PKTE_INT_CFG	Interrupt Configuration Register
PKTE_INT_CLR	Interrupt Clear Register
PKTE_INT_EN	Interrupt Enable Register
PKTE_IUMSK_STAT	Interrupt Unmasked Status Register
PKTE_LEN	Packet Engine Length Register
PKTE_OUTBUF_CNT	Packet Engine Output Buffer Count Register
PKTE_OUTBUF_DECR	Packet Engine Output Buffer Count Decrement Register
PKTE_RDRBASE_ADDR	Packet Engine Result Descriptor Ring Base Address
PKTE_RDSC_CNT	Packet Engine Result Descriptor Count Registers
PKTE_RDSC_DECR	Packet Engine Result Descriptor Count Decrement Registers
PKTE_RING_CFG	Packet Engine Ring Configuration

Table 15-1: ADSP-BF70x PKTE Register List (Continued)

Name	Description
PKTE_RING_PTR	Packet Engine Ring Pointer Status
PKTE_RING_STAT	Packet Engine Ring Status
PKTE_RING_THRESH	Packet Engine Ring Threshold Registers
PKTE_SA_ADDR	Packet Engine SA Address
PKTE_SA_CMD0	SA Command 0
PKTE_SA_CMD1	SA Command 1
PKTE_SA_IDIGEST[n]	SA Inner Hash Digest Registers
PKTE_SA_KEY[n]	SA Key Registers
PKTE_SA_NONCE	SA Initialization Vector Register
PKTE_SA_ODIGEST[n]	SA Outer Hash Digest Registers
PKTE_SA_RDY	SA Ready Indicator
PKTE_SA_SEQNUM[n]	SA Sequence Number Register
PKTE_SA_SEQNUM_MSK[n]	SA Sequence Number Mask Registers
PKTE_SA_SPI	SA SPI Register
PKTE_SRC_ADDR	Packet Engine Source Address
PKTE_STAT	Packet Engine Status Register
PKTE_STATE_ADDR	Packet Engine State Record Address
PKTE_STATE_BYTE_CNT[n]	State Hash Byte Count Registers
PKTE_STATE_IDIGEST[n]	State Inner Digest Registers
PKTE_STATE_IV[n]	State Initialization Vector Registers
PKTE_USERID	Packet Engine User ID

ADSP-BF70x PKTE Interrupt List

Table 15-2: ADSP-BF70x PKTE Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
67	PKTE0_IRQ	PKTE0 Security Packet Engine Interrupt	Level	

PKTE Definitions

Command Descriptor

An 8-word structure that is either written directly into the packet command MMR set or is placed in a Command Descriptor Ring (CDR) in the processor memory. The packet engine sequentially processes the structure. The

command descriptor contains the information that varies for every packet. This information includes pointers to the SA record, the state information, the source packet, and the destination packet.

Command Descriptor Ring (CDR)

A circular contiguous portion of memory which is used to manage one or more command descriptions for the packet engine.

Result Descriptor

When the packet engine completes the processing of a packet, it writes a result descriptor with the state information. The result descriptor can be read directly from the result register set or from the Result Descriptor Ring (RDR) in the processor memory.

Result Descriptor Ring (RDR)

A circular contiguous portion of memory which holds the mirror or copy of the CDR but contains the result descriptors. The RDR and CDR can be overlaid on top of each other.

Security Association (SA) Record

A structure that contains the remainder of the information the packet engine requires to process a packet. Most of the information fields in the SA record such as the key and encryption mode are static for the lifetime of the association. The fields do not require frequent manipulation by the processor core. The SA record non-static fields are the sequence number and sequence number mask. The SA record can have a corresponding state record for saving results from the current operations that are useful for future operations. The state record can hold the IV, the hash byte count, and the intermediate hash digest.

Hash

A cryptographic hash is a function that takes an arbitrary block of data and returns a fixed-size bit string. Four main properties define the function:

- It is easy to compute a hash value for any given input
- It is infeasible to generate the original input from a given hash
- It is infeasible to modify the input without changing the resulting hash
- It is infeasible to find two different inputs that result in the same hash

Autonomous Ring Mode (ARM)

Mode of operation in which most of the parameters as well as the data are set up in memory and moved to the engine for configuration and processing through DMA.

Target Command Mode (TCM)

Mode of operation where some parameters are set up in memory and moved into the packet engine through DMA while the other parameters are directly written to the registers. DMA moves the input and output data in and out of the engine.

Direct Host Mode (DHM)

Mode of operation that does not use DMA. All parameters are directly written to and read from the MMRs. The input and output are written to and read from the FIFO buffers.

Cipher Module

The cipher module does the symmetric encrypt or decrypt operations for:

- Data Encryption Standard (DES)
- Triple-DES
- Advanced Encryption Standard (AES) algorithms

The cipher module supports standard modes for DES and AES that include Electronic Code Book (ECB) and Cipher Block Chaining (CBC). The key size for DES is 56 bits, for Triple-DES is 168 bits. The AES module also provides support for AES counter-modes for IPsec and SRTP. All AES modes can use key sizes of 128 bits and 192/256 bits. Key scheduling is automatic and done in parallel with the encrypt or decrypt operation.

Hash Module

The hash module is tightly coupled with the encrypt or decrypt module and provides hardware accelerated one-way hash functions. Operations that combine both hash and encrypt or decrypt functions are provided to reduce processing time for data that needs both applied. For hash-then-decrypt operations, the packet engine performs parallel execution of both functions from the input buffer. For encrypt-then-hash operations, the processing is pipelined from the input buffer to provide minimum latency. An offset can be specified between the start of hashing and the start of encryption to support protocols such as IPsec or SRTP. The HMAC keyed hashing mechanism is supported for MD5, SHA-1, SHA-2-224 and SHA-2-256. The SSL-MAC is supported for MD5 and SHA-1. A second AES-CBC module for the hash function enables parallel processing for AES-CCM, a combined hash and encrypt algorithm.

Pseudo-Random Number Generator

Cipher algorithms that operate in CBC mode or counter-mode, require an IV. This IV must not be secret; however the IV must be unpredictable and unique for each execution of the encryption process. Pseudo-random number generators are deterministic algorithms that output statistically independent and unbiased numbers. True random number generators are non-deterministic and use the randomness that occurs in a physical process. The packet engine incorporates an ANSI X9.31 compliant Pseudo Random Number Generator (PRNG) that it can use to generate unique IVs using strong encryption. The ANSI X9.31 PRNG is defined as part of the ANSI X9 standards that

are used to secure financial transactions. The function can also be used for pseudo-random number generation as part of an implementation of the digital signature standard described in NIST FIPS PUB 186-2.

The PRNG function, as defined by ANSI X9.31, is based on the AES cipher. This section describes the function to promote understanding of the different inputs and outputs of the PRNG function itself.

NOTE: The PRNG in the packet engine is only based on the AES cipher with 128-bit keys. Other ciphers and key lengths are not supported for the PRNG based on ANSI X9.31.

Let $e \times K(Y)$ represent the AES encryption of Y under the key K.

The PRNG function uses three inputs:

- K, a 128-bit key
- V, a 128-bit seed value
- DT, a 128-bit date/ time vector which is updated on each iteration

The intermediate value I is the result of an AES encryption of the data and time vector under key K.

$I = e \times K(DT)$

That value I is then XOR-ed with the seed V and AES encrypted under key K. The result R is the output of the PRNG function.

$R = e \times K(I \text{ XOR } V)$

A new seed value V is generated from the AES encryption of the result R XOR-ed with the intermediate value I under the key K.

$V = e \times K(R \text{ XOR } I)$

The PRNG function is deeply integrated inside the datapath of the packet engine. The function is controlled indirectly through the PRNG mode bits in the `PKTE_CTL_STAT.PRNGMD` bit field of the command descriptor and the IV source selection bits in the `PKTE_SA_CMD0.IVSRC` bit field of the SA record.

The PKTE module supports four different modes.

1. Load IV from PRNG for the current operation: `PKTE_CTL_STAT.PRNGMD` = 0b00 and `PKTE_SA_CMD0.IVSRC` = 0b11.
2. PRNG init mode initializes the PRNG with a key, seed, and date/time value: `PKTE_CTL_STAT.PRNGMD` = 0b01.

   Before the PRNG function can be used, it must be initialized with a key, seed, and date/time value. At initialization, the key, seed, and date/time values are programmed. Other PRNG operations do not change the key, however the seed, and date/time values are updated (not re-programmed). The date/time is updated every 128 system clock cycles.

   The date/time is a 128-bit value with randomly distributed number of ones and zeros. It must not be all zeros.

The *SA Record for PRNG Init Operation* table shows how the key, seed and date/time values are loaded into the PKTE registers for initialization.

Table 15-3: SA Record for PRNG Init Operation

Parameter	SA Field	Description
K	SA_KEY0	PRNG key [127:96]
	SA_KEY1	PRNG key [95:64]
	SA_KEY2	PRNG key [63:32]
	SA_KEY3	PRNG key [31:0]
V	SA_IDIGEST0	PRNG seed [127:96]
	SA_IDIGEST1	PRNG seed [95:64]
	SA_IDIGEST2	PRNG seed [63:32]
	SA_IDIGEST3	PRNG seed [31:0]
DT	SA_ODIGEST0	PRNG date/time [127:96]
	SA_ODIGEST1	PRNG date/time [95:64]
	SA_ODIGEST2	PRNG date/time [63:32]
	SA_ODIGEST3	PRNG date/time [31:0]

3. PRNG generate mode generates pseudo-random data on initialized key, seed, and date/time value: `PKTE_CTL_STAT.PRNGMD` = 0b10.

   The PRNG function can be used to generated pseudo-random data for other purposes than IVs. For this mode, the PRNG must have been initialized once with the PRNG init mode.

   The PRNG generate mode uses the initialized key and unique changing date/time value as inputs. The pseudo-random data is output to the output buffer of the packet engine.

   The `LEN` field in the command descriptor indicates the amount of pseudo random data that is generated in multiples of 16 bytes. The maximum is 255 × 16 = 4080 bytes.

   In autonomous ring mode, the output data is copied to the host memory at the destination address in the command descriptor. In direct host mode, the host must read the data directly from the output buffer. No SA record is used for this function.

   Directly after the PRNG generate mode, a new pseudo-random number is generated and available for the next operation that uses the option `PKTE_SA_CMD0.IVSRC = PRNG`.

4. PRNG test mode generates pseudo-random data on initialized key, seed, and input (test) data: `PKTE_CTL_STAT.PRNGMD` = 0b11.

   The PRNG test mode can be used to test the correctness of the PRNG function. This mode is similar to the PRNG generate mode, except that the data is read from the input buffer of the packet engine, instead of the date/time value.

For this mode, the PRNG must have been initialized once with the PRNG Init mode.

The LEN field in the command descriptor indicates the amount of pseudo-random data to be generated in multiples of 16 bytes. The maximum is limited to the LEN field in bytes.

In autonomous ring mode, the output data is copied to the host memory at the destination address in the command descriptor. In the direct host mode, the host must read the data directly from the output buffer. No SA record is used for this function.

Directly after the PRNG test mode, a new pseudo-random number is generated and available for the next operation that uses the option PKTE_SA_CMD0.IVSRC = PRNG.

Packet Engine Processing Details

This section describes data processing through the packet engine. It describes padding and supported algorithms for each protocol.

A valid Security Association (SA) must be created before packet processing can start. A formatted SA record must reside in memory and be accessible to the packet engine. The host processor application is responsible for these tasks.

Crypto Padding

Padding is the process of adding data to fill-out a fixed-size plain text data structure. Three factors determine when to use a pad field:

1. If a block cipher encryption algorithm is used, a pad field is used to expand the plain text to a multiple of the block size.
2. Padding can be used to ensure that the cipher text terminates on an n-byte boundary.
3. Padding can conceal the actual length of the payload.

To facilitate peak encrypt or decrypt performance, the packet engine supports the following most commonly used padding functions in hardware:

1. Pad generation and insertion of pad bytes to the end of plain text prior to encryption, for outbound operations.
2. Pad verification to check for correct padding after decrypting a packet for inbound operations.
3. Pad consumption to strip the pad bytes from the plain text data after decrypting a packet, for inbound operations.

Pad Generation and Insertion

The pad type and how many bytes the packet engine inserts depends on the plain text length and the value of the following fields.

- PKTE_SA_CMD0.PADTYPE and PKTE_SA_CMD0.EXTPAD defines the type of padding.
- PKTE_CTL_STAT.PADVAL defines a value that is inserted in the pad.

- PKTE_SA_CMD0.CIPHER enforces a certain pad alignment.
- PKTE_SA_CMD1.CIPHERMD enforces a certain pad alignment.
- PKTE_SA_CMD0.SCPAD allows stream ciphers to be padded.
- PKTE_CTL_STAT.PADCTLSTAT controls the pad alignment.

The PKTE_CTL_STAT.PADCTLSTAT bit field of the result descriptor returns the total number of inserted pad bytes.

Pad Types

The pad type bit field (PKTE_SA_CMD0.PADTYPE) and the extended pad bit (PKTE_SA_CMD0.EXTPAD) select the pad type for the extended protocol group. The packet engine can generate different pad types in hardware as described in the ***Pad Examples*** table.

The PKTE_CTL_STAT.PADVAL bit field, together with the number of pad bytes, defines the value that is inserted in the pad. The format of the pad and the use of this field is best explained in an example (see the ***Pad Examples*** table).

For the IPsec pad type, this field holds the value that is inserted into the next header field (in the ESP trailer) of the innermost operation's header. For the Constant pad type or the Constant SSL pad type, this field holds the inserted fixed constant pad value. For all other pad types, this field is not used and must be zero.

Table 15-4: Pad Types

Pad Type	Value	Description
IPSec	0b000	Append 0 to 255 pad bytes, followed by a pad length byte and a next header byte. The first pad byte appended to the plain text is numbered 1, with subsequent pad bytes making up a monotonically increasing sequence: 1, 2, 3 and up. Append the pad length field that indicates the number of pad bytes (0–255), where a value of zero indicates no pad bytes present. Append the next header byte as specified in the PKTE_CTL_STAT.PADVAL field of the command descriptor. A minimum of 2 bytes are appended; zero pad bytes plus the pad length byte plus the next header byte, in which case the PKTE_CTL_STAT.PADCTLSTAT field in the result descriptor returns 0x02. A maximum of 257 bytes can be appended, in which case the PKTE_CTL_STAT.PADCTLSTAT field in the descriptor returns 0x01.
PKCS#7	0b001	Appends 1–128 pad bytes with a pad byte value equal to the pad length (1–128).
Constant	0b010	Appends 0–255 pad bytes of a user-specified character to the plain text data. This character is specified in the PKTE_CTL_STAT.PADVAL field of the command descriptor.
Zero	0b011	Appends 0–255 pad bytes of 0x00 to the plain text data.
Constant SSL	0b110	Appends 0–255 pad bytes of a user-specified character to the plain text data, followed by a ‘pad length’ byte (0–255). This character is specified in the PKTE_CTL_STAT.PADVAL field of the command descriptor. A total of 256 bytes can be appended, in which case the pad field returns 0x00.

For example, the ***Pad Examples*** table shows the appended pad for any of the pad types for an outbound (encrypt) operation. The table shows a plain text input of 2 bytes using the 8-byte block cipher crypto-algorithm DES-ECB and a PKTE_CTL_STAT.PADVAL field value of 0xAA.

Table 15-5: Pad Examples

Pad Type	Pad field (extended to Plain Text)	PKTE_CTL_STAT
IPSec	0x01, 0x02, 0x03, 0x04, 0x04, 0xAA	0x06
PKCS#7	0x06, 0x06, 0x06, 0x06, 0x06, 0x06	0x06
Constant	0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA	0x06
Zero	0x00, 0x00, 0x00, 0x00, 0x00, 0x00	0x06
Constant SSL	0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0x05	0x06

Pad Length

The ***Pad Alignment*** table lists the alignment (boundaries) to which the packet engine will pad, based on:

- The selected crypto-algorithm
- Crypto mode
- The value of the pad stream cipher bit
- The value of pad control

The minimum number of inserted pad bytes depends on the cipher algorithm, selected using the PKTE_SA_CMD0.CIPHER bit field, and the cipher mode, selected using the PKTE_SA_CMD1.CIPHERMD bit field.

For block ciphers, the plain text data is always (as a minimum) padded to the next block boundary. More pad bytes beyond the algorithm or protocol alignment requirements can be inserted using the pad control (PKTE_CTL_STAT.PADCTLSTAT) in the command descriptor. This feature can be used for *traffic flow security* to conceal the number of plain text bytes in an encrypted packet.

Encrypt operations that use block ciphers have minimum pad requirements based on their block size. The packet engine enforces a minimum pad alignment for block ciphers according to the ***Pad Alignment*** table. For ESP outbound operations, the minimum pad alignment is forced to 4 bytes.

For stream ciphers and null crypto the data is never padded when the stream cipher pad bit (PKTE_SA_CMD0.SCPAD) =0. When PKTE_SA_CMD0.SCPAD =1 the plain text data is padded to the length as defined by the PKTE_CTL_STAT.PADCTLSTAT bits in the command descriptor.

NOTE: The SSL protocol does not allow padding to exceed the ciphers block length. This length is 8 bytes for DES/Triple-DES and 16 bytes for AES. For SSL, the packet engine does not enforce this pad alignment value. The host processor must ensure that the PKTE_CTL_STAT.PADCTLSTAT bit field is configured correctly.

Table 15-6: Pad Alignment

Pad Control PADCTLSTAT = PKTE_CTL_STAT[31:24]			**0x00**	**0x01**[*1]	**0x02**	**0x04**	**0x08**	**0x10**	**0x20**	**0x40**	**0x80**[*2]
Crypto Algorithm	**Crypto Mode**	**Pad Stream Ciphers**	**%8**	**%1**	**%4**	**%8**	**%16**	**%32**	**%64**	**%128**	**%256**
DES	ECB, CBC	N/A	8	8	8	8	16	32	64	128	256
AES	ECB, CBC	N/A	16	16	16	16	16	32	64	128	256
	CTR, ICM with ESP	N/A	8	4	4	8	16	32	64	128	256
	CTR, ICM no ESP	no	0	0	0	0	0	0	0	0	0
		yes	8	0	4	8	16	32	64	128	256
NULL	with ESP	N/A	8	4	4	8	16	32	64	128	256
	no ESP	no	8	0	0	0	0	0	0	0	0
		yes	8	0	4	8	16	32	64	128	256

*1 If `PKTE_CTL_STAT.PADCTLSTAT` is configured for no padding (0x01), it does not mean that no padding bytes are inserted. When PKCS#7 padding is selected, a pad length field with a value =1 is inserted. When SSL or TLS padding is selected, a pad length field with a value =0 is inserted. When IPsec padding is selected, a pad length field is forced to (`PKTE_CTL_STAT.PADCTLSTAT`=0x20). When zero pad and constant pad are selected, no pad bytes are inserted.

*2 Pad type PKCS#7 supports a maximum length of 128 pad bytes, so the packet engine overrules a 256-byte alignment (`PKTE_CTL_STAT.PADCTLSTAT`=0x80) to a 128-byte boundary (`PKTE_CTL_STAT.PADCTLSTAT`=0x40).

The packet engine does not constrain the pad type that is used for an operation; any pad type can be used for each operation. The user must be aware that some protocol specifications only allow specific pad types. The SRTP specification does not have padding defined as padding performed by RTP. The host must pad the RTP packet.

The host software can implement padding that is not supported in hardware. In this case, the host must select the *zero pad* type and set the `PKTE_CTL_STAT.PADCTLSTAT` bit field in the packet descriptor to zero (no padding). The hardware padding engine does not add any bytes. When using a block cipher, the host must insert pad bytes. Then, the data to be encrypted (plain text and pad bytes) are a multiple of the block ciphers boundary. For stream ciphers, any number of pad bytes can be added.

Pad Verification and Consumption

The packet engine can validate a pad type against the expected values. The value of the following bits controls the pad verify function.

- `PKTE_SA_CMD0.PADTYPE` and `PKTE_SA_CMD0.EXTPAD` define the type of padding.
- `PKTE_SA_CMD0.CIPHER` enforces a certain pad alignment.
- `PKTE_SA_CMD1.CIPHERMD` enforces a certain pad alignment.
- `PKTE_SA_CMD0.SCPAD` allows stream ciphers to be padded.

When packet processing is complete, the STATUS byte in the first word of the result descriptor reports the pad verification status. Refer to the Table 15-24 Extended Error Codes - Status Encoding table.

The PKTE_CTL_STAT.PADCTLSTAT bits in the first word of the result descriptor return the total number of detected pad bytes and returns zero for a pad verify error.

When the IPsec pad type is selected, the PKTE_CTL_STAT.PADVAL bit field of the result descriptor returns the next header field. For IPsec ESP outbound operations, this field returns the decimal value 50. For IPsec inbound operations and basic inbound operations that use the IPsec pad mode, the packet engine returns the next header field it detects on the header of the innermost operation. This value is typical for the payload protocol, such as TCP or UDP. However, in bundling scenarios or in IPv6 with destination option headers, another header value could be seen. For all other inbound operations, the returned pad value is zero.

Pad verification is performed for inbound (decrypt) operations that use IPsec, TLS/DTLS or PKCS#7 pad type:

- In combination with a block cipher algorithm: DES-ECB, DES-CBC, AES-ECB, AES-CBC,
- In combination with a stream cipher and stream cipher padding PKTE_SA_CMD0.SCPAD enabled: AES-CTR, AES-ICM
- In combination with null-crypto (no encryption).

Pad Types

The PKTE_SA_CMD0.PADTYPE bit field and the extended pad PKTE_SA_CMD0.EXTPAD bit selects the pad type for the extended protocol group pad type. The packet engine can verify the different pad types in hardware as described in the ***Pad Types*** table.

The constant and zero pad types are not verified since they do not include a pad length field. The SSL pad type is not verified since it does not have a defined pattern.

Table 15-7: Pad Types

Pad Type	SA_CMD0[18, 7:6]	Description
IPSec	0b000	Verify that the pad field includes 0–255 pad bytes, followed by a correct pad length and a next header byte. Verify that pad bytes appended to the plain text are an incremental count, starting at one. Verify that the pad length field is the number of pad bytes (0–255), where a value of zero indicates no pad bytes present. Verify that a next header byte is present as the last byte of the packet, the value is not verified. This is after removal of the ICV. The total number of detected pad bytes is returned in the PKTE_CTL_STAT.PADCTLSTAT field in the result descriptor. **NOTE:** A minimum of 2 bytes must be present, zero pad bytes plus the pad length byte plus the next header byte, in which case the PKTE_CTL_STAT.PADCTLSTAT field in the descriptor returns 0x02. A maximum of 257 bytes can be present, in which case the PKTE_CTL_STAT.PADCTLSTAT field in the result descriptor returns 0x01. The value of the next header byte is returned in the PKTE_CTL_STAT.PADVAL field in the result descriptor.

Table 15-7: Pad Types (Continued)

Pad Type	SA_CMD0[18, 7:6]	Description
PKCS#7	0b001	Verify that the pad field includes 1–128 pad bytes, with a pad byte value equal to the pad length (1–128).

When a block cipher is used and the payload is not padded to the nearest block size boundary, as required by the protocol, a *block size error* is generated for all pad types.

Pad Consumption

The packet engine can optionally consume the decrypted pad bytes for an inbound operation that uses the IPsec, SSL, TLS/DTLS, or PKCS#7 pad types. The pad types constant and zero are not consumed since they do not include a pad length field.

Pad consumption (or stripping) is selected on a flow-by-flow basis in the SA-record with the `PKTE_SA_CMD1.CPYPAD` bit. When this bit is set, the length returned in the LEN field of the result descriptor is the total length of the plain text including the pad. When the `PKTE_SA_CMD1.CPYPAD` is disabled, the detected pad length, as returned in the `PKTE_CTL_STAT.PADCTLSTAT` bits, is subtracted from the total length and then returned in the LEN field of the result descriptor.

The pad is always written to the result packet buffer in memory. When the `PKTE_SA_CMD1.CPYPAD` bit is disabled, only the result length is corrected.

Crypto and Hash Algorithms

The packet engine supports a wide range of crypto and hash algorithms to accelerate basic operations and protocol operations. These algorithms are:

- Basic Encrypt and Basic Decrypt Operations
- Basic Hash Operations
- Basic Encrypt-Hash and Basic Hash-Decrypt Operations
- IPSec ESP Operations
- SRTP Operations

The following tables provide allowed algorithm combinations. Those algorithms not listed in the tables are invalid and can give unexpected results.

NOTE: Not all crypto and hash algorithms are available on all product models. For information on algorithm availability, see the product-specific data sheet.

Table 15-8: Algorithms for Basic Encrypt and Basic Decrypt Operations

Crypto Algorithm	Crypto Mode
DES, Triple-DES	ECB, CBC
AES	ECB, CBC, CRT, ICM
NULL	—

Table 15-9: Algorithms for Basic Encrypt and Basic Decrypt Operations

Crypto Algorithm	Crypto Mode
SHA-1	Basic hash, HMAC, SSL-MAC
SHA-224	Basic hash, HMAC
SHA-256	Basic hash, HMAC
NULL	—

Table 15-10: Algorithms for Basic Encrypt-Hash and Basic Hash-Decrypt Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	ECB, CBC	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC
		NULL	—
AES	ECB, CBC, CRT, ICM	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC
		NULL	—
NULL	—	SHA-1	Basic hash, HMAC, SSL-MAC
		SHA-224	Basic hash, HMAC
		SHA-256	Basic hash, HMAC
		MD5	Basic hash, HMAC, SSL-MAC

Table 15-11: Algorithms for IPsec ESP Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
AES	CBC, CTR	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
NULL	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	—

Table 15-12: Algorithms for Basic SSL and Extended SSL Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	SSL-MAC
		MD5	SSL-MAC
		NULL	—
AES	CBC	SHA-1	SSL-MAC
		MD5	SSL-MAC
		NULL	—
NULL	—	SHA-1	SSL-MAC
	—	MD5	SSL-MAC
	—	NULL	—

Table 15-13: Algorithms for Basic TLS, Extended TLS and DTLS Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
DES, Triple-DES	CBC	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
AES	CBC, CTR	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—
NULL	—	SHA-1	HMAC
		SHA-224	HMAC
		SHA-256	HMAC
		MD5	HMAC
		NULL	—

Table 15-14: Algorithms for SRTP Operations

Crypto Algorithm	Crypto Mode	Hash Algorithm	Hash Mode
AES	ICM	SHA-1	HMAC
NULL	—	SHA-1	HMAC

IV Processing

An initialization vector (IV) is necessary to start a cipher stream or a block cipher in any of the streaming modes of operation. The IV must be unique for each packet. The IV ensures that all cipher texts are unique even if produced by the same encryption key. This functionality prevents every packet from needing a unique encryption key.

Depending on the packet engine operation, the IV can be loaded from different sources. The IV format depends on the algorithm and the source of the IV. The ***Format of the IV*** table provides an overview of all IV formats.

Table 15-15: Format of the IV

Algorithm	IV Source (PKTE_SA_CMD0.IVSRC)	Format	IV Offset (PKTE_SA_CMD1.HSHCOFFST)
DES/Triple-DES (CBC)	Previous Result of IV	Internal IV register [63:0]	0x00
	Input Buffer	Input buffer [63:0]	0x02
	Saved IV	State Record Saved IV [63:0]	0x00
	Automatic	PRNG output [63:0]	0x00
AES (CBC)	Previous Result of IV	Internal IV register [127:0]	0x00
	Input Buffer	Input Buffer [127:0]	0x04
	Saved IV	State Record Saved IV [127:0]	0x00
	Automatic	PRNG output [127:0]	0x00
AES (ICM) for Basic and SRTP	Input Buffer	Input Buffer [127:0]	0x04
	Saved IV	State Record Saved IV [127:0]	0x00
AES (CTR) for Basic and IPsec	Input Buffer	SA_NONCE || Input Buffer[63:0] || 0x00000001	0x02
	Saved IV	State Record Saved IV [127:0]	0x00
	Automatic	SA_NONCE || PRNG output [95:32] || 0x00000001	0x00

Notes:

1. The PKTE_SA_CMD1.HSHCOFFST bit field provides the IV offset. The offset is only applicable for basic hash or encrypt operations. For protocol operations, the offset is automatically enforced.
2. AES-CTR: The Nonce value as described in RFC 3686, is mapped to the PKTE_SA_NONCE register. This Nonce value remains constant for the lifetime of the security association.
3. The host processor controls the IV update using the save-IV bit (PKTE_SA_CMD0.SVIV). When part of a packet is processed using a stream cipher and the encrypt or decrypt data is not an integer multiple of the block size, the saved IV is invalid. It must not be used to resume processing the packet.
4. The packet engine supports automatic IV generation for outbound operations. A new IV is generated for every packet with the internal PRNG. This automatic IV generation can be used for all DES, Triple-DES, and AES modes that use an IV, except for AES-ICM mode.
5. For outbound operations, automatic IV generation is the most efficient. No additional I/O is required, and the host processor does not need to provide the IV. When the saved IV option supplies the IV, the IV must be changed for each packet sent. This activity happens when the packet engine writes back the IV to the state record after processing.
6. Outbound IPsec operations put the IV explicitly at the front of the packet. For an inbound IPsec operation, loading from the input buffer is most efficient and always used.

7. CBC processing must not use a predictable IV. Do not use the saved IV and previous result IV options for CBC processing. Refer to RFC 3602 for more details.

Hash State Loading

The hash state can be loaded from various sources, depending on the selected protocol and hash algorithm. The ***Different Sources for Loading the Hash State*** table provides a list of all the options.

Table 15-16: Different Sources for Loading the Hash State

Hash Algorithm **PKTE_SA_CMD0.HASHSRC =**	**From SA** **0b00**	**RESERVED** **0b01**	**From State** **0b10**	**No Load** **0b11**
SHA-1	yes	-	yes	Yes
SHA-224	Yes	-	Yes	Yes
SHA-256	Yes	-	Yes	Yes
Null hash	x	x	x	x

Sequence Number Processing

The packet engine supports sequence number generation and verification for IPsec and extended SSL, TLS, and DTLS protocol operations.

A Sequence Number (SN) is an unsigned number that a sender must implement and a receiver can use to support anti-replay service (replay attacks) for a specific SA. This processing includes detection of the same packet that arrives more than once and detection of packets that arrive in an incorrect sequence and is outside an accepted level of order relative to the last received packet.

The packet engine supports the following options.

- IPsec ESP with 32-bit SN generation, verification, and overflow protection (see RFC 4303).
- Sequence number loaded from SA for outbound operations and is retrieved from the input packet for inbound operations.

The sequence number and sequence number mask fields are part of the SA record.

Sequence Number Processing in IPsec

IPsec uses a 32-bit explicit sequence number that is sent in the packet. For outbound operations, with header processing (`PKTE_SA_CMD0.HDRPROC`) enabled, the packet engine first increments the sequence number from the `PKTE_SA_SEQNUM[n]` registers in the SA, then inserts the sequence number in the packet. Then the packet engine stores the incremented sequence number in the `PKTE_SA_SEQNUM[n]` registers in the SA. The sequence number mask `PKTE_SA_SEQNUM_MSK[n]` registers in the SA are not used.

For outbound operations, with the `PKTE_SA_CMD0.HDRPROC` bit =1 and the anti-replay service bit (`PKTE_SA_CMD1.ENSQNCHK`) =1, the packet engine generates a sequence number error

(PKTE_CTL_STAT.SQNMERR). The packet engine generates the error when the 32-bit sequence number counter overflows (counter is 2^{32}–1 and increments to 0). The host processor must not send the packet.

For outbound operations, when the PKTE_SA_CMD0.HDRPROC bit =0 or the PKTE_SA_CMD1.ENSQNCHK bit =0, the packet engine does not increment and verify a sequence number counter overflow. It, therefore, never generates a sequence number error. When the PKTE_SA_CMD0.HDRPROC bit =0, the packet engine does not update the sequence number and sequence number mask fields in the SA. It expects the host processor to provide the sequence number through the input buffer as part of the header.

For inbound operations, when the PKTE_SA_CMD0.HDRPROC bit =1 and the PKTE_SA_CMD1.ENSQNCHK bit =1, the packet engine verifies the PKTE_SA_SEQNUM[n] bit field against the sequence number in the packet. It uses the PKTE_SA_SEQNUM_MSK[n] value from the SA. Three situations can occur:

1. If the received sequence number falls outside and above the 64-bit sequence number mask, the mask is shifted. The packet engine updates the SA with the received sequence number and the shifted sequence number mask.
2. If the received sequence number falls inside the 64-bit sequence number mask and is not a duplicate sequence number, the corresponding bit in the mask is set. (A duplicate sequence number is the same as one previously received). The packet engine updates the SA with the received sequence number and the updated sequence number mask.
3. If the received sequence number falls outside and below the 64-bit sequence number mask or matches an earlier received number, a sequence number error is generated. The packet engine does not update the sequence number and sequence number mask fields in the SA. The host must discard the packet.

For inbound operations, when the PKTE_SA_CMD0.HDRPROC bit =0 or the PKTE_SA_CMD1.ENSQNCHK bit =0, the packet engine does not increment and verify the sequence number against the sequence number in the packet. It, therefore, never generates a sequence number error. When the PKTE_SA_CMD0.HDRPROC bit =0, the packet engine does not update the sequence number and sequence number mask fields in the SA. It expects the host to provide the sequence number through the input buffer as part of the header.

PKTE Block Diagram

The ***PKTE Block Diagram*** shows the functional blocks within the PKTE.

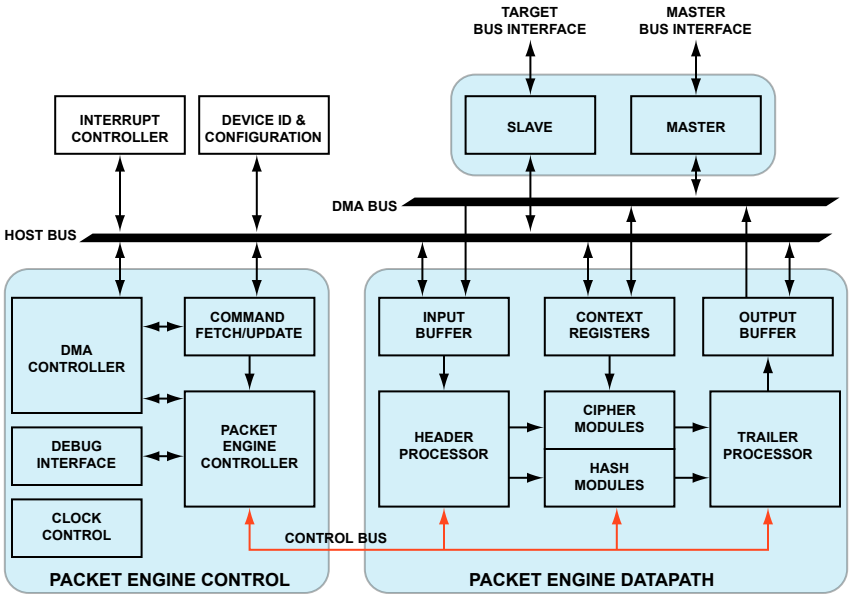

Figure 15-1: PKTE Block Diagram

PKTE Architectural Concepts

The following descriptions provide details on the functional blocks within the security packet engine.

Packet Engine

The packet engine contains symmetric cipher and hash engines. It is optimized to off-load intensive cryptographic operations from the host processor. It can perform parallel and pipelined encryption and hashing operations, reducing the latency, and processing time for packets that need both operations applied. The processor core provides the command information and packet data for the packet engine. The packet engine can run autonomously, using its local DMA controller to perform DMA transfers across the main bus to access the memory of the processor core. The DMA process incorporates flow-control to guarantee proper data flow. Two elements provide the command information that defines the processing for each packet:

- Command descriptor
- SA record

When the packet engine finishes the operation, it updates the SA record, when needed, and provides a result descriptor.

The packet engine has four different modes of operation. The modes give the processor core various levels of control over the command information and packet data transfers to and from the packet engine.

- Autonomous Ring Mode (ARM)
- Target Command Mode (with and without result descriptor ring) (TCM)
- Direct Host Mode (DHM)

Input/Output FIFO Buffers

The data for the packet engine is buffered at both input and output. These buffers decouple the DMA I/O process from the cipher and hash modules inside the packet engine. This functionality enables large DMA burst sizes and allows the crypto-engines to process data during I/O latency periods. Data moves automatically from the input buffer through the encryption and hash engines to the output buffer. If the output buffer is full, the process stops until the data is read and space is available in the output buffer. Each buffer is a 256-byte dual-port RAM.

Parallel Operations

The hash functionality and encrypt or decrypt functionality are tightly coupled. Operations that combine both hash and encrypt or decrypt functions are available to reduce processing time for data that must apply both. For hash-then-decrypt operations, the packet engine performs parallel execution of both functions from the input buffer. For encrypt-then-hash operations, the processing is pipelined from the input buffer to provide minimum latency. An offset can be specified between the start of the hashing and the start of the encryption to support protocols such as IPSec and SRTP.

DMA Controller

The packet engine uses a high-performance DMA controller for autonomous data transfers for:

- Command descriptor reads
- SA record and state record reads
- Packet data read
- Result packet writes
- SA record and state record writes
- Result descriptor writes

Interrupt Controller

The packet engine includes an interrupt controller that, under programmable configuration control, can generate an interrupt on completion of certain operations. Individual interrupts can be masked and cleared. The interrupt registers show both the raw and masked interrupt status of the internal interrupts. The processor core can use interrupts, together with their associated threshold settings, to optimize the overall packet processing in the system. One interrupt can inform the processor core that the input side of the packet engine is almost empty to avoid a stall-on-empty condition. One interrupt can inform the host that the output side is almost full to avoid a stall-on-full condition. The controller uses several interrupts to inform the processor core about errors inside the packet engine. All available interrupts are combined into a single output port as either a level- or edge-active programmable interrupt output.

Clock Controller

The packet engine includes a clock controller that generates clock enable signals. A clock manager external to the packet engine uses the clock enable signals to switch the clocks to modules in the packet engine, reducing power

consumption. The power saving can be significant, depending on the crypto-operation and the idle time of the packet engine. The clock controller generates the clock enable signals dynamically depending on the current crypto-operation. A clock control register provides the processor core the possibility to override this dynamic process.

PKTE Operating Modes

The packet engine can be configured in one of three command modes. For all modes the packet engine can generate an interrupt at completion of packet processing:

- Autonomous Ring Mode (ARM). The core prepares descriptors in the CDR and then initiates a descriptor fetch by triggering the packet engine. When a packet operation is complete, the packet engine writes the result descriptor out into a ring in host memory using the system master bus interface.
- Target Command Mode (TCM). The core directly writes the command descriptors to the packet command register set to initiate a packet operation. This process eliminates an extra DMA transfer to fetch a descriptor, but requires the core to synchronously initiate packet processing. This mode can be configured both without RDR or with RDR. In the latter case, the packet engine writes the result descriptor out into the RDR in host memory using the system master bus interface.
- Direct Host Mode (DHM). The core has full control over the packet engine and uses the system slave bus interface. The core provides all the command descriptors, SA record and state record, and packet input data. When the packet engine completes processing, the core must read the packet output data, the SA record, the state record, and the result descriptors.

Autonomous Ring Mode (ARM)

The *Autonomous Ring Mode* allows the packet engine and the host processor to operate asynchronously. A queue of multiple packets in the host processor memory can be processed continuously to provide the highest possible throughput. The packet engine autonomously fetches the command descriptor, the SA record, and optionally the state record and the input data from host processor memory. After the packet engine finishes processing, it autonomously writes the output data, updates the SA record and state record and writes the result descriptor in the host processor memory. It accesses the host processor memory through DMA read transfers across the system bus master.

This mode uses both command descriptor ring (CDR) and result descriptor ring (RDR).

Physically the CDR, RDR, SA record, source packet, and result packet can all be in different memories depending on the system memory architecture. The host processor writes command descriptors to the CDR in host processor memory. Then, it writes to the `PKTE_CDSC_INCR` register with the number of command descriptors that it prepared in the CDR. This write to the `PKTE_CDSC_INCR` register is the trigger for the packet engine to fetch the command descriptors sequentially from the CDR. When a command descriptor is fetched and written to the internal packet command register set, the descriptor is validated. If the ownership bits are set for the packet engine and the command is valid, processing starts. If not, that command is discarded and a result descriptor is written to the RDR with the error code *invalid command descriptor*. In this mode, the host processor can set a threshold on the CDR and enable an associated interrupt. The packet engine generates an interrupt when the number of command descriptors in the CDR is equal or below the threshold value.

The SA record and state record that contains the crypto context information are stored in a memory area. The packet engine autonomously accesses the memory area through DMA transfers across the system bus master. Also, the source packet and result packet are stored in a memory area that the packet engine autonomously accesses through the same bus master interface.

After decoding the command descriptor, the packet engine fetches the SA record and then, optionally, the state record.

Then, the source packet is fetched and stored in the input buffer. Packets less than the size of the input buffers are fetched entirely at once. Larger packets are fetched in parts that completely fill the input buffer. The packet engine initiates a new fetch each time the number of empty spaces in the input buffer reaches its threshold value. When the first packet data is available in the input buffer, the crypto engines start processing the data. After processing, the crypto engines write the result packet to the output buffer.

The packet engine writes the result packet from the output buffer to host processor memory when the number of bytes in the output buffer reaches its threshold value. Packets less than the threshold value are written entirely at once. Larger packets are written in parts that completely empty the output buffer

The source packet data fetching, data processing, and result packet data writing are parallel processes that continue until the last result packet is written to host processor memory. Then, the packet engine optionally writes the SA record and the state record to update the crypto context information. As a final step, the packet engine writes the result descriptor to the RDR. The host processor must either poll the RDR or wait for an interrupt from the packet engine to determine when packet processing is complete.

Target Command Mode (TCM)

This mode provides a synchronous interface between the processor core and the packet engine. The Command Descriptor Ring (CDR) is disabled and the processor core initiates packet processing by writing the command descriptor directly to the internal command descriptor MMRs of the packet engine. The Result Descriptor Ring (RDR) is optional.

- For `PKTE_CFG.MODE` =01, the RDR is disabled and the processor core reads the result descriptor directly from the internal result descriptor register set for the packet engine.
- For `PKTE_CFG.MODE` =10, the RDR is enabled and stored in a memory area that the packet engine can access through its master bus interface as in autonomous ring mode.

In target command mode, the packet engine autonomously fetches the SA record, state record, source packet data as in autonomous ring mode. Also, as in ARM, after processing, the packet engine updates state fields of the SA record and state record in the host processor memory.

Direct Host Mode (DHM)

This mode provides a synchronous interface between the processor core and the packet engine. The packet engine is under full control of the processor core. The host processor writes the command descriptors, SA record, and state record directly to the packet engine registers. Then, the processor core writes the source packet data into the input buffer. When processing is complete, the processor core reads back the result packet data from the output buffer.

Finally, it reads the result descriptor, the updated SA record, and state record directly from the packet engine registers.

PKTE Event Control

The following section provides information about interrupts in the PKTE module.

PKTE Interrupt Signals

The Packet Engine has an internal Interrupt Controller with 9 interrupt sources. There are 7 registers associated with the interrupt controller:

1. Interrupt Unmasked Status - PKTE_IUMSK_STAT
2. Interrupt Mask Status - PKTE_IMSK_STAT
3. Interrupt Clear Register - PKTE_INT_CLR
4. Interrupt Enable Register - PKTE_INT_EN
5. Interrupt Mask Disable - PKTE_IMSK_DIS
6. Interrupt Mask Enable - PKTE_IMSK_EN
7. Interrupt Configuration - PKTE_INT_CFG

Below shows a block diagram of the interrupt controller

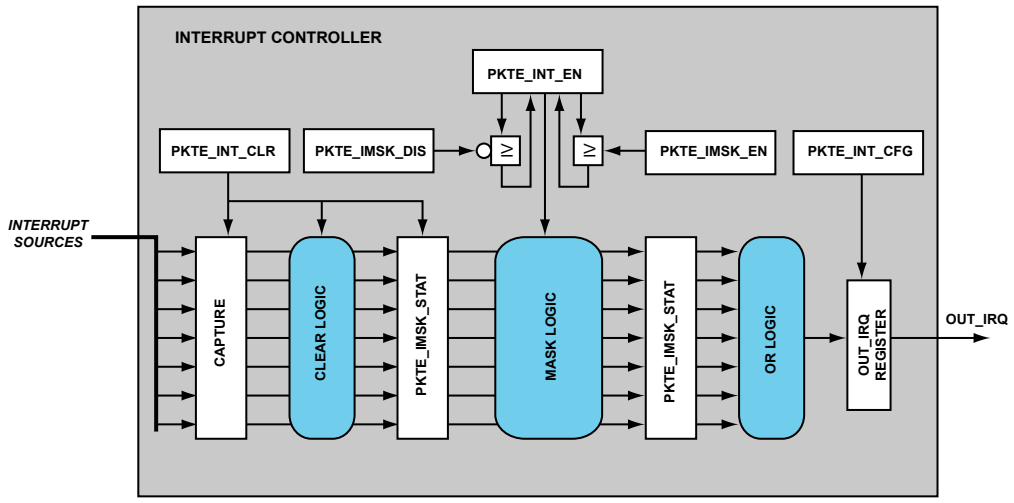

Figure 15-2: Block Diagram of Packet Engine Interrupt Controller

All of the interrupt sources are pulse or level events in their native form.

These interrupts are captured and stored at their unmasked and masked status in their respective PKTE_IUMSK_STAT and PKTE_IMSK_STAT registers. This allows the host processor to read the status of any interrupt source either before or after the mask is applied.

The PKTE_INT_EN register provides a mask to select what interrupt source are enabled to the output interrupt request. Writing a one to the PKTE_INT_CLR register resets both the masked and unmasked interrupt.

The PKTE_IMSK_EN and PKTE_IMSK_DIS registers can be set to enable and disable individual interrupts respectively in the PKTE_INT_EN register. This avoids the need for read-modify-write operations from the host processor.

Ring Interrupts

Two interrupts are provided for efficient ring management: The CDR threshold interrupt (*cdrthrs*h) and the RDR threshold interrupt (*rdrthrsh*).

Command Descriptor Ring

The CDR threshold interrupt (cdrthrsh) is a level-based interrupt and connects to the threshold value in the PKTE_RING_THRESH.CDRTHRSH bit field. It enables the host processor to efficiently fill the CDR. The host processor writes command descriptors to the CDR with this interrupt masked until the CDR is full. Then the host processor enables the CDR threshold interrupt. When the interrupt is activated, the host processor clears the interrupt and it is guaranteed that it can put CDR threshold number of descriptors in the CDR.

Example Configuration:

```
PKTE_RING_CFG.RINGSZ=256,
PKTE_RING_THRESH.CDRTHRSH=224,
PKTE_INT_EN.CDRTHRSH=0 /*(IRQ disabled)*/
```

1. The host writes 8 Command Descriptors at once, then writes the PKTE_CDSC_INCR register to 8.
2. This is repeated until there are less than 8 empty entries in the CDR.
3. The fill level is now equal to he threshold (224)
4. The host enables the CDR threshold IRQ (PKTE_INT_EN.CDRTHRSH=1)
5. The packet engine processes packets and the fill level (PKTE_CDSC_CNT register) decreases.
6. Then the CDR threshold IRQ is activated as the fill level equals 224.
7. The host handles the interrupt, clears it and continues with step 1.

Result Descriptor Ring

The RDR threshold interrupt (rdrthrsh) is a level-based interrupt and connects to the threshold value in the PKTE_RING_THRESH.RDRTHRSH bit field and the timeout value in the RD timeout (PKTE_RING_THRESH.RDTO) bit field. It enables the host processor to efficiently empty the RDR. The timeout reminds the host processor that when the threshold kicks in result descriptors stay long in the RDR and must be processed to reduce latency. The timeout counts when the ring is not empty, regardless of the fill level and restarts when the host processor writes the PKTE_RDSC_CNT register. Initially the host processor enables the RDR threshold interrupt. When the interrupt is activated the host processor reads result descriptors until the RDR is empty or contains less than the PKTE_RING_THRESH.RDRTHRSH number of descriptors.

Example Configuration:

```
PKTE_RING_CFG.RINGSZ=256,
PKTE_RING_THRESH.RDRTHRSH=32, timeout=1ms
PKTE_INT_EN.RDRTHRSH=1 /*(IRQ enabled)*/
```

1. The Packet Engine writes the Result Descriptor, timeout counter starts.
2. The Packet Engine writes 32 more Result Descriptors, fill level (PKTE_RDSC_CNT register) increases to 33.
3. The fill level exceeds threshold within 1ms, the RDR threshold IRQ is activated.
4. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the PKTE_RDSC_DECR register with 8. The write to the PKTE_RDSC_DECR register restarts the timeout counter. The fill level is now under the threshold but there are still 25 descriptors left.
5. The Packet Engine writes 8 more Result Descriptors, fill level increases to 33.
6. The fill level exceeds threshold within 1 ms, the RDR threshold IRQ is activated.
7. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the PKTE_RDSC_DECR register with 8. The write to the PKTE_RDSC_DECR register restarts the timeout counter. The fill level is now under the threshold but there are still 25 descriptors left.
8. After 1 ms, the timeout counter interrupt is activated.
9. The host handles the interrupt, reads 8 Result Descriptors at once, then writes the PKTE_RDSC_DECR register with 8. This is repeated until there are less than 8 full entries in the RDR. The fill level is now under the threshold and the RDR threshold IRQ interrupt is inactive. Each write to the PKTE_RDSC_DECR register restarts the timeout counter.

PKTE Programming Model

The host processor must always follow a pre-defined sequence of five phases required by the packet engine on a per packet basis when using direct host mode. The following sections describe the five phases.

Phase 1. Write the Command Descriptor

1. Write the first command descriptor word with status and control information to the PKTE_CTL_STAT register.
2. Optionally, write the user ID to the PKTE_USERID register.
3. Write the last descriptor word to the PKTE_LEN register.
4. Write the value 0x1 to the PKTE_CDSC_CNT register. This opertion triggers the packet engine to validate the command descriptor. If the command descriptor is invalid, an error is generated. (See the PKTE_CTL_STAT section in the Register Descriptions). If the command descriptor is valid, the packet engine waits for an PKTE_SA_RDY register write.

Phase 2. Write the State Registers and SA Registers

All required fields of the SA record and state record must be written. The fields required depend on the operation. The last field to be written is the PKTE_SA_RDY register. This register triggers the packet engine to start processing.

1. Write the required state record fields.
    - IV
    - Digest count
    - State digest
2. Write the required SA record data.
3. To complete the SA record and state record, write the PKTE_SA_RDY register.

Phase 3. Write the Source Packet Data and Read Result Packet Data

The packet engine has input and output buffers. If a source packet is smaller than the size of the input buffer, then the packet can be written in one part. Otherwise, it must be written in multiple parts. The same applies to the output data. If the result packet size is smaller than the size of the output buffer, then the packet can be read in one part. Otherwise, it must be read in multiple parts.

NOTE: An outbound packet that is smaller than the size of the input buffer can increase in size due to padding and does not always fit in the output buffer. Conversely, an inbound packet that is larger than the size of the input buffer can decrease in size, and due to de-padding, can fit in the output buffer. If the input buffer becomes empty or the output buffer becomes full, the engine stalls.

Two following steps describe different situations:

- Source packet smaller than the size of the input buffer, start at step 1.
- Source packet larger than the size of the input buffer, start at step 3.

The host processor must follow these steps:

1. Write the source packet data. Write the full source packet to the input buffer. Go to step 4.
2. Write the input buffer count register (PKTE_INBUF_CNT) with the number of valid bytes that are written to the input buffer. This value must correspond to the value in thePKTE_LEN.TOTLEN field of the command descriptor rounded up to the next multiple of 4 bytes. Go to step 5 to check the packet engine status.
3. Write part of the source packet data. The PKTE_STAT.IBUFEMPTYCNT field indicates the amount of free space in the input buffer. Programs write the number of bytes determined by the setting in the PKTE_BUF_THRESH register. Write the (partial) source packet to the input buffer. The host processor must resume where it ended the previous write operation. Do not write more than the buffer size at once. Go to step 4.
4. Write the PKTE_INBUF_CNT register with the number of valid bytes written to the input buffer. Go to step 5 to check the packet engine status.
5. Check packet engine status. Wait for an interrupt or poll the PKTE_STAT register for any of the following conditions:

- Condition 1 - An error interrupt or any of the bits [7:5] in the PKTE_STAT register becomes active to indicate a packet processing error. Depending on the type of error, the host processor must take appropriate action. Usually, the result packet is not valid after a processing error has occurred. Go to phase 5 to read the result descriptor.
- Condition 2 - An operation done interrupt or the PKTE_STAT.OPDN bit becomes active (the packet engine completed processing). Go to step 8 to read the remaining output data.
- Condition 3 - An output buffer threshold interrupt or the PKTE_STAT.OBUFREQ bit becomes active. Go to step 6 to read a block of output data.
- Condition 4 - An input buffer threshold interrupt or the PKTE_STAT.IBUFREQ bit becomes active. Go to step 3 to write a block of input data.

6. Read part of the output data. The PKTE_STAT.OBUFFULLCNT bit field indicates the number of bytes in the output buffer. This value is rounded up to full words. Programs read the number of bytes indicated in the PKTE_BUF_THRESH register. Read the (partial) output packet from the output buffer. The host processor must resume where it ended the previous read operation. Do not read more than the input buffer size at once. Go to step 7.
7. Write the PKTE_OUTBUF_CNT register with the number of valid bytes read from the output buffer. Go to step 5 to check the packet engine status.
8. Read the remaining output data. The PKTE_STAT.OBUFFULLCNT bit field indicates the number of bytes in the output buffer. This value is rounded up to words. Read the (partial) packet output data from the output buffer. The host processor must resume where it ended the previous read operation. Go to step 9.
9. Write the PKTE_OUTBUF_CNT register with the number of valid bytes read from the output buffer. Go to phase 4.

Phase 4. Read the Result Descriptor

1. Read the first result descriptor word from the PKTE_CTL_STAT.
2. Optionally read the user ID from the PKTE_USERID register.
3. Read the last result descriptor word from the PKTE_LEN register.
4. Write the value 0x1 to the PKTE_RDSC_DECR register. This operation allows the packet engine to accept new command descriptors. Go to phase 5.

Phase 5. Read the SA Record and State Record

Depending on the operation, the SA record or state record is updated. Check the bit fields [23:16] in the PKTE_CTL_STAT register for the following conditions:

- Condition 1 - At least one error bit in the bit fields [23:16] of the PKTE_CTL_STAT register is set. Do not update the local host processor maintained version of the SA record and state record but take any required action.

- Condition 2 - None of the error bits in the bit fields [23:16] of the PKTE_CTL_STAT register is set, the packet is processed normally (without errors). Update the host processor maintained version of the SA record and state record with the result read from the packet engine registers:
  - Sequence number
  - Sequence number mask
  - Result IV
  - Result digest count
  - Result digest

PKTE Mode Configuration

Before using the packet engine, it must be configured. The mode of the packet engine must be defined and the PRNG (if used) must be initialized.

Configure the packet engine in one of three command modes:

- Autonomous Ring Mode: PKTE_CFG.MODE =b'11
- Target Command Mode: PKTE_CFG.MODE =b'10
- Direct Host Mode: PKTE_CFG.MODE =b'00

PKTE Programming Concepts

The following sections provide conceptual information for programming the PKTE.

Packet Engine Descriptor

IMPORTANT: Depending on the mode, ARM, TCM, or DHM, the descriptor is either:

- in the memory of the host processor in the command descriptor ring, or
- written directly to the descriptor registers in the packet engine

References to descriptor registers are for either the register that is mirrored in the descriptor structure in memory or for the actual register itself.

Command descriptors are host-supplied commands that control the real-time operation of the packet engine. The packet engine returns result descriptors at the end of an operation that provide the status information to the host. The ***Command Descriptor Structure*** and the ***Result Descriptor Structure*** tables show these descriptors.

Table 15-17: Command Descriptor Structure

Word Offset	31:24	23:20	19:16	15:8	7:0	Address Offset
0	Pad Control	—		Next Header/ Pad Value	Control	0x000
1	Source Address					0x004
2	Destination Address					0x008
3	SA Address					0x00C
4	SA State Address					0x010
5	Reserved					0x014
6	User ID					0x018
7	Bypass (words)	Control	Reserved	Input Packet Length (bytes)		0x01C

Table 15-18: Result Descriptor Structure

Word Offset	31:24	23:20	19:16	15:8	7:0	Address Offset
0	Pad Status	Status		Next Header/ Pad Value	Control	0x000
1	Source Address					0x004
2	Destination Address					0x008
3	SA Address					0x00C
4	SA State Address					0x010
5	Reserved					0x014
6	User ID					0x018
7	Bypass (words)	Control	Reserved	Input Packet Length (bytes)		0x01C

When the packet engine is configured for autonomous ring mode, command descriptors and result descriptors reside in a ring in host memory. Command descriptors are automatically fetched from the Command Descriptor Ring (CDR) through DMA into the command descriptor registers. When an operation is complete, the result descriptors are automatically read from the packet engine and through DMA to the Result Descriptor Ring (RDR).

When the packet engine is configured for direct host mode, the host processor manually writes the command descriptor directly to the internal command descriptor MMR set. When an operation is complete, the host processor manually reads the result descriptor directly from the result descriptor MMR set.

The target command mode is a combination of the direct host mode and the autonomous ring mode. The host processor writes the command descriptor directly to the internal command descriptor register set. When an operation is complete there are two options.

1. The host processor can read the result descriptor directly from the result descriptor registers.

2. The result descriptors reside in a ring in host memory and the result descriptors are automatically DMA'd from the packet engine to the RDR.

When the host processor writes a command descriptor to the command descriptor registers, the packet engine is triggered when the host processor updates the PKTE_CDSC_CNT register. This functionality guarantees that all fields in the command descriptor are valid before the command is executed.

Descriptor Processing

This section describes the functional steps of the packet engine while processing the command descriptors.

Descriptor Ring Configuration

At initialization, the host processor specifies the size of the Command Descriptor Ring (CDR). The Result Descriptor Ring (RDR) has the same size.

NOTE: In some configurations, these two rings overlay each other with the results written on top of the command descriptors. This configuration is called overlaid ring mode.

When the packet engine is configured and enabled, it fetches the descriptors from the CDR using system bus master reads.

Descriptor Ring Processing

To validate the descriptor exchange between the host processor and the packet engine, the ownership bits PKTE_CTL_STAT.PERDY and PKTE_CTL_STAT.HOSTRDY are used. One pair of ownership bits is in the first word of the descriptor (PKTE_CTL_STAT), and one pair is in the last word (PKTE_LEN). The 'consumer' of a descriptor must verify that both ownership pairs match to ensure that a race condition did not occur between one party writing and the other party reading the descriptor. A race condition can occur when a memory locking scheme is not used.

Each pair [PKTE_CTL_STAT.PERDY, PKTE_CTL_STAT.HOSTRDY] of ownership bits provide 3 states:

- b'00 = idle or null descriptor
- b'01 = host processor has written a descriptor in the CDR and passed ownership to the packet engine
- b'10 = packet engine processing complete: packet engine has written the descriptor in the RDR and passed ownership back to the host.
- b'11 = Reserved

At initialization, the host sets the entire CDR memory area to zero, when the CDR is used.

1. The host processor writes one or more command descriptors to the CDR. The host processor must set the PKTE_CTL_STAT.HOSTRDY bit to 1 and the PKTE_CTL_STAT.PERDY bit to 0 to indicate that ownership has passed to the packet engine. These bits are mirrored in the PKTE_LEN descriptor word.
2. The host processor must write the PKTE_CDSC_INCR register with the number of new valid command descriptors in the CDR.

3. The packet engine reads and validates one command descriptor.
4. The packet engine reads the SA record and state record, processes the packet and updates the SA record and state record.
5. If the rings are not overlaid and the PKTE_CFG.ENCDRUPDT bit is 1, the packet engine writes the result descriptor to the CDR with the PKTE_CTL_STAT.HOSTRDY bit set to 0 and PKTE_CTL_STAT.PERDY bit set to 1. These bits are mirrored in the PKTE_LEN descriptor word.
6. The packet engine writes the result descriptor to the RDR. The packet engine sets the PKTE_CTL_STAT.PERDY bit to 1 and the PKTE_CTL_STAT.HOSTRDY bit to 0 to indicate that the ownership has passed to the host. These bits are mirrored in the PKTE_LEN descriptor word.
7. The packet engine decrements the value in the PKTE_CDSC_CNT register. If the value is not zero, the packet engine reads with the next command descriptor (step 3).
8. The packet engine increments the value in the PKTE_RDSC_CNT register.
9. The host processor reads one or more result descriptors from the RDR and processes the results.
10. The host processor must write the PKTE_RDSC_DECR register with the number of processed result descriptors in the RDR.

Ownership of the Descriptor

The ownership of the command descriptor and result descriptor is set by ownership bits in the first and last word of the respective descriptor, in the PKTE_CTL_STAT register and the PKTE_LEN register. For the command descriptor, it is the processor core that sets the ownership to the packet engine. For the result descriptor, it is the packet engine that sets the ownership to the host processor.

The packet engine reads the ownership bits before processing, irrespective of the mode of the packet engine. The ownership bits are used to validate and identify the descriptor. When two separate rings are used, the packet engine can be programmed to clear the ownership bits of the command descriptors in the CDR so the host processor knows which descriptors are processed.

NOTE: This update of the ownership bits can be disabled in the PKTE_CFG register when the host processor actively counts the number of descriptors in the CDR to prevent ring wrapping.

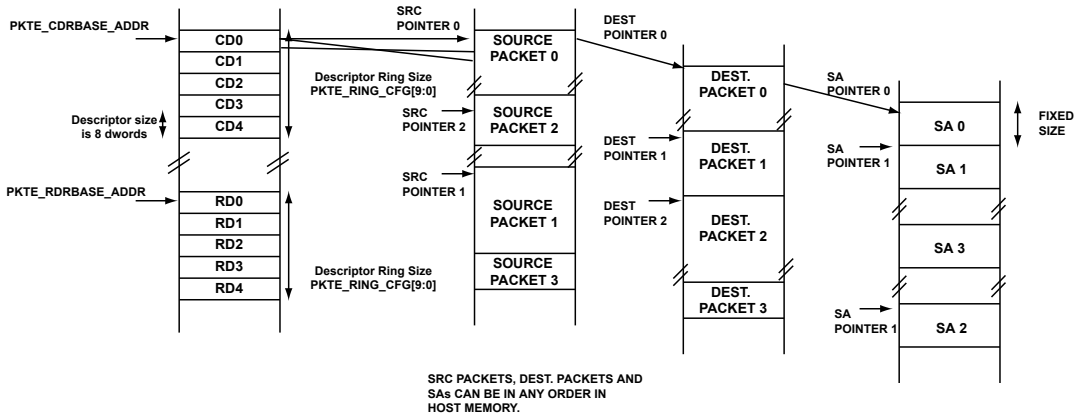

Figure 15-3: Descriptor Rings in Autonomous Ring Mode

Description and Use of the SA Record and State Record Structure

The SA record is a packed structure that contains the remainder of the information needed by the packet engine to process a packet. Most of the information fields in the SA record, such as the key and encryption mode, are static for the lifetime of the association. The fields do not require frequent manipulation by the host processor. The SA record non-static fields are the sequence number and sequence number mask.

The SA record can have a corresponding state record that is used to save results from the current operations that can be used for future operations. The state record can hold the IV, the hash byte count, and the intermediate hash digest.

There is no practical limit to how many SA records and corresponding state records the packet engine can support.

In the autonomous ring mode and target command mode, once the packet engine has validated a command descriptor, it automatically fetches the SA record and optional state record. After processing, the packet engine updates the stateful fields in the SA record and state record in the host processor memory.

In direct host mode, after the descriptor is validated, the host must write the SA record directly into the internal registers of the packet engine. After processing, the host reads the stateful fields from the SA registers of the packet engine and saves them back to the SA record in the host processor memory.

SA Record Structure

The ***SA Record Structure*** table shows the structure for an SA Record. When using direct host mode, the corresponding elements are accessed directly with the registers. When using autonomous ring mode or target command mode, the SA Record is defined, configured and accessed in host memory.

Table 15-19: SA Record Structure

Word Offset	Description (name)	Use
0	PKTE_SA_CMD0[31:0]	SA Control word 0 (all operations)
1	PKTE_SA_CMD1[31:0]	SA Control word 1 (all operations)

Table 15-19: SA Record Structure (Continued)

Word Offset	Description (name)	Use
2	PKTE_SA_KEY0[31:0]	Key word (DES, Triple-DES, AES-128/192/256,)
3	PKTE_SA_KEY1[63:32]	Key word (DES, Triple-DES, AES-128/192/256,)
4	PKTE_SA_KEY2[95:64]	Key word (Triple-DES, AES-128/192/256,)
5	PKTE_SA_KEY3[127:96]	Key word (Triple-DES, AES-128/192/256,)
6	PKTE_SA_KEY4[159:128]	Key word (Triple-DES, AES-192/256)
7	PKTE_SA_KEY5[191:160]	Key word (Triple-DES, AES-192/256)
8	PKTE_SA_KEY6[223:192]	Key word (AES-256)
9	PKTE_SA_KEY7[255:224]	Key word (AES-256)
10	PKTE_SA_IDIGEST0[31:0]	Inner Hash digest (Basic Hash and HMAC with ,SHA-1, SHA-224, SHA-256)
11	PKTE_SA_IDIGEST1[63:32]	Inner Hash digest (Basic Hash and HMAC with ,SHA-1, SHA-224, SHA-256)
12	PKTE_SA_IDIGEST2[95:64]	Inner Hash digest (Basic Hash and HMAC with ,SHA-1, SHA-224, SHA-256)
13	PKTE_SA_IDIGEST3[127:96]	Inner Hash digest (Basic Hash and HMAC with ,SHA-1, SHA-224, SHA-256)
14	PKTE_SA_IDIGEST4[159:128]	Inner Hash digest (Basic Hash and HMAC with SHA-1, SHA-224, SHA-256)
15	PKTE_SA_IDIGEST5[191:160]	Inner Hash digest (Basic Hash and HMAC with SHA-224, SHA-256)
16	PKTE_SA_IDIGEST6[223:192]	Inner Hash digest (Basic Hash and HMAC with SHA-224, SHA-256)
17	PKTE_SA_IDIGEST7[255:224]	Inner Hash digest (Basic Hash and HMAC with SHA-256)
18	PKTE_SA_ODIGEST0[31:0]	Outer Hash digest (HMAC with , SHA-1, SHA-224, SHA-256)
19	PKTE_SA_ODIGEST1[63:32]	Outer Hash digest (HMAC with , SHA-1, SHA-224, SHA-256)
20	PKTE_SA_ODIGEST2[95:64]	Outer Hash digest (HMAC with , SHA-1, SHA-224, SHA-256)
21	PKTE_SA_ODIGEST3[127:96]	Outer Hash digest (HMAC with , SHA-1, SHA-224, SHA-256)
22	PKTE_SA_ODIGEST4[159:128]	Outer Hash digest (HMAC with SHA-1, SHA-224, SHA-256)
23	PKTE_SA_ODIGEST5[191:160]	Outer Hash digest (HMAC with SHA-224, SHA-256)

Table 15-19: SA Record Structure (Continued)

Word Offset	Description (name)	Use
24	PKTE_SA_ODIGEST6[223:192]	Outer Hash digest (HMAC with SHA-224, SHA-256)
25	PKTE_SA_ODIGEST7[255:224]	Outer Hash digest (HMAC with SHA-256)
26	PKTE_SA_SPI[31:0]	SPI (IPsec), Type[23:16] / Version [15:0] (SSL, TLS, DTLS)
27	PKTE_SA_SEQNUM0[31:0]	Sequence Number (IPsec, SSL, TLS, DTLS with Header Processing)
28	PKTE_SA_SEQNUM1[63:32]	
29	PKTE_SA_SEQNUM_MSK0[31:0]	Sequence Number Mask (IPsec, DTLS inbound with Header Processing)
30	PKTE_SA_SEQNUM_MSK1[63:32]	
31	PKTE_SA_NONCE[31:0] / PKTE_SA_READY	Nonce value (AES-CTR, AES-ICM)/ i & j pointers ()/SA ready indicator (Direct Host Mode)

Some of these fields may be updated by the packet engine. These include:

- PKTE_SA_SEQNUM0
- PKTE_SA_SEQNUM1
- PKTE_SA_SEQNUM_MSK0
- PKTE_SA_SEQNUM_MSK1

All the other fields remain unchanged.

SA State Structure

The security association state structure contains information that may be updated after each packet, such as the IV and the intermediate hash result. The ***SA State Structure*** table shows the SA state structure and usage. In direct host mode, the elements are accessed directly using the PKTE registers. In target command mode and autonomous ring mode, this structure is defined and updated in host memory.

Table 15-20: SA State Structure

Word Offset	Description (name)	Use
0	PKTE_STATE_IV0[31:0]	Initialization Vector (DES, Triple DES, AES)
1	PKTE_STATE_IV1[63:32]	Initialization Vector (DES, Triple DES, AES)
2	PKTE_STATE_IV2[95:64]	Initialization Vector (AES)
3	PKTE_STATE_IV3[127:96]	Initialization Vector (AES)
4	PKTE_STATE_BYTE_CNT0[31:0]	Current hash byte count (SHA-1, SHA-224, SHA-256)
5	PKTE_STATE_BYTE_CNT1[63:32]	Current hash byte count (SHA-1, SHA-224, SHA-256)

Table 15-20: SA State Structure (Continued)

Word Offset	Description (name)	Use
6	PKTE_STATE_IDIGEST0[31:0]	Inner Hash digest (mirror of PKTE_SA_IDIGEST0)
7	PKTE_STATE_IDIGEST1[63:32]	Inner Hash digest (mirror of PKTE_SA_IDIGEST1)
8	PKTE_STATE_IDIGEST2[95:64]	Inner Hash digest (mirror of PKTE_SA_IDIGEST2)
9	PKTE_STATE_IDIGEST3[127:96]	Inner Hash digest (mirror of PKTE_SA_IDIGEST3)
10	PKTE_STATE_IDIGEST4[159:128]	Inner Hash digest (mirror of PKTE_SA_IDIGEST4)
11	PKTE_STATE_IDIGEST5[191:160]	Inner Hash digest (mirror of PKTE_SA_IDIGEST5)
12	PKTE_STATE_IDIGEST6[223:192]	Inner Hash digest (mirror of PKTE_SA_IDIGEST6)
13	PKTE_STATE_IDIGEST7[255:224]	Inner Hash digest (mirror of PKTE_SA_IDIGEST7)

Configuring Operations in the PKTE

The operation (cipher, hash function, and others) that the PKTE performs is configured primarily in the `PKTE_SA_CMD0` register. The following sections include a series of tables to help configure the least significant 16 bits of the `PKTE_SA_CMD0` register. These fields include:

- The operation code field (`PKTE_SA_CMD0.OPCD`)
- The direction field (`PKTE_SA_CMD0.DIR`)
- The operation group field (`PKTE_SA_CMD0.OPGRP`)
- The padding type (`PKTE_SA_CMD0.PADTYPE`)
- The cipher selection (`PKTE_SA_CMD0.CIPHER`)
- The hash selection (`PKTE_SA_CMD0.HASH`)

Basic Operations and Decoding

Table 15-21: Basic Operation Decoding

Outbound				Inbound			
OpGroup	**Dir**	**OpCode**	**Operation**	**OpGroup**	**Dir**	**OpCode**	**Operation**
0b00	0	0b000	Encrypt	0b00	1	0b000	Decrypt
0b00	0	0b001	Encrypt - Hash	0b00	1	0b001	Hash - Decrypt
0b00	0	0b010	Reserved	0b00	1	0b010	Reserved
0b00	0	0b011	Hash	0b00	1	0b011	Hash
0b00	0	0b100... 0b110	Reserved	0b00	1	0b100... 0b110	Reserved

Table 15-21: Basic Operation Decoding (Continued)

Outbound				Inbound			
OpGroup	**Dir**	**OpCode**	**Operation**	**OpGroup**	**Dir**	**OpCode**	**Operation**
0b00	0	0b111	PRNG	0b00	1	0b111	Reserved

Table 15-22: Protocol Operation Decoding

Outbound				Inbound			
OpGroup	**Dir**	**OpCode**	**Operation**	**OpGroup**	**Dir**	**OpCode**	**Operation**
0b01	0	0b000	ESP Outbound	0b01	1	0b000	ESP Inbound
0b01	0	0b001... 0b011	Reserved	0b01	1	0b001	Reserved
0b01	0	0b100	Basic SSL Outbound	0b01	1	0b010	Basic SSL Inbound
0b01	0	0b101	Basic TLS Outbound	0b01	1	0b011	Basic TLS Inbound
0b01	0	0b110	Reserved	0b01	1	0b100... 0b110	Reserved
0b01	0	0b111	SRTP Outbound	0b01	1	0b111	SRTP Inbound

NOTE: For SSL/TLS and SRTP, no header processing is performed in hardware.

Table 15-23: Extended Protocol Operation Decoding

Outbound				Inbound			
OpGroup	**Dir**	**OpCode**	**Operation**	**OpGroup**	**Dir**	**OpCode**	**Operation**
0b11 0	0	0b000	Reserved	0b11	1	0b000	Reserved
0b11 0	0	0b001	DTLS Outbound	0b11	1	0b001	DTLS Inbound
0b11 0	0	0b010... 0b011	Reserved	0b11	1	0b010... 0b011	Reserved
0b11 0	0	0b100	Ext. SSL Outbound	0b11	1	0b100	Ext. SSL Inbound
0b11 0	0	0b101	Ext. TLS v1.0 Outbound	0b11	1	0b101	Ext. TLS v1.0 Inbound

Table 15-23: Extended Protocol Operation Decoding (Continued)

Outbound				Inbound			
OpGroup	Dir	OpCode	Operation	OpGroup	Dir	OpCode	Operation
0b11 0	0	0b110	Ext. TLS v1.1 Outbound	0b11	1	0b110	Ext. TLS v1.1 Inbound
0b11 0	0	0b111	Reserved	0b11	1	0b111	Reserved

Error Code Description

The PKTE_CTL_STAT register is used to configure the packet engine for processing in Direct Host Mode (DHM) or Target Command Mode (TCM). The PKTE_CTL_STAT structure element in memory is used when the packet engine is configured for Autonomous Ring Mode (ARM). In both cases, when an operation is started, errors are reported in the status field (bits [23:16]) of this register or structure element. The ***Extended Error Codes - Status Encoding*** table provides a guide on how to decipher the meaning of the bits that are set when an error occurs.

Extended Error Codes

The following table provides information about the extended errors associated with the PKTE module.

Table 15-24: Extended Error Codes - Status Encoding

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
0b0000_0000	0x00	NA	Successful completion. No errors occurred during processing of the packet.	Packet fully processed
0b----_---1	0x-1	NA	Authentication Error. For an inbound IPsec ESP operation, the Integrity Check Value (ICV) does not match the computed value. For an inbound SRTP operation, the authentication tag does not match the computed value. For a basic SSL/TLS, Extended SSL/TLS or DTLS operation the Message Authentication Code (MAC) does not match the computed value.	Packet fully processed
0b----_--1-	0x-2	NA	Pad Verify Error. For inbound operations that use pad type Constant TLS, IPsec or PKCS#7, the decrypted pad does not match the expected values for the selected pad type.	Packet fully processed
0b----_-1--	0x-4	NA	Sequence Number Error. For an inbound IPsec or DTLS operation, there was a fault in the Anti-Replay Sequence Number.	Packet fully processed

Table 15-24: Extended Error Codes - Status Encoding (Continued)

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
			For an outbound IPsec packet, the sequence number overflows; count is 2^{32}– 1 and increments to 0. For an outbound DTLS operation, the sequence number overflows; count is 2^{48}– 1 and increments to 0. For an outbound SSL or TLS operation, the sequence number overflows; count is 2^{64}– 1 and increments to 0.	
0b0000_1---	0x08	1	System Bus error. The master bus interface generates an error due to ERROR response from system slave. The slave bus interface generates an error due to request for non-word (32-bit) access.	Packet is aborted. The host must reject the packet and apply a hardware reset to the system.
0b0001_1---	0x18	2	Invalid Command Descriptor Error. The ownership bits in the command descriptor are not set to the packet engine, after the PKTE_CDSC_CNT register is incremented.	Command descriptor is ignored, no packet is processed. The packet must be re-queued or discarded.
0b0010_1---	0x28	3	Invalid Crypto Operation Error. A reserved operation is selected.	The SA record is ignored, no packet is processed. The packet must be re-queued or discarded.
0b0011_1---	0x38	4	Invalid Crypto Algorithm Error. A reserved cipher is selected, refer to PKTE_SA_CMD0.CIPHER. A reserved hash is selected, refer to PKTE_SA_CMD0.HASH.	The SA record is ignored, no packet is processed. The packet must be re-queued or discarded.
0b0100_1---	0x48	5	SPI Error. On an inbound packet, the 32-bit SPI value in the packet does not match the value in the SA while header processing is enabled. Note: A failure caused by an SPI mismatch, in general should not occur because the host checks the SPI and does not send an incorrect SPI to the packet engine.	Packet is fully processed. The host must reject the packet.
0b0101_1---	0x58	3	Zero Length Error. The packet length defined in the command descriptor PKTE_LEN.TOTLEN is zero, which is illegal.	Packet command is ignored, no packet processed. The host must reject the packet.
0b0110_1xxx	0x68	6	Invalid Packet Length Error. For Basic Encrypt-Hash and Hash-Decrypt operations: PKTE_LEN.TOTLEN < Hash/Encrypt Offset For IPsec ESP inbound operations:	Packet processing is aborted. Result packet length is zero.

Table 15-24: Extended Error Codes - Status Encoding (Continued)

STATUS bits [23:16]	Hex Value	Priority	Description	Processing Result
			PKTE_LEN.TOTLEN < ICV length or PKTE_LEN.TOTLEN is non-4 byte aligned For SRTP inbound operations: PKTE_LEN.TOTLEN ≤ IV (opt.) + Bypass Offset + ROC For SSL inbound operations: PKTE_LEN.TOTLEN ≤ 1 or packet length > 65535 bytes (SSL packet-bypass length) For TLS and DTLS inbound operations: PKTE_LEN.TOTLEN ≤ 13 or payload length > 65535 bytes (data to be hashed) Note: For IPsec ESP the ICV is stripped before the length is checked.	The host must reject the packet and apply a software reset of the packet engine.
0b0111_1xxx	0x78	7	Block Size Error. The length of the inbound packet defined in the Command Descriptor PKTE_LEN.TOTLEN is not a multiple of the DES or AES block cipher length. For outbound packets the size is always automatically aligned (padded) to the correct block size. The hashed packet length is not a multiple of the hash block size for intermediate hash operation. For a final hash operation no error is generated. Note: For IPsec ESP operations the ICV is stripped before the block size is checked.	Packet is fully processed. The host must reject the packet.
0b1000_1xxx	0x88	8	Processing Error. The number of bytes in the input buffer is more than defined in the PKTE_LEN.TOTLEN field. The number of bytes written to the output buffer is less than processed in the datapath.	Packet processing aborted. Result packet length is zero. The host must reject the packet and apply a software reset.
0b1010_1xxx 0b1111_1xxx	Reserved			

Number Format

When dealing with cryptographic functions, data and keys are large vectors. For instance, AES supports keys of sizes 128, 192, and 256 bits. When a key needs to be loaded or read, multiple 32-bit key registers are used, namely PKTE_SA_KEY[n] registers. The first key register PKTE_SA_KEY[n]0 holds bits 31:0, and PKTE_SA_KEY[n]1 holds the next thirty two bits 63:32, and so on.

Generally, for large vectors defined as Byte0, Byte1, Byte2, Byte3 and so on, the values are stored in the PKTE registers as PKTE_REG0 = Byte3Byte2Byte1Byte0, followed by PKTE_REG1 = Byte7Byte6Byte5Byte4 and so on.

PKTE Programming Examples

Use these examples to extend your understanding of PKTE features, operating modes, event control, and programming modes.

Calculating SHA in Direct Host Mode

This section describes how to configure the packet engine to calculate a hash digest using one of supported SHA algorithms in direct host mode. This configuration follows the procedure outlined in the *PKTE Programming Model* section.

1. Configure the packet engine for direct host mode by setting the PKTE_CFG.MODE bit =0
2. Set the ownership back to the packet engine to process a command descriptor by setting the PKTE_CTL_STAT.PERDY bit =0 and the PKTE_CTL_STAT.HOSTRDY bit =1.
3. Set the PKTE_CTL_STAT.HASHFINAL bit to indicate this command descriptor handles all the input data for the hash calculation. This configuration is needed for the packet engine because the last block requires special handling (see FIPS 180-4 for details).
4. Set size of the input data in bytes in the PKTE_LEN.TOTLEN bit field.
5. Also set the PKTE_LEN.PEDONE bit =0 and the PKTE_LEN.HSTRDY bit =1. These bits must be the same as the PKTE_CTL_STAT.PERDY and PKTE_CTL_STAT.HOSTRDY bits to guarantee ownership.
6. Set the PKTE_CDSC_CNT register =1 to trigger the packet engine to start validating the command descriptor. In this case, the PKTE_CTL_STAT, PKTE_LEN and PKTE_CDSC_CNT registers are the only command descriptor registers modified.
7. Configure the PKTE_SA_CMD0 and PKTE_SA_CMD1 registers to define the operation. For an SHA, set the PKTE_SA_CMD0.OPCD bit field =0b011 for hash operation and the PKTE_SA_CMD0.OPGRP bit field =0b00 for basic operation.
8. Select the specific SHA function using the PKTE_SA_CMD0.HASH bit field as follows.
    - For SHA-1, PKTE_SA_CMD0.HASH =0b0001
    - SHA-224, PKTE_SA_CMD0.HASH =0b0010
    - for SHA-256, PKTE_SA_CMD0.HASH =0b0011
9. Depending on the SHA selected, the appropriate digest length must be chosen for the PKTE_SA_CMD0.DIGESTLEN bit field as follows.
    - For SHA-1, PKTE_SA_CMD0.DIGESTLEN =0b0101 (5 words)
    - For SHA-224, PKTE_SA_CMD0.DIGESTLEN =0b0111 (7 words)
    - For SHA-256, PKTE_SA_CMD0.DIGESTLEN =0b1000 (8 words)

10. The SHA specifies initial constants. These constants can be pre-loaded or read from memory. In this example, by setting the PKTE_SA_CMD0.HASHSRC bit field =0b11, the packet engine provides the correct initial constants depending on the SHA chosen.
11. Next, set the PKTE_SA_CMD1.CPYDGST bit =1 and PKTE_SA_CMD1.CPYPAD bit =1 to move the result to the output buffer of the packet engine at the PKTE_DATAIO_BUF location.
12. At this point, write to the PKTE_SA_RDY register with any value to trigger the operation.
13. Start writing the input to the data buffer of the packet engine starting at the PKTE_DATAIO_BUF location.
14. Write the PKTE_INBUF_CNT register with the length of the input rounded up to the next multiple of 4. For example, if the input length is 30 bytes, set this register to 32.
15. Poll the PKTE_STAT register to see if any errors occurred or if the operation completed without errors.

Once the operation is done, the digest is available in the packet engine data I/O buffer.

NOTE: The input data or message is input into the packet engine data buffer in big endian format while the result or digest is little endian format.

Performing AES Decryption in Direct Host Mode

This section describes how to configure the packet engine to decrypt using AES-128 in direct host mode. This configuration follows the procedure outlined in the *PKTE Programming Model* section.

1. Configure the packet engine for direct host mode by setting the PKTE_CFG.MODE bit =0
2. Start configuring the command descriptor registers. Set the ownership back to the packet engine to process a command descriptor by setting the PKTE_CTL_STAT.PERDY bit =0 and the PKTE_CTL_STAT.HOSTRDY bit =1.
3. Next, configure the PKTE_LEN.TOTLEN bit field with the size of the packet or message to decrypt. If the entire input message (cipher text) fits into the 256-byte data I/O buffer of the packet engine, the process can be done in one shot.
4. Set the PKTE_LEN.PEDONE bit =0 and the PKTE_LEN.HSTRDY bit =1. These bits must have the same setting as the PKTE_CTL_STAT.PERDY and PKTE_CTL_STAT.HOSTRDY bits to guarantee ownership.
5. Set the PKTE_CDSC_CNT register =1 to trigger the packet engine to start validating the command descriptor. In this case, the PKTE_CTL_STAT, PKTE_LEN, and PKTE_CDSC_CNT registers are the only command descriptor registers modified.
6. Next, configure the PKTE_SA_CMD0 and PKTE_SA_CMD1 registers to define the operation.
    - For a AES decrypt inbound cipher operation, set the PKTE_SA_CMD0.OPCD bit field =0b000 and the PKTE_SA_CMD0.DIR bit field =0b1.
    - Set thePKTE_SA_CMD0.OPGRP bit field =0b00 for basic operation.

- To choose the AES cipher, set the PKTE_SA_CMD0.CIPHER bit field =0b0011. Set the PKTE_SA_CMD0.HASH bit field to 0b1111 to choose the NULL function.

7. Next, set the PKTE_SA_CMD1.AESKEYLEN bit field to select the appropriate key length. In this case, setting it to 0b10 select 128 bits. Also, set PKTE_SA_CMD1.CIPHERMD bit field to select the mode. In this case, setting it to 0b01 select CBC mode.
8. Continue configuring the Security Association (SA) record by loading the key in the PKTE_SA_KEY[n] registers.
9. Next load the initialization vector in the SA state registers (PKTE_STATE_IV[n]).
10. Finally, write anything in to the PKTE_SA_RDY register to trigger the operation.
11. The input data can now be written into the data I/O buffer starting at PKTE_DATAIO_BUF.
12. After the data is written, write the length (or next multiple of 4) into PKTE_INBUF_CNT register.
13. Poll PKTE_STAT to see if any errors occurred or if the operation completed without errors.

Once the operation is done, the result can be found in the same data I/O buffer.

ADSP-BF70x PKTE Register Descriptions

Security Packet Engine (PKTE) contains the following registers.

Table 15-25: ADSP-BF70x PKTE Register List

Name	Description
PKTE_BUF_PTR	Packet Engine Buffer Pointer Register
PKTE_BUF_THRESH	Packet Engine Buffer Threshold Register
PKTE_CDRBASE_ADDR	Packet Engine Command Descriptor Ring Base Address
PKTE_CDSC_CNT	Packet Engine Command Descriptor Count Register
PKTE_CDSC_INCR	Packet Engine Command Descriptor Count Increment Register
PKTE_CFG	Packet Engine Configuration Register
PKTE_CLK_CTL	PE Clock Control Register
PKTE_CONT	PKTE Continue Register
PKTE_CTL_STAT	Packet Engine Control Register
PKTE_DATAIO_BUF	Starting Entry of 256-byte Data Input/Output Buffer
PKTE_DEST_ADDR	Packet Engine Destination Address
PKTE_DMA_CFG	Packet Engine DMA Configuration Register
PKTE_ENDIAN_CFG	Packet Engine Endian Configuration Register
PKTE_HLT_CTL	Packet Engine Halt Control Register

Table 15-25: ADSP-BF70x PKTE Register List (Continued)

Name	Description
PKTE_HLT_STAT	Packet Engine Halt Status Register
PKTE_IMSK_DIS	Interrupt Mask Disable Register
PKTE_IMSK_EN	Interrupt Mask Enable Register
PKTE_IMSK_STAT	Interrupt Masked Status Register
PKTE_INBUF_CNT	Packet Engine Input Buffer Count Register
PKTE_INBUF_INCR	Packet Engine Input Buffer Count Increment Register
PKTE_INT_CFG	Interrupt Configuration Register
PKTE_INT_CLR	Interrupt Clear Register
PKTE_INT_EN	Interrupt Enable Register
PKTE_IUMSK_STAT	Interrupt Unmasked Status Register
PKTE_LEN	Packet Engine Length Register
PKTE_OUTBUF_CNT	Packet Engine Output Buffer Count Register
PKTE_OUTBUF_DECR	Packet Engine Output Buffer Count Decrement Register
PKTE_RDRBASE_ADDR	Packet Engine Result Descriptor Ring Base Address
PKTE_RDSC_CNT	Packet Engine Result Descriptor Count Registers
PKTE_RDSC_DECR	Packet Engine Result Descriptor Count Decrement Registers
PKTE_RING_CFG	Packet Engine Ring Configuration
PKTE_RING_PTR	Packet Engine Ring Pointer Status
PKTE_RING_STAT	Packet Engine Ring Status
PKTE_RING_THRESH	Packet Engine Ring Threshold Registers
PKTE_SA_ADDR	Packet Engine SA Address
PKTE_SA_CMD0	SA Command 0
PKTE_SA_CMD1	SA Command 1
PKTE_SA_IDIGEST[n]	SA Inner Hash Digest Registers
PKTE_SA_KEY[n]	SA Key Registers
PKTE_SA_NONCE	SA Initialization Vector Register
PKTE_SA_ODIGEST[n]	SA Outer Hash Digest Registers
PKTE_SA_RDY	SA Ready Indicator
PKTE_SA_SEQNUM[n]	SA Sequence Number Register
PKTE_SA_SEQNUM_MSK[n]	SA Sequence Number Mask Registers
PKTE_SA_SPI	SA SPI Register

Table 15-25: ADSP-BF70x PKTE Register List (Continued)

Name	Description
PKTE_SRC_ADDR	Packet Engine Source Address
PKTE_STAT	Packet Engine Status Register
PKTE_STATE_ADDR	Packet Engine State Record Address
PKTE_STATE_BYTE_CNT[n]	State Hash Byte Count Registers
PKTE_STATE_IDIGEST[n]	State Inner Digest Registers
PKTE_STATE_IV[n]	State Initialization Vector Registers
PKTE_USERID	Packet Engine User ID

Packet Engine Buffer Pointer Register

The PKTE_BUF_PTR register contains the offset of the next buffer address (entry) to be read or written by the packet engine. This register is used in direct host mode only.

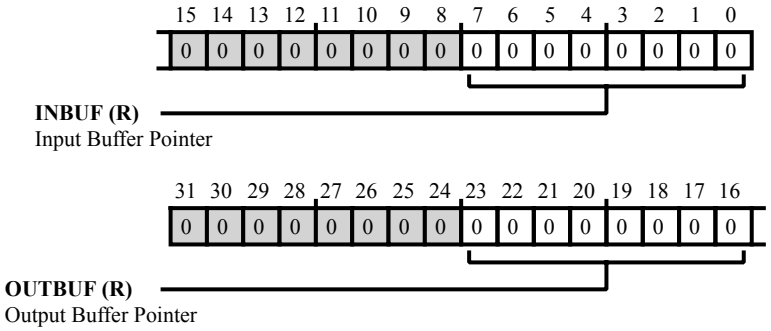

Figure 15-4: PKTE_BUF_PTR Register Diagram

Table 15-26: PKTE_BUF_PTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/NW)	OUTBUF	Output Buffer Pointer. The PKTE_BUF_PTR.OUTBUF bit field indicates the offset of the next address (entry) in the output buffer that will be written next by the packet engine. This bit field is reset to zero after starting up and decremented by 4 at every output buffer write operation. Pointers wrap around; the maximum value this field can have equals the output buffer size minus 4.
7:0 (R/NW)	INBUF	Input Buffer Pointer. The PKTE_BUF_PTR.INBUF bit field indicates the offset of the next address (entry) in the input buffer that will be read next by the packet engine. The bit field is reset to zero after starting up and incremented by 4 at every input buffer read operation. Pointers wrap around; the maximum value this field can have equals the input buffer size minus 4.

Packet Engine Buffer Threshold Register

When in autonomous ring mode or target command mode, the PKTE_BUF_THRESH register defines the high- and low-level value at which the packet engine starts to transfer packet data in or out of the internal packet buffers. These parameters can be used to control the DMA burst size for packet data input and output from the packet engine. In direct host mode, this register contains both threshold values to reduce the amount of packet engine interrupts.

The input buffer threshold (ibufthrsh) interrupt indicates that the input buffer counter is less than or equal to the input buffer threshold value set in this register - this interrupt can be used to wake up a process that stalled on a full input buffer.

The output buffer threshold (obufthrsh) interrupt indicates that the output buffer counter exceeds the output buffer threshold setting. The output buffer interrupt remains active until the output buffer counter is decremented to zero again.

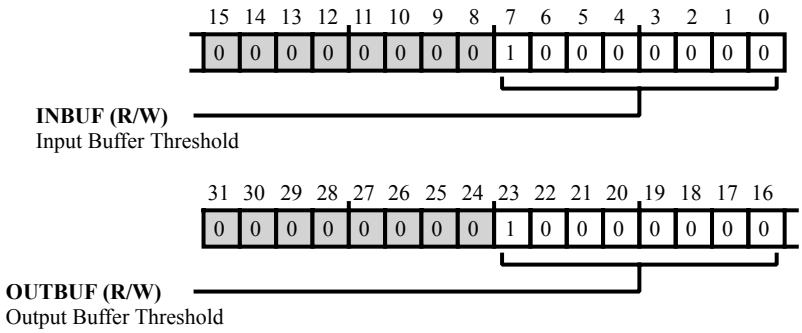

Figure 15-5: PKTE_BUF_THRESH Register Diagram

Table 15-27: PKTE_BUF_THRESH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	OUTBUF	Output Buffer Threshold. The PKTE_BUF_THRESH.OUTBUF bit field specifies how many bytes must be available in the packet engine output buffer before an output transfer starts. Valid values range from 0 to 252, in multiples of 4. In autonomous ring mode, a value of 128 generally gives a good performance, but the optimal value depends on the system and application. In direct host mode, the output buffer threshold (obufthrsh) interrupt activates when the output buffer counter for the output buffer exceeds the value set in this field. A value of 128 generally gives a good performance, but the optimal value depends on the system and application.
7:0 (R/W)	INBUF	Input Buffer Threshold.

Table 15-27: PKTE_BUF_THRESH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The `PKTE_BUF_THRESH.INBUF` bit field specifies how many bytes must be free in the packet engine input buffer before an input transfer starts. Valid values range from 0 to 252, in multiples of 4. In autonomous ring mode, a value of 128 generally gives a good performance, but the optimal value depends on the system and application. In direct host mode, the input buffer threshold (ibufthrsh) interrupt activates when the input buffer counter for the input buffer is below or equal the value set in this field. A value of 128 generally gives a good performance, but the optimal value depends on the system and application.

Packet Engine Command Descriptor Ring Base Address

The PKTE_CDRBASE_ADDR register holds the command descriptor ring base address in host memory. It is only applicable in autonomous ring mode. The PKTE_CDRBASE_ADDR register is ignored for all other modes when command descriptors are directly written into the descriptor registers.

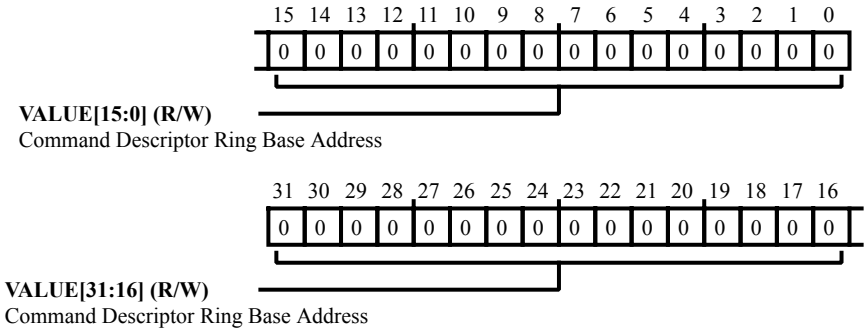

Figure 15-6: PKTE_CDRBASE_ADDR Register Diagram

Table 15-28: PKTE_CDRBASE_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Descriptor Ring Base Address. The PKTE_CDRBASE_ADDR.VALUE bit field specifies the base location of the command descriptor ring in the host memory space.

Packet Engine Command Descriptor Count Register

The PKTE_CDSC_CNT register holds the counter for the number of descriptors in the Command Descriptor Ring (CDR). It is decremented by the packet engine each time a valid descriptor is read from the CDR and processed.

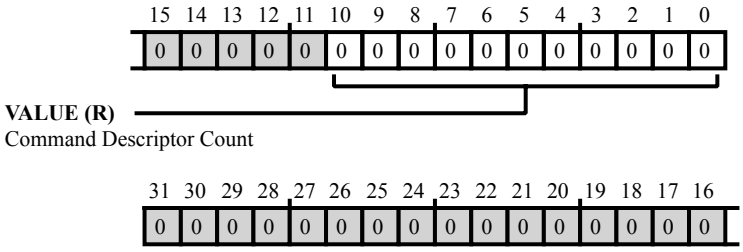

Figure 15-7: PKTE_CDSC_CNT Register Diagram

Table 15-29: PKTE_CDSC_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Command Descriptor Count. The PKTE_CDSC_CNT.VALUE bit field provides the number of command descriptors in the command descriptor ring. The packet engine decrements the counter when a valid command descriptor is read from the CDR and processed.

Packet Engine Command Descriptor Count Increment Register

The PKTE_CDSC_INCR register is accessible by the host connected through the system slave bus. The host can increment the command descriptor counter by writing a value between 1 and 255 to the lowest byte of this register.

In autonomous ring mode, the host must prepare 1 to 255 valid command descriptors in the CDR and then write this register with a value between 1 and 255. The write triggers the packet engine to fetch the command descriptors from the CDR. In direct host mode or target command mode, the host must write one valid command descriptor to the internal descriptor registers and then write this register with the value 1, to indicate that one valid descriptor is available.

A CDR threshold interrupt is activated when the command descriptor counter is less than or equal to the threshold value set in the PKTE_RING_THRESH register. This interrupt can be used to wake up a process that stalled on a full CDR.

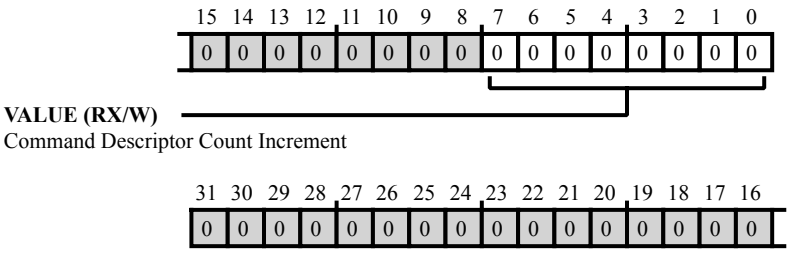

Figure 15-8: PKTE_CDSC_INCR Register Diagram

Table 15-30: PKTE_CDSC_INCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	VALUE	Command Descriptor Count Increment. The value written to the PKTE_CDSC_INCR.VALUE bit field is added to the command descriptor counter. The counter is protected against overflow (see the PKTE_RING_STAT register description). Note that bits[10:8] should be written with zeros.

Packet Engine Configuration Register

The PKTE_CFG register is used to select static settings that control the packet-processing path. This register is typically the last one to be written during the initialization sequence. These settings are typically set at initialization and not changed again.

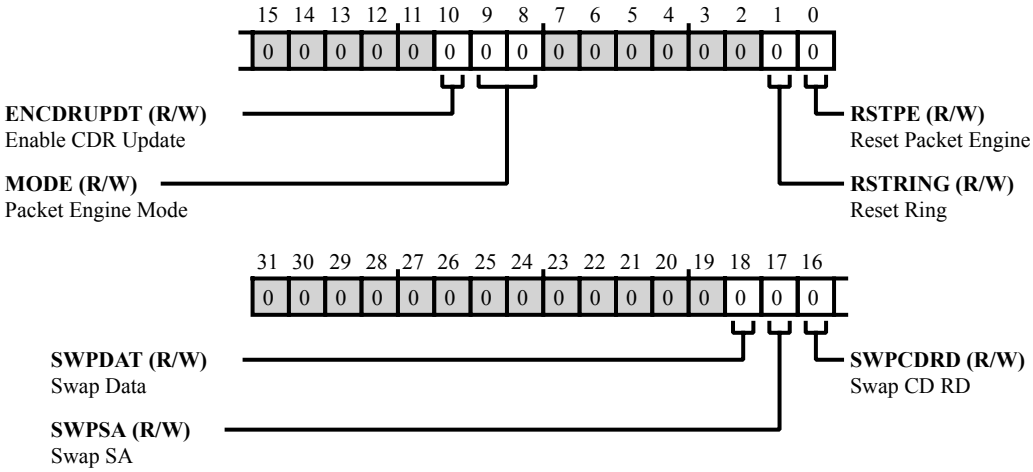

Figure 15-9: PKTE_CFG Register Diagram

Table 15-31: PKTE_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
18 (R/W)	SWPDAT	Swap Data. The PKTE_CFG.SWPDAT bit enables endian swap for packet data as configured in the PKTE_ENDIAN_CFG.MSTRBSWP bits for the packet data DMA read and write.	
		0	No Endian Swap
		1	Apply Endian Swap
17 (R/W)	SWPSA	Swap SA. The PKTE_CFG.SWPSA bit enables endian swap for a SA record as configured in the PKTE_ENDIAN_CFG.MSTRBSWP bits for the SA record and state record DMA read and write. If the PKTE_ENDIAN_CFG.MSTRBSWP bits specify no endian swap, this bit is ignored.	
		0	No Endian Swap
		1	Apply Endian Swap

Table 15-31: PKTE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	SWPCDRD	Swap CD RD. The PKTE_CFG.SWPCDRD bit enables endian swap for descriptors as configured in the PKTE_ENDIAN_CFG.MSTRBSWP bits for the command descriptor DMA read and result descriptor DMA write. If the PKTE_ENDIAN_CFG.MSTRBSWP bits specify no endian swap, this bit is ignored.	
		0	No Endian Swap
		1	Apply Endian Swap
10 (R/W)	ENCDRUPDT	Enable CDR Update. The PKTE_CFG.ENCDRUPDT bit enables the packet engine to update, (clear the ownership bits) in the command descriptor in the CDR.	
		0	Do Not Clear Ownership Bits. The packet engine does not clear the ownership bits in the command descriptor when it completes an operation. The host application must clear the ownership bits in "old descriptors" before the packet engine is allowed to wrap around the CDR to re-encounter these "old descriptors". This setting has the advantage of eliminating a separate DMA write to the CDR.
		1	Clear Ownership Bits. The packet engine clears (set to zero) the ownership bits in the current command descriptor in the CDR. This prevents the packet engine from re-processing an "old descriptor" when it wraps around the CDR.
9:8 (R/W)	MODE	Packet Engine Mode. The PKTE_CFG.MODE bit field selects how the packet engine receives commands.	
		0	Direct Host Mode.
		1	Target Command Mode with Result Descriptor Ring Disabled.
		2	Target Command Mode with Result Descriptor Ring Enabled.
		3	Autonomous Ring Mode

Table 15-31: PKTE_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	RSTRING	Reset Ring. The PKTE_CFG.RSTRING bit resets the internal counters for the CDR and RDR, PKTE_CDSC_CNT and PKTE_RDSC_CNT registers) to zero. Resets the PKTE_RING_PTR register to the base address. After the reset the rings are empty. This bit must be written with a '1' to reset the descriptor ring manager and then re-written with a '0' to release the reset. Note that this bit can remain in the reset state if the CDR ring is disabled (PKTE_CFG.MODE is not 0b11). Note that this reset must be coordinated with the 'owner' of the descriptor ring to ensure that the pointers are in sync after the reset.	
		0	Release the Descriptor Ring Manager Reset
		1	Reset the Descriptor Ring Manager
0 (R/W)	RSTPE	Reset Packet Engine. The PKTE_CFG.RSTPE bit resets the packet engine and the state machine logic that drives header processing, DMA, and context management. The PKTE_CFG.RSTPE bit resets the PKTE_CTL_STAT and PKTE_LEN internal registers. This bit must be written with a 1 to reset the packet engine and then re-written with a 0 to release the reset. Note that this bit should not be used by a typical application. It is provided to use during development testing or to recover from critical errors. Note that the PKTE_CTL_STAT.PADVAL and PKTE_CTL_STAT.PADCTLSTAT bit fields are only reset when in autonomous ring mode, but the PKTE_CTL_STAT.PRNGMD bit is not reset. Halt mode is not affected by this reset as well. When exiting out of halt mode, a HW reset is required or a write to the PKTE_CONT register.	
		0	Release the Packet Engine Reset
		1	Reset the Packet Engine

PE Clock Control Register

The PKTE_CLK_CTL register controls the clock enable signals. This register can be used to enable the clock for read and write access to SA registers or to enable the required clock signals for certain crypto functions. The setting of this register overrides the packet engine dynamic clock enable.

In autonomous ring mode and target command modes, this register can be all zeros; the packet engine dynamically requests the external clock manager to activate the module clocks. This register can be used in combination with the debugging interface for internal register access.

In direct host mode, the clock enable bits for the packet engine (PKTE_CLK_CTL.ENPECLK) must be enabled to write and read the SA record and state record registers. All module clocks that are required for the current operation must be enabled during processing.

Note that all the clocks are enabled by default to reset all the registers within the packet engine. After a system reset the host can program this register to disable clocks for power reduction.

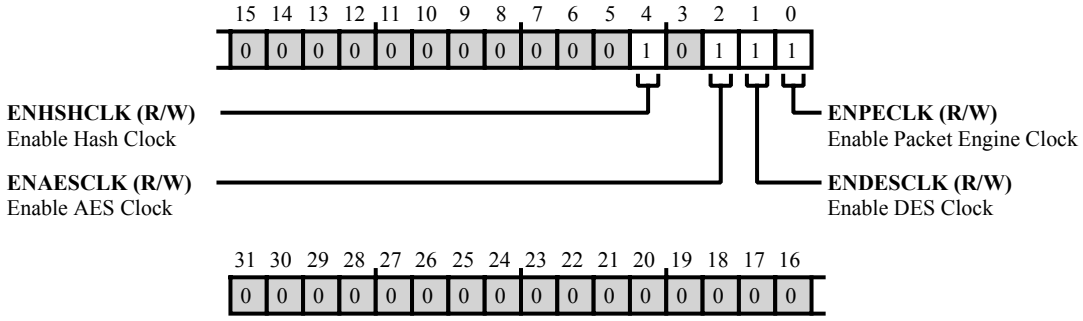

Figure 15-10: PKTE_CLK_CTL Register Diagram

Table 15-32: PKTE_CLK_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	ENHSHCLK	Enable Hash Clock. The PKTE_CLK_CTL.ENHSHCLK bit enables the clock to the hash functions.	
		0	Do not enable the hash clock
		1	Enable the hash clock
2 (R/W)	ENAESCLK	Enable AES Clock. The PKTE_CLK_CTL.ENAESCLK bit enables the clock to the AES encrypt/decrypt function.	
		0	Do not enable the AES clock
		1	Enable the AES clock

Table 15-32: PKTE_CLK_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	ENDESCLK	Enable DES Clock. The `PKTE_CLK_CTL.ENDESCLK` bit enables the clock to the DES function.	
		0	Do not enable the DES clock
		1	Enable the DES clock
0 (R/W)	ENPECLK	Enable Packet Engine Clock. The `PKTE_CLK_CTL.ENPECLK` bit enables the clock in the PKTE data path.	
		0	Do not enable the PKTE data path clock
		1	Enable the PKTE data path clock

PKTE Continue Register

A write to the PKTE_CONT register (with any value) releases the packet engine from a halt state when in halt mode.

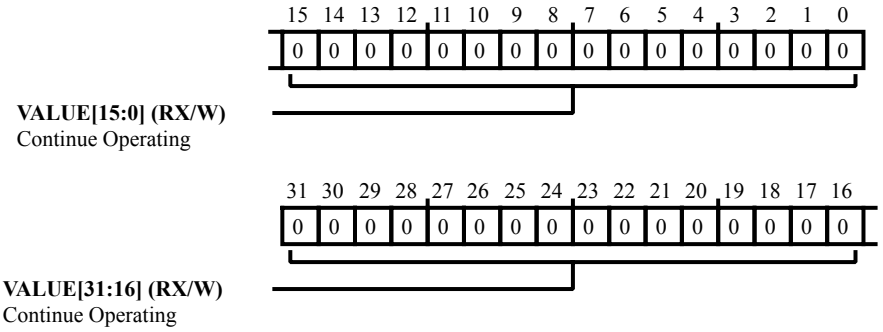

Figure 15-11: PKTE_CONT Register Diagram

Table 15-33: PKTE_CONT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Continue Operating. The PKTE_CONT.VALUE bit field releases the packet engine from a halt state when written with any value in halt mode.

Packet Engine Control Register

The PKTE_CTL_STAT register has a dual function. Together with the data in the SA, this register provides the basic command information for the packet engine to process a packet. When the packet engine successfully or unsuccessfully completes an operation, the packet engine control/status register provides the result status for the host.

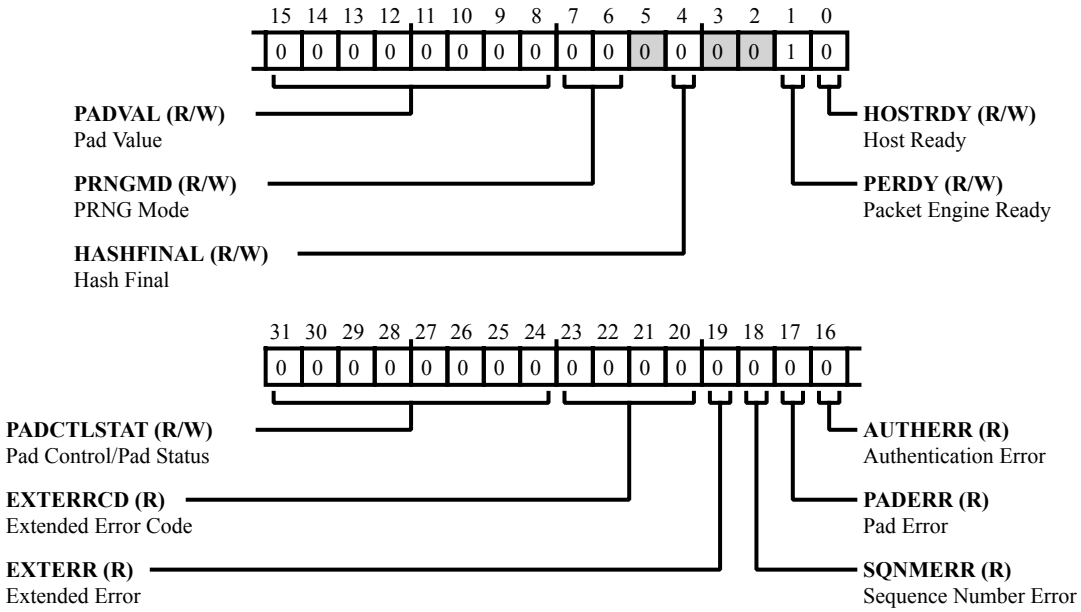

Figure 15-12: PKTE_CTL_STAT Register Diagram

Table 15-34: PKTE_CTL_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:24 (R/W)	PADCTLSTAT	Pad Control/Pad Status. The PKTE_CTL_STAT.PADCTLSTAT bit field is used to control the pad boundary for pad insertion (outbound) and after processing returns the number of inserted (outbound) or detected (inbound) pad bytes. For the command descriptor, the enumerations below provide the codes for the pad boundary for the outbound operations. This can be used for traffic flow security to conceal the number of payload bytes in an encrypted packet. For the result descriptor inbound operations that use pad types SSL, TLS, IPsec or PKCS#7, it returns the number of detected pad bytes. For all other inbound operations, it returns zero since the other pad modes do not allow implicit determination of pad count. If a pad verify failure occurs, it returns zero. For an outbound operation, it returns the number of inserted pad bytes for all pad types. The pad value includes added bytes such as the pad length and the next header field in an IPsec ESP pad type.	
		0	Align packet end to modulo 8-byte boundary
		1	Align packet end to modulo 1-byte boundary

Table 15-34: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	Align packet end to modulo 4-byte boundary
		4	Align packet end to modulo 8-byte boundary
		8	Align packet end to modulo 16-byte boundary
		16	Align packet end to modulo 32-byte boundary
		32	Align packet end to modulo 64-byte boundary
		64	Align packet end to modulo 128-byte boundary
		128	Align packet end to modulo 256-byte boundary
23:20 (R/NW)	EXTERRCD	Extended Error Code. The PKTE_CTL_STAT.EXTERRCD bit field represents an encoded error condition.	
19 (R/NW)	EXTERR	Extended Error. The PKTE_CTL_STAT.EXTERR bit field provides an extended error code.	
		0	No Extended Error
		1	Extended Error
18 (R/NW)	SQNMERR	Sequence Number Error. The PKTE_CTL_STAT.SQNMERR bit indicates that for an inbound operation, there was a fault in the anti-replay sequence number. For an outbound operation, there was a sequence number overflow condition.	
		0	No Sequence Number Error
		1	Sequence Number Error
17 (R/NW)	PADERR	Pad Error. The PKTE_CTL_STAT.PADERR bit indicates that for an inbound operation the decrypted pad does not match the expected values.	
		0	No Pad Error
		1	Pad Error
16 (R/NW)	AUTHERR	Authentication Error. The PKTE_CTL_STAT.AUTHERR bit indicates that for an inbound operation the authentication value in the packet does not match the computed value.	
		0	No Authentication Error
		1	Authentication Error
15:8 (R/W)	PADVAL	Pad Value. The PKTE_CTL_STAT.PADVAL bit field is used to pass the pad value between the host and the packet engine. Command Descriptor: (write-only)	

Table 15-34: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		For outbound operations that use pad type IPsec, the host must populate this field with the value that is to be inserted into the next header field. For the IPsec ESP operation, this next header is part of the ESP trailer of the innermost operation's header and the value must be 50 decimal. For outbound encrypt operations that use the pad type constant or constant SSL, the host must specify the fixed constant value in this field. For all other outbound and inbound operations, this field is not used. Result Descriptor: (read only) For inbound operations that use pad type IPsec, the packet engine returns the next header field that it detects. For IPsec ESP inbound operations, this is the next header field in the innermost operation's header, which will typically be the value for the payload protocol, such as TCP or UDP. However, in bundling scenarios or in IPv6 with destination option headers, another header value could be seen. For all other outbound operations, the packet engine will not update this field. For all other inbound operations, the returned pad value is zero.	
7:6 (R/W)	PRNGMD	PRNG Mode. The PKTE_CTL_STAT.PRNGMD bits select the pseudo-random number generator mode.	
		0	Operation does not use the PRNG function.
		1	PRNG Init. PRNG is initialized with a SEED, KEY and an LFSR value as defined in the SA.
		2	PRNG Generate. Pseudo-random data is generated with the LFSR as input value. Before this mode can be used, the PRNG must be initialized with a valid SEED, KEY and LFSR using PRNG Init (PKTE_CTL_STAT.PRNGMD=b'01).
		3	PRNG Test. It can be used to test the PRNG function with custom input data. Before this mode can be used, the PRNG must be initialized once with a valid SEED using PRNG Init (PKTE_CTL_STAT.PRNGMD=b'01).

Table 15-34: PKTE_CTL_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	HASHFINAL	Hash Final. When the PKTE_CTL_STAT.HASHFINAL bit is zero, the data to be hashed must be a multiple of the hash block size, 64 bytes for SHA-1, MD5, SHA-224, SHA-256. This bit is only applicable for Basic Hash, Basic Encrypt-Hash and Basic Hash-Decrypt operations that use the SHA-1, MD5, SHA-224, SHA-256 hash algorithm. The PKTE_CTL_STAT.HASHFINAL bit is overruled for HMAC operations that always completes the hash and always returns the last written value on a read by the host.	
		0	Perform Intermediate Hash Operation. The packet engine performs an intermediate hash operation by generating an intermediate hash digest on the data presented on the input. No hash pad is applied.
		1	Perform Final Hash Operation. The packet engine appends the required final hash pad and generates the final hash digest on the data presented on the input. This completes the hash operation.
1 (R/W)	PERDY	Packet Engine Ready. The PKTE_CTL_STAT.PERDY bit indicates that the packet engine has completed processing the command descriptor and returns the result descriptor with ownership set to the host. This bit can be reset to 0 by the host and the packet engine, but only the packet engine can set this bit. When the packet engine is idle (not processing), this bit always returns '1' on a read by the host.	
0 (R/W)	HOSTRDY	Host Ready. The PKTE_CTL_STAT.HOSTRDY bit indicates that the host has populated the command descriptor. This bit can be reset to 0 by the host and the packet engine, but only the host can set this bit. When the packet engine is idle (not processing), this bit always returns '0' on a read by the host.	

Starting Entry of 256-byte Data Input/Output Buffer

When in direct host mode, the source packet data is written here to be transferred to the packet engine. The host can monitor the available space in the input buffer through the PKTE_STAT register. This is also the location in the packet engine from where output data is read when in direct host mode. The host can monitor the available bytes in the output buffer through the PKTE_STAT register.

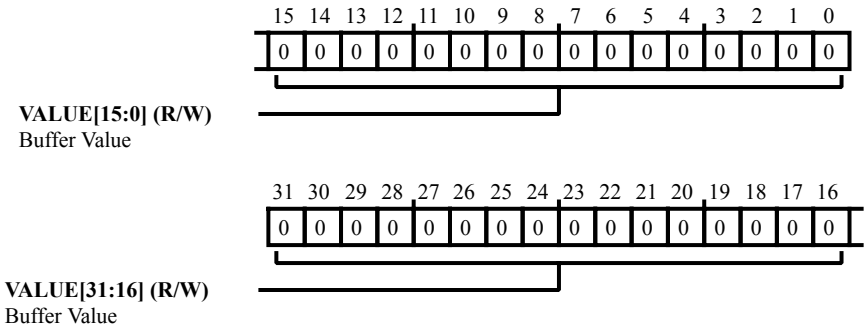

Figure 15-13: PKTE_DATAIO_BUF Register Diagram

Table 15-35: PKTE_DATAIO_BUF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Buffer Value.

Packet Engine Destination Address

The PKTE_DEST_ADDR register holds the starting (byte) address to write the result packet from the requested operation.

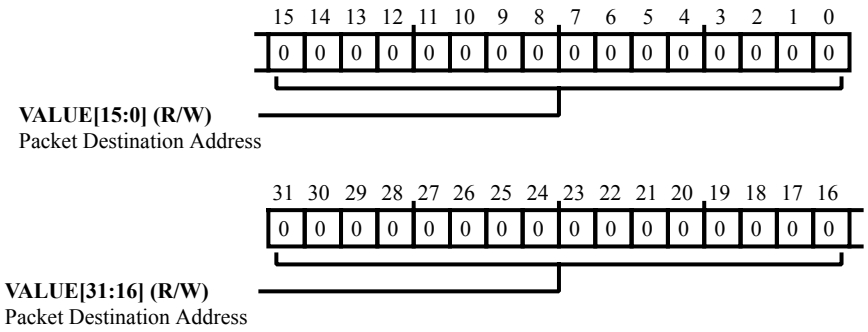

Figure 15-14: PKTE_DEST_ADDR Register Diagram

Table 15-36: PKTE_DEST_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Packet Destination Address.

Packet Engine DMA Configuration Register

The PKTE_DMA_CFG register configures the maximum burst transfer size, enables incremental transfers, and insertions IDLE cycles between two bus transfers.

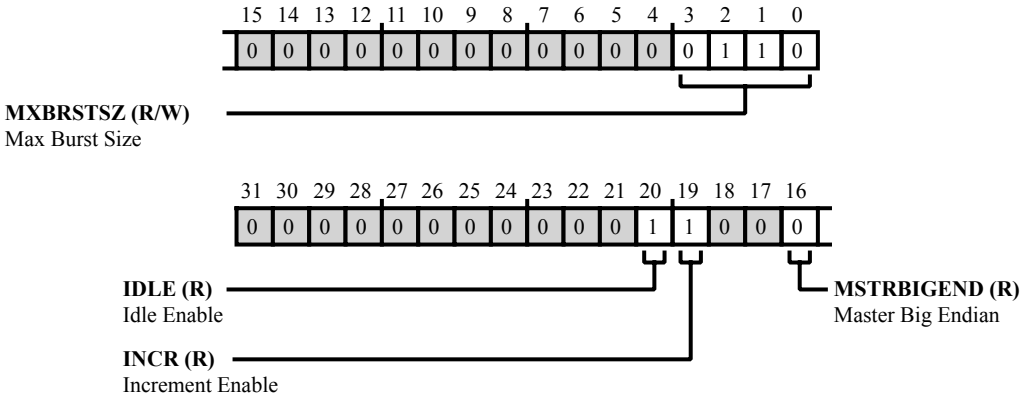

Figure 15-15: PKTE_DMA_CFG Register Diagram

Table 15-37: PKTE_DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	IDLE	Idle Enable. The PKTE_DMA_CFG.IDLE bit allows the peripheral bus master to insert one additional IDLE transfer between two successive peripheral bus master burst operations. This provides the arbiter one additional cycle to hand over the grant to another peripheral bus master.	
		0	The peripheral bus master inserts no IDLE cycle between two successive burst operations
		1	The peripheral bus master inserts one additional IDLE transfer between two successive burst operations

Table 15-37: PKTE_DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
19 (R/NW)	INCR	Increment Enable. The PKTE_DMA_CFG.INCR bit lets the peripheral bus master generate INC4, INC8 and INC16 type of burst transfers. By default, the peripheral bus master generates the largest possible incremental burst of unspecified length (INCR) with a maximum length (in bytes) as configured by the PKTE_DMA_CFG.MXBRSTSZ bit field. In case there are less than 4 bytes of data available or the 1kB boundary will be crossed using a burst operation, then a single transfer of size byte is generated. When the PKTE_DMA_CFG.INCR bit is set, the peripheral bus master generates one or more incremental burst of specified length (INC4, INC8, INC16). In case there is less data available then the smallest possible burst (INC4) or the 1kB boundary will be crossed using a burst operation, then an unspecified length burst or a single transfer of size byte is generated.	
		0	The bus master will generate only INCR burst types
		1	The bus master will generate INC4, INC8 and INC16 burst types
16 (R/NW)	MSTRBIGEND	Master Big Endian. The PKTE_DMA_CFG.MSTRBIGEND bit determines whether the engine is used in a little or big endian system.	
		0	Little endian
		1	Big endian
3:0 (R/W)	MXBRSTSZ	Max Burst Size. The PKTE_DMA_CFG.MXBRSTSZ bit field configures the maximum size of an unspecified length burst (INC) at the bus in bytes. When there is less data available than the PKTE_DMA_CFG.MXBRSTSZ bit field setting or the 1kB boundary will be crossed using a burst operation, then the length of the burst can be less than PKTE_DMA_CFG.MXBRSTSZ. Any requested transfers larger than this size are broken up in to multiple burst transfers of this size or less.	

Packet Engine Endian Configuration Register

The packet engine incorporates a powerful interface specific endian handler. This endian handler allows byte lane swapping in each direction for data passing through the host interface.

The PKTE_ENDIAN_CFG register configures the byte order function for the peripheral bus master and peripheral bus slave interface. The bits for the peripheral bus master are combined in four sets of two bits; each group configures a byte swap function for a particular DMA transfer. The same applies for the peripheral bus slave interface.

The PKTE_ENDIAN_CFG register also defines the endian swapping that occurs for host-initiated target transfers and for packet engine master DMA read and write transfers. Individual endian swap enable bits in the configuration (PKTE_CFG) register can enable the endian swap for various transaction types: command descriptors and result descriptors, SA records and state records, packet data.

In direct host mode, only target operations are supported. Only the target endian configuration of this register is applicable.

Note: This register is typically programmed once during the initialization phase, although software is allowed to dynamically change the setting in this register just before initiating a data transfer. The developer will have to analyze the benefit of the cycles needed to write the endian register dynamically versus handling endian swapping for some data structures in the host system (most modern processors support a byte swap in zero cycles). Certainly the endian swap should be set correctly for the packet data, since this represents the majority of the data transferred.

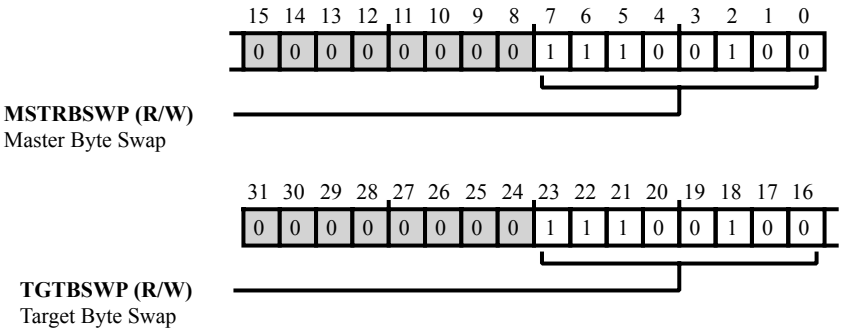

Figure 15-16: PKTE_ENDIAN_CFG Register Diagram

Table 15-38: PKTE_ENDIAN_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:16 (R/W)	TGTBSWP	Target Byte Swap. The PKTE_ENDIAN_CFG.TGTBSWP bit field configures the byte swap for peripheral bus target transfers. Note that only target word transfers are supported. Each double-bit field in this register specifies the source of the indicated byte lane. The field values are interpreted as follows: 00 = byte 0, 01 = byte 1, 10 = byte 2, 11 = byte 3. Note: Setting the value 0xE4 defines no swap (little endian) and setting value 0x1B defines a full byte swap within a 32-bit word (big endian).

Table 15-38: PKTE_ENDIAN_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	MSTRBSWP	Master Byte Swap. The PKTE_ENDIAN_CFG.MSTRBSWP bit field configures the byte swap for peripheral bus master multi-byte transfers, including command descriptors, result descriptors, SA records, state records and packet data. Separate controls in the PKTE_CFG register can enable this swap individually for each of the 4 types of data. Each double-bit field in this register specifies the source of the indicated peripheral bus byte lane. The field values are interpreted as follows: 00=byte 0, 01=byte 1, 10=byte 2, 11=byte 3. Note: Setting the value 0xE4 defines no swap (little endian) and setting value 0x1B defines a full byte swap within a 32-bit word (big endian).

Packet Engine Halt Control Register

Th PKTE_HLT_CTL register controls the packet engine halt mode. This register can be used for debugging purposes while processing in autonomous ring mode or target command mode. During the halt mode, the host can read all internal registers for examination without side-effects. When halted, the host should not write to any registers. To continue packet engine operation, the host must write to the PKTE_CONT (PKTE continue) register.

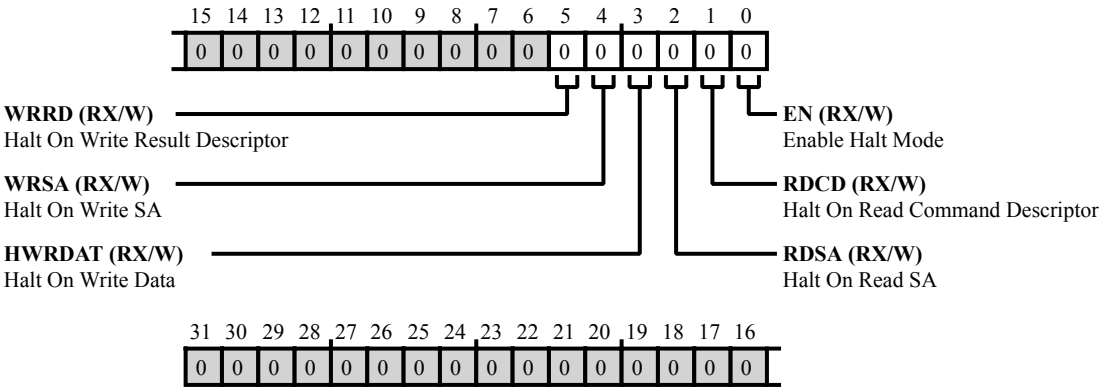

Figure 15-17: PKTE_HLT_CTL Register Diagram

Table 15-39: PKTE_HLT_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RX/W)	WRRD	Halt On Write Result Descriptor. The PKTE_HLT_CTL.WRRD bit halts the packet engine in the HALT_WRITE_STATUS state after it completes a result descriptor write operation to the result descriptor ring. The host can use this bit to examine the result descriptor that is currently in the host memory.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation
4 (RX/W)	WRSA	Halt On Write SA. The PKTE_HLT_CTL.WRSA bit halts the packet engine in the HALT_WRITE_SA state after it completes an SA write operation to the host memory. The host can use this bit to examine the security context that is currently in the host memory.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation

Table 15-39: PKTE_HLT_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RX/W)	HWRDAT	Halt On Write Data. The PKTE_HLT_CTL.HWRDAT bit halts the packet engine in the HALT_DATA state after it completes writing the result packet data to the host memory. The host can use this bit to examine the result packet that is currently in the host memory.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation
2 (RX/W)	RDSA	Halt On Read SA. The PKTE_HLT_CTL.RDSA bit halts the packet engine in the HALT_READ_SA state after it completes an SA read operation from the host memory. The host can use this bit to examine the security context that is currently in the SA registers.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation
1 (RX/W)	RDCD	Halt On Read Command Descriptor. The PKTE_HLT_CTL.RDCD bit halts the packet engine in the HALT_READ_DESCR state after it completes a command descriptor read operation from the command descriptor ring. It will halt whether the descriptor is valid or invalid. The host can use this bit to examine the command descriptor that is currently in the internal command descriptor registers.	
		0	Do not halt the Packet Engine operation
		1	Halt the Packet Engine operation
0 (RX/W)	EN	Enable Halt Mode. The PKTE_HLT_CTL.EN bit enables halt mode where the packet engine can halt processing at any processing state as indicated by bits [5:1]. When halted, the packet engine continues operation on a write to the PKTE_CONT register.	
		0	Do not enable halt mode
		1	Enable halt mode

Packet Engine Halt Status Register

The PKTE_HLT_STAT register reflects the status of the packet engine in halt mode. This register can be used for debugging purposes while processing in autonomous ring mode or target command mode. When the packet engine is halted, the host can read all internal registers for examination without side effects. The host should not write to any registers.

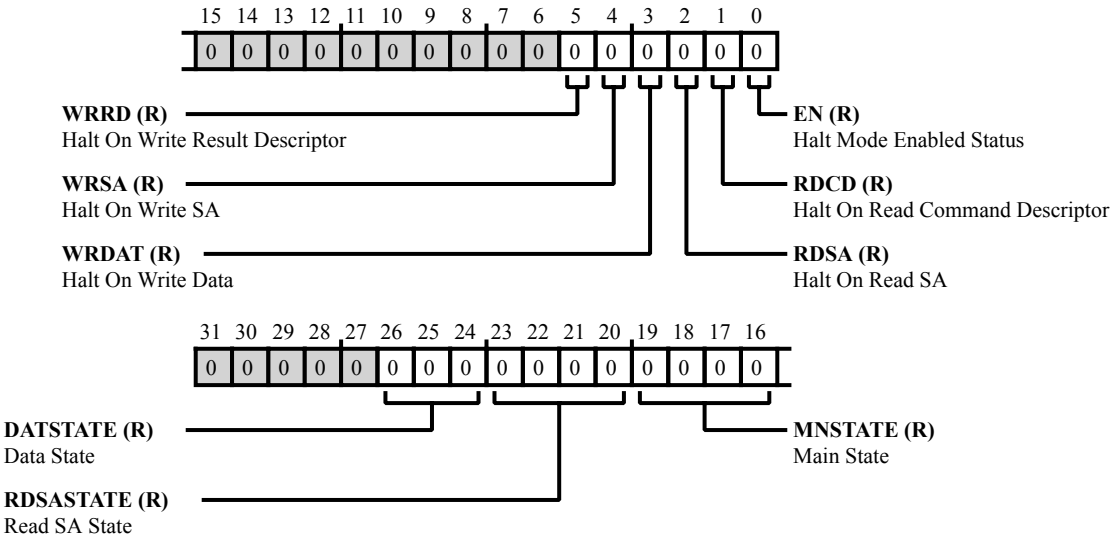

Figure 15-18: PKTE_HLT_STAT Register Diagram

Table 15-40: PKTE_HLT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26:24 (R/NW)	DATSTATE	Data State. The PKTE_HLT_STAT.DATSTATE bit field indicates the state of the packet engine read data FSM.	
		0	DATA_IDLE, no operation
		1	DATA_READ
		2	DATA_WRITE
		3	DATA_WAIT
		5	DATA_PAD_READ
		6	DATA_BYP_READ
		7	RESERVED
23:20 (R/NW)	RDSASTATE	Read SA State. The PKTE_HLT_STAT.RDSASTATE bit field indicates the state of the packet engine read SA FSM.	

Table 15-40: PKTE_HLT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	SA_IDLE, no operation
		1	SA_READ_CMD
		2	SA_READ_STATE_IV
		3-5	RESERVED
		6	RESERVED
		7	SA_READ_WAIT
		8	RESERVED
		9	SA_WRITE_PROT_HDR
		10	RESERVED
		11	SA_WRITE_IV
		12	SA_WRITE_DIGEST
		13	RESERVED
		14	RESERVED
		15	SA_WRITE_WAIT
19:16 (R/NW)	MNSTATE	Main State. The `PKTE_HLT_STAT.MNSTATE` bit field indicates the state of the packet engine main FSM.	
		0	MAIN_IDLE, no operation
		1	MAIN_READ_CD, reading command descriptor
		2	MAIN_READ_SA, reading SA
		3	MAIN_DATA, processing data
		4	MAIN_WRITE_SA, writing SA
		5	MAIN_WRITE_STATUS, writing status
		6	MAIN_WRITE_CD, updating command descriptor
		7	MAIN_WRITE_RD, updating result descriptor
		8	MAIN_INIT_WAIT, wait single clock
		9	MAIN_HALT_READ_CD, halt after read command descriptor
		10	MAIN_HALT_READ_SA, halt after read SA
		11	MAIN_HALT_DATA, halt after processing data
		12	MAIN_HALT_WRITE_SA, halt after write SA

Table 15-40: PKTE_HLT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		13	MAIN_WAIT_FOR_CLOCK, wait for clocks to be active
		15	MAIN_HALT_WRITE_RD, halt after write result descriptor
5 (R/NW)	WRRD	Halt On Write Result Descriptor. The PKTE_HLT_STAT.WRRD bit reflects the value in the PKTE_HLT_CTL.WRRD bit.	
4 (R/NW)	WRSA	Halt On Write SA. The PKTE_HLT_STAT.WRSA bit reflects the value in the PKTE_HLT_CTL.WRSA bit.	
3 (R/NW)	WRDAT	Halt On Write Data. The PKTE_HLT_STAT.WRDAT bit reflects the value in the PKTE_HLT_CTL.HWRDAT bit.	
2 (R/NW)	RDSA	Halt On Read SA. The PKTE_HLT_STAT.RDSA bit reflects the value in the PKTE_HLT_CTL.RDSA bit.	
1 (R/NW)	RDCD	Halt On Read Command Descriptor. The PKTE_HLT_STAT.RDCD bit reflects the value in the PKTE_HLT_CTL.RDCD bit.	
0 (R/NW)	EN	Halt Mode Enabled Status.	
		0	Halt mode not enabled
		1	Halt mode enabled

Interrupt Mask Disable Register

The host can use the PKTE_IMSK_DIS register to clear individual bits in the PKTE_INT_EN register for the host interrupt. This register is a bitmap for each of the possible interrupt sources: A 1 clears the interrupt enable bit, a 0 does not affect the interrupt enable bit in the PKTE_INT_EN register. Clearing the enable bits through this register avoids the time-consuming read-modify-write operation on the host.

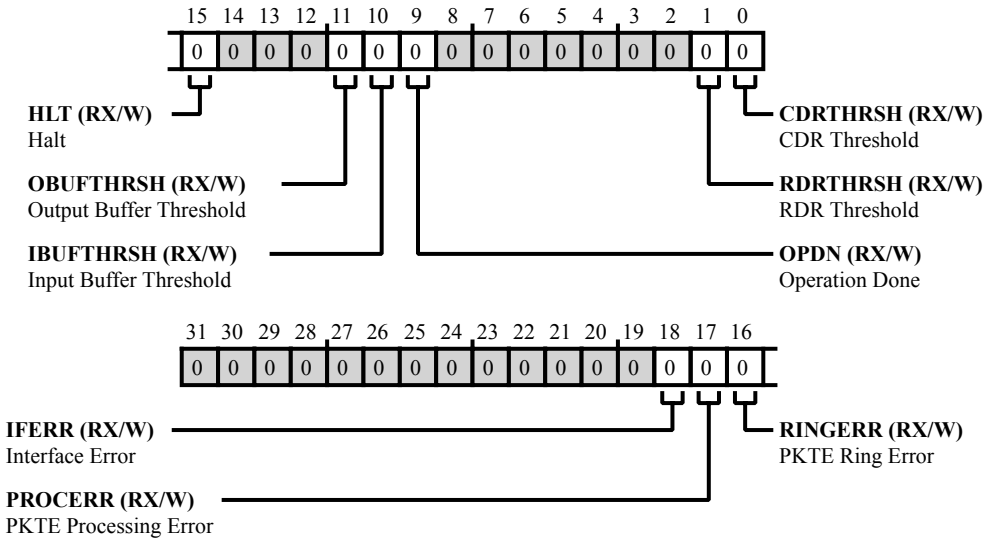

Figure 15-19: PKTE_IMSK_DIS Register Diagram

Table 15-41: PKTE_IMSK_DIS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. Write the PKTE_IMSK_DIS.IFERR bit to clear when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. Write the PKTE_IMSK_DIS.PROCERR bit to clear an extended error that occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error. Write the PKTE_IMSK_DIS.RINGERR bit to clear a CDR overflow or an RDR underflow.
15 (RX/W)	HLT	Halt. Write the PKTE_IMSK_DIS.HLT bit to clear when the packet engine is in the halt state.
11	OBUFTHRSH	Output Buffer Threshold.

Table 15-41: PKTE_IMSK_DIS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(RX/W)		Write the PKTE_IMSK_DIS.OBUFTHRSH bit to clear the output buffer counter exceeds the output buffer threshold value defined in PKTE_BUF_THRESH.OUTBUF bit.
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold. Write the PKTE_IMSK_DIS.IBUFTHRSH bit to clear when the input buffer counter is less than or equal to the input buffer threshold value defined in PKTE_BUF_THRESH.INBUF bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. Write the PKTE_IMSK_DIS.RDRTHRSH bit to clear when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the PKTE_RING_THRESH.RDRTHRSH bit, or the RD counter for the RDR in the PKTE_RDSC_CNT register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. Write the PKTE_IMSK_DIS.CDRTHRSH bit to clear when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the PKTE_RING_THRESH.CDRTHRSH bit.

Interrupt Mask Enable Register

The host can use the PKTE_IMSK_EN register to set individual bits in the PKTE_INT_EN register for the host interrupt. This register is a bitmap for each of the possible interrupt sources: A 1 sets the interrupt enable bit, a 0 does not affect the interrupt enable bit in the PKTE_INT_EN register. Setting the enable bits through this register avoids the time-consuming read-modify-write operation on the host.

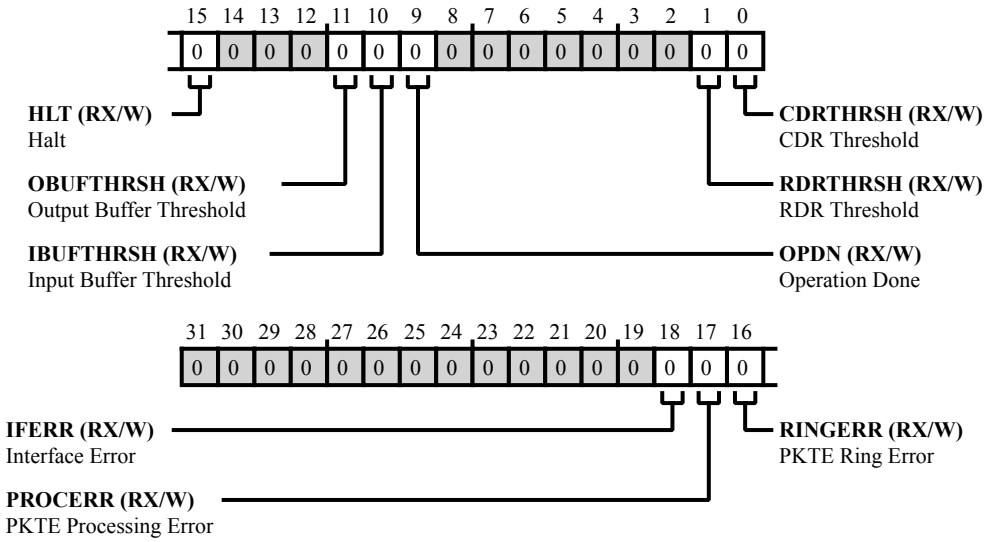

Figure 15-20: PKTE_IMSK_EN Register Diagram

Table 15-42: PKTE_IMSK_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. Set the PKTE_IMSK_EN.IFERR bit to indicate a host request for a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. Set the PKTE_IMSK_EN.PROCERR bit to indicate an extended error occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error.
15 (RX/W)	HLT	Halt. Set the PKTE_IMSK_EN.HLT bit to indicate when the packet engine is in the halt state.
11 (RX/W)	OBUFTHRSH	Output Buffer Threshold.

Table 15-42: PKTE_IMSK_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		Set the PKTE_IMSK_EN.OBUFTHRSH bit to indicate that the output buffer counter exceeds the output buffer threshold value defined in the PKTE_BUF_THRESH.OUTBUF bit.
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold. Set the PKTE_IMSK_EN.IBUFTHRSH bit to indicate the input buffer counter is less than or equal to the input buffer threshold value defined in PKTE_BUF_THRESH.INBUF bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. Set the PKTE_IMSK_EN.RDRTHRSH bit to indicate when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the PKTE_RING_THRESH.RDRTHRSH bit, or the RD counter for the RDR in PKTE_RDSC_CNT register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. Set the PKTE_IMSK_EN.CDRTHRSH bit to indicate when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the PKTE_RING_THRESH.CDRTHRSH bit.

Interrupt Masked Status Register

The PKTE_IMSK_STAT register provides interrupt status visibility to the host, after the interrupt mask is applied. This lets the host view the selected sources of interrupts that are directed to the interrupt output signal, that is connected to the system interrupt controller. As with the unmasked status register, all interrupt bits are latched and must be cleared using the PKTE_INT_CLR register in order to capture a subsequent event. A 1 indicates that the associated interrupt is present.

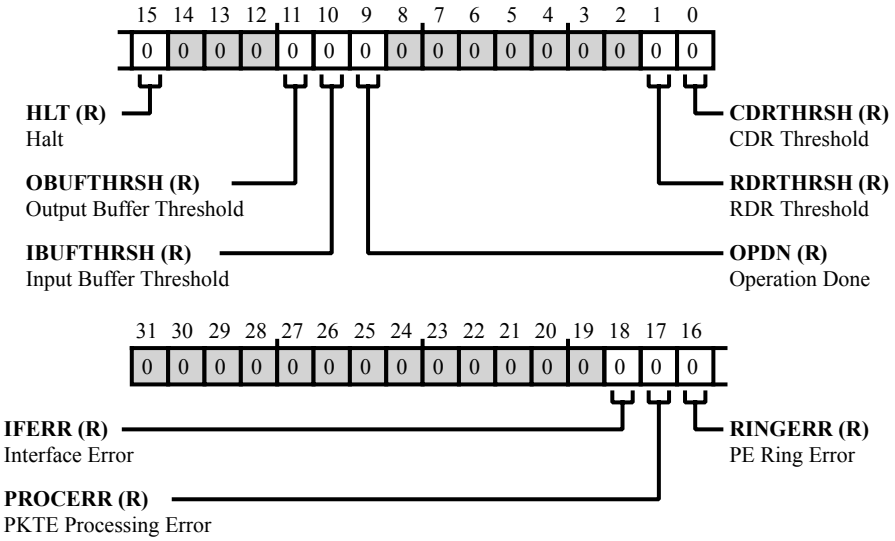

Figure 15-21: PKTE_IMSK_STAT Register Diagram

Table 15-43: PKTE_IMSK_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/NW)	IFERR	Interface Error. The PKTE_IMSK_STAT.IFERR bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (R/NW)	PROCERR	PKTE Processing Error. The PKTE_IMSK_STAT.PROCERR bit is when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (R/NW)	RINGERR	PE Ring Error. The PKTE_IMSK_STAT.RINGERR bit is set on a CDR overflow or an RDR underflow.
15 (R/NW)	HLT	Halt. The PKTE_IMSK_STAT.HLT bit is set when the packet engine is in the HALT state.
11	OBUFTHRSH	Output Buffer Threshold.

Table 15-43: PKTE_IMSK_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/NW)		The PKTE_IMSK_STAT.OBUFTHRSH bit is set when the output buffer counter exceeds the output buffer threshold value defined in PKTE_BUF_THRESH.OUTBUF bit.
10 (R/NW)	IBUFTHRSH	Input Buffer Threshold. The PKTE_IMSK_STAT.IBUFTHRSH bit is set when the input buffer counter is less than or equal to the input buffer threshold value defined in PKTE_BUF_THRESH.INBUF bit.
9 (R/NW)	OPDN	Operation Done.
1 (R/NW)	RDRTHRSH	RDR Threshold. The PKTE_IMSK_STAT.RDRTHRSH bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the PKTE_RING_THRESH.RDRTHRSH bit, or the RD counter for the RDR in PKTE_RDSC_CNT register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/NW)	CDRTHRSH	CDR Threshold. The PKTE_IMSK_STAT.CDRTHRSH bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the PKTE_RING_THRESH.CDRTHRSH bit.

Packet Engine Input Buffer Count Register

The PKTE_INBUF_CNT register provides the number of bytes available in the input buffer. The PKTE_INBUF_CNT register is used in direct host mode only.

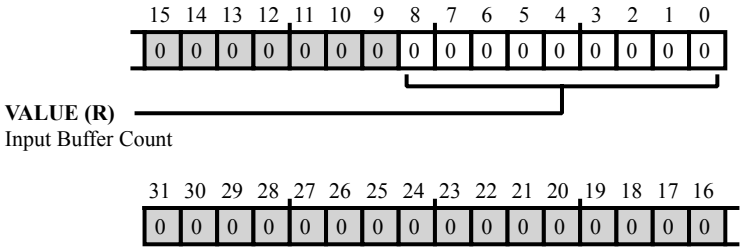

Figure 15-22: PKTE_INBUF_CNT Register Diagram

Table 15-44: PKTE_INBUF_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/NW)	VALUE	Input Buffer Count. The PKTE_INBUF_CNT.VALUE bit field provides the number of bytes in the input buffer. The packet engine decrements the counter by 4 when a 32-bit word is read from the input buffer.

Packet Engine Input Buffer Count Increment Register

A host connected through the system slave bus can increment the input buffer counter by writing a value between 4 and 256, in multiples of 4, to the lowest bits of this register. The PKTE_INBUF_INCR register is used in direct host mode only.

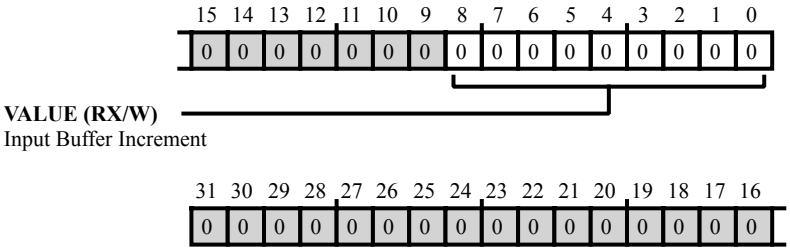

Figure 15-23: PKTE_INBUF_INCR Register Diagram

Table 15-45: PKTE_INBUF_INCR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (RX/W)	VALUE	Input Buffer Increment. The value written is added to the input buffer counter. Valid values range from 4 to 256, in multiples of 4.

Interrupt Configuration Register

The PKTE_INT_CFG register configures the interrupt type that is sent to the interrupt line connected to the system interrupt controller. (Note that this only effects the final output of the interrupt subsystem).

Configuring the interrupt output type for pulse causes the interrupt signal to pulse low for two clock cycles when activated. When set for level, the interrupt signal is set low until cleared by the host (it follows the bit in the masked status register). For the host, this is typically set to level.

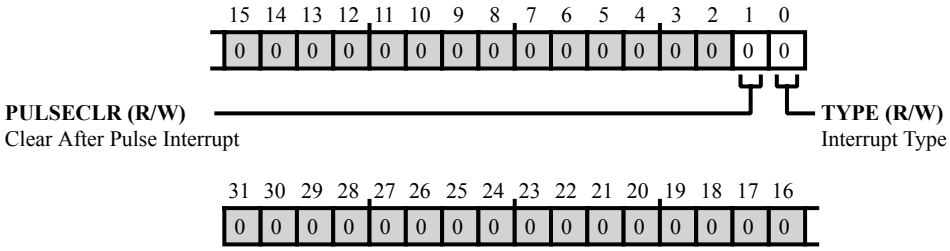

Figure 15-24: PKTE_INT_CFG Register Diagram

Table 15-46: PKTE_INT_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	PULSECLR	Clear After Pulse Interrupt. The PKTE_INT_CFG.PULSECLR bit clears the latched interrupt source after the pulse interrupt.	
		0	Manually clear pulse interrupt source. Do not automatically clear the interrupt sources after pulsing the interrupt output. Clear the source by writing to the PKTE_INT_CLR register.
		1	Automatically clear pulse interrupt source. After pulsing the interrupt output, automatically clear the sources.
0 (R/W)	TYPE	Interrupt Type. The PKTE_INT_CFG.TYPE bit selects the type, pulse or level, for the interrupt output to the system.	
		0	Level. The interrupt output is a level signal that is set low when an enabled interrupt is active until the interrupt is cleared.
		1	Pulse. The interrupt output is a two clock cycle low-active pulse, activated when an enabled interrupt is active.

Interrupt Clear Register

The PKTE_INT_CLR register allows the host processor to clear pending interrupts. A 1 written to a given bit in this register clears the corresponding interrupt. A 0 leaves the interrupt latch unchanged for that position.

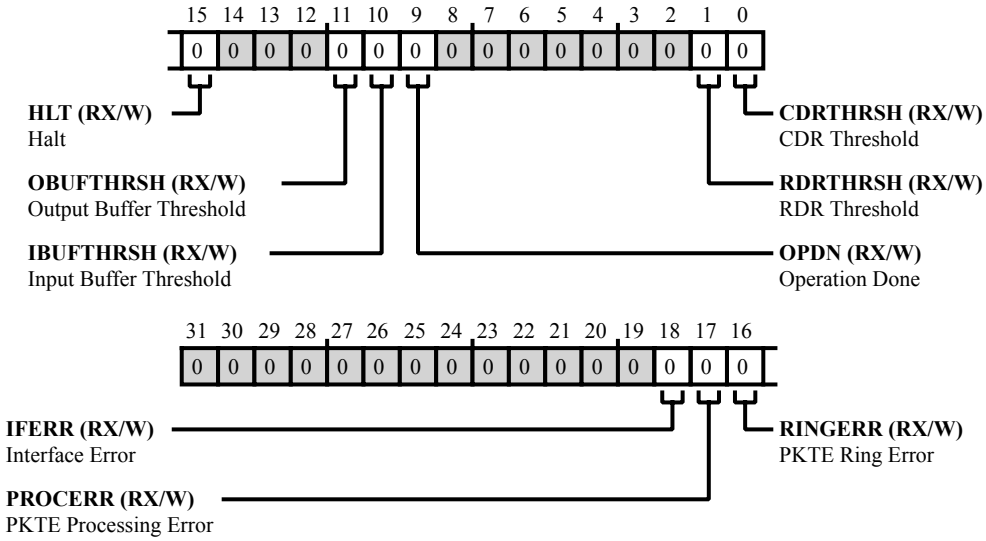

Figure 15-25: PKTE_INT_CLR Register Diagram

Table 15-47: PKTE_INT_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (RX/W)	IFERR	Interface Error. The PKTE_INT_CLR.IFERR bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (RX/W)	PROCERR	PKTE Processing Error. The PKTE_INT_CLR.PROCERR bit is set when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (RX/W)	RINGERR	PKTE Ring Error. The PKTE_INT_CLR.RINGERR bit is set on a CDR overflow or an RDR underflow.
15 (RX/W)	HLT	Halt. The PKTE_INT_CLR.HLT bit is set when the packet engine is in the HALT state.
11 (RX/W)	OBUFTHRSH	Output Buffer Threshold. The PKTE_INT_CLR.OBUFTHRSH bit is set when the output buffer counter exceeds the output buffer threshold value defined in PKTE_BUF_THRESH.OUTBUF bit.
10 (RX/W)	IBUFTHRSH	Input Buffer Threshold.

Table 15-47: PKTE_INT_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The PKTE_INT_CLR.IBUFTHRSH bit is set when the input buffer counter is less than or equal to the input buffer threshold value defined in PKTE_BUF_THRESH.INBUF bit.
9 (RX/W)	OPDN	Operation Done.
1 (RX/W)	RDRTHRSH	RDR Threshold. The PKTE_INT_CLR.RDRTHRSH bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the PKTE_RING_THRESH.RDRTHRSH bit, or the RD counter for the RDR in PKTE_RDSC_CNT register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (RX/W)	CDRTHRSH	CDR Threshold. The PKTE_INT_CLR.CDRTHRSH bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the PKTE_RING_THRESH.CDRTHRSH bit.

Interrupt Enable Register

The PKTE_INT_EN register configures the interrupt mask for the host interrupt. This register is a bitmap for each of the possible interrupt sources. A 1 enables the interrupt source and a 0 disables the source. If an interrupt source is disabled, a cleared bit also clears the matching interrupt in the PKTE_IMSK_STAT register.

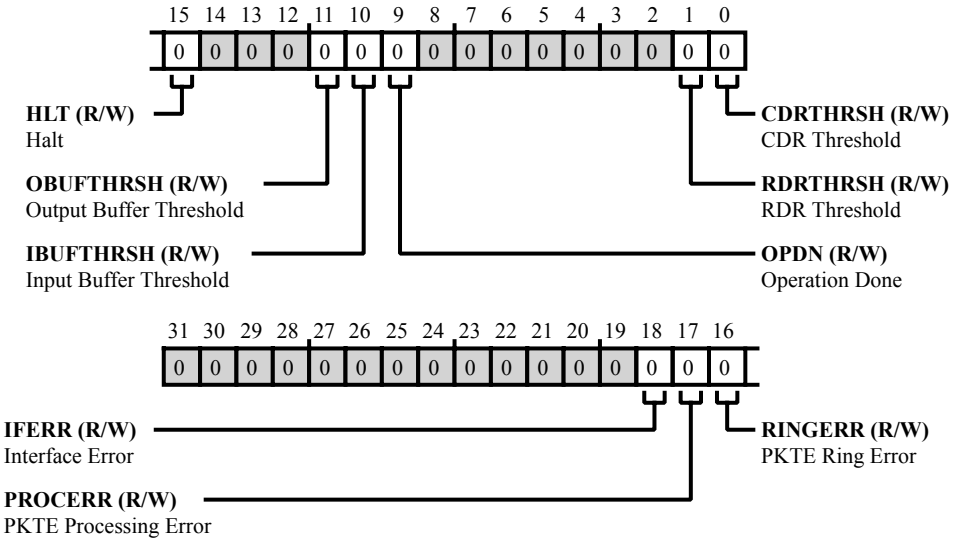

Figure 15-26: PKTE_INT_EN Register Diagram

Table 15-48: PKTE_INT_EN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/W)	IFERR	Interface Error. Set the PKTE_INT_EN.IFERR bit for host requests for a non 32-bit access to the packet engine interrupt or when the packet engine receives an error writing data back out to the host memory system.
17 (R/W)	PROCERR	PKTE Processing Error. Set the PKTE_INT_EN.PROCERR bit to enable the extended error occurred before, during or after processing the current packet in the packet engine interrupt.
16 (R/W)	RINGERR	PKTE Ring Error. Set the PKTE_INT_EN.RINGERR bit to enable the CDR overflow or RDR underflow interrupt.
15 (R/W)	HLT	Halt. Set the PKTE_INT_EN.HLT bit for when the packet engine is in the HALT state.
11 (R/W)	OBUFTHRSH	Output Buffer Threshold. Set the PKTE_INT_EN.OBUFTHRSH bit for to trigger an interrupt when the output buffer counter exceeds the output buffer threshold value defined in the PKTE_BUF_THRESH.OUTBUF bit.

Table 15-48: PKTE_INT_EN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
10 (R/W)	IBUFTHRSH	Input Buffer Threshold. Set the PKTE_INT_EN.IBUFTHRSH bit for to trigger an interrupt when the input buffer counter is less than or equal to the input buffer threshold value defined in the PKTE_BUF_THRESH.INBUF bit.
9 (R/W)	OPDN	Operation Done.
1 (R/W)	RDRTHRSH	RDR Threshold. Set the PKTE_INT_EN.RDRTHRSH bit for to trigger an interrupt when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the PKTE_RING_THRESH.RDRTHRSH bit, or the RD counter for the RDR in PKTE_RDSC_CNT register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/W)	CDRTHRSH	CDR Threshold. Set the PKTE_INT_EN.CDRTHRSH bit for to trigger an interrupt when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the PKTE_RING_THRESH.CDRTHRSH bit.

Interrupt Unmasked Status Register

The PKTE_IUMSK_STAT register provides interrupt status visibility to the host, prior to the interrupt mask being applied. Using this register, the host can view all potential sources of incoming interrupts. All of these sources, whether masked in or out, are latched in this register and must be cleared using the PKTE_INT_CLR register in order to capture a subsequent event. A 1 indicates that the associated interrupt is present.

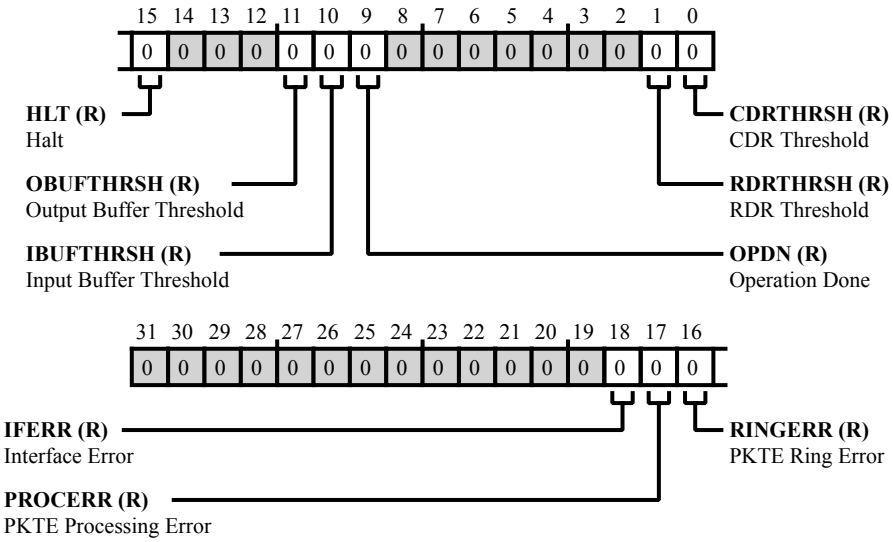

Figure 15-27: PKTE_IUMSK_STAT Register Diagram

Table 15-49: PKTE_IUMSK_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
18 (R/NW)	IFERR	Interface Error. The PKTE_IUMSK_STAT.IFERR bit is set when the host requests a non 32-bit access to the packet engine or when the packet engine receives an error writing data back out to the host memory system.
17 (R/NW)	PROCERR	PKTE Processing Error. The PKTE_IUMSK_STAT.PROCERR bit is set when an extended error occurred before, during or after processing the current packet in the packet engine.
16 (R/NW)	RINGERR	PKTE Ring Error. The PKTE_IUMSK_STAT.RINGERR bit is set on a CDR overflow or an RDR underflow.
15 (R/NW)	HLT	Halt. The PKTE_IUMSK_STAT.HLT bit is set when the packet engine is in the HALT state.
11 (R/NW)	OBUFTHRSH	Output Buffer Threshold.

Table 15-49: PKTE_IUMSK_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The PKTE_IUMSK_STAT.OBUFTHRSH interrupt is triggered when the output buffer counter exceeds the output buffer threshold value defined in PKTE_BUF_THRESH.OUTBUF bit.
10 (R/NW)	IBUFTHRSH	Input Buffer Threshold. The PKTE_IUMSK_STAT.IBUFTHRSH interrupt is triggered when the input buffer counter is less than or equal to the input buffer threshold value defined in PKTE_BUF_THRESH.INBUF bit.
9 (R/NW)	OPDN	Operation Done.
1 (R/NW)	RDRTHRSH	RDR Threshold. The PKTE_IUMSK_STAT.RDRTHRSH bit is set when the number of result descriptors for the host in the RDR exceeds the RD threshold value in the PKTE_RING_THRESH.RDRTHRSH, or the RD counter for the RDR in PKTE_RDSC_CNT register is non-zero for more than $2^{(N+10)}$ internal system clock cycles.
0 (R/NW)	CDRTHRSH	CDR Threshold. The PKTE_IUMSK_STAT.CDRTHRSH bit is set when the number of command descriptors for the packet engine in the CDR is less than or equal to the CD threshold value in the PKTE_RING_THRESH.CDRTHRSH bit.

Packet Engine Length Register

The PKTE_LEN register gives the length of the packet, the bypass data and a second set of ownership bits.

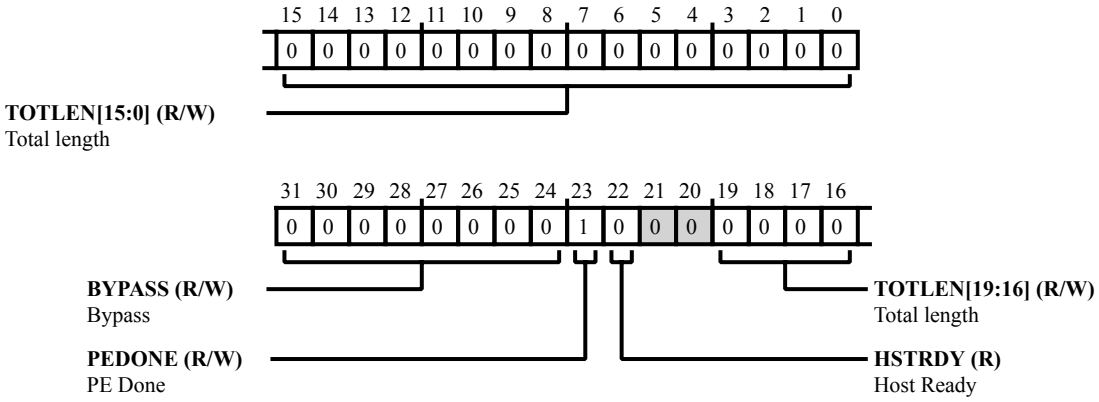

Figure 15-28: PKTE_LEN Register Diagram

Table 15-50: PKTE_LEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	BYPASS	Bypass. The PKTE_LEN.BYPASS bit field indicates the length of data in words that must bypass the packet engine and are directly copied from the source buffer to the destination buffer. The packet engine does not process this data. Valid bypass offsets range from 0 (0x00) to 255 (0xFF) words. For SRTP operations, this field specifies the offset in words between the hash and encrypt/decrypt data.
23 (R/W)	PEDONE	PE Done. The PKTE_LEN.PEDONE bit is a mirrored bit from the PKTE_CTL_STAT.PERDY bit. The bit is repeated here to guarantee ownership consistency between the first and last word. When the packet engine fetches a descriptor, these bits must match or the descriptor is discarded and fetched again.
22 (R/NW)	HSTRDY	Host Ready. The PKTE_LEN.HSTRDY bit is a mirrored bit of the PKTE_CTL_STAT.HOSTRDY bit. The bit is repeated here to guarantee ownership consistency between the first and last word. It should also be set along with the PKTE_CTL_STAT.HOSTRDY bit when the command descriptor is finished being populated. When the packet engine fetches a descriptor, these bits must match or the descriptor is discarded and fetched again.
19:0 (R/W)	TOTLEN	Total length. Command Descriptor: The PKTE_LEN.TOTLEN bit field indicates the total length (in bytes) of all data to be passed to the packet engines input buffer for an operation. Exceptions are the PRNG init and PRNG generate operations. The PRNG init operation does not require any input data; this field must be zero.

Table 15-50: PKTE_LEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		For the PRNG generate operation, this field indicates the number of pseudo-random bytes to be generated. Valid lengths range from 16 (0x00010) to 255*16 = 4080 (0x00FF0) bytes in multiples of 16 bytes. Valid lengths for the basic operation range from 1 (0x00001) to 1,048,575 (0xFFFFF) bytes. This is the length of the data to be encrypted or hashed and includes the bypass data and padding bytes. Valid lengths for IPsec ESP range from 1 (0x00001) to 65535 (0x0FFFF) bytes. This is the length of the IP payload. Valid lengths for SSL v3.0, TLS v1.x and DTLS range from 1 (0x00001) to 16383 (0x03FFF). This is the length of the payload. Valid lengths for SRTP range from 1 (0x00001) to 65535 (0x0FFFF). This is the length of the payload. Note: A length of zero bytes is illegal and will result in an error status code in the result descriptor. Result Descriptor: Upon completion of an operation, the PKTE_LEN.TOTLEN field indicates the result length of the result packet. Valid lengths range from 1 (0x001) to 1,048,575 (0xFFFFF) bytes. This includes the bypass data and padding bytes. Note: When an extended error (PKTE_CTL_STAT[18]=1) is reported in the result descriptor and no packet data is processed, this field returns zero.

Packet Engine Output Buffer Count Register

The PKTE_OUTBUF_CNT register provides the number of data bytes there are in the output buffer. The PKTE_OUTBUF_CNT register is used in direct host mode only.

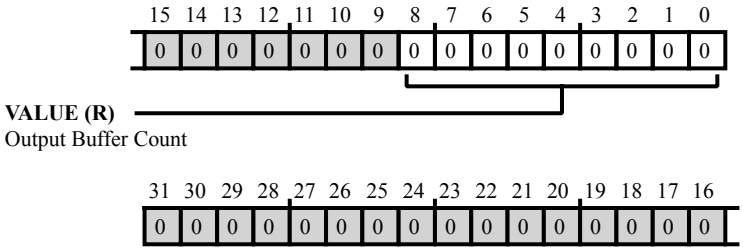

Figure 15-29: PKTE_OUTBUF_CNT Register Diagram

Table 15-51: PKTE_OUTBUF_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/NW)	VALUE	Output Buffer Count. The PKTE_OUTBUF_CNT.VALUE bit field provides the number of bytes in the output buffer. The packet engine increments the counter by 4 when a 32-bit word is written to the output buffer.

Packet Engine Output Buffer Count Decrement Register

A host connected via the system slave bus can decrement the output buffer counter by writing a value between 4 and 256, in multiples of 4, to the lowest bits of this register. The PKTE_OUTBUF_DECR register is used in direct host mode only.

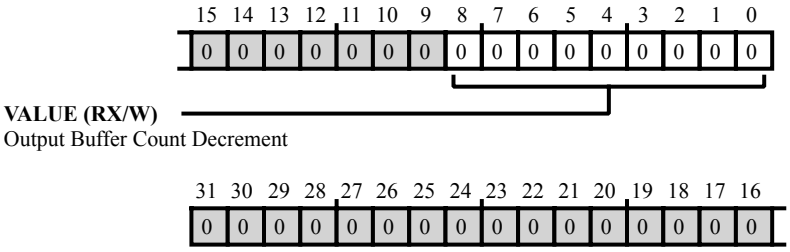

Figure 15-30: PKTE_OUTBUF_DECR Register Diagram

Table 15-52: PKTE_OUTBUF_DECR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (RX/W)	VALUE	Output Buffer Count Decrement. The PKTE_OUTBUF_DECR.VALUE bit field is the value written is subtracted to the output buffer counter. Valid values range from 4 to 256, in multiples of 4.

Packet Engine Result Descriptor Ring Base Address

The PKTE_RDRBASE_ADDR register holds the result descriptor ring base address in host memory. It is only applicable in autonomous ring mode and target command mode with RDR enabled. Note that in target command mode, the CDR is not used, but the RDR must be configured when enabled so that the packet engine knows where to write the result descriptors.

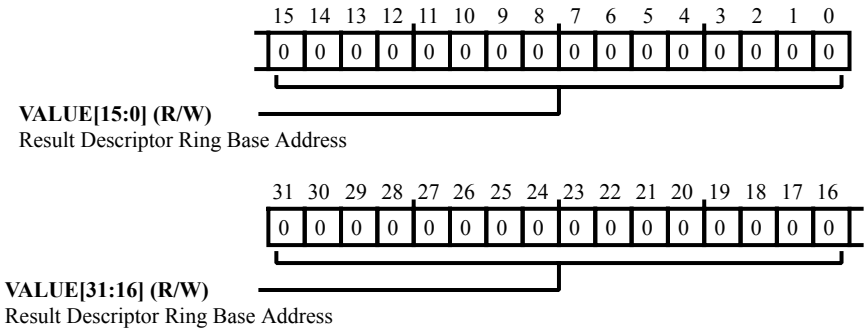

Figure 15-31: PKTE_RDRBASE_ADDR Register Diagram

Table 15-53: PKTE_RDRBASE_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Result Descriptor Ring Base Address. The PKTE_RDRBASE_ADDR.VALUE bit field specifies the base location of the result descriptor ring in the host memory space.

Packet Engine Result Descriptor Count Registers

The PKTE_RDSC_CNT register holds the counter for the number of descriptors in the Result Descriptor Ring (RDR). It is incremented by the packet engine each time a valid result descriptor is written to the RDR.

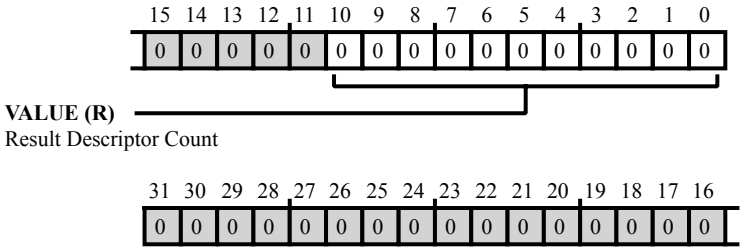

Figure 15-32: PKTE_RDSC_CNT Register Diagram

Table 15-54: PKTE_RDSC_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Result Descriptor Count. The PKTE_RDSC_CNT.VALUE bit field provides the number of result descriptors in the result descriptor ring. The packet engine increments the counter when a valid result descriptor is written to the RDR.

Packet Engine Result Descriptor Count Decrement Registers

The PKTE_RDSC_DECR register is accessible by the host connected through the system slave bus can decrement the result descriptor counter by writing a value between 1 and 255 to the lowest byte of this register.

With an RDR enabled, this is the number of result descriptors that have been read by the host. With an RDR disabled, this indicates that the host has read one valid result descriptor.

In autonomous ring mode or target command mode with the RDR enabled, the host must process 1 to 255 result descriptors from the RDR and then write this register with the number of result descriptors that have been processed by the host.

In direct host mode or target command mode with the RDR disabled, the host must read one result descriptor from the internal descriptor registers and then write this register with the value 1, to indicate that one valid descriptor is read. An RDR threshold interrupt is activated when the result descriptor counter exceeds the threshold value set in the PKTE_RING_THRESH register. This interrupt can be used to wake up a process that stalled on an empty RDR.

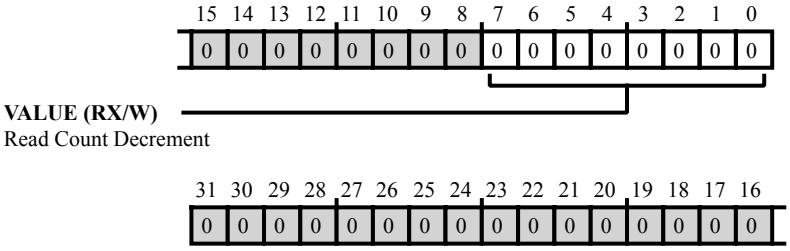

Figure 15-33: PKTE_RDSC_DECR Register Diagram

Table 15-55: PKTE_RDSC_DECR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (RX/W)	VALUE	Read Count Decrement. The value written to the PKTE_RDSC_DECR.VALUE bit field is subtracted from the result descriptor counter. The counter is protected against underflow (See the PKTE_RING_STAT register). Note that bits [10:8] should be written with zeros.

Packet Engine Ring Configuration

The PKTE_RING_CFG register configures the size (in number of descriptor ring entries minus 1) for both the command descriptor ring and result descriptor ring in host memory. This register is only applicable for autonomous ring mode and target command mode with RDR enabled.

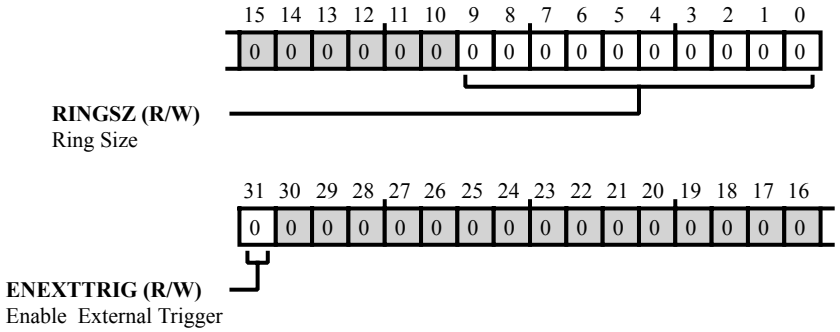

Figure 15-34: PKTE_RING_CFG Register Diagram

Table 15-56: PKTE_RING_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	ENEXTTRIG	Enable External Trigger. The PKTE_RING_CFG.ENEXTTRIG signal enables the increment of the PKTE_CDSC_CNT register through the external input pin ext_cd_cnt_incr and enables the decrement of the PKTE_RDSC_CNT fields through the external input pin ext_rd_cnt_decr.
9:0 (R/W)	RINGSZ	Ring Size. The PKTE_RING_CFG.RINGSZ bit field specifies the size of the command ring in number of descriptors, minus 1. Valid sizes range from 1 (for 2 descriptors) to 1023 (for 1024 descriptors). The accompanying result ring will have the same size.

Packet Engine Ring Pointer Status

The PKTE_RING_PTR register holds the pointers to the current entry of the Command Descriptor Ring (CDR) and Result Descriptor Ring (RDR).

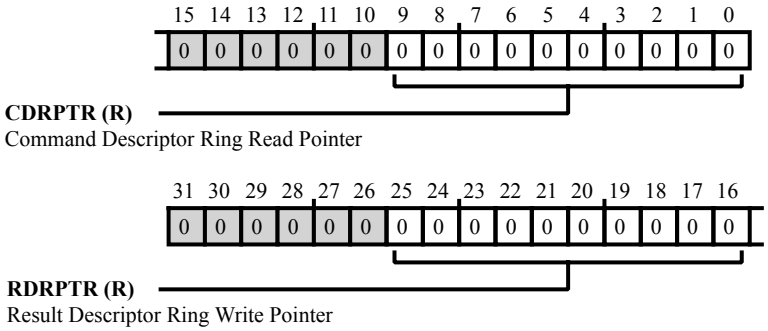

Figure 15-35: PKTE_RING_PTR Register Diagram

Table 15-57: PKTE_RING_PTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:16 (R/NW)	RDRPTR	Result Descriptor Ring Write Pointer. The PKTE_RING_PTR.RDRPTR bit field indicates the entry number in the RDR that will be written next by the packet engine. The PKTE_RING_PTR.RDRPTR bit field is reset to zero after starting up and updated after every result descriptor write DMA operation. Pointers wrap around; the maximum value this field can have equals the contents of the ring size (PKTE_RING_CFG.RINGSZ) bit field.
9:0 (R/NW)	CDRPTR	Command Descriptor Ring Read Pointer. The PKTE_RING_PTR.CDRPTR bit field indicates the entry number in the CDR that will be read next by the packet engine. The PKTE_RING_PTR.CDRPTR bit field is reset to zero after starting up and updated after every command descriptor read DMA operation. Pointers wrap around; the maximum value this field can have equals the contents of the ring size (PKTE_RING_CFG.RINGSZ) field.

Packet Engine Ring Status

The PKTE_RING_STAT register gives indication of either a Command Descriptor Ring (CDR) overflow or a Result Descriptor Ring (RDR) underflow. A ring error (ringerr) interrupt in the interrupt controller is activated on a command descriptor ring overflow or a result descriptor ring underflow. This type of error can occur when the host and the packet engine get out-of-sync. The host can read this register to retrieve information on which ring is corrupted. The corrupted ring must be reset and reinitialized. See the PKTE_CFG.RSTRING bit.

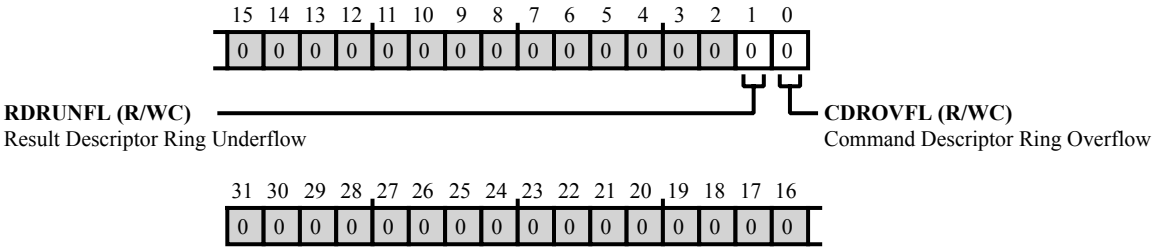

Figure 15-36: PKTE_RING_STAT Register Diagram

Table 15-58: PKTE_RING_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1 (R/WC)	RDRUNFL	Result Descriptor Ring Underflow. The PKTE_RING_STAT.RDRUNFL bit is set when the command descriptor count (PKTE_RDSC_CNT) register is decremented below zero. This bit is reset with a write of any value.
0 (R/WC)	CDROVFL	Command Descriptor Ring Overflow. The PKTE_RING_STAT.CDROVFL bit is set when the command descriptor count (PKTE_CDSC_CNT) register is incremented above the ring size (PKTE_RING_CFG.RINGSZ) bits. This bit is reset with a write of any value.

Packet Engine Ring Threshold Registers

To reduce the amount of packet engine result interrupts, the PKTE_RING_THRESH register contains threshold and time-out values.

The CDR threshold (cdrthrsh) interrupt indicates that the command descriptor counter is less than or equal to the CDR threshold (cdrthrsh) value set in this register. This interrupt can be used to wake up a process that stalled on a full CDR.

The RDR threshold (rdrthrsh) interrupt indicates that the result descriptor counter exceeds the result descriptor threshold set here, or that the result descriptor counter is non-zero for a time longer than the result descriptor time-out setting. The RDR result interrupt remains active until the result descriptor counter is decremented below the RDR threshold (rdrthrsh) value. In case the interrupt is the result of a time-out and the result descriptor counter is below the threshold value, the result descriptor counter must be decremented once before the interrupt can be cleared in the interrupt controller.

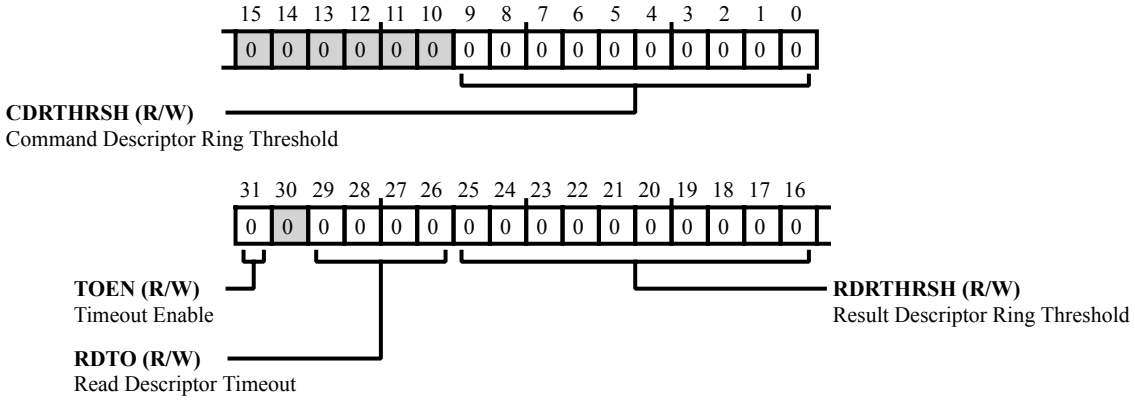

Figure 15-37: PKTE_RING_THRESH Register Diagram

Table 15-59: PKTE_RING_THRESH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	TOEN	Timeout Enable. A 1 in the PKTE_RING_THRESH.TOEN bit indicates the result descriptor timeout counter is enabled. This bit can be used to de-activate the timeout counter to save power.
29:26 (R/W)	RDTO	Read Descriptor Timeout. The timeout enable (PKTE_RING_THRESH.TOEN) bit in this register must be set to activate this PKTE_RING_THRESH.RDTO result descriptor timeout counter. The rdrthrsh interrupt activates when the RD counter for the RDR is non-zero for more than $2^{(N+10)}$ internal system clock cycles, where 'N' is the value set in this field. Valid settings range from 0 to 15. The minimum time-out value for N=0 is 1024 clock cycles and the maximum time-out value for N=15 is 33554432 clock cycles. At 100 MHz, this is 5.12 us for N=0 and ~335.55 ms for N=15.

Table 15-59: PKTE_RING_THRESH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		Note: The time-out delay may not be exact - expect a variation on the order of 1024 system clock cycles (just more than one microsecond at 100 MHz system clock frequency).
25:16 (R/W)	RDRTHRSH	Result Descriptor Ring Threshold. The rdrthrsh interrupt activates when the RD counter for the RDR exceeds the value set in the `PKTE_RING_THRESH.RDRTHRSH` field. Valid settings range from 0 to 1023.
9:0 (R/W)	CDRTHRSH	Command Descriptor Ring Threshold. The cdrthrsh interrupt activates when CD counter for the CDR is below or equal the value set in the `PKTE_RING_THRESH.CDRTHRSH` field. Valid settings range from 0 to 1023.

Packet Engine SA Address

The PKTE_SA_ADDR register holds the start address of the SA record.

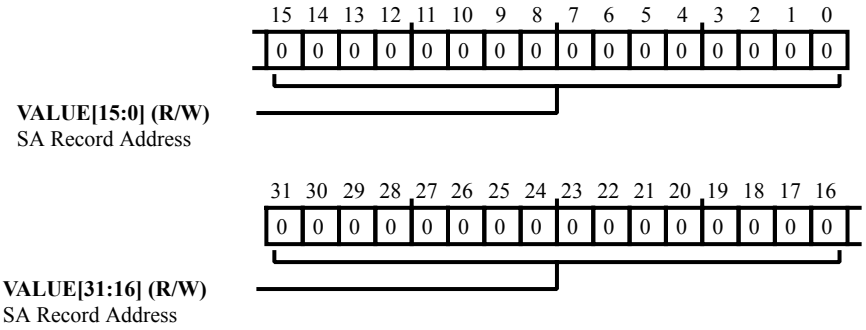

Figure 15-38: PKTE_SA_ADDR Register Diagram

Table 15-60: PKTE_SA_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Record Address. The PKTE_SA_ADDR.VALUE bit field holds the start address of the SA record.

SA Command 0

The two SA command registers, PKTE_SA_CMD0 and PKTE_SA_CMD1, are used to control the cryptographic operation of the packet engine. The PKTE_SA_CMD0 register contains the major control bits to define an operation while the PKTE_SA_CMD1 register contains the minor control bits. In direct host mode, this is a write-only register.

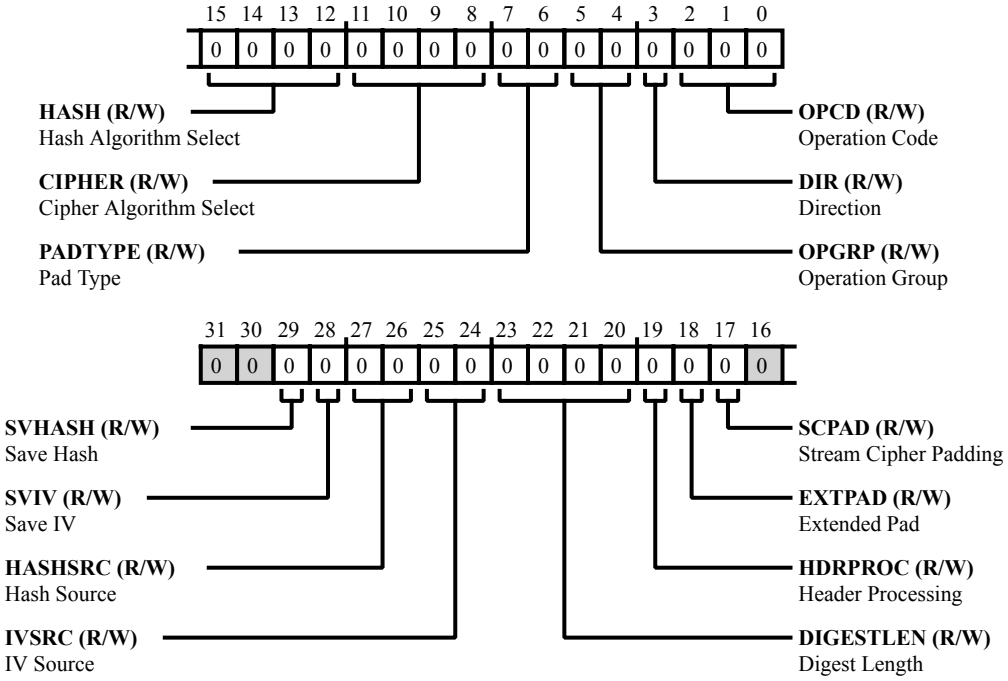

Figure 15-39: PKTE_SA_CMD0 Register Diagram

Table 15-61: PKTE_SA_CMD0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	SVHASH	Save Hash. The PKTE_SA_CMD0.SVHASH bit indicates that the Hash State is saved to the STATE_BYTE_CNT_X and STATE_IDIGEST_X fields in the SA record in memory after completion of a crypto operation.	
		0	Hash state is not saved
		1	Hash state is saved
28 (R/W)	SVIV	Save IV. The PKTE_SA_CMD0.SVIV bit field indicates that for DES or the AES the Initialization Vector (IV) is saved to the STATE_IV_X fields in the state record after completion of the crypto operation.	
		0	IV
		1	IV

Table 15-61: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
27:26 (R/W)	HASHSRC	Hash Source. The `PKTE_SA_CMD0.HASHSRC` bit field selects the source of the hash digest used by the algorithm.	
		0	From SA. Digest only hash byte count is forced to 0x40.
		1	Reserved
		2	From State. Read saved inner hash digest and saved hash byte count.
		3	No Load. Use the hash algorithm defined constants for the initial hash. Hash byte count is 0x00.
25:24 (R/W)	IVSRC	IV Source. The `PKTE_SA_CMD0.IVSRC` bit field selects the source of the initialization vector used by the crypto algorithm.	
		0	No load. Use previous result IV, not applicable for inbound data. This option should never be used for operations with DES-CBC or AES-CBC, (see RFC3602) or any AES counter modes
		1	From input buffer. The IV is provided as part of the input data stream.
		2	From State. Read STATE_IV_X, from the SA structure. Refer to inner hash digest register structure. Useful for resume operations.
		3	From internal PRNG. Not applicable for inbound operations.
23:20 (R/W)	DIGESTLEN	Digest Length. The `PKTE_SA_CMD0.DIGESTLEN` bit field defines the length of the hash digest in words as put in the output buffer.	
		0	3 Words (96-bit output)
		1	1 Word
		2	2 Words
		3	3 Words (IPsec)
		4	4 Words (MD5 and AES-based hash)
		5	5 Words (SHA-1)
		6	6 Words
		7	7 Words (SHA-224)
		8	8 Words (SHA-256)

Table 15-61: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		9	Reserved
		10	10 bytes (SRTP and TLS)
		11-15	Reserved
19 (R/W)	HDRPROC	Header Processing. The PKTE_SA_CMD0.HDRPROC bit enables header processing for protocol operations. There is no header-processing support for basic SSL, basic TLS and SRTP protocol operations as defined in the protocol group (see the Crypto and Hash Algorithms section). This bit must be zero for these operations; however, the protocol header must be supplied to the packet engine since it is part of the hash calculation. Refer to the protocol specifications for more information about header-processing support for a protocol.	
		0	No header processing
		1	Header processing; insert the protocol header for outbound operations, verify the protocol header for inbound operations.
18 (R/W)	EXTPAD	Extended Pad. The PKTE_SA_CMD0.EXTPAD bit extends the number of padding types. Used in combination with PKTE_SA_CMD0.PADTYPE.	
17 (R/W)	SCPAD	Stream Cipher Padding. The PKTE_SA_CMD0.SCPAD bit enables padding for stream ciphers algorithms.	
15:12 (R/W)	HASH	Hash Algorithm Select. The PKTE_SA_CMD0.HASH bit field selects the hash algorithm.	
		0	MD5
		1	SHA-1
		2	SHA-224
		3	SHA-256
		4-14	Reserved
		15	Null
11:8 (R/W)	CIPHER	Cipher Algorithm Select. The PKTE_SA_CMD0.CIPHER bit field selects the cipher algorithm to be used for encryption and decryption. Note: Each type of protocol operation supports different sets of crypto algorithms. Refer to the Crypto and Hash Algorithms general processing section for details of the supported algorithms.	
		0	DES
		1	Triple-DES
		2	Reserved

Table 15-61: PKTE_SA_CMD0 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		3	AES
		4-14	Reserved
		15	Null
7:6 (R/W)	PADTYPE	Pad Type. The PKTE_SA_CMD0.PADTYPE bit field indicates the type of crypto that must be generated for outbound packets or checked for inbound packets.	
		0	Select IPSec operation (if Bit 18=0); Reserved (if Bit 18=1)
		1	PKCS#7 (if Bit 18=0); Select TLS/DTLS Pad, required for TLS/DTLS operation (if Bit 18=1)
		2	Constant pad (if Bit 18=0); Select Constant SSL Pad, required for SSL operation (if Bit 18=1)
		3	Zero pad (if Bit 18=0), Reserved (if Bit 18=1)
5:4 (R/W)	OPGRP	Operation Group. The PKTE_SA_CMD0.OPGRP bit field defines the operation groups. Refer to the Basic Operations and Decoding section for more information.	
		0	Basic operation group
		1	Protocol operation group
		2	Extended protocol operations group
		3	Reserved
3 (R/W)	DIR	Direction. The PKTE_SA_CMD0.DIR bit field selects the direction of operation.	
		0	Outbound operations
		1	Inbound operations
2:0 (R/W)	OPCD	Operation Code. The PKTE_SA_CMD0.OPCD bit field selects the operation within the operation group.	

SA Command 1

The PKTE_SA_CMD1 register contains the minor control bits that define an operation. In direct host mode, this is a write-only register.

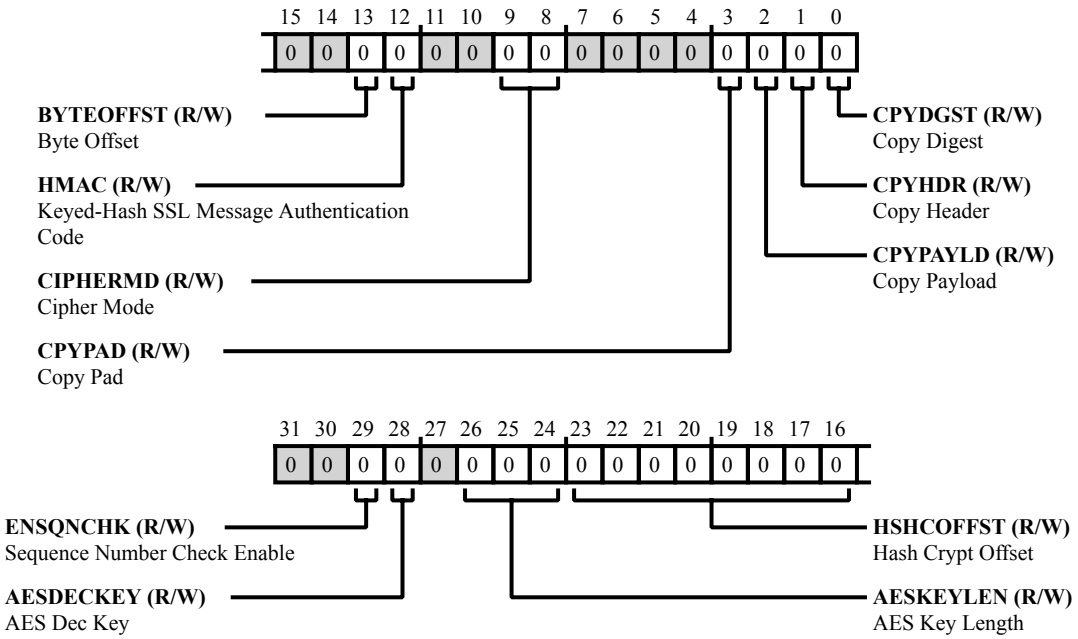

Figure 15-40: PKTE_SA_CMD1 Register Diagram

Table 15-62: PKTE_SA_CMD1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	ENSQNCHK	Sequence Number Check Enable. The PKTE_SA_CMD1.ENSQNCHK bit defines that the key in the SA key field is an AES encrypt key or an AES decrypt key.	
		0	Disable sequence number check
		1	Enable sequence number check
28 (R/W)	AESDECKEY	AES Dec Key. If the PKTE_SA_CMD1.AESDECKEY bit is set, the key in loaded in the PKTE_SA_KEY[n] registers are expected to be the key from the last round from key expansion. If not set, the key loaded in the PKTE_SA_KEY[n] registers are expected to be the same key used during the encryption process.	
		0	AES key is an encrypt key.
		1	AES key is a decrypt key.
26:24 (R/W)	AESKEYLEN	AES Key Length.	

Table 15-62: PKTE_SA_CMD1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The PKTE_SA_CMD1.AESKEYLEN bit field select the size of the key used for the AES algorithm in increments of 64 bits.	
		0-1	Reserved
		2	128 Bits
		3	192 Bits
		4	256 Bits
		5-7	Reserved
23:16 (R/W)	HSHCOFFST	Hash Crypt Offset. For Basic Encrypt-Hash and Basic Hash-Decrypt operations, the PKTE_SA_CMD1.HSHCOFFST bit field specifies the offset between the hash data and the encrypt/decrypt data. The data to be hashed is assumed to come first, with an offset to the beginning of encrypt/decrypt data. When PKTE_SA_CMD1.BYTEOFFST, bit 13, is zero, then the offset is defined in 32-bit words. When an initialization vector is loaded through the input buffer, valid values range from IV size to 255. In all other cases, valid values range from 0 to 255. When PKTE_SA_CMD1.BYTEOFFST, bit 13, is one, then the offset is defined in 8-bit bytes. When an initialization vector is loaded through the input buffer, valid values range from IV size to 255. In all other cases, valid values range from 4 to 255. (The IV size is two words for DES, Triple-DES and AES-CTR and four words for AES-CBC and AES-ICM operations). Other operations do not use these bits (a default value is applied by the packet engine).	
13 (R/W)	BYTEOFFST	Byte Offset. The PKTE_SA_CMD1.BYTEOFFST bit defines how the PKTE_SA_CMD1.HSHCOFFST, bits of this register are used.	
		0	HASH_CRYPT_OFFSET is defined in 32-bit words
		1	HASH_CRYPT_OFFSET is defined in 8-bit bytes
12 (R/W)	HMAC	Keyed-Hash SSL Message Authentication Code. For basic operations that include hashing, the PKTE_SA_CMD1.HMAC bit enables the HMAC processing, which calls for an extra outer hash operation.	
		0	Standard Hash
		1	HMAC Processing

Table 15-62: PKTE_SA_CMD1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	CIPHERMD	Cipher Mode. The PKTE_SA_CMD1.CIPHERMD bit field selects the crypto mode to be used for the cipher algorithm.	
		0	Electronic Code Book (ECB) used for DES and AES
		1	Cipher Block Chaining (CBC) used for DES and AES
		2	AES Counter Mode (CTR) for IPsec using a 32-bit counter
		3	AES Integer Counter Mode (ICM) for SRTP using a 16-bit counter. Note: This is implemented as a 32-bit counter, the Host must check for an overflow from the value 0xFF to zero and then refresh the IV.
3 (R/W)	CPYPAD	Copy Pad. The PKTE_SA_CMD1.CPYPAD bit indicates that the padding data for an inbound operation is copied to the output buffer and saved in memory.	
		0	Do not copy the padding to output
		1	Copy padding to output
2 (R/W)	CPYPAYLD	Copy Payload. The PKTE_SA_CMD1.CPYPAYLD bit indicates that the payload data is copied to the output buffer and saved in memory.	
		0	Do not copy the payload to output
		1	Copy payload to output
1 (R/W)	CPYHDR	Copy Header. The PKTE_SA_CMD1.CPYHDR bit indicates that the protocol header is copied to the output buffer and saved in memory. For Basic Encrypt-Hash and Basic Hash-Decrypt operations, the header is defined as the Hash/Crypt Offset data (authenticated only).	
		0	Do not copy the header to output
		1	Copy header to output
0 (R/W)	CPYDGST	Copy Digest. The PKTE_SA_CMD1.CPYDGST bit copies the hash result is to the output buffer and saves in memory. The length of the hash result is defined by the PKTE_SA_CMD0.DIGESTLEN field.	
		0	Do not copy hash result to output
		1	Copy hash result to output, when the command descriptor PKTE_CTL_STAT.HASHFINAL bit is set.

SA Inner Hash Digest Registers

The PKTE_SA_IDIGEST[n] registers are a set of eight 32-bit read/write registers.

For MD5, SHA-1, SHA-224 and SHA-256, these read/write registers are used to enter a start hash state, and to read the interim or final hash digest.

For IPsec, TLS and DTLS operations that make use of MD5, SHA-1, SHA-224 or SHA-256 with basic hash or HMAC authentication with the PKTE_SA_CMD0.HASHSRC bits = 00 (from SA), these registers hold the pre-computed inner hash digest. This is the hash of the hash-key padded with 0x36 hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-MD5 with the PKTE_SA_CMD0.HASHSRC bits = 00 (from SA), these registers hold the inner hash pre-compute; this is the hash of the MAC_WRITE_SECRET padded with 0x36 hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-SHA-1 with the PKTE_SA_CMD0.HASHSRC bits = 00 (from SA), these registers hold the MAC_WRITE_SECRET. Note that it is not possible to calculate a hash pre-compute for SHA-1 in combination with SSL-MAC (specification flaw). The packet engine appends the hash-key pad (0x36 hex) and sets the starting hash byte count automatically to 60 decimal / 0x3C hex (to indicate that 60 bytes have already been prepared for the hash).

The reset value for these registers is zero.

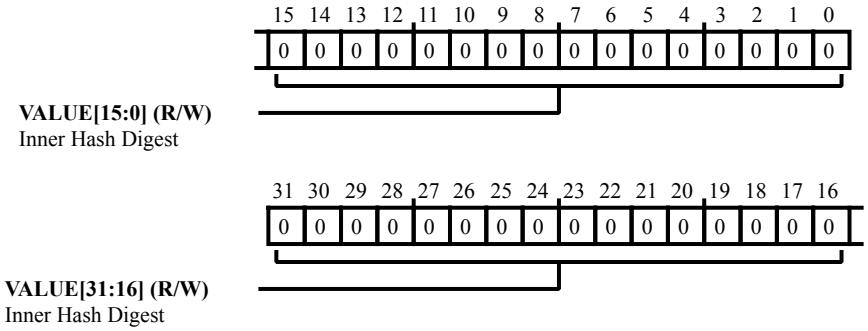

Figure 15-41: PKTE_SA_IDIGEST[n] Register Diagram

Table 15-63: PKTE_SA_IDIGEST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Inner Hash Digest.

SA Key Registers

These are the PKTE_SA_KEY[n] registers for DES, Triple-DES, and AES: A set of eight 32-bit write only registers. The reset value of these registers is zero.

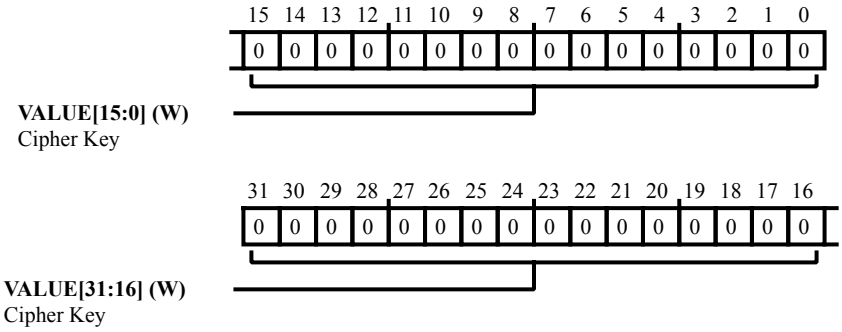

Figure 15-42: PKTE_SA_KEY[n] Register Diagram

Table 15-64: PKTE_SA_KEY[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Cipher Key.

SA Initialization Vector Register

The PKTE_SA_NONCE register is used for operations that make use of the IV value loaded from the SA record. This register is used both to enter a starting IV state, as well as for reading the interim or final IV. For IPsec outbound operations, it is recommended that the automatic IV insertion mode be used, this register is not needed. For IPsec inbound operations, the IV is extracted from the header of the packet.

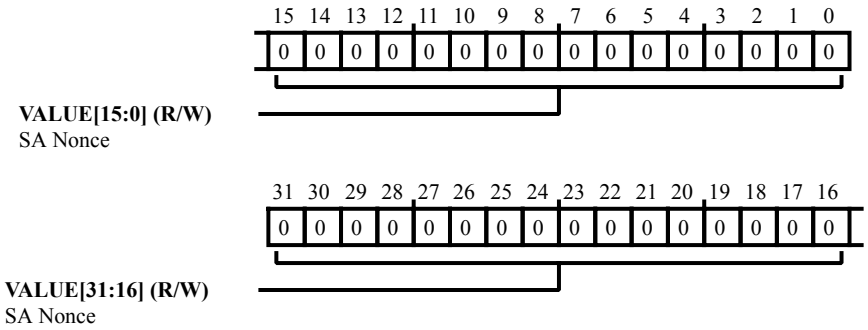

Figure 15-43: PKTE_SA_NONCE Register Diagram

Table 15-65: PKTE_SA_NONCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Nonce.

SA Outer Hash Digest Registers

The PKTE_SA_ODIGEST[n] registers are a set of five eight 32-bit write-only registers.

For write operations, these registers contain the pre-computed outer hash digest for IPsec operations with basic HMAC operations with the PKTE_SA_CMD0.HASHSRC bits = 00 (from SA).

For MD5, SHA-1, SHA-224 and SHA-256, these read/write registers hold a start hash state, or the interim outer hash digest. They are only used for HMAC processing.

For IPsec, SSL, TLS, DTLS and SRTP operations that make use of MD5, SHA-1, SHA-224 or SHA-256 with HMAC authentication with the PKTE_SA_CMD0.HASHSRC bits = 00 (from SA), these registers hold the pre-computed outer hash digest. This is the hash of the hash-key padded with 0x5C hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-MD5 with the PKTE_SA_CMD0.HASHSRC bits = 00 (from SA), these registers hold the outer hash pre-compute; this is the hash of the MAC_WRITE_SECRET padded with 0x5C hex. The starting hash byte count is automatically set to 64 decimal / 0x40 hex (to indicate that 64 bytes have already been processed through the hash).

For SSL operations that make use of SSL-MAC-SHA-1 with the PKTE_SA_CMD0.HASHSRC bits = 00 (from SA), these registers hold the MAC_WRITE_SECRET. Note that it is not possible to calculate a hash pre-compute for SHA-1 in combination with SSL-MAC (specification flaw). The packet engine appends the required hash-key pad (0x5C hex) and sets the starting hash byte count automatically to 60 decimal / 0x3C hex (to indicate that 60 bytes have already been prepared for the hash).

The reset value for these registers is zero.

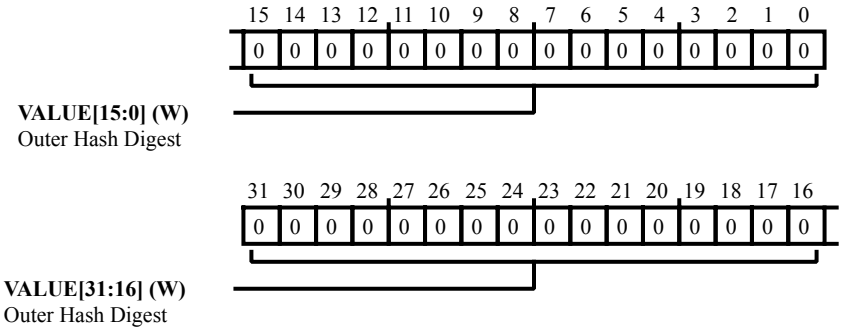

Figure 15-44: PKTE_SA_ODIGEST[n] Register Diagram

Table 15-66: PKTE_SA_ODIGEST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Outer Hash Digest.

SA Ready Indicator

In direct host mode, a write to the PKTE_SA_RDY register triggers the packet engine to start processing using the command descriptor, SA record and state record in the packet engine registers. This register MUST be written for all direct host mode packet operations. It is intended that this register is written in sequence; as the entire SA record is written.

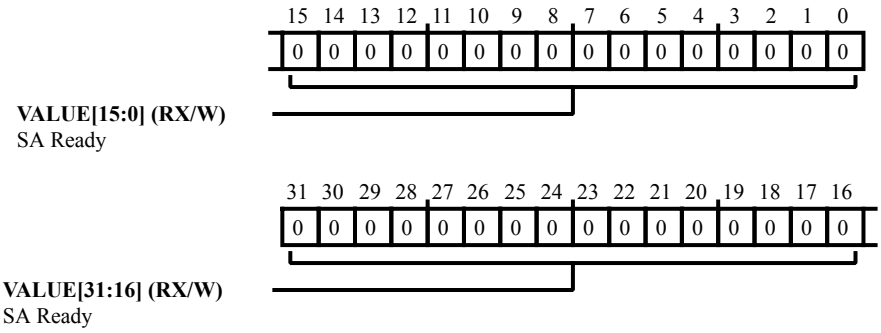

Figure 15-45: PKTE_SA_RDY Register Diagram

Table 15-67: PKTE_SA_RDY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	SA Ready.

SA Sequence Number Register

The PKTE_SA_SEQNUM[n] registers are a set of two read/write registers and are used for IPsec ESP, SSL, TLS, DTLS operations to specify the anti-replay sequence number value that is to be placed in the ESP header (outbound), or to be checked against for inbound packets. The packet engine manages this counter value for both inbound and outbound operations.

Outbound: The host writes the counter value stored in the SA record to this register to start an IPsec, SSL, TLS, DTLS operation. The packet engine automatically increments the count if header processing is selected. Upon successful completion, the host reads back this value and writes it to the SA record.

Inbound: The host writes the counter value stored in the SA record to this register to start an IPsec or DTLS operation. The packet engine automatically performs the specified inbound processing (per RFC 4303) as it processes the packet. As a result, the expected count value may or may not be updated during processing. Upon successful completion, the host should read back this value and write it to the SA record.

Note: The description is only for the direct host mode. The sequence number for autonomous ring mode and target command mode are updated by the packet engine.

The reset value of this register is zero.

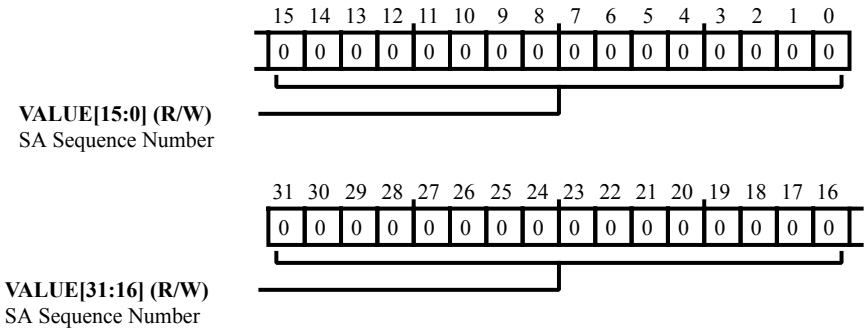

Figure 15-46: PKTE_SA_SEQNUM[n] Register Diagram

Table 15-68: PKTE_SA_SEQNUM[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Sequence Number.

SA Sequence Number Mask Registers

The PKTE_SA_SEQNUM_MSK[n] registers are a set of two read/write registers and are used for IPsec ESP and DTLS operations to specify the anti-replay sequence number mask value for inbound operations. The packet engine manages this counter value automatically.

Inbound: The host writes the counter value stored in the SA record into this register upon starting an IPsec, DTLS operation. The packet engine automatically performs the specified inbound processing (per RFC 4303) as it processes the packet. As a result, the new mask value may or may not be updated during processing. Upon successful completion, the host should read back this value and write it to the SA record.

Outbound: not used.

Note that the above description only applies to the direct host mode, for autonomous ring mode and target command mode the packet engine extracts the sequence number mask from the SA record.

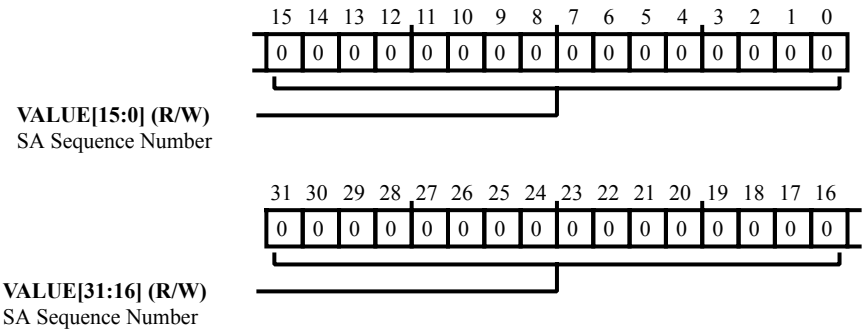

Figure 15-47: PKTE_SA_SEQNUM_MSK[n] Register Diagram

Table 15-69: PKTE_SA_SEQNUM_MSK[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SA Sequence Number.

SA SPI Register

For IPsec operations, the PKTE_SA_SPI register is written with the SPI (Security Parameters Index) associated with the inbound or outbound flow.

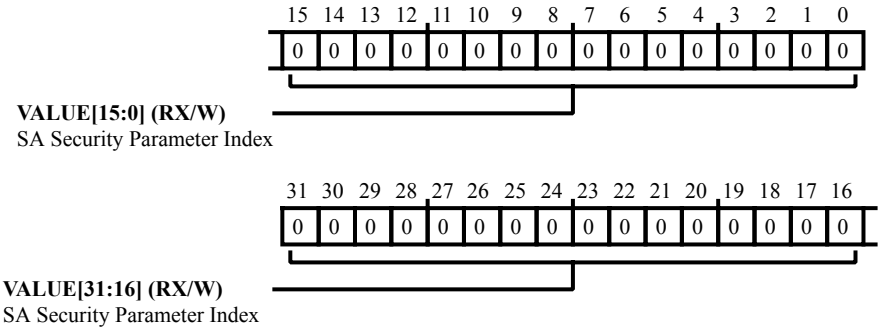

Figure 15-48: PKTE_SA_SPI Register Diagram

Table 15-70: PKTE_SA_SPI Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	SA Security Parameter Index.

Packet Engine Source Address

The PKTE_SRC_ADDR register holds the starting (byte) address for the packet to be processed.

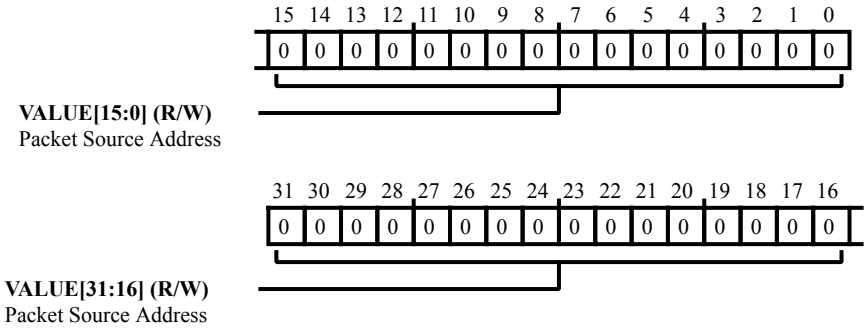

Figure 15-49: PKTE_SRC_ADDR Register Diagram

Table 15-71: PKTE_SRC_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Packet Source Address. The PKTE_SRC_ADDR.VALUE bit field holds the starting (byte) address for the packet to be processed.

Packet Engine Status Register

The PKTE_STAT register is used to provide the status of the packet engine. This register is useful in the direct host mode to determine when data must be written to or read from the packet engine, or for debugging the software when errors occur. This register can be ignored in autonomous ring mode and target command mode where the DMA engine controls the packet data I/O. This is a read-only register. A write to any of the bits has no effect.

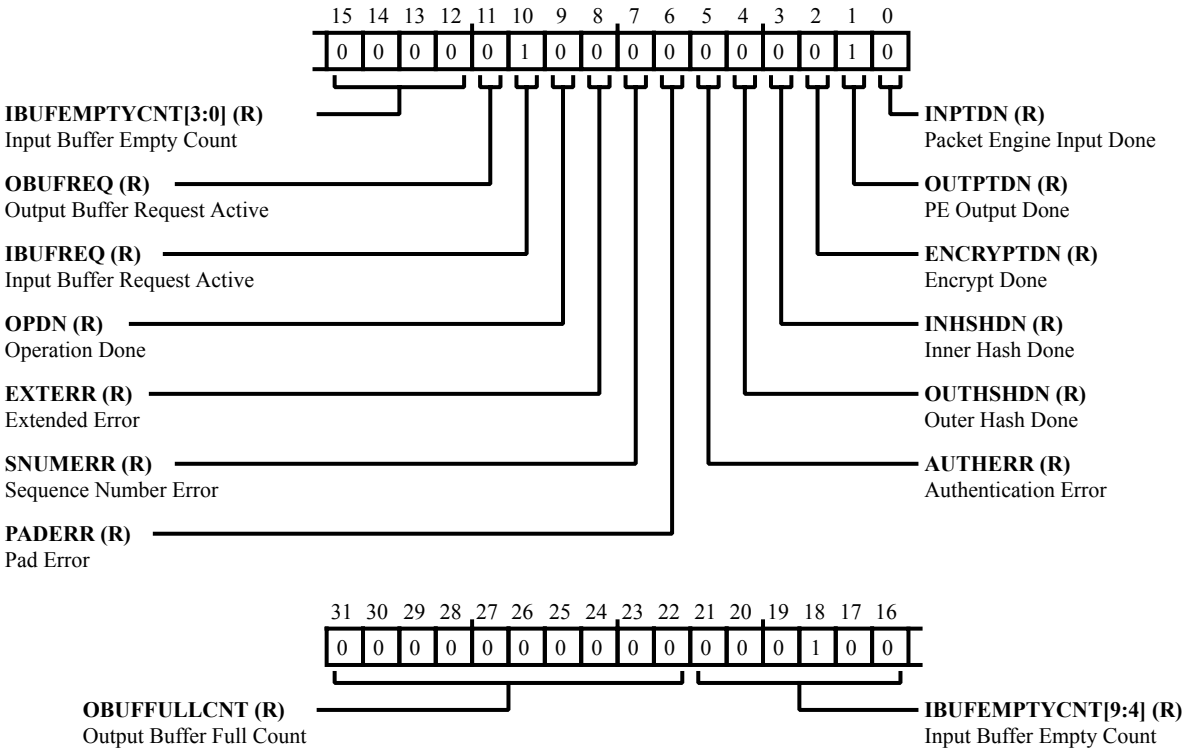

Figure 15-50: PKTE_STAT Register Diagram

Table 15-72: PKTE_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:22 (R/NW)	OBUFFULLCNT	Output Buffer Full Count. The PKTE_STAT.OBUFFULLCNT bit field indicates the number of 32-bit words that are available in the packet engine output buffer. It works in conjunction with bit 11 from this register. When bit 11 is asserted, to indicate a request for output, the word count matches the specified output buffer threshold setting in the PKTE_BUF_THRESH register. For the last output for a given packet, any value from 1 dword to the full output buffer threshold can be seen. Transfers must be a multiple of full dwords. The application must read the PKTE_LEN field in the result descriptor to determine the exact byte-length of the result.
21:12 (R/NW)	IBUFEMPTYCNT	Input Buffer Empty Count.

Table 15-72: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The PKTE_STAT.IBUFEMPTYCNT bit field indicates the number of 32-bit empty spaces that are available in the packet engine input buffer. It works in conjunction with the PKTE_STAT.IBUFREQ bit (10) from this register. The value in the register is deducted from the specified packet length, so will never exceed the number of dwords that remain in the packet. For packets smaller than the buffer size, this register typically indicates that buffer space is available for the entire packet (rounded up to the nearest dword). For very large packets, these bits usually have a value around the maximum buffer size, indicating that the full input buffer is available.	
11 (R/NW)	OBUFREQ	Output Buffer Request Active. The PKTE_STAT.OBUFREQ bit indicates that the packet engine requests output data to be read from the output buffer.	
		0	No request for output data
		1	Request for output data
10 (R/NW)	IBUFREQ	Input Buffer Request Active. The PKTE_STAT.IBUFREQ bit indicates that the packet engine requests input data to be written to the input buffer.	
		0	No request for input data
		1	Request for input data
9 (R/NW)	OPDN	Operation Done. The PKTE_STAT.OPDN bit indicates that the packet engine has finished processing a packet when in direct host mode. This bit is zero in autonomous ring mode and target command mode.	
		0	Packet engine is idle
		1	Packet engine has finished processing a packet
8 (R/NW)	EXTERR	Extended Error. The PKTE_STAT.EXTERR bit indicates that an extended error occurred for this packet. For more information, refer to table Extended Error Codes - Status Encoding.	
		0	No extended error
		1	Extended error

Table 15-72: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	SNUMERR	Sequence Number Error. For an inbound operation, the PKTE_STAT.SNUMERR bit indicates that there was a fault in the anti-replay sequence number. For an outbound operation, there was a sequence number overflow condition. For more information, refer to table Extended Error Codes - Status Encoding.	
		0	No sequence number error
		1	Input done, all bytes written to input buffer
6 (R/NW)	PADERR	Pad Error. The PKTE_STAT.PADERR bit indicates that an inbound crypto pad fault is detected. For more information about pad verification, refer to the Pad Verification and Consumption section.	
		0	No pad error
		1	Pad error
5 (R/NW)	AUTHERR	Authentication Error. The PKTE_STAT.AUTHERR bit indicates that an inbound ICV (for IPsec) or TAG (for SRTP) or MAC (for SSL/TLS/DTLS) fault is detected; the value carried within the packet did not match the value just computed.	
		0	No authentication error
		1	Authentication error
4 (R/NW)	OUTHSHDN	Outer Hash Done. The PKTE_STAT.OUTHSHDN bit indicates that the outer hash processing for this packet is finished.	
		0	Outer hash busy
		1	Outer hash done
3 (R/NW)	INHSHDN	Inner Hash Done. The PKTE_STAT.INHSHDN bit indicates that the inner hash processing for this packet is finished.	
		0	Inner hash busy
		1	Inner hash done
2 (R/NW)	ENCRYPTDN	Encrypt Done. The PKTE_STAT.ENCRYPTDN bit indicates that the encryption or decryption for this packet is finished.	
		0	Encryption or decryption busy
		1	Encryption or decryption done

Table 15-72: PKTE_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	OUTPTDN	PE Output Done. The PKTE_STAT.OUTPTDN bit indicates that the output data for the current packet is read from the packet engine output buffer.	
		0	Output not done, more output bytes available
		1	Output done, all bytes read from the output buffer
0 (R/NW)	INPTDN	Packet Engine Input Done. The PKTE_STAT.INPTDN bit indicates that the number of bytes specified in the command descriptor PKTE_LEN field is written into the packet engine input buffer.	
		0	Input not done, more input bytes expected
		1	Input done, all bytes written to input buffer

Packet Engine State Record Address

The PKTE_STATE_ADDR register holds the start address of the SA state record.

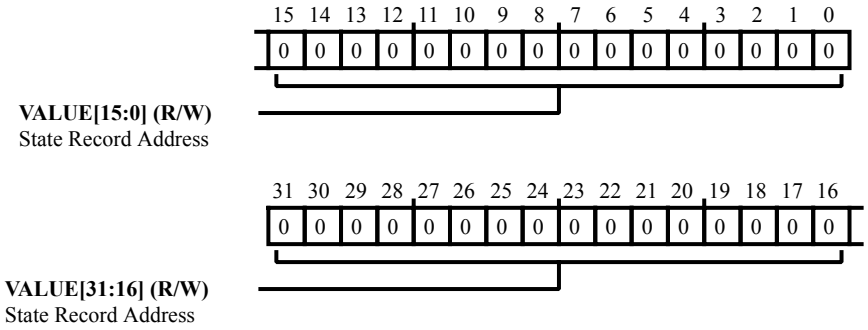

Figure 15-51: PKTE_STATE_ADDR Register Diagram

Table 15-73: PKTE_STATE_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	State Record Address. The PKTE_STATE_ADDR.VALUE bit field holds the start address of the SA state record.

State Hash Byte Count Registers

The PKTE_STATE_BYTE_CNT[n] registers are used to enter a starting hash byte count, as well as to read the interim or final byte count.

For some hash operations, these registers are ignored and the byte count is internally set to 64 (0x40 hex) to indicate that the first 64 bytes (512 bits) hash block has been processed using a pre-computed hash state. These operations are:

All IPsec, SSL, TLS, DTLS and SRTP operations that use authentication; the "pre-computed" inner and outer hash digests are loaded from SA words 10 - 19.

Basic operations with PKTE_SA_CMD0.HASHSRC bits = 00 (from SA) specified. For Basic Hash with no HMAC, a pre-computed digest is loaded from SA words 10 - 14. For Basic Hash with HMAC, the inner and outer digests are loaded from SA words 10 - 19.

Note: Protocol operations can not be suspended in mid-packet and resumed later, therefore protocol operations do not use these registers.

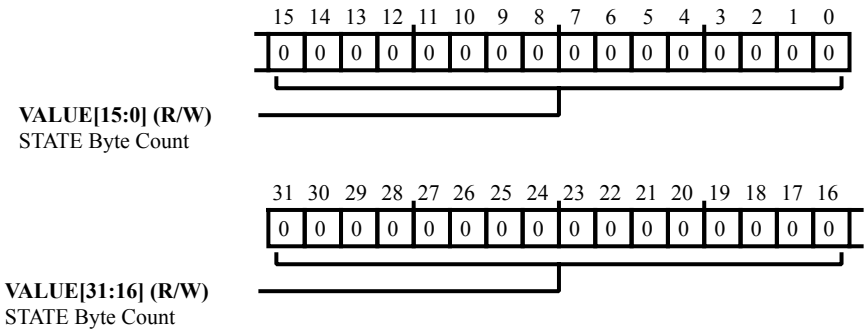

Figure 15-52: PKTE_STATE_BYTE_CNT[n] Register Diagram

Table 15-74: PKTE_STATE_BYTE_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	STATE Byte Count.

State Inner Digest Registers

The PKTE_STATE_IDIGEST[n] registers consist of eight 32-bit registers. These read/write registers are used to read the interim or final hash digest. The PKTE_STATE_IDIGEST[n] registers are only used with basic operations involving basic hash, and are typically used for operations that must be suspended and resumed in the middle of a hash. The interim hash state can be read from these registers along with the hash byte-count from the previous register. Both can be restored when resuming the hash. The appropriate save hash state (PKTE_SA_CMD0.SVHASH=1) and load hash from state (PKTE_SA_CMD0.HASHSRC=0b10) settings must be used. These registers are a mirror of the SA record inner hash digest register.

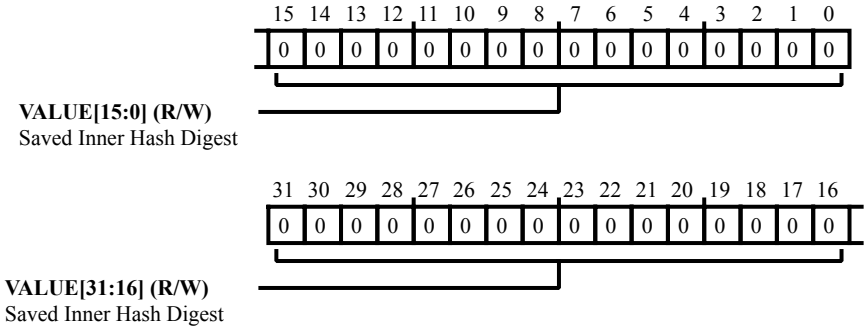

Figure 15-53: PKTE_STATE_IDIGEST[n] Register Diagram

Table 15-75: PKTE_STATE_IDIGEST[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Saved Inner Hash Digest.

State Initialization Vector Registers

The `PKTE_STATE_IV[n]` consists of four 32-bit registers. These registers are used to enter a starting IV state and to read the interim or final IV. PKTE_STATE_IV0 and PKTE_STATE_IV1 are used with DES/3DES cipher while PKTE_STATE_IV0 to PKTE_STATE_IV3 are used with AES cipher. The reset value of these registers is zero.

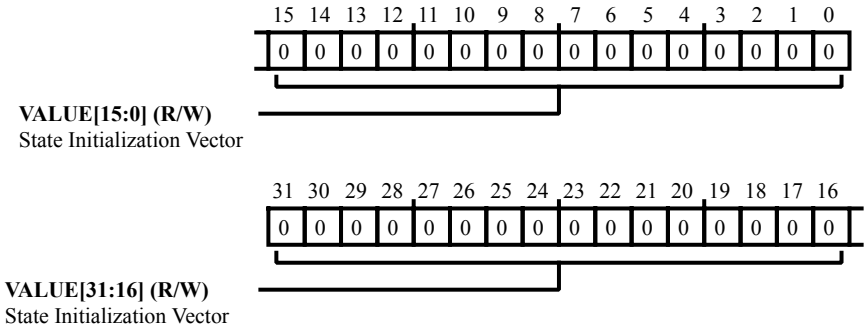

Figure 15-54: PKTE_STATE_IV[n] Register Diagram

Table 15-76: PKTE_STATE_IV[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	State Initialization Vector. The `PKTE_STATE_IV[n].VALUE` bit field is used to enter a starting IV state and to read the interim or final IV.

Packet Engine User ID

The PKTE_USERID register is a read/write register that gives identification to a command descriptor and the resultant result descriptor. The host is free to use this field for its own purpose. The host can write a unique identifier to the register in direct host mode or includes it as part of the command descriptor in autonomous ring mode. The PKTE_USERID register value passes though the packet engine without alteration to the result descriptor to be read back by the host.

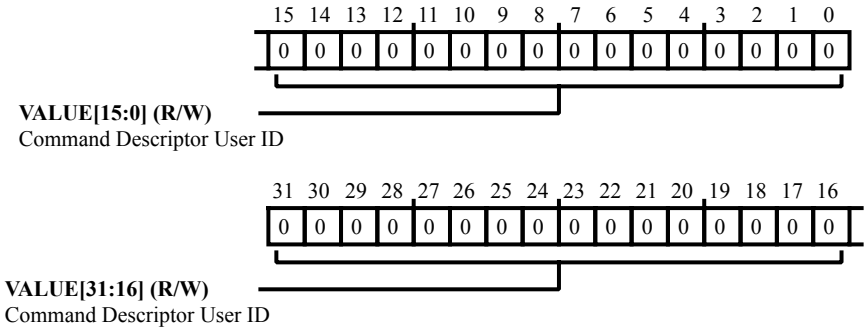

Figure 15-55: PKTE_USERID Register Diagram

Table 15-77: PKTE_USERID Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Descriptor User ID. The PKTE_USERID.VALUE bit field gives identification to a command descriptor and the resultant result descriptor.

16 Public Key Accelerator (PKA)

The PKA helps offload computationally-intensive operations commonly found in public key cryptography algorithms.

PKA Features

The PKA engine provides the following basic operations:

- Large vector addition, subtraction, and combined addition/subtraction
- Large vector shift right or left
- Large vector multiplication, division (with and without quotient)
- Large vector compare and copy

The PKA engine provides the following complex operations:

- Large vector unsigned value modular exponentiation
- Large vector unsigned value modular exponentiation using the 'Chinese Remainders Theorem' (CRT) method with pre-calculated Q inverse vector
- Modular inversion: Given A and M, calculate B such that ((A × B) MOD M) = 1
- ECC point addition/doubling on elliptic curve $y^2 = x^3 + ax + b \pmod p$ with prime number *p* and input values *a* and *b* to the operation. Adding two identical points automatically performs point doubling.
- ECC point multiplication on elliptic curve $y^2 = x^3 + ax + b \pmod p$ with prime number *p* and input values *a* and *b* to the operation. A version of the 'Montgomery ladder' algorithm is used to provide side channel attack resistance.

The PKA also contains hardware logic to automatically zero out the PKA RAM buffer to clear out any information that is considered sensitive or secure.

PKA Functional Description

The following sections provide details on the function of the PKA module.

ADSP-BF70x PKA Register List

The Public Key Accelerator module (PKA) provides security-related features. A set of registers governs PKA operations. For more information on PKA functionality, see the PKA register descriptions.

Table 16-1: ADSP-BF70x PKA Register List

Name	Description
PKA_ALEN	PKA Vector_A Length
PKA_APTR	PKA Vector_A Address
PKA_BLEN	PKA Vector_B Length
PKA_BPTR	PKA Vector_B Address
PKA_COMPARE	PKA Compare Result
PKA_CPTR	PKA Vector_C Address
PKA_DIVMSW	PKA Most-Significant-Word of Divide Remainder
PKA_DPTR	PKA Vector_D Address
PKA_FUNC	PKA Function
PKA_RAM	Start of PKA RAM space
PKA_RESULTMSW	PKA Most-Significant-Word of Result Vector
PKA_SHIFT	PKA Bit Shift Value

PKA Definitions

The following definitions are helpful when using the PKA module.

Elliptic Curve Cryptography (ECC)

A form of public key cryptography based on elliptic curves over finite fields.

RSA

An acronym for Ron Rivest, Adi Shamir, and Leonard Adleman. It is another form of a public key cryptosystem.

Chinese Remainder Theorem (CRT)

A mathematical theorem used for simplifying time-consuming arithmetic used in public key algorithm computations.

Addition Chaining Table (ACT)

A method of speeding up exponentiation by repeatedly squaring the input and storing the result and reusing the result as input. ACT2 uses a table with 2 address bits (4 entries) and ACT4 uses a table with 4 address bits (16 entries).

PKA Architectural Concepts

The following sections describe the PKA architecture.

Public Key Co-Processor (PKCP)

The Public Key Co-Processor (PKCP) handles the basic large vector processing such as addition, subtraction, multiplication, etc.

Sequencer

The sequencer is small processor that is part of the PKA which handles the more complicated vector processing for public key algorithms. Algorithms include modular exponentiation and the ECC addition and ECC multiply used in Elliptic Curve Cipher algorithms. It executes instructions stored from an internal pre-programmed ROM that handles these operations.

RAM

Input and output vectors are stored in a 4 kB RAM buffer that is part of the MMR space. The address of PKA_RAM is the beginning of the RAM space. This memory is also used as a scratchpad or workspace for the sequencer and PKCP. Programs must place the vectors appropriately following the constraints described in the ***Functional Description*** section of this chapter.

PKA Block Diagram

The ***PKA Block Diagram*** shows the top-level block diagram of the PKA engine. The PKA engine is comprised of five parts:

1. Registers for input, output, status, and control
2. Public Key Co-processor (PKCP) module which performs the basic suite of big number (vector) operations typically found in public key cryptography applications
3. Sequencer which controls modular exponentiation, elliptic curve cryptography, and modular inversion operations.
4. Program ROM associated with the PKA engine exclusively for the sequencer
5. PKA RAM holds the large input and output values as well as the workspace/scratchpad required from the sequencer and PKCP for operations.

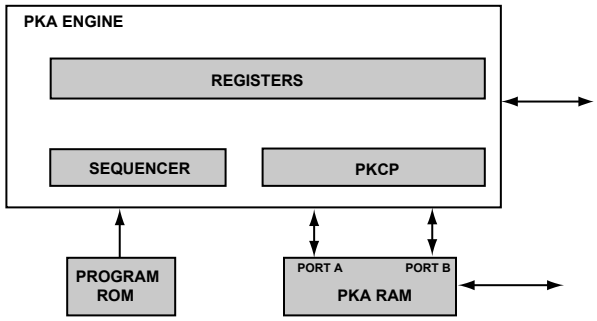

Figure 16-1: PKA Block Diagram

PKCP Vector Operations

The ***Summary of PKCP Vector Operations*** table lists the arguments and results for each PKCP vector operation.

Table 16-2: Summary of PKCP Vector Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
Multiply	A x B → C	Multiplicand	Multiplier	Product	N/A
Add	A + B → C	Addend	Addend	Sum	N/A
Subtract	A - B → C	Minuend	Subtracthend	Difference	N/A
AddSub	A + C - B → D	Addend	Subtracthend	Addend	Result
Right Shift	A >> Shift → C	Input	N/A	Result	N/A
Left Shift	A << Shift → C	Input	N/A	Result	N/A
Divide	A mod B → C, A div B → D	Dividend	Divisor	Remainder	Quotient
Modulo	A mod B → C	Dividend	Divisor	Remainder	N/A
Compare	A = B, A < B, A > B	Input 1	Input 2	N/A	N/A
Copy	A → C	Input	N/A	Result	N/A

To obtain correct result, the input vectors must meet the requirements presented in the ***Operational Restrictions on Input Vectors for PKCP Operations*** table.

Note the following:

- The PKCP does not check input restrictions
- A_Len and B_Len indicate the size of vectors A and B in (32-bit) words
- Max_Len equals 128 (32-bit) words, for example, the standard maximum vector size is 4096 bit

Table 16-3: Operational Restrictions on Input Vectors for PKCP Operations

Function	Requirement
Multiply	0 < A_Len, B_Len <= Max_Len
Add	0 < A_Len, B_Len <= Max_Len
Subtract	0 < A_Len, B_Len <= Max_Len Result must be positive (A>=B)
AddSub	0 < A_Len <= Max_Len (B and C operands have A_Len as length, B_Len ignored) Result must be positive ((A + C) >= B)
Right Shift	0 < A_Len <= Max_Len
Left Shift	0 < A_Len <= Max_Len
Divide, Modulo	1 < B_Len <= A_Len <= Max_Len Most significant 32-bit word of B operand cannot be zero
Compare	0 < A_Len <= Max_Len (B operand has A_Len as length, B_Len ignored)
Copy	0 < A_Len <= Max_Len

The host processor is responsible for allocating a block of contiguous memory in PKA RAM for the result vectors. The ***PKCP Result Vector Memory Allocation*** table indicates how much memory is allocated for the result vectors.

Table 16-4: PKCP Result Vector Memory Allocation

Function	Result Vector	Result Vector Length (in 32-bit words)
Multiply	C	A_Len + B_Len + 6 (the 6 'scratchpad' words should be discarded)
Add	C	Max(A_Len, B_Len) + 1
Subtract	C	Max(A_Len, B_Len)
AddSub	D	A_Len + 1
Right Shift	C	A_Len
Left Shift	C	A_Len + 1 (when Shift Value is non-zero) A_Len (when Shift Value is zero)
Divide	C	Remainder → B_Len + 1 (one 'scratchpad' word should be discarded)
	D	Quotient → A_Len – B_Len + 1
Modulo	C	Remainder → B_Len + 1 (one 'scratchpad' word should be discarded)
Compare	None	Compare updates the PKA_COMPARE register
Copy	C	A_Len

Input vectors for an operation are always allowed to overlap in memory (partially or completely). The ***PKCP Result Vector/Input Overlap Restrictions*** table gives restrictions for the overlap of output and input vectors of the operations.

Table 16-5: PKCP Result Vector/Input Overlap Restrictions

Function	Result Vector	Restrictions
Multiply	C	No overlap with A or B vectors allowed
Add, Subtract	C	May overlap with A and/or B vector, provided the start address of the C vector does not lie above the start address of the vectors with which it overlaps
AddSub	D	May overlap with A, B and/or C vector, provided the start address of the D vector does not lie above the start address of the vectors with which it overlaps
Right Shift, Left Shift	C	May overlap with A vector, provided the start address of the C vector does not lie above the start address of the A vector
Divide	C	No overlap with A, B, or D vectors allowed
	D	No overlap with A, B, or C vectors allowed
Modulo	C	No overlap with A or B vectors allowed
Compare	None	Compare does not write a result vector
Copy	None	Same restrictions as for right or left shift, copy of a vector to a lower address is always allowed even if source and destination overlap†

†The copy operation can be used to fill memory by breaking the overlap restrictions, but it requires setting up TWO initial (32-bit) words. To zero a block of memory, set the A vector pointer to the block start, set the C vector pointer two words higher and the A vector length to the block length minus two (words). Fill the first two words of the block with constant zero and perform a PKCP copy operation to zero the remainder of the block.

Modular Exponentiation Operations

The ***Summary of ExpMod Operations*** table summarizes the modular exponentiation operations that the PKA supports.

Table 16-6: Summary of ExpMod Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	$C^A \bmod B \rightarrow D$	Exponent, length = A_Len	Modulus, length = B_Len	Base, length = B_Len	Result and Workspace
ExpMod-CRT	See below	Exp P followed by Exp Q at next higher even word address†, both A_Len long	Mod P + buffer word followed by Mod Q at next higher even word address‡, both B_Len long	Q inverse, length = B_Len	Input, Result (both 2xB-Len long) and Workspace

† If A_Len is even, Exp Q follows Exp P immediately – if A_Len is odd, there is one empty word between Exp Q and Exp P.

‡ If B_Len is even, there are two empty words between Mod P and Mod Q – if B_Len is odd, there is one empty (buffer) word between Mod Q and Mod P. Note that the engine may zero the words following Mod P and Mod Q.

The ExpMod-CRT operation performs the following computation steps. (These steps implement Garner's recombination algorithm after the basic exponentiations.)

- X <- (Input mod Mod P)Exp P mod Mod P
- Y <- (Input mod Mod Q)Exp Q mod Mod Q
- Z <- ((((X – Y) mod Mod P) · Q inverse) mod Mod P) · Mod Q
- Result <- Y + Z

The ExpMod-ACT2, -ACT4, and -variable functions implement the same mathematical operation but with a differently sized table with pre-calculated *odd powers*. The ExpMod-ACT2 function uses a table with two entries whereas ExpMod-ACT4 uses a table with eight entries. The ACT4 version gives better performance but needs more memory. ExpMod-variable and ExpMod-CRT operations allow the selection of a variable number (from 1 up to and including 16) of odd powers through the register normally used to specify the number of bits to shift for shift operations.

The exponentiation functions are extensions of the set of PKA functions. Input and result vectors are passed the same way as basic PKCP operations. The ***Restrictions on Input Vectors for ExpMod Operations*** table shows the restrictions on the input and result vectors for the exponentiation operations.

Table 16-7: Restrictions on Input Vectors for ExpMod Operations

Function	Requirements
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	0 < A_Len <= Max_Len 1 < B_Len <= Max_Len Modulus B must be odd (for example, the least significant bit must be ONE) Modulus B > 2^{32} Base C < Modulus B Vectors B and C must be followed by an empty 32-bit *buffer* word
ExpMod-CRT	0 < A_Len <= Max_Len 1 < B_Len <= Max_Len Mod P and Mod Q must be odd (for example, the least significant bits must be ONE) Mod P > Mod Q > 2^{32}(note that Mod P must be larger than Mod Q) Mod P and Mod Q must be co-prime (their GCD must be 1) 0 < Exp P < (Mod P – 1) 0 < Exp Q < (Mod Q – 1) (Q inverse · Mod Q) 1 (modulo Mod P) Input < (Mod P · Mod Q) Mod P and Mod Q must be followed by an empty 32-bit *buffer* word

The ***ExpMod Result Vector/Scratchpad Area Memory Allocation Starting at PKA_DPTR*** table shows the required scratchpad sizes for the exponentiation operations. These sizes depend on the PKA type. The 'M_Len' used in the table is the 'real' Modulus length in 32-bit words, for example, without trailing zero words at the end. (This description also applies to Mod P in an ExpMod-CRT operation and Modulus B in the other operations.) If the last word of the modulus vector as given is non-zero, 'M_Len' equals B_Len.

Table 16-8: ExpMod Result Vector/Scratchpad Area Memory Allocation Starting at PKA_DPTR

Function	Scratchpad Area Size (in 32-bit words), Result Vector is either M_Len or 2xM_Len 32-bit words long
ExpMod-ACT2	5 x (M_Len + 2)
ExpMod-ACT4	11 x (M_Len + 2)
ExpMod-variable	(# odd powers + 3) x (M_Len + 2)
ExpMod-CRT	(# odd powers + 3) x (M_Len + 2) + (M_Len + 2 – (M_Len MOD 2))

NOTE: During execution of an ExpMod-ACT2, -ACT4 or -variable operation, the last 34 bytes of the PKA RAM are used as the general scratchpad for the sequencer program execution. The ExpMod-CRT operation requires the last 72 bytes of the PKA RAM as the scratchpad. These (fixed location) areas may not overlap with any of the input vectors and/or the D vector scratchpad area. They can be used freely when executing basic PKCP operations.

Table 16-9: ExpMod Scratchpad Area / Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	D	Scratchpad area starting at D may not overlap with any of the other vectors, except that base C may be co-located with result vector D to save space (for example, `PKA_CPTR = PKA_DPTR` is allowed).
ExpMod-CRT	D	Scratchpad area starting at D may not overlap with any of the other vectors. This area is also the location of the main input vector (with length 2 x B_Len)

The ***Maximum Number of Odd Powers*** table indicates the maximum number of odd powers that can be used for different standard PKA RAM sizes and PKA types (non-CRT operations using `PKA_CPTR = PKA_DPTR`). As a rule of thumb, for optimal performance, use one odd power for *Verify* operations and 4 (or as many as the implemented PKA RAM size allows) for *Sign* operations. Note the following points about odd powers:

- Using more than eight odd powers is not advisable as the speed advantage for each extra odd power decreases rapidly (and can even become negative for short exponent vector lengths due to the extra pre-processing required).
- The maximum number of odd powers is 16 (limited by the firmware). All '16 odd powers' entries in the table above hit this limit – they are not limited by the PKA RAM size.

Table 16-10: Maximum Number of Odd Powers

Operation	Modulus and Exponent Sizes	Maximum Number of Odd Powers
Non-CRT	1024 bits	16
	2048 bits	10
	4096 bits	2
CRT	2 × 512 bits	16
	2 ×1024 bits	16
	2 × 2048 bits	6

The ***2K-bit Modular Exponentiation PKA RAM Allocation Examples*** table shows example PKA RAM vector allocations for modular exponentiation operations with and without using CRT. The *free space* start address is the first free byte following the vector workspace. The sequencer execution scratchpad of 34 bytes (non-CRT) or 72 bytes (using CRT) must fit between this address and the end of the PKA RAM. Note that the non-CRT operations use `PKA_CPTR` = `PKA_DPTR` to save space.

Table 16-11: 2K-bit Modular Exponentiation PKA RAM Allocation Examples

Operation	(sub-)vector	Start address (Byte Offset)	Size (words)	Buffer (words)
non-CRT (`PKA_ALEN` = 0x040, `PKA_BLEN` = 0x040, 4 odd-powers)	Exponent	0x000 (`PKA_APTR` = 0x000)	64	0
	Modulus	0x100 (`PKA_BPTR` = 0x040)	64	2
	Base	0x208 (`PKA_CPTR` = 0x082)	64	2
	Result	0x208 (`PKA_DPTR` = 0x082)	64	2
	Vector Workspace	0x208 (= Result)	7 × (64+2)=462	0
	Free space	0x940 (2368 bytes used)	-	-
using CRT (`PKA_ALEN` = 0x020, `PKA_BLEN` = 0x020, 4 odd-powers)	Exp P	0x000 (`PKA_APTR` = 0x000)	32	0
	Exp Q	0x080	32	0
	Mod P	0x100 (`PKA_BPTR` = 0x040)	32	2
	Mod Q	0x188	32	2
	Q inverse	0x210 (`PKA_CPTR` = 0x084)	32	0
	Input, Result	0x290 (`PKA_DPTR` = 0x0A4)	64	0
	Vector workspace	0x290 (= Result)	7 × (32+2)+32+2-0 = 272	0
	Free space	0x6D0 (1744 bytes used)	-	-

The following example in pseudo-code describes the execution of a non-CRT modular exponentiation operation using a 512 bits modulus and a 160 bits exponent, using actual test vectors:

```
// Perform a 512/160 bits
modular exponentiation without CRT (using 4 'odd-powers')
// Exponent equals value 0x8FD84098_8A0930CC_9CDC1E8A_B246EB46_2D39F064
```

```
// write as vector A to PKA
RAM Byte offset 0x000:
Write PKA_RAM_BASE+0x000+0x00
0x2D39F064
Write PKA_RAM_BASE+0x000+0x04
0xB246EB46
Write PKA_RAM_BASE+0x000+0x08
0x9CDC1E8A
Write PKA_RAM_BASE+0x000+0x0C
0x8A0930CC
Write PKA_RAM_BASE+0x000+0x10
0x8FD84098
// Modulus equals value 0xF42F559D
1877CA5F_449492B9_42DC7C01_...
// A3C9085B_7236A085_2102B000_A093C6B4_...
// 9D0EDA0C_292DE841_29C23723_4048BDA3_...
// 373C4C9F_45CF15A7_5F049ABF_D8A01B9B
// write as vector B to PKA
RAM Byte offset 0x018 (following exp at next aligned 64-bit word):
Write PKA_RAM_BASE+0x018+0x00
0xD8A01B9B
Write PKA_RAM_BASE+0x018+0x04
0x5F049ABF
Write PKA_RAM_BASE+0x018+0x08
0x45CF15A7
Write PKA_RAM_BASE+0x018+0x0C
0x373C4C9F
Write PKA_RAM_BASE+0x018+0x10
0x4048BDA3
Write PKA_RAM_BASE+0x018+0x14
0x29C23723
Write PKA_RAM_BASE+0x018+0x18
0x292DE841
Write PKA_RAM_BASE+0x018+0x1C
0x9D0EDA0C
Write PKA_RAM_BASE+0x018+0x20
0xA093C6B4
Write PKA_RAM_BASE+0x018+0x24
0x2102B000
Write PKA_RAM_BASE+0x018+0x28
0x7236A085
Write PKA_RAM_BASE+0x018+0x2C
0xA3C9085B
Write PKA_RAM_BASE+0x018+0x30
0x42DC7C01
Write PKA_RAM_BASE+0x018+0x34
0x449492B9
Write PKA_RAM_BASE+0x018+0x38
0x1877CA5F
Write PKA_RAM_BASE+0x018+0x3C
```

```
0xF42F559D
// Base equals value 0x3D291F48_49064887_1149594B_67935110_...
// 14EB8FF0_AB291F3A_54A1B4D1_5E611E44_...
// C989251B_44904B45_0B060482_317F8352_...
// 18CE440E_9BF509F1_6EAF26F2_95F19F12
// write as vector C to PKA
RAM Byte offset 0x060 (following mod after buffer + align words):
Write PKA_RAM_BASE+0x060+0x00
0x95F19F12
Write PKA_RAM_BASE+0x060+0x04
0x6EAF26F2
Write PKA_RAM_BASE+0x060+0x08
0x9BF509F1
Write PKA_RAM_BASE+0x060+0x0C
0x18CE440E
Write PKA_RAM_BASE+0x060+0x10
0x317F8352
Write PKA_RAM_BASE+0x060+0x14
0x0B060482
Write PKA_RAM_BASE+0x060+0x18
0x44904B45
Write PKA_RAM_BASE+0x060+0x1C
0xC989251B
Write PKA_RAM_BASE+0x060+0x20
0x5E611E44
Write PKA_RAM_BASE+0x060+0x24
0x54A1B4D1
Write PKA_RAM_BASE+0x060+0x28
0xAB291F3A
Write PKA_RAM_BASE+0x060+0x2C
0x14EB8FF0
Write PKA_RAM_BASE+0x060+0x30
0x67935110
Write PKA_RAM_BASE+0x060+0x34
0x1149594B
Write PKA_RAM_BASE+0x060+0x38
0x49064887
Write PKA_RAM_BASE+0x060+0x3C
0x3D291F48
// The result value and scratchpad
(vector D) may be co-located with the base vector C for
// a normal modular exponentiation,
so these are located at PKA RAM Byte offset 0x060 too.
// Load pointer and length
registers:
Write PKA_APTR 0x000>>2 //
Exponent pointer
Write PKA_BPTR 0x018>>2 //
Modulus pointer
Write PKA_CPTR 0x060>>2 //
```

```
Base pointer
Write PKA_DPTR 0x060>>2 //
Result/scratchpad pointer
Write PKA_ALENGTH 0x00000005
// Exponent length in 32-bit words
Write PKA_BLENGTH 0x00000010
// Mod/base/result length in 32-bit words
// Start modular exponentiation
and wait until it's done:
Write PKA_SHIFT 0x00000004
// Number of 'odd powers'
Write PKA_FUNCTION 0x0000E000
// 'Run' bit set, 'Sequencer Operations' = 0b110
Wait PKA_FUNCTION[15] == '0'
// 'Run' bit clears itself - Host can also use interrupt!
// Result value equals 0xA497BF8B_DB729088_954005B0_B5CA6691_...
// A3EC491B_091A3D62_03C24214_0863A389_...
// 0C7C03CD_2333E231_35EC10ED_8F91281C_...
// 30F4253B_FE38FAFB_BB4A39DB_C14F2661
// written as vector D at
PKA RAM Byte offset 0x060:
Check PKA_RAM_BASE+0x060+0x00
== 0xC14F2661
Check PKA_RAM_BASE+0x060+0x04
== 0xBB4A39DB
Check PKA_RAM_BASE+0x060+0x08
== 0xFE38FAFB
Check PKA_RAM_BASE+0x060+0x0C
== 0x30F4253B
Check PKA_RAM_BASE+0x060+0x10
== 0x8F91281C
Check PKA_RAM_BASE+0x060+0x14
== 0x35EC10ED
Check PKA_RAM_BASE+0x060+0x18
== 0x2333E231
Check PKA_RAM_BASE+0x060+0x1C
== 0x0C7C03CD
Check PKA_RAM_BASE+0x060+0x20
== 0x0863A389
Check PKA_RAM_BASE+0x060+0x24
== 0x03C24214
Check PKA_RAM_BASE+0x060+0x28
== 0x091A3D62
Check PKA_RAM_BASE+0x060+0x2C
== 0xA3EC491B
Check PKA_RAM_BASE+0x060+0x30
== 0xB5CA6691
Check PKA_RAM_BASE+0x060+0x34
== 0x954005B0
Check PKA_RAM_BASE+0x060+0x38
```

```
== 0xDB729088
Check PKA_RAM_BASE+0x060+0x3C
== 0xA497BF8B
```

Modular Inversion

Besides modular exponentiation, the sequencer also controls modular inversion operations.

Table 16-12: Summary of ModInv Operation

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ModInv	A^{-1} mod B → D	NumToInvert, length = A_Len	Modulus, length = B_Len	Not Used	Result and Workspace

The above function appears to be an extension of the set of basic PKCP functions with the following exceptions:

- Vector D not only addresses the result but also a workspace
- The `PKA_SHIFT` register field is used to return info on the operation's result.

Table 16-13: PKA_SHIFT Result Values for ModInv Operation

Function	PKA_SHIFT Register Field Value At Conclusion
ModInv	0 → success; VectorD holds result 7 → no inverse exists (GCD(A, B) != 1, for example, A and B have common factors); result undefined 31 → error, modulus even; result undefined other values are reserved

The following tables list the restrictions on the input and result vectors for the ModInv operation:

Table 16-14: Operational Restrictions on Input Vectors for the ModInv Operation

Function	Requirements
ModInv	0 < A_Len <= Max_Len 0 < B_Len <= Max_Len Modulus B must be odd (for example, the least significant bit must be ONE) Modulus B may not have value 1 (result is undefined, no error indicated) The highest word of the modulus vector, as indicated by B_Len, may not be zero.

Table 16-15: ModInv Scratchpad Area/Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ModInv	D	Scratchpad area starting at D may not overlap with any of the other vectors

The following table shows the required scratchpad sizes for the ModInv Operation:

Table 16-16: ModInv Result Vector/Scratchpad Area Memory Allocation (Both Starting at PKA_DPTR)

Function	Scratchpad area size (in 32-bit words), Result Vector is B_Length 32-bit words long
ModInv	5 x (M + ε(M)), with M = Max(A_Length, B_Length) ε(n) = 2 + (n MOD 2), for example, 2 (for n even) or 3 (for n odd)

NOTE: During execution of a ModInv operation, the last 34 bytes of the PKA RAM are used as general scratch-pad for the sequencer program execution. This (fixed location) area may not overlap with any of the input vectors and/or the D vector scratchpad area during execution.

Modular Inversion with an Even Modulus

The ModInv operation requires the modulus to be odd. At first, this requirement appears to make the operation useless in the case of RSA key generation where the private key exponent *d* is derived from a chosen public exponent *e* as follows:

d = ModInv(e, φ); where φ = (p-1) × (q-1) and *p* and *q* both prime

Note that φ is even. However, since *e* must be odd (otherwise no inverse exists), *d* can be calculated as:

d = (1 + (φ × (e – ModInv(φ, e))) / e

With four more basic PKCP operations, ModInv can also be used to find inverse values in case the modulus is even.

Modular Inversion with a Prime Modulus

Modular inversion can be performed with a modular exponentiation using the modulus value minus two as exponent, provided that the modulus value is a prime. This is due to the following:

$(A^M) \bmod M = A \Rightarrow$

$(A^{M-1}) \bmod M = 1 \Rightarrow$

$(A^{M-2}) \bmod M = A–1 \pmod M$

Under the constraint that M is a prime value.

Especially with the large PKA engines containing an LNME, it is worthwhile to check whether this method is faster than using the ModInv operation directly. The modulus values for the ECC curves supported by this PKA engine must be prime, so this method can be used in ECDSA operations.

ECC Operations

Besides modular exponentiation and modular inversion, the sequencer also controls ECC operations.

Table 16-17: Summary of ECC Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ECC-ADD	Point addition/ doubling† on elliptic curve: $y^2 = x^3 + ax + b \pmod p$ pntA + pntC → pntD	pntA.x followed‡ by pntA.y both B_Len long (A_Len *not* used)	Curve parameter *p* followed‡ by *a* (*b* is not needed) all B_Len long	pntC.x followed by pntC.y both B_Len long	Result, for example, pntD.x followed‡ by pntD.y and workspace
ECC-MUL	Point multiplication on elliptic curve: $y^2 = x^3 + ax + b \pmod p$ *k* x pntC → pntD	Scalar *k* A_Len long	Curve parameter *p* followed‡ by *a* and *b* all B_Len long.	pntC.x followed‡ by pntC.y both B_Len long	Result, for example pntD.x followed‡ by pntD.y and workspace

† If pntA = pntC, a point doubling operation is performed automatically.

‡ All input components must be located on a 64-bit boundary and must have extra 'buffer' words (of 32 bits each) after their most significant word. ε must be 3 (B_Len odd) or 2 (B_Len even). Each result component (for example, pntD.x, pntD.y) is followed by ε buffer (zero) words.

The above functions appear to be extensions of the set of PKCP basic functions with the following exceptions:

- Input and result vectors can now be composite (for example, consist of two or three equal-sized subvectors)
- Vector D not only addresses the result but also a workspace
- The `PKA_SHIFT` register is used to return info on the result of the operation

Table 16-18: PKA_SHIFT Result Values for ECC Operations

Function	PKA_SHIFT Register Field Value at Conclusion
ECC-ADD ECC-MUL	0 → success; VectorD holds result point 7 → result is point-at-infinity; VectorD result point undefined 31 → error, (p not odd, p too short, etc); VectorD result point undefined Other values are reserved

The following tables below list the restrictions on the input and result vectors for the ECC operations.

Table 16-19: Operational Restrictions on Input Vectors for ECC Operations

Function	Requirements
ECC-ADD	1 < B_Len <= 24 (maximum vector length is 768 bits) Modulus p must be a prime > 2^{63} Effective modulus size (in bits) must be a multiple of 32† The highest word of the modulus vector, as indicated by B_Len, may not be zero.

Table 16-19: Operational Restrictions on Input Vectors for ECC Operations (Continued)

Function	Requirements
	a< p and *b*< p pntA and pntC must be on the curve (this condition is not checked) Neither pntA nor pntC can be the point-at-infinity, although ECC-ADD can return this point as a result
ECC-MUL	0 < A_Len <= 24 (maximum vector length is 768 bits) 1 < B_Len <= 24 (maximum vector length is 768 bits) Modulus p must be a prime > 2^{63} Effective modulus size (in bits) must be a multiple of 32† The highest word of the modulus vector, as indicated by B_Len, may not be zero. *a*< p and *b*< p pntC must be on the curve (this condition is not checked) pntC cannot be the point-at-infinity, although ECC-MUL can return this point as a result

† Modulus lengths of 112 and 521 bits are exceptions to this rule.

Table 16-20: ECC Scratchpad Area/Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ECC-ADD ECC-MUL	D	Scratchpad area starting at D may not overlap with any of the other vectors

The >***ECC Result Vector/Scratchpad Area Memory Allocation*** table shows the required scratchpad sizes for the ECC operations:

Table 16-21: ECC Result Vector/Scratchpad Area Memory Allocation (Both Starting at PKA_DPTR)

Function	**Scratchpad area size (in 32-bit words), Result Vector is 2x(B_Length+ ε †(B_Length)) 32-bit words long**
ECC-ADD	2 × L + 5 x M, where L = B_Length + ε(B_Length) M = B_Length + 1 + ε(B_Length + 1)
ECC-MUL	18 x L + Max(8, L), where L = (B_Length + ε(B_Length))

† ε(*n*) = 2 + (*n* MOD 2), for example, 2 (for *n* even) or 3 (for *n* odd)

NOTE: During execution of an ECC-ADD or ECC-MUL operation, the last 72 bytes of the PKA RAM are used as a general scratchpad for the sequencer program execution. These (fixed location) areas must not overlap with any of the input vectors and the D vector scratchpad area during execution.

The ***ECC Point Multiplication PKA RAM Allocation Examples*** table shows example PKA RAM vector allocations for ECC point multiplication operations. The *free space* start address is the first free byte following the vector scratchpad. The sequencer execution scratchpad of 72 bytes must fit between this address and the end of the PKA

RAM. Because of this requirement, a 521-bit ECC point multiplication cannot be performed with 2K byte PKA RAM.

Table 16-22: ECC Point Multiplication PKA RAM Allocation Examples

Modulus Length	(sub-)vector	Start Address (byte offset)	Size (words)	Buffer (words)
192 bits (=6 words, PKA_ALEN=0x006, PKA_BLEN=0x006)	Scalar *k*	0x000 (PKA_APTR = 0x000)	6	0
	p	0x018 (PKA_BPTR = 0x006)	6	2
	a	0x038	6	2
	b	0x058	6	2
	PntC.x (base)	0x078 (PKA_CPTR = 0x01E)	6	2
	PntC.y (base)	0x098	6	2
	PntD.x (result)	0x0B8 (PKA_DPTR = 0x02E)	6	2
	PntD.y (result)	0x0D8	6	0
	Vector scratchpad	0x0B8 (= PntD.x)	(18 × 8) + 8 = 152	0
	Free space	0x318 (792 bytes used)	-	-
384 bits (=12 words, ALENGTH=0x00C, BLENGTH=0x00C)	Scalar *k*	0x000 (PKA_APTR = 0x000)	12	0
	p	0x030 (PKA_BPTR = 0x00C)	12	2
	a	0x068	12	2
	b	0x0A0	12	2
	PntC.x (base)	0x0D8 (PKA_CPTR = 0x036)	12	2
	PntC.y (base)	0x110	12	2
	PntD.x (result)	0x148 (PKA_DPTR = 0x052)	12	2
	PntD.y (result)	0x180	12	0
	Vector scratchpad	0x148 (= PntD.x)	(18 × 14) + 14 = 266	0
	Free space	0x570 (1392 bytes used)	-	-
521 bits (=17 words, ALENGTH=0x011, BLENGTH=0x011)	Scalar *k*	0x000 (PKA_APTR = 0x000)	17	1 (to align *p*)
	p	0x048 (PKA_BPTR = 0x012)	17	3
	a	0x098	17	3
	b	0x0E8	17	3
	PntC.x (base)	0x138 (PKA_CPTR = 0x04E)	17	3
	PntC.y (base)	0x188	17	3
	PntD.x (result)	0x1D8 (PKA_DPTR = 0x076)	17	3
	PntD.y (result)	0x228	17	0
	Vector scratchpad	0x1D8 (= PntD.x)	(18 × 20) + 20 = 380	0

Table 16-22: ECC Point Multiplication PKA RAM Allocation Examples (Continued)

Modulus Length	(sub-)vector	Start Address (byte offset)	Size (words)	Buffer (words)
	Free space	0x7C8 (1992 bytes used)	-	-

The following example in pseudo-code describes the execution of a 192 bits ECC point multiplication, using actual test vectors (the curve parameters and generator point are from standard curve 'secp192r1').

```
    // Perform a 192 bits ECC
point multiplication using PKA RAM layout from table above.
    // Scalar 'k' equals value
0x8D98D058_9EFD018A_C9BCF3CF_2C33AEC0_24867D7F_6ADACBFF
    // write as vector A to PKA
RAM Byte offset 0x000:
    Write PKA_RAM_BASE+0x000+0x00
0x6ADACBFF
    Write PKA_RAM_BASE+0x000+0x04
0x24867D7F
    Write PKA_RAM_BASE+0x000+0x08
0x2C33AEC0
    Write PKA_RAM_BASE+0x000+0x0C
0xC9BCF3CF
    Write PKA_RAM_BASE+0x000+0x10
0x9EFD018A
    Write PKA_RAM_BASE+0x000+0x14
0x8D98D058
    // Curve parameter 'p' equals
value 0xFFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFE_FFFFFFFF_FFFFFFFF
    // write as 1st part of vector
B immediately following vector A at PKA RAM Byte offset 0x018
    // (no buffer word needed
after 'k' vector, 64-bit alignment is OK):
    Write PKA_RAM_BASE+0x018+0x00
0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x04
0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x08
0xFFFFFFFE
    Write PKA_RAM_BASE+0x018+0x0C
0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x10
0xFFFFFFFF
    Write PKA_RAM_BASE+0x018+0x14
0xFFFFFFFF
    // Curve parameter 'a' equals
value 0xFFFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFE_FFFFFFFF_FFFFFFFC
    // write as 2nd part of vector
B after one buffer word and one re-alignment word at 0x038:
```

```
    Write PKA_RAM_BASE+0x038+0x00
0xFFFFFFFC
    Write PKA_RAM_BASE+0x038+0x04
0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x08
0xFFFFFFFE
    Write PKA_RAM_BASE+0x038+0x0C
0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x10
0xFFFFFFFF
    Write PKA_RAM_BASE+0x038+0x14
0xFFFFFFFF
    // Curve parameter 'b' equals
value 0x64210519_E59C80E7_0FA7E9AB_72243049_FEB8DEEC_C146B9B1
    // write as 3rd part of vector
B after one buffer word and one re-alignment word at 0x058:
    Write PKA_RAM_BASE+0x058+0x00
0xC146B9B1
    Write PKA_RAM_BASE+0x058+0x04
0xFEB8DEEC
    Write PKA_RAM_BASE+0x058+0x08
0x72243049
    Write PKA_RAM_BASE+0x058+0x0C
0x0FA7E9AB
    Write PKA_RAM_BASE+0x058+0x10
0xE59C80E7
    Write PKA_RAM_BASE+0x058+0x14
0x64210519
    // X-coord of generator point
is value 0x188DA80E_B03090F6_7CBF20EB_43A18800_F4FF0AFD_82FF1012
    // write as 1st part of vector
C following vector B after buffer + alignment words at 0x078:
    Write PKA_RAM_BASE+0x078+0x00
0x82FF1012
    Write PKA_RAM_BASE+0x078+0x04
0xF4FF0AFD
    Write PKA_RAM_BASE+0x078+0x08
0x43A18800
    Write PKA_RAM_BASE+0x078+0x0C
0x7CBF20EB
    Write PKA_RAM_BASE+0x078+0x10
0xB03090F6
    Write PKA_RAM_BASE+0x078+0x14
0x188DA80E
    // Y-coord of generator point
is value 0x07192B95_FFC8DA78_631011ED_6B24CDD5_73F977A1_1E794811
    // write as 2nd part of vector
C after one buffer word and one re-alignment word at 0x098:
    Write PKA_RAM_BASE+0x098+0x00
0x1E794811
```

```
    Write PKA_RAM_BASE+0x098+0x04
0x73F977A1
    Write PKA_RAM_BASE+0x098+0x08
0x6B24CDD5
    Write PKA_RAM_BASE+0x098+0x0C
0x631011ED
    Write PKA_RAM_BASE+0x098+0x10
0xFFC8DA78
    Write PKA_RAM_BASE+0x098+0x14
0x07192B95
    // The result point and scratchpad
(vector D) follow vector C after one buffer word and one
    // re-alignment word, so these
are located at PKA RAM Byte offset 0x0B8.
    // Load pointer and length
registers:
    Write PKA_APTR 0x000>>2 //
Scalar 'k' pointer
    Write PKA_BPTR 0x018>>2 //
Curve parameters 'p', 'a' & 'b' pointer
    Write PKA_CPTR 0x078>>2 //
Generator point X & Y coordinates pointer
    Write PKA_DPTR 0x0B8>>2 //
Result point X & Y coordinates/scratchpad pointer
    Write PKA_ALENGTH 0x00000006
// Scalar 'k' length in 32-bit words
    Write PKA_BLENGTH 0x00000006
// Curve parameters and coordinate lengths in 32-bit words
    // Start ECC point multiplication
and wait until it's done:
    Write PKA_FUNCTION 0x0000D000
// 'Run' bit set, 'Sequencer Operations' = 0b101
    Wait PKA_FUNCTION[15] == '0'
// 'Run' bit clears itself - Host can also use interrupt!
    Check PKA_SHIFT == 0x00000000
// Shift field value 0 indicates success - check this
    // X-coord of result point
is value 0x759B9F39_0E81D268_18C82BB9_CB42BCF5_0E0AE958_85BA3097
    // written as 1st part of
vector D at PKA RAM Byte offset 0x0B8:
    Check PKA_RAM_BASE+0x0B8+0x00
== 0x85BA3097
    Check PKA_RAM_BASE+0x0B8+0x04
== 0x0E0AE958
    Check PKA_RAM_BASE+0x0B8+0x08
== 0xCB42BCF5
    Check PKA_RAM_BASE+0x0B8+0x0C
== 0x18C82BB9
    Check PKA_RAM_BASE+0x0B8+0x10
== 0x0E81D268
```

```
    Check PKA_RAM_BASE+0x0B8+0x14
== 0x759B9F39
    // Y-coord of result point
is value 0xECA14640_F92EFF07_CAF2BD55_3FBE28EF_D043F28E_1CC3D238
    // written as 2nd part of
vector D at PKA RAM Byte offset 0x0D8:
    Check PKA_RAM_BASE+0x0D8+0x00
== 0x1CC3D238
    Check PKA_RAM_BASE+0x0D8+0x04
== 0xD043F28E
    Check PKA_RAM_BASE+0x0D8+0x08
== 0x3FBE28EF
    Check PKA_RAM_BASE+0x0D8+0x0C
== 0xCAF2BD55
    Check PKA_RAM_BASE+0x0D8+0x10
== 0xF92EFF07
    Check PKA_RAM_BASE+0x0D8+0x14
== 0xECA14640
```

ADSP-BF70x PKA Register Descriptions

Public Key Accelerator (PKA) contains the following registers.

Table 16-23: ADSP-BF70x PKA Register List

Name	Description
PKA_ALEN	PKA Vector_A Length
PKA_APTR	PKA Vector_A Address
PKA_BLEN	PKA Vector_B Length
PKA_BPTR	PKA Vector_B Address
PKA_COMPARE	PKA Compare Result
PKA_CPTR	PKA Vector_C Address
PKA_DIVMSW	PKA Most-Significant-Word of Divide Remainder
PKA_DPTR	PKA Vector_D Address
PKA_FUNC	PKA Function
PKA_RAM	Start of PKA RAM space
PKA_RESULTMSW	PKA Most-Significant-Word of Result Vector
PKA_SHIFT	PKA Bit Shift Value

PKA Vector_A Length

During execution of basic PKCP operations, the PKA_ALEN register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_ALEN register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

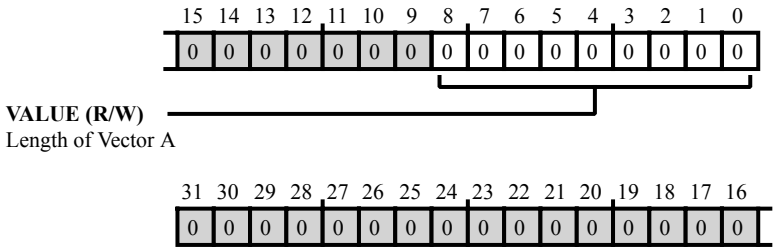

Figure 16-2: PKA_ALEN Register Diagram

Table 16-24: PKA_ALEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Length of Vector A. Length (in 32-bit words) of Vector A.

PKA Vector_A Address

During execution of basic PKCP operations, the PKA_APTR register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_APTR register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

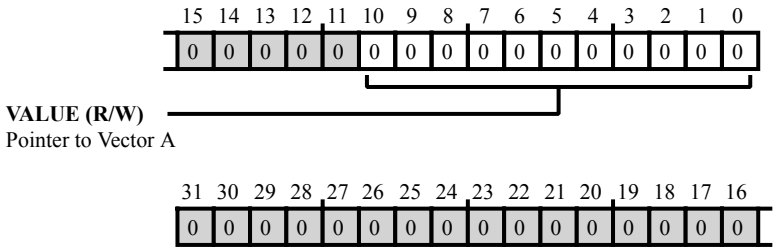

Figure 16-3: PKA_APTR Register Diagram

Table 16-25: PKA_APTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector A. The PKA_APTR.VALUE bit field is the location of Vector A within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Vector_B Length

During execution of basic PKCP operations, the PKA_BLEN register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_BLEN register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

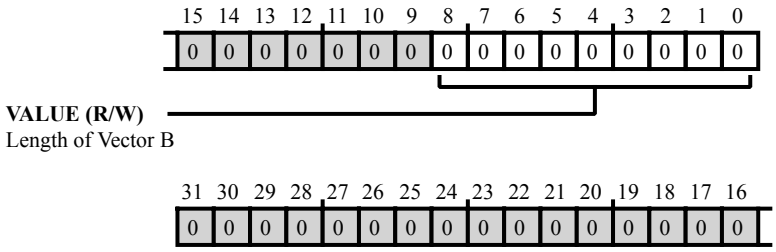

Figure 16-4: PKA_BLEN Register Diagram

Table 16-26: PKA_BLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
8:0 (R/W)	VALUE	Length of Vector B. Length (in 32-bit words) of Vector B.

PKA Vector_B Address

During execution of basic PKCP operations, the PKA_BPTR register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_BPTR register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

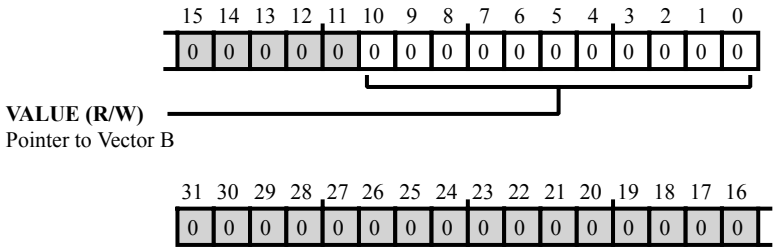

Figure 16-5: PKA_BPTR Register Diagram

Table 16-27: PKA_BPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector B. The PKA_BPTR.VALUE bit field is the location of Vector B within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Compare Result

The PKA_COMPARE register provides the result of a basic PKCP Compare operation. It is updated when the PKA_FUNC.RUN bit is reset at the end of that operation. The status after a complex sequencer operation is unknown.

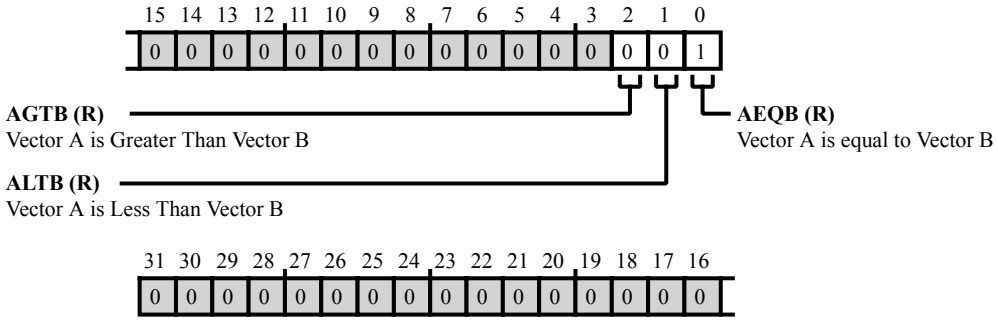

Figure 16-6: PKA_COMPARE Register Diagram

Table 16-28: PKA_COMPARE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
2 (R/NW)	AGTB	Vector A is Greater Than Vector B. The PKA_COMPARE.AGTB bit shows the result of the basic compare operation is PKCP Vector_A is greater than Vector_B.
1 (R/NW)	ALTB	Vector A is Less Than Vector B. The PKA_COMPARE.ALTB bit shows the result of the basic compare operation is Vector_A is less than Vector_B.
0 (R/NW)	AEQB	Vector A is equal to Vector B. The PKA_COMPARE.AEQB bit shows the result of the basic compare operation is Vector_A is equal to Vector_B.

PKA Vector_C Address

During execution of basic PKCP operations, the PKA_CPTR register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_CPTR register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

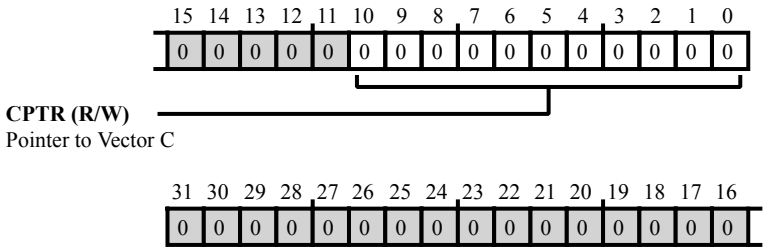

Figure 16-7: PKA_CPTR Register Diagram

Table 16-29: PKA_CPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	CPTR	Pointer to Vector C. The PKA_CPTR.CPTR bit field is the location of Vector C within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Most-Significant-Word of Divide Remainder

The PKA_DIVMSW register indicates the (32-bit word) address in the PKA RAM where the most significant non-zero 32-bit word of the Remainder result for the basic Divide and Modulo operations is stored. Bits [4:0] are loaded with the bit number of the most significant non-zero bit in the most significant non-zero word when MS one control bit is set. For Divide, Modulo and MS one reporting, this register is updated when the PKA_FUNC.RUN bit is reset at the end of the operation.

For the complex sequencer controlled operations, updating bits [4:0] of this register with the actual result's most significant bit location is done near the end of the operation. Note that the result is only meaningful if no errors were detected and that for ECC operations, the PKA_DIVMSW register provides information for the x-coordinate of the result point only.

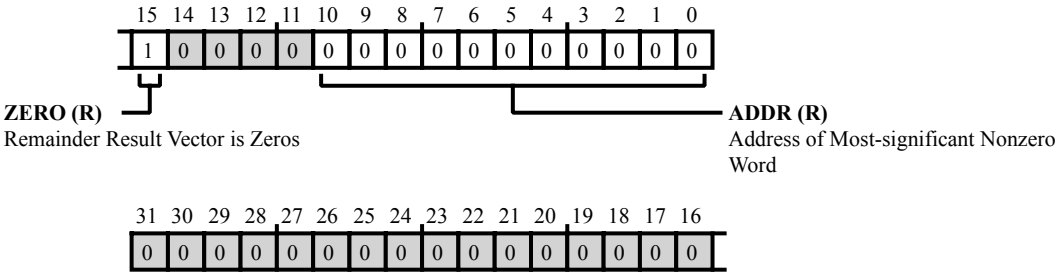

Figure 16-8: PKA_DIVMSW Register Diagram

Table 16-30: PKA_DIVMSW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/NW)	ZERO	Remainder Result Vector is Zeros. The PKA_DIVMSW.ZERO bit shows the remainder result vector is all zeros, ignore the address returned in bits [10:0].
10:0 (R/NW)	ADDR	Address of Most-significant Nonzero Word. The PKA_DIVMSW.ADDR bit shows the address of the most significant non-zero 32-bit word of the remainder result vector in PKA RAM.

PKA Vector_D Address

During execution of basic PKCP operations, the PKA_DPTR register is double buffered and can be written with a new value for the next operation. When not written, the value remains intact. During the execution of sequencer controlled complex operations, the PKA_DPTR register may not be written and its value is undefined at the conclusion of the operation. The driver software cannot rely on the written value to remain intact.

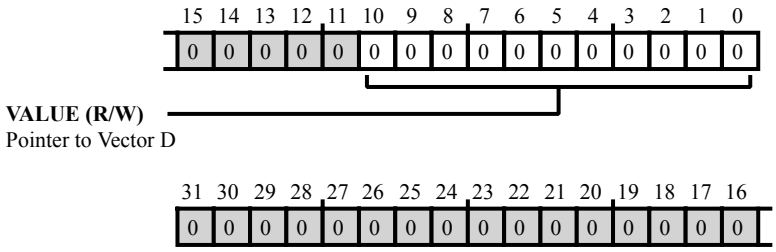

Figure 16-9: PKA_DPTR Register Diagram

Table 16-31: PKA_DPTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/W)	VALUE	Pointer to Vector D. The PKA_DPTR.VALUE bit field is the location of Vector D within the PKA RAM. Vectors are identified through the location of their least-significant 32-bit word. Note that bit [0] must be zero to ensure that the vector starts at an 8-byte boundary.

PKA Function

The PKA_FUNC register contains the control bits to start basic PKCP as well as complex sequencer operations. The PKA_FUNC.RUN bit can be used to poll for the completion of the operation. Modifying bits [11:0] is made impossible during the execution of a basic PKCP operation.

During the execution of Sequencer controlled complex operations, this register is modified - the PKA_FUNC.RUN and PKA_FUNC.STALLRSLT bits are set to zero at the conclusion, but other bits are undefined.

Continuously reading this register to poll the PKA_FUNC.RUN bit is NOT allowed when executing complex sequencer operations (the sequencer cannot access the PKCP when this is done).

Leave at least one SCLK cycle between poll operations.

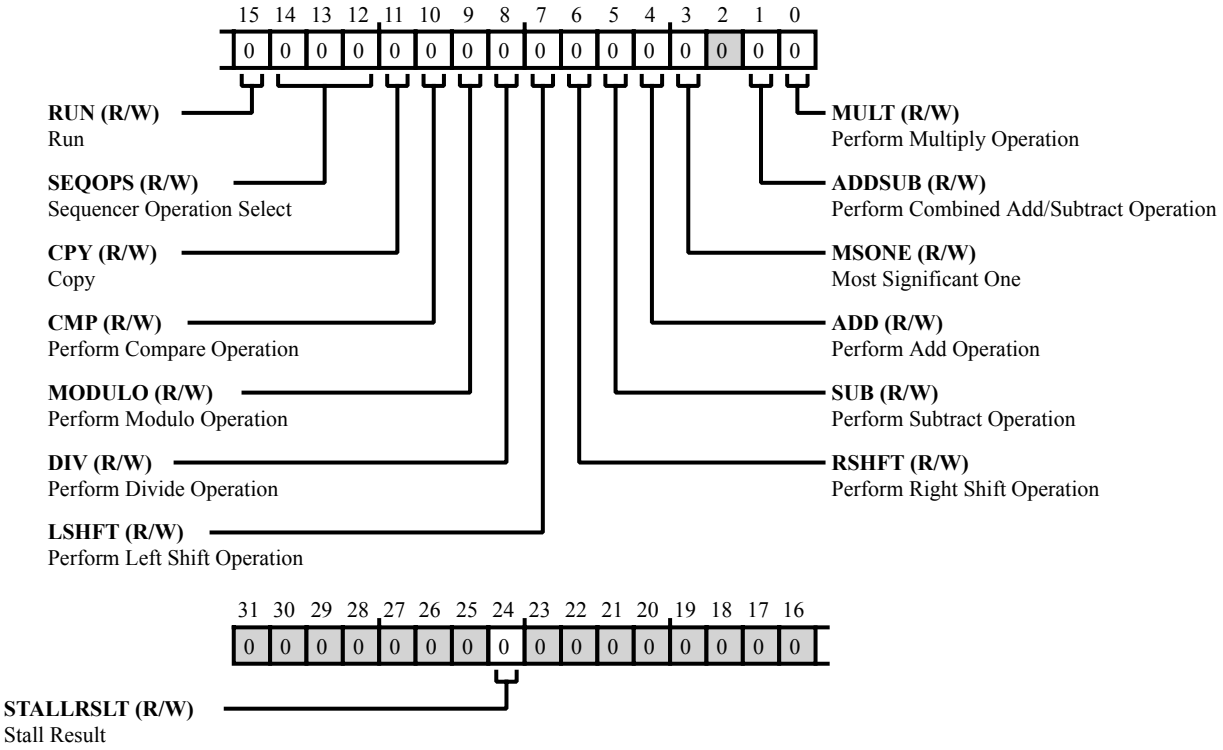

Figure 16-10: PKA_FUNC Register Diagram

Table 16-32: PKA_FUNC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/W)	STALLRSLT	Stall Result. When the PKA_FUNC.STALLRSLT bit is set, updating the PKA_COMPARE, PKA_RESULTMSW and PKA_DIVMSW registers, as well as resetting the Run bit, is stalled beyond the point that a running operation is actually finished. Use this to allow

Table 16-32: PKA_FUNC Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		software enough time to read results from a previous operation when the newly started operation is known to take only a short amount of time. If a result is waiting, the result registers is updated and the Run bit is reset in the clock cycle following writing the Stall Result bit back to 0. The Stall result function may only be used for basic PKCP operations.
15 (R/W)	RUN	Run. Set the PKA_FUNC.RUN bit to instruct the PKA module to begin processing the basic PKCP or complex Sequencer operation. This bit is reset low automatically when the operation is complete. The complement of this bit is output as the pkaint1 interrupt. After a reset, the Run bit is always set to 1b but the first Sequencer firmware instruction sets this bit to 0 immediately after the hardware reset is released. A few clock cycles are needed before the first instruction is executed and the Run bit state has been propagated.
14:12 (R/W)	SEQOPS	Sequencer Operation Select. The PKA_FUNC.SEQOPS bit field select the complex Sequencer operation to perform.
		0 None
		1 ExpMod-CRT
		2 ExpMod-ACT4
		3 ECC-ADD
		4 ExpMod-ACT2
		5 ECC-MUL
		6 ExpMod-variable
		7 ModInv
11 (R/W)	CPY	Copy. Setting the PKA_FUNC.CPY bit performs the copy operation. For more information, see the "PKCP Vector Operations" section.
10 (R/W)	CMP	Perform Compare Operation. Setting the PKA_FUNC.CMP bit performs the compare operation. For more information, see the "PKCP Vector Operations" section.
9 (R/W)	MODULO	Perform Modulo Operation. Setting the PKA_FUNC.MODULO bit performs the modulo operation. For more information, see the "PKCP Vector Operations" section.
8 (R/W)	DIV	Perform Divide Operation. Setting the PKA_FUNC.DIV bit performs the divide operation. For more information, see the "PKCP Vector Operations" section.
7	LSHFT	Perform Left Shift Operation.

Table 16-32: PKA_FUNC Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		Setting the PKA_FUNC.LSHFT bit performs the Left shift operation. For more information, see the "PKCP Vector Operations" section.
6 (R/W)	RSHFT	Perform Right Shift Operation. Setting the PKA_FUNC.RSHFT bit performs the right shift operation. For more information, see the "PKCP Vector Operations" section.
5 (R/W)	SUB	Perform Subtract Operation. Setting the PKA_FUNC.SUB bit performs the subtract operation. For more information, see the "PKCP Vector Operations" section.
4 (R/W)	ADD	Perform Add Operation. Setting the PKA_FUNC.ADD bit performs the add operation. For more information, see the "PKCP Vector Operations" section.
3 (R/W)	MSONE	Most Significant One. Setting the PKA_FUNC.MSONE bit loads the location of the Most Significant one bit within the result word indicated in the PKA_RESULTMSW register into bits [4:0] of the PKA_DIVMSW register can only be used with basic PKCP operations, except for Divide, Modulo and Compare.
1 (R/W)	ADDSUB	Perform Combined Add/Subtract Operation. Setting the PKA_FUNC.ADDSUB bit performs the combined Add/Subtract operation. For more information, see the "PKCP Vector Operations" section.
0 (R/W)	MULT	Perform Multiply Operation. Setting the PKA_FUNC.MULT bit performs the multiply operation. For more information, see the "PKCP Vector Operations" section.

Start of PKA RAM space

The PKA_RAM register provides the the starting location of the RAM space to hold the input, output and other vectors.

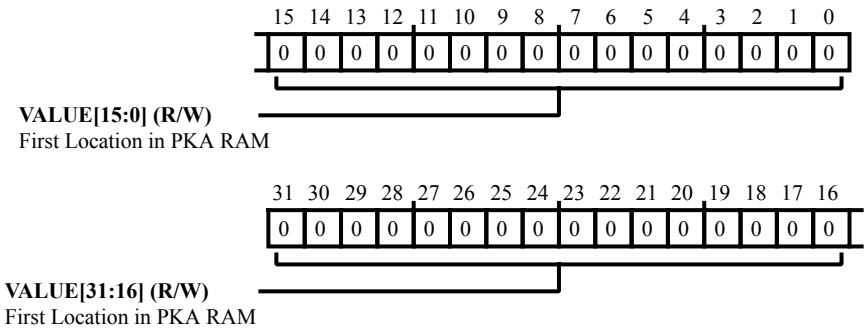

Figure 16-11: PKA_RAM Register Diagram

Table 16-33: PKA_RAM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	First Location in PKA RAM. The PKA_RAM.VALUE bit field provides the the starting location of the RAM space to hold the input, output and other vectors.

PKA Most-Significant-Word of Result Vector

The PKA_RESULTMSW register indicates the (word) address in the PKA RAM where the most significant non-zero 32-bit word of the result is stored and should be ignored for modulo operations. For basic PKCP operations, the PKA_RESULTMSW register is updated when the PKA_FUNC.RUN bit is reset at the end of the operation.

For the complex sequencer controlled operations, updating the final value matching the actual result is done near the end of the operation. Note that the result is only meaningful if no errors are detected and that for ECC operations, the PKA_DIVMSW register provides information for the x-coordinate of the result point only.

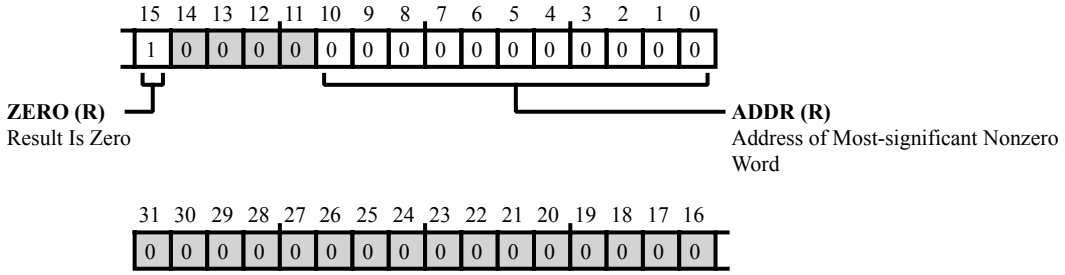

Figure 16-12: PKA_RESULTMSW Register Diagram

Table 16-34: PKA_RESULTMSW Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15 (R/NW)	ZERO	Result Is Zero. The PKA_RESULTMSW.ZERO bit indicates the result vector is all zeros, ignore the address returned in bits [10:0].
10:0 (R/NW)	ADDR	Address of Most-significant Nonzero Word. The PKA_RESULTMSW.ADDR bit is the address of the most significant non-zero 32-bit word of the result vector in PKA RAM.

PKA Bit Shift Value

For basic PKCP operations, modifying the contents of the PKA_SHIFT register is made impossible while the operation is being performed. For the ExpMod-variable and ExpMod-CRT operations, the PKA_SHIFT register is used to indicate the number of odd powers to use (directly as a value in the range 1-16). For the ModInv and ECC operations, this register is used to hold a completion code.

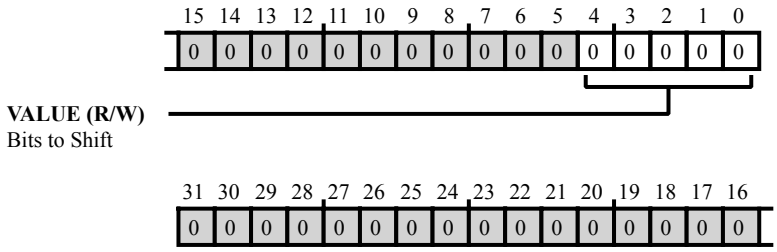

Figure 16-13: PKA_SHIFT Register Diagram

Table 16-35: PKA_SHIFT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Bits to Shift. The PKA_SHIFT.VALUE bit field is the number of bits to shift the input vector (in the range 0-31) during a Rshift or Lshift operation.

17 Public Key Interrupt Controller (PKIC)

The Public Key Accelerator (PKA) and the True Random Number Generator (TRNG) share a common interrupt controller, the Public Key Interrupt Controller (PKIC). The host processor configures the PKIC to generate interrupts when certain operations are complete or interrupts are caused by errors.

PKIC Functional Description

The main purpose and function of the PKIC is to capture the interrupts from different sources, either from the PKA or the TRNG and combine them to one interrupt output. The interrupt controller is managed using the following register groups:

- Control for polarity, edge, and level detection and enabling of individual interrupts
- Acknowledgment (to clear edge detected interrupts)
- Status:
  - A raw source status register after edge detection, if edge selected.
  - A status register after masking with the interrupt enable control bits.

ADSP-BF70x PKIC Register List

The Public Key Processor Interrupt Controller (PKIC) provides security-related features. A set of registers governs PKIC operations. For more information on PKIC functionality, see the PKIC register descriptions.

Table 17-1: ADSP-BF70x PKIC Register List

Name	Description
PKIC_ACK	Acknowledge Register
PKIC_EN_CLR	Enable Clear Register
PKIC_EN_CTL	Enable Control Register
PKIC_EN_SET	Enable Set Register
PKIC_EN_STAT	Enabled Status Register
PKIC_POL_CTL	Polarity Control Register

Table 17-1: ADSP-BF70x PKIC Register List (Continued)

Name	Description
PKIC_RAW_STAT	Raw Status Register
PKIC_TYPE_CTL	Type Control Register

ADSP-BF70x PKIC Interrupt List

Table 17-2: ADSP-BF70x PKIC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
68	PKIC0_IRQ	PKIC0 Public Key Interrupt	Level	

PKIC Programming Model

The following sections provide information on how to program the PKIC.

Enabling/Disabling and Status

The PKIC_EN_STAT register provides the mask to which interrupt source is enabled. There are two status registers, PKIC_RAW_STAT and PKIC_EN_STAT. They allow the host processor to read the status of the interrupt source before and after the mask is applied.

Level or Edge

All of the interrupt sources are level or edge events. The PKIC_TYPE_CTL register configures each interrupt to either level or edge. The PKIC_POL_CTL register controls the polarity of the signal.

These interrupts are latched at both status registers in case edge detection is selected. The edge detectors are reset by clearing the interrupts using the PKIC_ACK registers.

PKIC Programming Concepts

The following concepts help with proper programming for the PKIC module.

Interrupt Handling

When an interrupt is triggered, the handler must first examine this module, the PKIC to determine what triggered the interrupt. By reading the PKIC_EN_STAT, the bits that are set are the pending interrupts of the ones that were not masked. After determining the source of the interrupt, the appropriate action must be taken to service the interrupt from the corresponding module, the PKA, or the TRNG. After handling the interrupt in a particular module, the corresponding interrupt must be acknowledged and cleared in the PKIC to allow further interrupts.

While handling an interrupt, any events that would cause another interrupt would happen without triggering another interrupt.

Overlapping Registers

There are two sets of overlapping registers in the PKIC. The PKIC_EN_STAT and PKIC_ACK registers share one address. If read, the register tells which enabled interrupts are pending. If written to (W1C), the interrupt is acknowledged and cleared. The PKIC_RAW_STAT and PKIC_EN_SET registers are another pair that share the address. When read, the register tells which interrupts are pending and if written to will enabled certain interrupts. This register cannot be used to disable any interrupts.

ADSP-BF70x PKIC Register Descriptions

Public Key Processor Interrupt Controller (PKIC) contains the following registers.

Table 17-3: ADSP-BF70x PKIC Register List

Name	Description
PKIC_ACK	Acknowledge Register
PKIC_EN_CLR	Enable Clear Register
PKIC_EN_CTL	Enable Control Register
PKIC_EN_SET	Enable Set Register
PKIC_EN_STAT	Enabled Status Register
PKIC_POL_CTL	Polarity Control Register
PKIC_RAW_STAT	Raw Status Register
PKIC_TYPE_CTL	Type Control Register

Acknowledge Register

The PKIC_ACK register is used to acknowledge the interrupt and clear the corresponding interrupt bit in the status register.

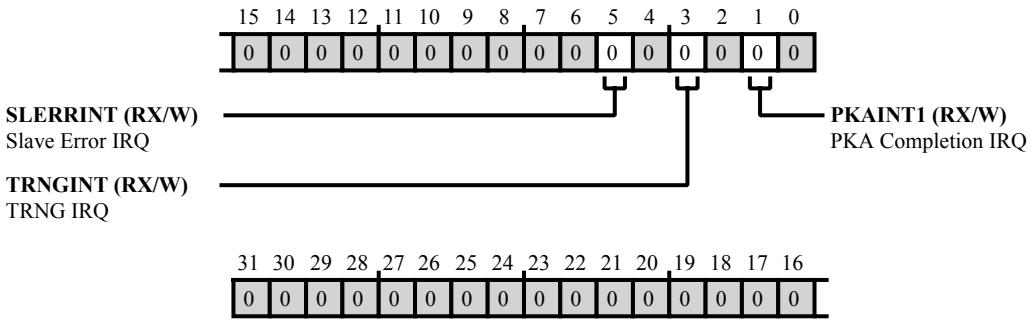

Figure 17-1: PKIC_ACK Register Diagram

Table 17-4: PKIC_ACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RX/W)	SLERRINT	Slave Error IRQ. The PKIC_ACK.SLERRINT bit is the acknowledge bit for the Slave Error interrupt. When set =1 the PKIC_ACK.SLERRINT bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).	
		0	Do not acknowledge interrupt and clear status bit
		1	Acknowledge interrupt and clear status bit
3 (RX/W)	TRNGINT	TRNG IRQ. The PKIC_ACK.TRNGINT bit is the acknowledge bit for the TRNG interrupt. When set =1 the PKIC_ACK.TRNGINT bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).	
		0	Do not acknowledge interrupt and clear status bit
		1	Acknowledge interrupt and clear status bit
1 (RX/W)	PKAINT1	PKA Completion IRQ. The PKIC_ACK.PKAINT1 bit is the acknowledge bit for the PKA completion interrupt. When set =1 the PKIC_ACK.PKAINT1 bit acknowledges the interrupt signal and clears the status bit (and is cleared automatically).	

Enable Clear Register

The PKIC_EN_CLR register allows the user to disable certain interrupts without enabling others. The disabled interrupts are also reflected in PKIC_EN_CTL register.

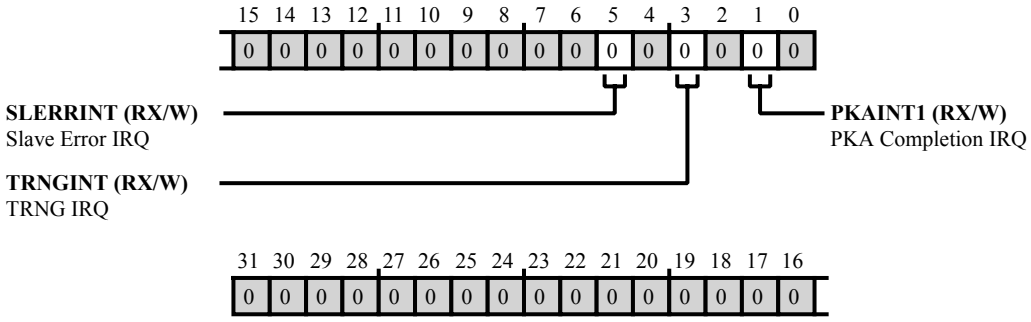

Figure 17-2: PKIC_EN_CLR Register Diagram

Table 17-5: PKIC_EN_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RX/W)	SLERRINT	Slave Error IRQ. The PKIC_EN_CLR.SLERRINT bit is the individual disable for the Slave Error interrupt. When set =1 this bit clears/resets the corresponding bit in the PKIC_EN_CTL register to 0 (disables the interrupt). This bit is cleared automatically.	
		0	No action
		1	Clear/reset corresponding CTL bit
3 (RX/W)	TRNGINT	TRNG IRQ. The PKIC_EN_CLR.TRNGINT bit is the individual disable for the TRNG interrupt. When set =1 this bit clears/resets the corresponding bit in the PKIC_EN_CTL register to 0 (disables the interrupt). This bit is cleared automatically. 0= no effect.	
		0	No action
		1	Clear/reset corresponding CTL bit
1 (RX/W)	PKAINT1	PKA Completion IRQ. The PKIC_EN_CLR.PKAINT1 bit is the individual disable for the PKA Completion interrupt. When set =1 this bit clears/resets the corresponding bit in the PKIC_EN_CTL register to 0 (disables the interrupt). This bit is cleared automatically. 0= no effect.	
		0	No action
		1	Clear/reset corresponding CTL bit

Enable Control Register

The PKIC_EN_CTL register provides individual enable bits for the interrupt sources.

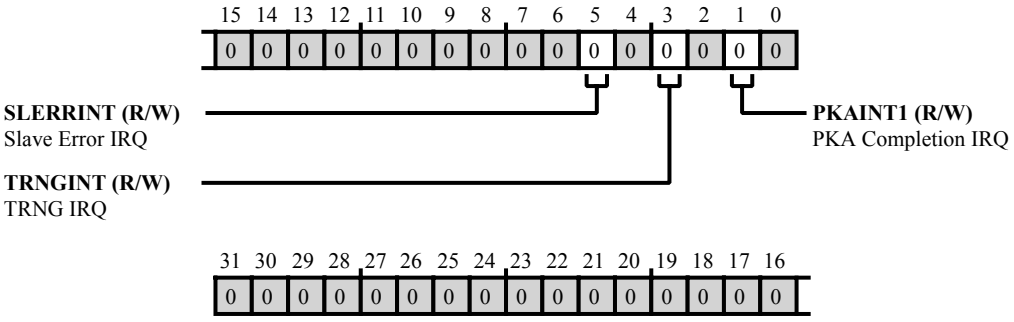

Figure 17-3: PKIC_EN_CTL Register Diagram

Table 17-6: PKIC_EN_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SLERRINT	Slave Error IRQ. The PKIC_EN_CTL.SLERRINT bit enables control for the Slave Error interrupt.	
		0	Disable interrupt
		1	Enable interrupt
3 (R/W)	TRNGINT	TRNG IRQ. The PKIC_EN_CTL.TRNGINT bit enables control for the TRNG interrupt.	
		0	Disable interrupt
		1	Enable interrupt
1 (R/W)	PKAINT1	PKA Completion IRQ. The PKIC_EN_CTL.PKAINT1 bit enables control for the PKA completion interrupt.	
		0	Disable interrupt
		1	Enable interrupt

Enable Set Register

The PKIC_EN_SET register allows the user to only set certain interrupt sources without disabling any others. The enabled interrupts are reflected in PKIC_EN_CTL register.

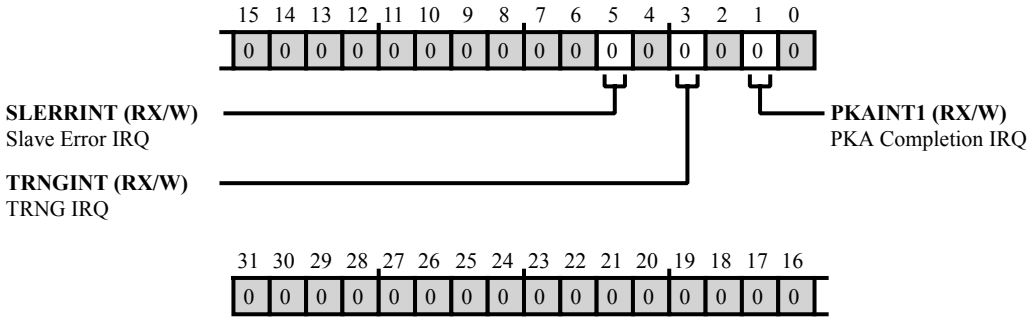

Figure 17-4: PKIC_EN_SET Register Diagram

Table 17-7: PKIC_EN_SET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RX/W)	SLERRINT	Slave Error IRQ. The PKIC_EN_SET.SLERRINT bit is the individual enable for the Slave Error interrupt. If =1, sets the corresponding bit in the PKIC_EN_CTL register to 1 (enables interrupt). This bit is cleared automatically. 0=no effect	
		0	No effect
		1	Enable interrupt
3 (RX/W)	TRNGINT	TRNG IRQ. The PKIC_EN_SET.TRNGINT bit is the individual enable for the TRNG interrupt. If =1, sets the corresponding bit in the PKIC_EN_CTL register to 1 (enables interrupt). This bit is cleared automatically.	
		0	No effect
		1	Enable interrupt
1 (RX/W)	PKAINT1	PKA Completion IRQ. The PKIC_EN_SET.PKAINT1 bit is the individual enable for the PKA Completion interrupt. If =1, sets the corresponding bit in the PKIC_EN_CTL register to 1 (enables interrupt). This bit is cleared automatically.	
		0	No effect
		1	Enable interrupt

Enabled Status Register

The PKIC_EN_STAT register is used to tell the status of the interrupts after the gating with the PKIC_EN_CTL register.

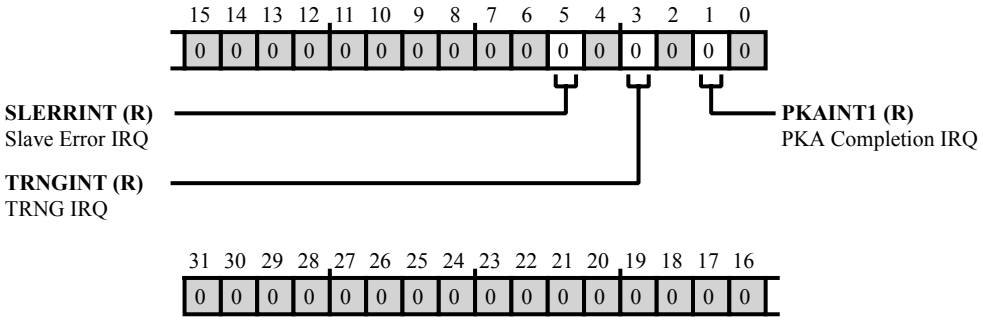

Figure 17-5: PKIC_EN_STAT Register Diagram

Table 17-8: PKIC_EN_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	SLERRINT	Slave Error IRQ. The PKIC_EN_STAT.SLERRINT bit provides the status of the Slave Error interrupt (after masking from the PKIC_EN_CTL register).	
		0	Interrupt is inactive
		1	Interrupt is pending
3 (R/NW)	TRNGINT	TRNG IRQ. The PKIC_EN_STAT.TRNGINT bit provides the status of the TRNG interrupt (after masking from the PKIC_EN_CTL register).	
		0	Interrupt is inactive
		1	Interrupt is pending
1 (R/NW)	PKAINT1	PKA Completion IRQ. The PKIC_EN_STAT.PKAINT1 bit provides the status of the PKA Completion interrupt (after masking from the PKIC_EN_CTL register).	
		0	Interrupt is inactive
		1	Interrupt is pending

Polarity Control Register

The PKIC_POL_CTL register is used to configure the signal polarity for each individual interrupt. During the initialization phase of the PKA the host processor must set each interrupt in this register to (high level/rising edge or low level/falling edge).

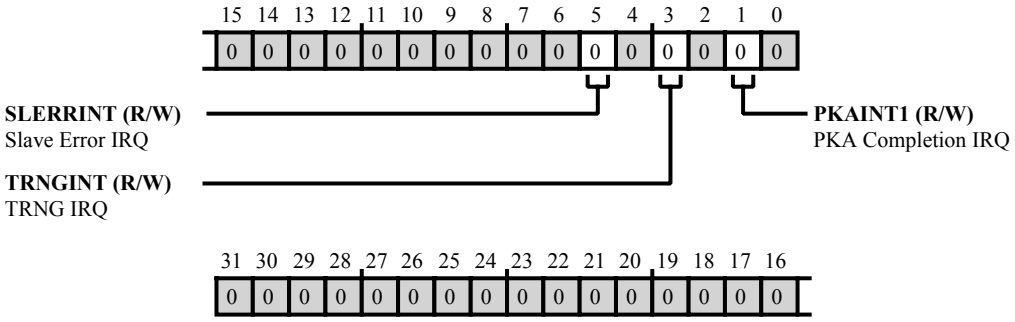

Figure 17-6: PKIC_POL_CTL Register Diagram

Table 17-9: PKIC_POL_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SLERRINT	Slave Error IRQ. The PKIC_POL_CTL.SLERRINT bit provides polarity control for the Slave Error interrupt.	
		0	Low level/falling edge
		1	High level/rising edge
3 (R/W)	TRNGINT	TRNG IRQ. The PKIC_POL_CTL.TRNGINT bit provides polarity control for the TRNG interrupt.	
		0	Low level/falling edge
		1	High level/rising edge
1 (R/W)	PKAINT1	PKA Completion IRQ. The PKIC_POL_CTL.PKAINT1 bit provides polarity control for PKA completion interrupt.	
		0	Low level/falling edge
		1	High level/rising edge

Raw Status Register

The PKIC_RAW_STAT register reflects the status of the individual interrupts before masking with the PKIC_EN_CTL register.

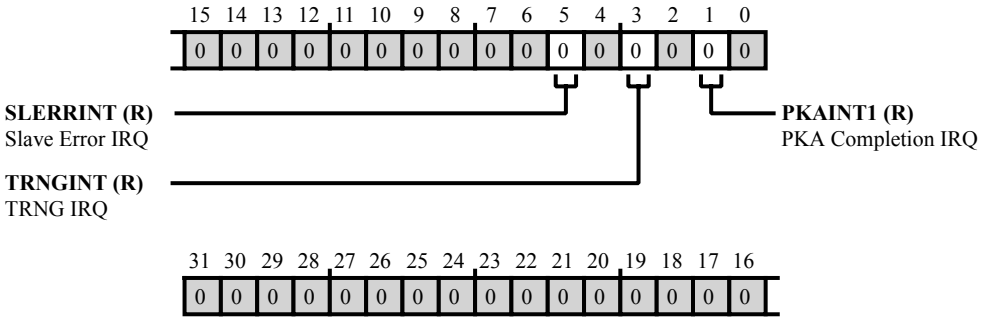

Figure 17-7: PKIC_RAW_STAT Register Diagram

Table 17-10: PKIC_RAW_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/NW)	SLERRINT	Slave Error IRQ. The PKIC_RAW_STAT.SLERRINT bit provides the raw status of the Slave Error interrupt.	
		0	Inactive interrupt
		1	Pending interrupt
3 (R/NW)	TRNGINT	TRNG IRQ. The PKIC_RAW_STAT.TRNGINT bit provides the raw status of the TRNG interrupt where 1=pending and 0=inactive.	
		0	Inactive interrupt
		1	Pending interrupt
1 (R/NW)	PKAINT1	PKA Completion IRQ. The PKIC_RAW_STAT.PKAINT1 bit provides raw status of the PKA completion interrupt where 1=pending and 0=inactive.	
		0	Inactive interrupt
		1	Pending interrupt

Type Control Register

The PKIC_TYPE_CTL register is used to configure the signal type for each individual interrupt. During the initialization phase of the PKA the host processor must set each interrupt in this register to level or edge.

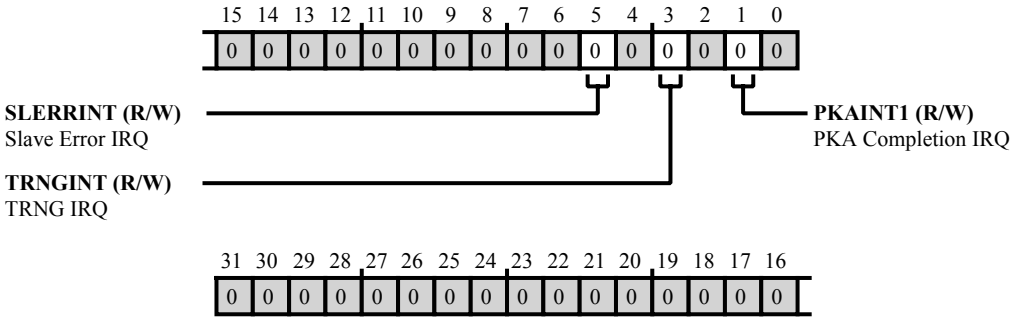

Figure 17-8: PKIC_TYPE_CTL Register Diagram

Table 17-11: PKIC_TYPE_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SLERRINT	Slave Error IRQ. The PKIC_TYPE_CTL.SLERRINT bit provides signal type control for the Slave Error interrupt.	
		0	Level
		1	Edge
3 (R/W)	TRNGINT	TRNG IRQ. The PKIC_TYPE_CTL.TRNGINT bit provides signal type control for the TRNG interrupt.	
		0	Level
		1	Edge
1 (R/W)	PKAINT1	PKA Completion IRQ. The PKIC_TYPE_CTL.PKAINT1 bit provides signal type control for the PKA completion interrupt.	
		0	Level
		1	Edge

18 True Random Number Generator (TRNG)

The TRNG engine provides a true, non-deterministic, noise source for generating keys, Initialization Vectors (IVs), and other random number requirements. Other non-cryptographic purposes include statistical sampling, retry timers for communications protocols and noise generation.

TRNG Features

The TRNG features include:

- Hardware-based non-deterministic random number generator
- ANSI X9.31 postprocessing (depending on processor)
- Redundant 'fail-safe' design with self-test circuits
- Reliable shot noise oscillator implementation with auto-tuning
- Debug output to allow monitoring of internal operation
- Alarm count overflow and auto-tuning error interrupts
- Buffer to allow generation of large blocks of random data in the background

TRNG Functional Description

The following sections provide details on the function of the TRNG module.

ADSP-BF70x TRNG Register List

The True Random Number Generator (TRNG) provides random number generation, intended mainly for security-related applications. A set of registers governs TRNG operations. For more information on TRNG functionality, see the TRNG register descriptions.

Table 18-1: ADSP-BF70x TRNG Register List

Name	Description
TRNG_ALMCNT	TRNG Alarm Counter Register

Table 18-1: ADSP-BF70x TRNG Register List (Continued)

Name	Description
TRNG_ALMMSK	TRNG Alarm Mask Register
TRNG_ALMSTP	TRNG Alarm Stop Register
TRNG_BLKCNT	TRNG Block Count Register
TRNG_CFG	TRNG Configuration Register
TRNG_CNT	Counter Register
TRNG_CTL	TRNG Control Register
TRNG_FRODETUNE	TRNG FRO De-tune Register
TRNG_FROEN	TRNG FRO Enable Register
TRNG_INPUT[n]	TRNG Input Registers
TRNG_INTACK	TRNG Interrupt Acknowledge Register
TRNG_KEY[n]	Post-Process Key Registers
TRNG_LFSR_H	TRNG LFSR Access Register
TRNG_LFSR_L	TRNG LFSR Access Register
TRNG_LFSR_M	TRNG LFSR Access Register
TRNG_MONOBITCNT	TRNG Monobit Test Result Register
TRNG_OUTPUT[n]	TRNG Output Registers
TRNG_POKER[n]	TRNG Poker Test Result Registers
TRNG_RUNCNT	TRNG Run Count Registers
TRNG_RUN[n]	TRNG Run Test State and Result Registers
TRNG_STAT	TRNG Status Register
TRNG_TEST	TRNG Test Register
TRNG_V[n]	TRNG Post-Process "V" Value Registers

Random Number Generation

The random numbers that the TRNG generates are produced by sampling Free Running Oscillators (FRO). The (TRNG_CTL.STARTUPCYC) bit field along with TRNG_CFG.MINREFCYC (minimum refill cycles) and TRNG_CFG.MAXREFCYC (maximum refill cycles) bit fields determine the number of samples taken to generate the first random value and subsequent random values.

1. After enabling the TRNG (TRNG_CTL.TRNGEN bit =1), a number of FRO output samples defined by the TRNG_CTL.STARTUPCYC bit field are gathered in the main linear-feedback shift register (LFSR) before taking a snapshot of the LFSR and storing that snapshot in the random data buffer (after optional post-processing).

2. After taking a snapshot of the LFSR, a number of FRO output samples defined by the TRNG_CFG.MINREFCYC bit field, are gathered in the main LFSR. If the random data buffer is full, sample-taking continues until the number of samples (counting from the snapshot) matches the number of samples defined by the TRNG_CFG.MAXREFCYC bit field. At that point, the TRNG switches off the FROs and powers down.
3. If, after the TRNG_CFG.MINREFCYC sampling period, the random data buffer is not completely filled, a new snapshot of the LFSR is taken and stored in the random data buffer (after optional postprocessing). Control branches back to point step 2 above and a new TRNG_CFG.MINREFCYC sample period starts.

Locking Detection and Prevention

Lock detection in functional mode uses the sampled outputs of the individual FROs. A FRO alarm event is declared when a repeating pattern (of up to four sample lengths) is detected continuously for the number of samples defined in the alarm threshold TRNG_ALMCNT.ALMTHRESH bit field. The alarm event is logged by setting the bit that corresponds to the FRO that caused the alarm in the TRNG_ALMMSK register. If that bit was already set, the corresponding bit in the TRNG_ALMSTP register is set. The FRO is switched off to prevent further alarm events from that FRO. If the TRNG_ALMMSK register bit was not yet set, the FRO is restarted automatically in an attempt to break locking.

The shutdown count field in the alarm count TRNG_ALMCNT.SHDNCNT register monitors the number of FROs switched off. (It counts the number of 1 bits in the TRNG_ALMSTP register.) The shutdown threshold field (TRNG_ALMCNT.SHDNTHRESH) can be configured to generate the shutdown overflow interrupt (TRNG_STAT.SHDNOVR). When the shutdown count in the TRNG_ALMCNT.SHDNCNT bit field exceeds the shutdown threshold in the TRNG_ALMCNT.SHDNTHRESH bit field, the shutdown overflow bit (TRNG_STAT.SHDNOVR) is set to 1 (which can be used to generate an interrupt).

Software can use two strategies for the TRNG operation:

- ***Monitored Operation***. Software checks the TRNG_ALMMSK register at regular intervals (on the order of seconds). If a bit is set in that register, then the program must also check the TRNG_ALMSTP register to determine if a FRO was shut down due to multiple alarm events. If no FROs are shut down, the program clears the TRNG_ALMMSK register to remove the incidental alarm events. If one or more FROs are shut down, the host processor can modify the delay selection of those FROs using the TRNG_FRODETUNE register to prevent further locking. For this type of operation, the shutdown threshold is normally set to a low value (two, for example). The shutdown overflow interrupt can then be used to signal abnormal operation conditions or the breakdown of FROs.
- ***Unmonitored Operation.*** Software sets the shutdown threshold to the acceptable number of FROs to be shut down before taking corrective actions. It then uses the shutdown overflow interrupt to initiate corrective actions (clearing the TRNG_ALMMSK and TRNG_ALMSTP registers, toggling bits in the TRNG_FRODETUNE register). The software must monitor the time interval between these interrupts. If they occur too often (for example, within a minute after each other), this frequency indicates abnormal operating conditions or the breakdown of FROs.

Run Testing

Run Test

The TRNG block counts the number of consecutive zeros and ones (runs) in the data stream shifted into the main LFSR. The run length and bit value is then used to increment a specific bucket counter for these values. After 20,000 bits, the bucket counters must be within specified limits for this test to pass. If not, a run fail interrupt (RUNFAIL) is generated.

Table 18-2: Allowable Limits on Runs of 0's and 1's

Run Count	Bit Value	Min (inclusive)	Max (inclusive)
1	0	2267	2733
	1	2267	2733
2	0	1079	1421
	1	1079	1421
3	0	502	748
	1	502	748
4	0	233	402
	1	233	402
5	0	90	223
	1	90	223
6 and up	0	90	233
	1	90	233

Long Run Test

The long run test fails immediately when a run longer than 33 bits is found and a long run fail (`TRNG_STAT.LRUNFAIL`) interrupt is generated.

Noise Source Test

A noise source failure is declared when a run of 48 or more identical bits is found and a noise fail interrupt (`TRNG_STAT.NOISEFAIL`) is generated.

The status and counts are stored in the `TRNG_RUNCNT` and `TRNG_RUN[n]` registers. Unless otherwise indicated, all counters and state bits in these registers are reset when writing a 1 to either the monobit fail acknowledge (`TRNG_INTACK.MBITFAIL`), the run fail acknowledge (`TRNG_INTACK.RUNFAIL`), or the poker fail acknowledge (`TRNG_INTACK.PKRFAIL`) bits.

Monobit Testing

The TRNG block performs the monobit test on blocks of 20,000 bits (in parallel with the run test and poker test). It monitors the number of zeros and ones in the data stream shifted into the main LFSR. At the start of the block,

the counter is initialized to 10,000. Each 1 value increments the counter, each 0 value decrements the counter. After 20,000 bits, the counter value must be within 9310–10690 (inclusive) for this test to pass. If not, a monobit fail interrupt (TRNG_STAT.MBITFAIL) is generated. The AIS-31 standard (test T1, ref 0) specifies this runtime testing of the TRNG and the parameters.

NOTE: The actual limits stated here are different than the limits stated in the AIS-31 standard due to the implementation. The circuitry in the TRNG uses an up-down counter while the standard just evaluates the sum of the 20,000 bits.

When the continue poker (TRNG_TEST.CONTPKR) bit is set to 1, the test is not stopped after 20,000 bits. The counter keeps incrementing and decrementing (protected against overflow and underflow). But, no actual limit checking happens. The offset from starting value 10,000 indicates the balance of 0 and 1 bits that were checked since the start of the continuous test. The offset is twice the number of missing or extra 1 bits. (An extra 1 bit adds an increment operation and removes one decrement operation. So, having 10,001 1 bits in the block gives a counter value of 10,002.)

Poker Testing

The poker test is run in parallel with the monobit test and run test. Counters in the TRNG_POKER[n] registers are used to count the occurrences of one specific 4-bit value in the data stream fed into the main LFSR. All of the counters are decremented by one every 64 data bits and reset to their start value every 20,000 bits. All counters start at a value of –1 and are decremented 312 times during the 20,000-bit test run.

Each 8-bit counter holds a two's complement value and does not overflow past the range of –128 to +127. At the end of the 20,000-bits block, the values of the counters that contain a single 1 bit appended at the least significant bit side are individually squared and then added. The poker test fails with a poker fail (PKRFAIL) interrupt when:

- the resulting sum (accumulated in the TRNG_MONOBITCNT register) is outside the range 1288 – 71750 (inclusive), or
- one of the counters tries to increment or decrement outside its limit range

NOTE: The poker test fails when the 4-bit values of the data stream are distributed too evenly (with eight counters having incremented 312 times and the others incremented 313 times). This failure is intentional. The minimum mean deviation from the expected value of 312.5 is 4.5. Failure at counter overflow is not an official part of the poker test as specified in the AIS-31 standard (ref 0). It can be shown that the maximum deviation for one counter's value from the mean value of 312.5 (without the poker test failing) is 129.5. Since this deviation is more than 40% of the mean value, it indicates that something is wrong. Here, the counter overflow failure is combined with the official poker test failure.

When the continue poker (TRNG_TEST.CONTPKR) bit is set to 1, the test does not stop after 20,000 bits. The counters keep incrementing and decrementing (the latter every 64 bits). A PKRFAIL interrupt is generated when one of the counters tries to increment or decrement outside its limit range.

Data for Tests

For the monobit test, run test and poker test circuits self-test, the test data written to the TRNG_INPUT[n]0 register is used for the monobit test and run test bit-by-bit. It executes from bit 31 down to bit 0. For the poker test, the written test data is used in eight blocks of 4 bits, starting from bits 31:28 down to bits 3:0.

When the TRNG_TEST.CONTPKR bit =0 during run test and poker test circuits self-tests, the state of the poker and runs test status registers is frozen after all calculations are made and after inputting 20,000 test bits. This state allows time to read the test results. These calculations take around 20 clock cycles to complete. The status registers are reset to their starting states when the first word for the next test block of test bits is written to the TRNG_INPUT[n]0 register. Then, the contents of this word are processed.

X9.31 Postprocessing

Postprocessing is available on some parts using the TRNG. If available, the postprocessing block is situated after the main LFSRs and before the output buffer that stores the random numbers for consumption. The online test logic for monobit, run, and poker testing is before the postprocessor block. The bits used for testing are from the main LFSR.

The postprocessor is based on the ANSI X9.31 specification using 3-Key 3DES cipher algorithm. It does not provide any more entropy than is what is already achieved from sampling the Free Running Oscillators (FROs). The use of the postprocessor only helps with applications requiring the use of an X9.31 compliant RNG.

The following section is an example of postprocessing.

The variables include:

- DT is a 64-bit date/time vector. This value is the input coming from random bits from the main LFSR.
- I is an intermediate value, a 64-bit value
- R is the final result, a 64-bit value
- V is a 64-bit seed value that is to be kept secret
- K is the 3-key for 3DES, each being 64-bits

The postprocessing uses the steps:

1. The intermediate value I is calculated: I = $3DES_{EDE}$(K, DT) with 3DES.
2. The result R is calculated: R = $3DES_{EDE}$(K, I XOR V).
3. Finally, V is updated: V = $3DES_{EDE}$(K, R XOR I).

TRNG Block Diagram

The ***System Block Diagram of the TRNG*** diagram shows the system block for the TRNG. The system includes:

- Free Running Oscillators (FROs) that are the source of sampled bits
- A post-processing unit (processor dependent) for standards compliance

- Test circuitry to detect non-randomness due to failures in the system

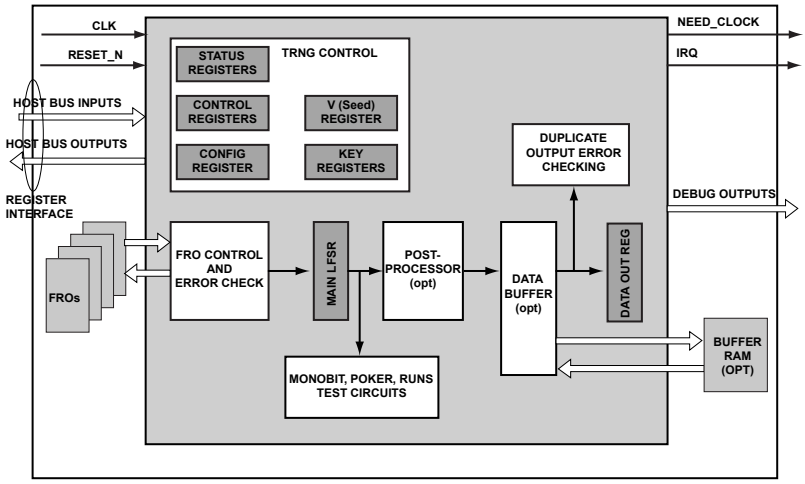

Figure 18-1: System Block Diagram of the TRNG

TRNG Architectural Concepts

The random numbers are accessible to the host processor in four 32-bit registers allowing a single burst read of a 128-bit random number. Acknowledging the data ready (interrupt) state causes the TRNG to move a new value (when available in the data buffer) to the TRNG output register. The TRNG always tries to keep the data buffer completely full. Pulling out data starts the regeneration of a new number by:

- Enabling the FROs
- Capturing their outputs in the LFSR, and
- Cryptographically postprocessing the values from the LFSR

The process produces new random values to replenish the buffer.

The major functional blocks of the TRNG module are:

- The actual TRNG core with control and test circuits, optional post-processor, and optional data buffer control logic
- Free Running Oscillators (FROs) instantiated outside the TRNG core
- A 128-byte buffer RAM instantiated outside the TRNG core

In the TRNG core, the true entropy source uses FROs as the basic building block. The accumulation of timing jitter, caused (for the largest part) by shot noise, creates uncertainty intervals for the output transitions of each FRO. Sampling within an uncertainty interval generates a single bit of entropy, which is ‘accumulated’ in a LFSR. As the uncertainty interval is very narrow compared to the cycle time of a FRO, the mean amount of entropy generated per sample is small (less than 1/100 bit per sample). To increase the entropy generation rate, multiple FROs are used in parallel.

The FROs are asynchronous to one another and asynchronous to the sampling clock to make their behavior truly non-deterministic.

The output signals of the FROs are sampled at regular intervals (in general, at the TRNG core module clock frequency). The samples feed into an error detection circuit that checks for repeating patterns coming out of a FRO. If a repeating pattern persists for a configurable number of samples, the FRO is suspect of having synchronized to (a harmonic of) the sampling interval. This activity drastically reduces the amount of entropy generated by that FRO. The error detection circuit signals this activity as a FRO error event.

Error events can occur during normal operation. The FRO control circuit attempts to restart a FRO that on a first error event. A second error event causes an automatic shutdown of the FRO. Because there are multiple FROs, shutting down a FRO reduces the amount of entropy generated, but it does not immediately jeopardize the TRNG operation. A limit can be configured below which the number of operational FROs does not drop. If this limit is crossed, an interrupt can be generated on the host processor. Software on the host processor can then attempt to prevent frequent locking of a FRO by *de-tuning* it to a slightly different frequency.

An XOR tree combines the sampled outputs and feeds them into an 81-bit LFSR to accumulate entropy and whiten the random bits stream.

NOTE: Here, *whitening* means balancing the number of one and zero bits in the stream.

TRNG Operating Modes

The TRNG has the following operating modes.

Normal Reading Mode

In normal reading mode, random data can only be read out of the output registers of the TRNG module when the ready bit (`TRNG_STAT.RDY`) =1. Acknowledging the data by writing a 1 to the ready acknowledge bit in the `TRNG_INTACK.RDY` register clears the ready bit from the status register and clears the output registers. The registers remain at zero until the next 128-bit data block is available.

Secure Reading Mode

An attacker can try to read the output registers (without acknowledging the data) to obtain a copy of data to be read later by an application. To block this attack, secure reading mode is used. In this mode, enable reading from the output registers (by writing 0x00 to the open read gate bits (`TRNG_INTACK.OPENRDGATE`) before it is possible to access the output registers. Enabling the reading starts a timeout (controlled by the `TRNG_CFG.RDTIMEOUT` bit field). When this timeout expires, reading is disabled and the offered data is acknowledged so that it is not offered again. The host processor must set this timeout such that there is enough time to read the output registers and perform a normal data acknowledge (which aborts the timeout).

Test Mode

In addition to the test circuitry that operates during normal operation, the TRNG has a test mode that allows further diagnosis when errors occur. In test mode, programs have access to the main LFSR through the `TRNG_LFSR_L`,

TRNG_LFSR_M, and TRNG_LFSR_H registers. Programs can also control the finite state machine sample counter using the TRNG_CNT register.

In test mode, the TRNG can be configured to test individual or a chosen set of FROs. It can also be configured to feed test patterns to the delay chain.

TRNG Data Transfer Modes

The host processor reads four 32-bit registers to access the random numbers. Once the registers are read and the data ready interrupt has been acknowledged, the TRNG moves more data from the internal buffer to the output registers (TRNG_OUTPUT[n]).

TRNG Event Control

There are eight events that the TRNG generates. The events are common error events from the runtime testing of the TRNG. While the TRNG is operating and generating random bits, test circuitry is also running statistical tests. The tests determine if the sources of the random bits have started to fail and are not truly producing random bits. There is also a single event to signal when data is ready in the output registers.

These events are captured in the TRNG_STAT register and can generate a single interrupt in the Public Key Interrupt Controller (PKIC). The individual events can be masked so the interrupt is not triggered. This configuration uses the TRNG_CTL register. The event and associated interrupt can be acknowledged in the TRNG_INTACK register.

The ***TRNG Interrupt Signals*** table lists all events or interrupts that the TRNG can generate.

TRNG Interrupt Signals

The TRNG provides a total of eight interrupts multiplexed into one output.

Table 18-3: TRNG Interrupt Signals

Interrupt	Name	Description
0	RDY	Ready. Random number is ready to be read from registers
1	SHDNOVR	Shutdown Overflow. The number of FRO's automatically shut down due to failures or errors have gone above the threshold specified in TRNG_ALMCNT.SHDNTHRESH.
2	STUCKOUT	Stuck Out. Logic circuitry around the output registers has detected the same output has been provided twice.
3	NOISEFAIL	Noise Fail. Logic circuitry monitoring the data shifted into the main LFSR detected 48 identical bits, which is considered a noise source failure.
4	RUNFAIL	Run Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds value for at least one of the TRNG_RUN[n] counters after checking 20,000 bits.
5	LRUNFAIL	Long Run Fail. Logic circuitry monitoring the data shifted into the main LFSR detected 34 identical bits.

Table 18-3: TRNG Interrupt Signals (Continued)

Interrupt	Name	Description
6	PKRFAIL	Poker Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds value in at least one of the 16 TRNG_POKER[n] counters or an out-of-bounds sum of squares values after checking 20,000 bits.
7	MBITFAIL	Monobit Fail. Logic circuitry monitoring the data shifted into the main LFSR detected an out-of-bounds number of 1's after checking 20,000 bits.

ADSP-BF70x TRNG Register Descriptions

True Random Number Generator (TRNG) contains the following registers.

Table 18-4: ADSP-BF70x TRNG Register List

Name	Description
TRNG_ALMCNT	TRNG Alarm Counter Register
TRNG_ALMMSK	TRNG Alarm Mask Register
TRNG_ALMSTP	TRNG Alarm Stop Register
TRNG_BLKCNT	TRNG Block Count Register
TRNG_CFG	TRNG Configuration Register
TRNG_CNT	Counter Register
TRNG_CTL	TRNG Control Register
TRNG_FRODETUNE	TRNG FRO De-tune Register
TRNG_FROEN	TRNG FRO Enable Register
TRNG_INPUT[n]	TRNG Input Registers
TRNG_INTACK	TRNG Interrupt Acknowledge Register
TRNG_KEY[n]	Post-Process Key Registers
TRNG_LFSR_H	TRNG LFSR Access Register
TRNG_LFSR_L	TRNG LFSR Access Register
TRNG_LFSR_M	TRNG LFSR Access Register
TRNG_MONOBITCNT	TRNG Monobit Test Result Register
TRNG_OUTPUT[n]	TRNG Output Registers
TRNG_POKER[n]	TRNG Poker Test Result Registers
TRNG_RUNCNT	TRNG Run Count Registers
TRNG_RUN[n]	TRNG Run Test State and Result Registers
TRNG_STAT	TRNG Status Register
TRNG_TEST	TRNG Test Register

Table 18-4: ADSP-BF70x TRNG Register List (Continued)

Name	Description
TRNG_V[n]	TRNG Post-Process "V" Value Registers

TRNG Alarm Counter Register

The TRNG_ALMCNT register, together with the TRNG_ALMMSK and TRNG_ALMSTP registers, can be used by the host processor to determine if the FRO/sample cycle locking is a problem. Note that incidental alarm events are expected to occur during normal operation. This register also controls the way the monobit test and poker test circuits operate (using the standard 20,000 bit blocks or running continuously).

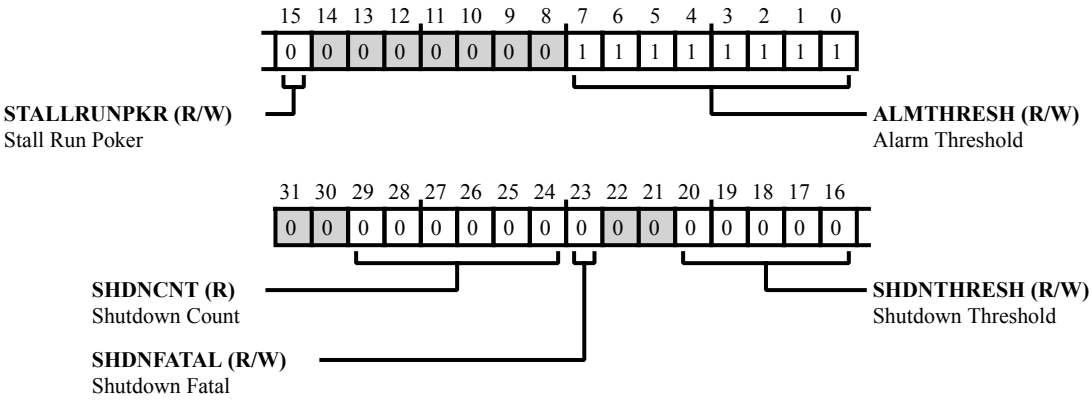

Figure 18-2: TRNG_ALMCNT Register Diagram

Table 18-5: TRNG_ALMCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/NW)	SHDNCNT	Shutdown Count. This read-only field indicates the number of 1 bits in the TRNG_ALMSTP register, the number of FRO's that's been turned off.
23 (R/W)	SHDNFATAL	Shutdown Fatal. When the TRNG_ALMCNT.SHDNFATAL bit field is set, the shutdown overflow (SHDNOVR) interrupt is considered a fatal error requiring taking the complete TRNG engine off-line.
20:16 (R/W)	SHDNTHRESH	Shutdown Threshold. The TRNG_ALMCNT.SHDNTHRESH bit field provides the threshold setting for generating the shutdown overflow (SHDNOVR) interrupt, which is activated when the shutdown count (TRNG_ALMCNT.SHDNCNT) value in this register exceeds the threshold value set here.
15 (R/W)	STALLRUNPKR	Stall Run Poker. When the TRNG_ALMCNT.STALLRUNPKR bit is set, stalls the Monobit Test, Run Test and Poker Test circuits when either the TRNG_STAT.MBITFAIL, TRNG_STAT.RUNFAIL or TRNG_STAT.PKRFAIL bits =1. This allows inspection of the state of the result counters (which would otherwise be reset immediately for the next 20,000 bits block to test).
7:0	ALMTHRESH	Alarm Threshold.

Table 18-5: TRNG_ALMCNT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		The TRNG_ALMCNT.ALMTHRESH bit field sets the alarm detection threshold for the repeating pattern detectors on each FRO. A FRO alarm event is declared when a repeating pattern (of up to four samples length) is detected continuously for the number of samples defined by this fields value. Reset value 255 (decimal) should keep the number of alarm events to a manageable level.

TRNG Alarm Mask Register

A set bit (=1) in the TRNG_ALMMSK register signifies an alarm event and is used by the host processor to determine which of the individual FROs generated the alarm. If a bit in this register is set, the corresponding bit in the TRNG_ALMSTP register is set and the FRO is turned off by clearing the corresponding bit in the TRNG_FROEN register. If a bit is not set, the FRO restarts automatically to try to break sample cycle locking that could have caused the alarm event.

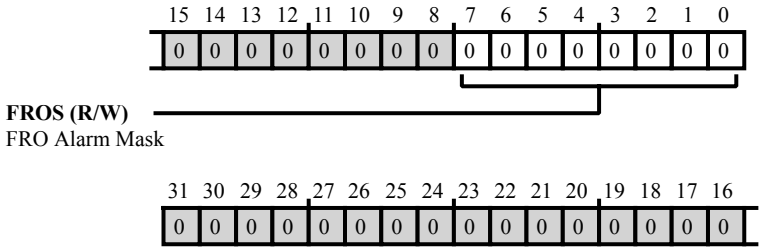

Figure 18-3: TRNG_ALMMSK Register Diagram

Table 18-6: TRNG_ALMMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	FRO Alarm Mask. The TRNG_ALMMSK.FROS bit field provides logging for the alarm events of individual FROs. A 1 in bit [n] indicates FRO n experienced an alarm event.

TRNG Alarm Stop Register

The TRNG_ALMSTP register is used by the host processor to determine which of the individual FROs generated more than one alarm event in quick succession. If a FRO generates an alarm event while a previous event is still logged in the TRNG_ALMMSK register, the corresponding bit in this register is set (=1) and the FRO is turned off by clearing (=0) the corresponding bit in the TRNG_FROEN register. The TRNG_ALMCNT.SHDNCNT bit field keeps track of the number of bits that are set in this register.

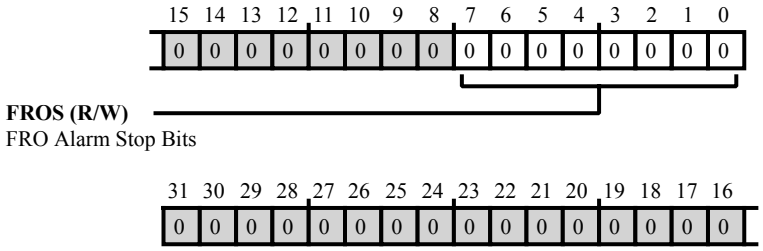

Figure 18-4: TRNG_ALMSTP Register Diagram

Table 18-7: TRNG_ALMSTP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	FRO Alarm Stop Bits. The TRNG_ALMSTP.FROS bit field provides logging for the alarm events of individual FROs. A 1 in bit [n] indicates FRO n experienced more than one alarm event in quick succession and has been turned off. A 1 in this field forces the corresponding bit in the TRNG_FROEN register to 0.

TRNG Block Count Register

The TRNG_BLKCNT register is the counter for the 128-bit blocks generated by the post-processor. These bits are forced to zero when the post-processor is disabled and are cleared to zero when an internal re-seed operation has finished. This register can be used by driver software to determine when to re-seed the post-processor.

The whole 32 bits of this register represent the amount of data (in bytes) generated since a re-seed. The TRNG_BLKCNT register is only present when a post-processor is available.

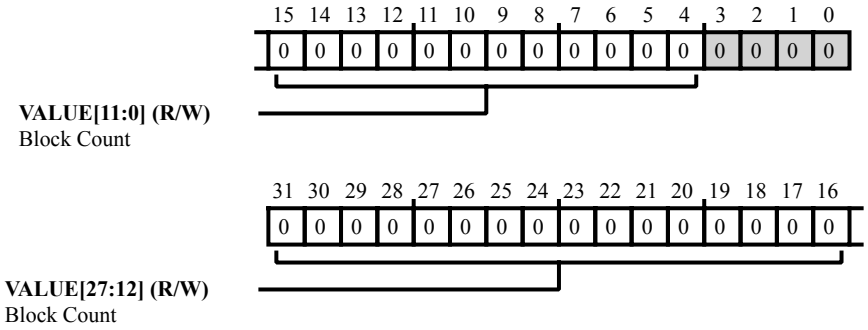

Figure 18-5: TRNG_BLKCNT Register Diagram

Table 18-8: TRNG_BLKCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:4 (R/W)	VALUE	Block Count. The TRNG_BLKCNT.VALUE bit field is the counter for the 128-bit blocks generated by the post-processor. These bits are forced to zero when the post-processor is disabled and are cleared to zero when an internal re-seed operation has finished.

TRNG Configuration Register

The TRNG_CFG register holds the lower and upper limits of the samples taken from the FROs in order to refill the random data buffer. This register also holds the time out value used for secure reading mode.

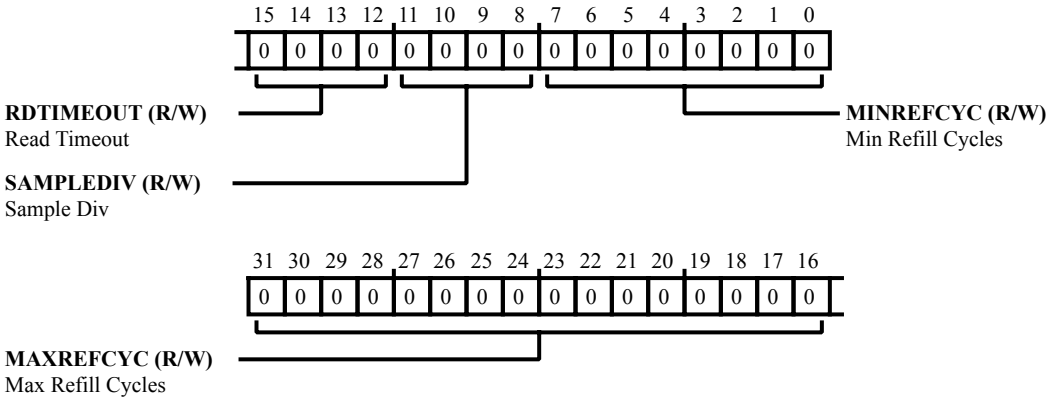

Figure 18-6: TRNG_CFG Register Diagram

Table 18-9: TRNG_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	MAXREFCYC	Max Refill Cycles. The TRNG_CFG.MAXREFCYC bit field determines the maximum number of samples (between 2^8 and 2^{24}) taken to re-generate entropy from the FROs after reading out a 64 bit random number. If the written value of this field is zero, the number of samples is 2^{24}, otherwise the number of samples equals the written value times 2^8. This field can only be modified while the TRNG_CTL.TRNGEN bit =0.
15:12 (R/W)	RDTIMEOUT	Read Timeout. The TRNG_CFG.RDTIMEOUT bit field controls the Secure Reading Mode. When this field is 0, Secure Reading Mode is disabled. Values in the range 115 enable Secure Reading and set a read gate closure timeout of approximately (read_timeout + 1) x 16 clock input cycles. This field can only be modified while the TRNG_CTL.TRNGEN bit =0.
11:8 (R/W)	SAMPLEDIV	Sample Div. The TRNG_CFG.SAMPLEDIV bit field directly controls the number of input cycles between samples taken from the FROs. The default value 0 indicates that samples are taken every cycle, maximum value 15 (decimal) takes one sample every 16 cycles. This field must be set to a value such that the slowest FRO (even under worst-case conditions) has a cycle time less than twice the sample period. The default configuration of the FROs allows this field to remain 0. This field can only be modified while TRNG_CTL.TRNGEN=0.
7:0	MINREFCYC	Min Refill Cycles.

Table 18-9: TRNG_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		The TRNG_CFG.MINREFCYC bit field determines the minimum number of samples (between 2^6 and 2^{24}) taken to re-generate entropy from the FROs after reading out a 64 bit random number. If the value of this field is zero, the number of samples is fixed to the value determined by the maximum refill cycles (TRNG_CFG.MAXREFCYC) field, otherwise the minimum number of samples equals the written value times 64 (which can be up to 2^{14}). The number of samples defined here cannot be higher than the number defined by the TRNG_CFG.MAXREFCYC field (i.e. that field takes precedence). This field can only be modified while the TRNG_CTL.TRNGEN bit =0.

Counter Register

The TRNG_CNT register is used to access the main control Finite State Machine's (FSM) sample counter while the TRNG_CTL.TSTMODE bit =1.

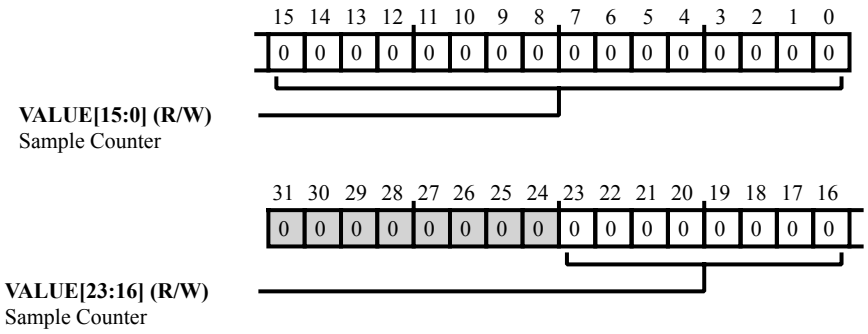

Figure 18-7: TRNG_CNT Register Diagram

Table 18-10: TRNG_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Sample Counter. The TRNG_CNT.VALUE bit field is the sample counter used by control finite state machine. This counter can only be accessed when the TRNG_CTL.TSTMODE bit =1.

TRNG Control Register

The TRNG_CTL register must be written to start accumulating entropy before random numbers can be generated. In most cases, the TRNG_CFG register must also be written prior to writing the TRNG_CTL register. To enable the TRNG, set the TRNG_CTL.TRNGEN bit. This register also controls post-processing (if available). Note that when the TRNG_CTL.TRNGEN bit =1, the start up cycles field (TRNG_CTL.STARTUPCYC) and the post-processing enable bit (if available) are locked. Any writes to these fields are ignored.

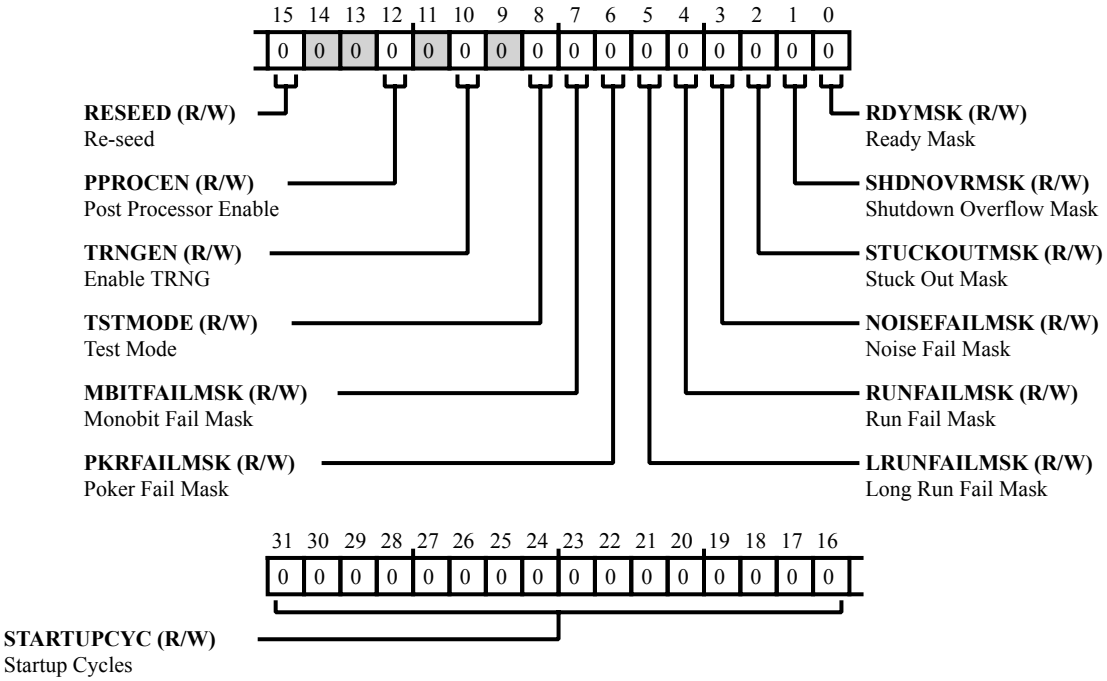

Figure 18-8: TRNG_CTL Register Diagram

Table 18-11: TRNG_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	STARTUPCYC	Startup Cycles. The TRNG_CTL.STARTUPCYC bit field determines the number of samples (between 2^8 and 2^{24}) taken to gather entropy from the FROs during startup. If the written value of this field is zero, the number of samples is 2^{24}, otherwise the number of samples equals the written value times 2^8 . This field can only be written when TRNG_CTL.TRNGEN=0 before the write.
15 (R/W)	RESEED	Re-seed. The TRNG_CTL.RESEED bit is set-only, writing a 1 starts a re-seed cycle and writing a 0 has no effect. A re-seed cycle entails loading the TRNG_KEY[n] and TRNG_V[n] registers with random values generated internally these values are not visible outside the TRNG core.

Table 18-11: TRNG_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		This bit falls back to 0 automatically after the re-seed operation is complete at that time the TRNG_BLKCNT register is reset to zero and the random data buffer is zeroized so that any new data read from the TRNG will use the new seed values. This bit is only present when post-processing is available and can only be set to 1 when TRNG_CTL.TRNGEN=1 before the write. Note that re-seeding can be done with the post-processor disabled (normally used to seed the post-processor before enabling it). When writing a 1 to this bit, all other bits in this register remain unchanged.
12 (R/W)	PPROCEN	Post Processor Enable. Setting the TRNG_CTL.PPROCEN bit enables the FIPS post-processor. If this bit is reset to 0, the post-processor is forced back into the idle state immediately. This bit is only present when post-processing is available and can only be changed when the TRNG_CTL.TRNGEN bit (enable TRNG) was 0 before the write. To change the TRNG_CTL.PPROCEN bit during operation, first put the TRNG into reset (TRNG_CTL.TRNGEN =0). If the post-processor is enabled, it can be disabled by a subsequent write of 0 to this bit (writing this bit when the TRNG_CTL.TRNGEN bit is still 1 has no effect). The post-processor then stops immediately. To enable it, this bit bit must be written with 1. Changing the enabled/disabled state does not affect the contents of the the TRNG_KEY[n] and TRNG_V[n] registers. After changing the state, the TRNG must be started again by setting the TRNG_CTL.TRNGEN bit to 1. Note that it is required to re-gather entropy, so the same number of start-up cycles must be used as when starting the TRNG out of a system reset state.
10 (R/W)	TRNGEN	Enable TRNG. Setting the TRNG_CTL.TRNGEN bit to 1 starts the TRNG, gathering entropy from the FROs for the number of samples determined by the value in the TRNG_CTL.STARTUPCYC (Startup Cycles) field. Resetting this bit to 0 forces all TRNG logic back into the idle state immediately. Resetting this bit to 0 also performs the Un-instantiate operation, clearing all internal post-processor registers.
8 (R/W)	TSTMODE	Test Mode. When the TRNG_CTL.TSTMODE bit is set, access is enabled to the TRNG_CNT and TRNG_LFSR_L, TRNG_LFSR_M and TRNG_LFSR_H registers (the latter are cleared before enabling access) and sets the TRNG_STAT.NEEDCLK bit for testing purposes. This bit must be set to 1 before various test modes in the TRNG_TEST register can be enabled.
7 (R/W)	MBITFAILMSK	Monobit Fail Mask. When the TRNG_CTL.MBITFAILMSK bit is set, this mask allows the TRNG_STAT.MBITFAIL bit to activate the (active HIGH) interrupt output.
6 (R/W)	PKRFAILMSK	Poker Fail Mask. When the TRNG_CTL.PKRFAILMSK bit is set, this mask allows the TRNG_STAT.PKRFAIL bit to activate the (active HIGH) interrupt output.

Table 18-11: TRNG_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
5 (R/W)	LRUNFAILMSK	Long Run Fail Mask. When the TRNG_CTL.LRUNFAILMSK bit is set, this mask allows the TRNG_STAT.LRUNFAIL bit to activate the (active HIGH) interrupt output.
4 (R/W)	RUNFAILMSK	Run Fail Mask. When the TRNG_CTL.RUNFAILMSK bit is set, this mask allows the TRNG_STAT.RUNFAIL bit to activate the (active HIGH) interrupt output.
3 (R/W)	NOISEFAILMSK	Noise Fail Mask. When the TRNG_CTL.NOISEFAILMSK bit is set, this mask allows the TRNG_STAT.NOISEFAIL bit to activate the (active HIGH) interrupt output.
2 (R/W)	STUCKOUTMSK	Stuck Out Mask. When the TRNG_CTL.STUCKOUTMSK bit is set, this mask allows the TRNG_STAT.STUCKOUT bit to activate the (active HIGH) interrupt output.
1 (R/W)	SHDNOVRMSK	Shutdown Overflow Mask. When the TRNG_CTL.SHDNOVRMSK bit is set, this mask allows the TRNG_STAT.SHDNOVR bit to activate the (active HIGH) interrupt output.
0 (R/W)	RDYMSK	Ready Mask. When the TRNG_CTL.RDYMSK bit is set, this mask allows the TRNG_STAT.RDY bit to activate the (active HIGH) interrupt output.

TRNG FRO De-tune Register

The TRNG_FRODETUNE register is used by the host processor to change the frequencies of individual FROs. This can reduce the number of alarm events generated by a specific FRO.

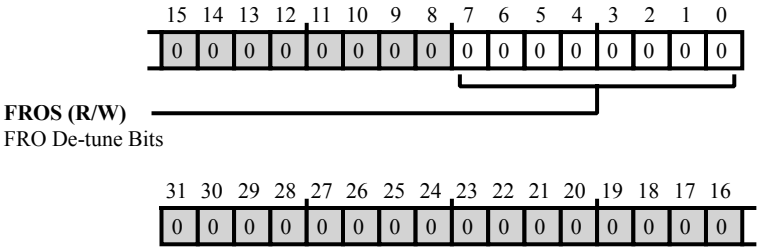

Figure 18-9: TRNG_FRODETUNE Register Diagram

Table 18-12: TRNG_FRODETUNE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	FRO De-tune Bits. The TRNG_FRODETUNE.FROS bits De-tune the FROs. A 1 in bit [n] lets FRO n run approximately 5% faster. The value of one of these bits may only be changed while the corresponding FRO is turned off (by temporarily writing a 0 in the corresponding bit of the TRNG_FROEN register).

TRNG FRO Enable Register

The TRNG_FROEN register can be used by the host processor to enable and disable FROs individually. Only enabled FROs contribute to entropy generation, but require power to do so. Disabled FROs cannot generate alarm events.

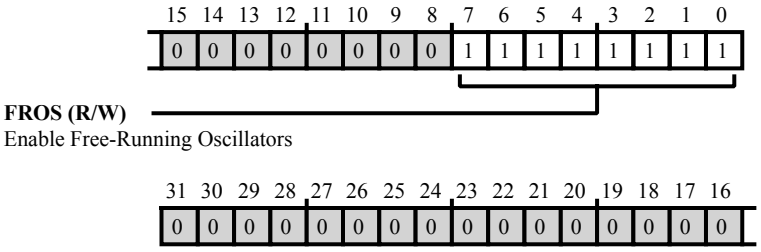

Figure 18-10: TRNG_FROEN Register Diagram

Table 18-13: TRNG_FROEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FROS	Enable Free-Running Oscillators. The TRNG_FROEN.FROS bits are the enables for the individual FROs. A 1 in bit [n] enables FRO n. The default state is all ones to enable all FROs after power-up. Note that the FROs are not actually started up before the TRNG_CTL.TRNGEN bit is set to 1. These bits are automatically forced to 0 (and cannot be written to 1) when the corresponding bit in the TRNG_ALMSTP register has value 1.

TRNG Input Registers

The TRNG_INPUT[n] registers are used as input for post-processor testing (if post processing is available) and as input for Monobit Test, Run Test and Poker Test functionality tests (TRNG_INPUT0 only). They share their addresses with the corresponding TRNG_OUTPUT[n] registers. The least significant word is contained in the TRNG_INPUT0 register.

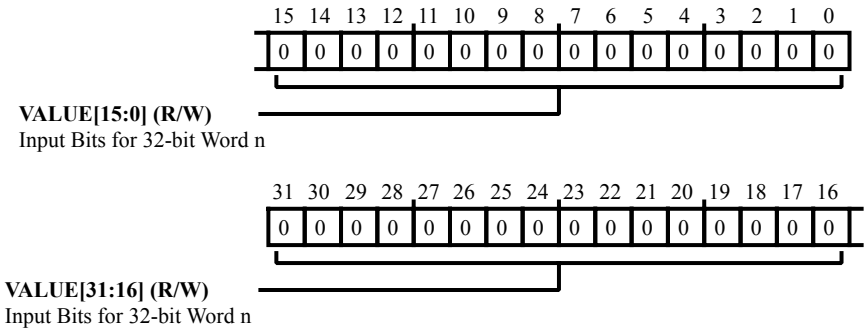

Figure 18-11: TRNG_INPUT[n] Register Diagram

Table 18-14: TRNG_INPUT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Input Bits for 32-bit Word n. The TRNG_INPUT[n].VALUE bit field is used to hold 32-bits of the 64-bit word of test data for 3-DES post-processor (if available). Or, the TRNG_INPUT[n].VALUE bit field is used to hold 32-bit data word for Run Test and Poker Test circuits self test. Can only be written to when the TRNG_STAT.TSTRDY bit =1.

TRNG Interrupt Acknowledge Register

The TRNG_INTACK register is written to acknowledge interrupts indicated in bits [7:0] of the TRNG_STAT register. Writing a 1 to any of the bits [7:2] has side effects in resetting various parts of the TRNG core logic which can also be used even if no interrupts are actually active.

When acknowledging the interrupts, these bits are write '1' to clear the associated bit in TRNG_STAT register. The bit in this register will also automatically be reset to zero.

When secure reading mode is enabled, write bits [7:0] of this register with zeros to enable TRNG data reads from the TRNG_OUTPUT[n] registers. Writing bits [7:0] also starts the (configurable) timeout counter that automatically acknowledges the TRNG data (and disables reads) if the TRNG_INTACK.RDY bit is not written with a 1 within that timeout period.

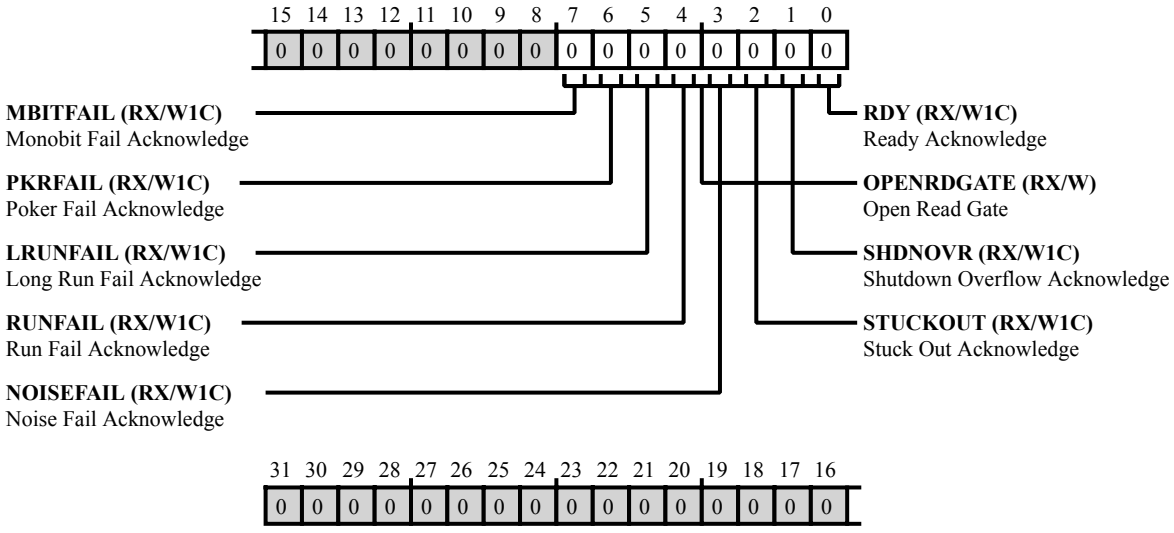

Figure 18-12: TRNG_INTACK Register Diagram

Table 18-15: TRNG_INTACK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (RX/W1C)	MBITFAIL	Monobit Fail Acknowledge. Set the TRNG_INTACK.MBITFAIL bit to acknowledge the Monobit Fail Interrupt. This also resets all counter and state bits in the TRNG_RUN[n], TRNG_MONOBITCNT and the TRNG_POKER[n] registers (except for the TRNG_RUNCNT.LENMAX field).
6 (RX/W1C)	PKRFAIL	Poker Fail Acknowledge. Set the TRNG_INTACK.PKRFAIL bit to acknowledge the Poker Fail Interrupt. This also resets all counter and state bits in the TRNG_RUN[n], TRNG_MONOBITCNT and the TRNG_POKER[n] registers (except for the TRNG_RUNCNT.LENMAX field).
5	LRUNFAIL	Long Run Fail Acknowledge.

Table 18-15: TRNG_INTACK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(RX/W1C)		Set the TRNG_INTACK.LRUNFAIL bit to acknowledge the Long Run Fail Interrupt. Also clears the TRNG_RUNCNT.LENMAX field.
4 (RX/W1C)	RUNFAIL	Run Fail Acknowledge. Set the TRNG_INTACK.RUNFAIL bit to acknowledge the Run Fail Interrupt. Also resets all counter and state bits in the TRNG_RUN[n], TRNG_MONOBITCNT and the TRNG_POKER[n] registers (except for the TRNG_RUNCNT.LENMAX field).
3 (RX/W1C)	NOISEFAIL	Noise Fail Acknowledge. Set the TRNG_INTACK.NOISEFAIL bit to acknowledge the Noise Fail Interrupt. Setting this bit also clears the TRNG_RUNCNT.LENMAX (Run Length Max) field, the random data buffer, the TRNG_OUTPUT[n] registers and the TRNG_STAT.RDY bit.
2 (RX/W1C)	STUCKOUT	Stuck Out Acknowledge. Set the TRNG_INTACK.STUCKOUT bit to acknowledge the Stuck Out Interrupt. Setting this bit also clears the random data buffer, the TRNG_OUTPUT[n] registers and the TRNG_STAT.RDY bit.
1 (RX/W1C)	SHDNOVR	Shutdown Overflow Acknowledge. Set the TRNG_INTACK.SHDNOVR bit to acknowledge the Shutdown Overflow Interrupt.
0 (RX/W1C)	RDY	Ready Acknowledge. The TRNG_INTACK.RDY bit allows a new number (if it is ready in the random data buffer), to directly move into the result register. Once done, the TRNG_STAT.RDY bit is reset, after at most size clock cycles.
7:0 (RX/W)	OPENRDGATE	Open Read Gate. In Secure Reading Mode, the TRNG_INTACK.OPENRDGATE bit writes an all zeros value to bits [7:0] to enable reading of TRNG data from the TRNG_OUTPUT[n] registers. This starts the timeout counter that automatically acknowledges the TRNG data (and disables reading) if the TRNG_INTACK.RDY bit is not written with a 1 within that timeout period.

Post-Process Key Registers

The TRNG_KEY[n] registers are used to load the key used for post-processing (if available). These registers are write-only. Reads return the values of the other registers mapped at the same addresses.

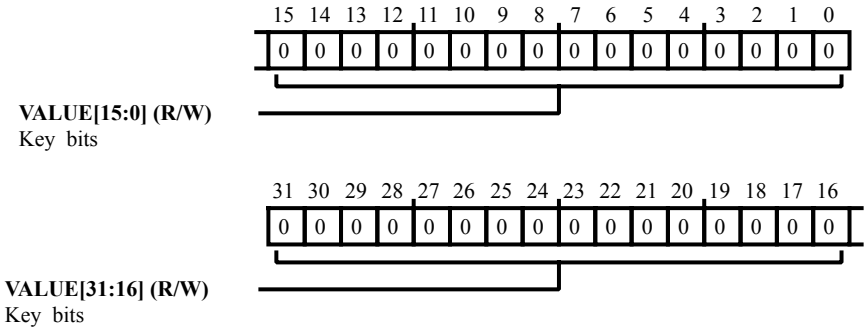

Figure 18-13: TRNG_KEY[n] Register Diagram

Table 18-16: TRNG_KEY[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Key bits. Bits for cipher key used in the post-processor.

TRNG LFSR Access Register

The TRNG_LFSR_H register is used to access bits [80:64] of the main entropy accumulation LFSR while in test mode (TRNG_CTL.TSTMODE =1).

For security reasons, the LFSR contents are zeroed before enabling access.

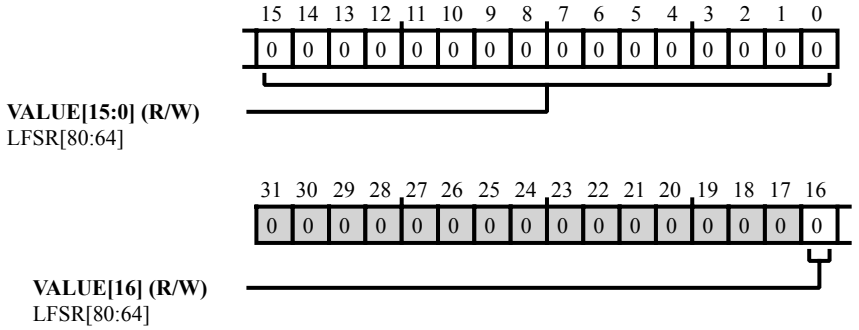

Figure 18-14: TRNG_LFSR_H Register Diagram

Table 18-17: TRNG_LFSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:0 (R/W)	VALUE	LFSR[80:64]. The TRNG_LFSR_H.VALUE bit field contains bits [80:64] of the main entropy accumulation LFSR. This field can only be accessed when the TRNG_CTL.TSTMODE bit =1. Contents are cleared (=0) before access is enabled.

TRNG LFSR Access Register

The TRNG_LFSR_L register is used to access bits [31:0] of the main entropy accumulation LFSR while in test mode (TRNG_CTL.TSTMODE =1).

For security reasons, the LFSR contents are zeroed before enabling access.

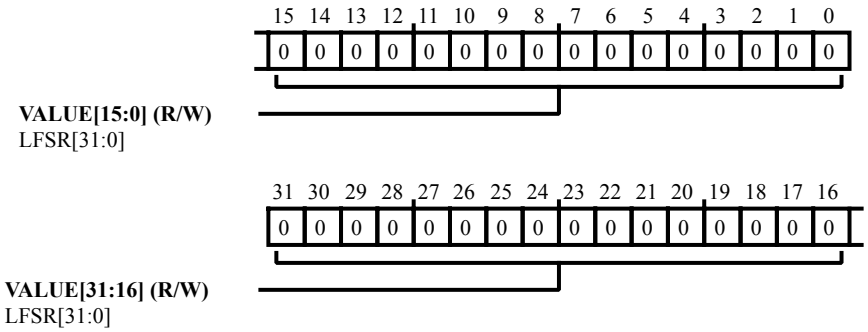

Figure 18-15: TRNG_LFSR_L Register Diagram

Table 18-18: TRNG_LFSR_L Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	LFSR[31:0]. The TRNG_LFSR_L.VALUE bit field contains bits [31:0] of the main entropy accumulation LFSR. This field can only be accessed when the TRNG_CTL.TSTMODE bit =1. Contents are cleared (=0) before access is enabled.

TRNG LFSR Access Register

The TRNG_LFSR_M register is used to access bits [63:32] of the main entropy accumulation LFSR while in test mode (TRNG_CTL.TSTMODE =1).

For security reasons, the LFSR contents are zeroed before enabling access.

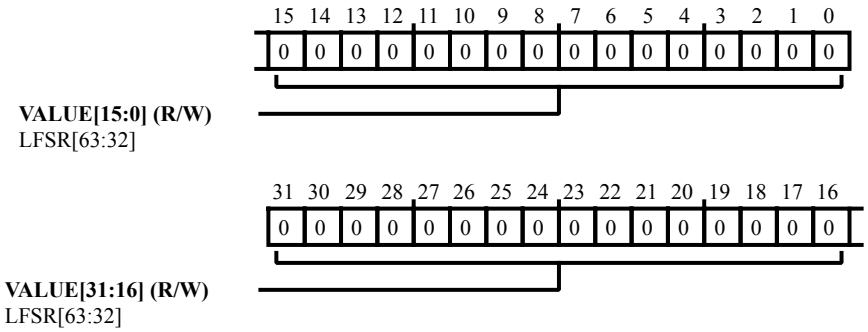

Figure 18-16: TRNG_LFSR_M Register Diagram

Table 18-19: TRNG_LFSR_M Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	LFSR[63:32]. The TRNG_LFSR_M.VALUE bit field contains bits [63:32] of the main entropy accumulation LFSR. This field can only be accessed when the TRNG_CTL.TSTMODE bit =1. Contents are cleared (=0) before access is enabled.

TRNG Monobit Test Result Register

The TRNG_MONOBITCNT register accesses the counter used perform a Monobit Test as specified by the AIS-31 standard (test T1, ref 4). This test is performed on blocks of 20,000 bits (in parallel to the run test and Poker Test).

Note: Immediately after performing the actual Monobit Test at the end of the 20,000 bits block, the counter is used to accumulate the Poker Test results. As a result, the actual Monobit Test count result value can only be read in the TRNG_MONOBITCNT register if the test fails and the stall run Poker (TRNG_ALMCNT.STALLRUNPKR) bit =1.

The monobit test result register is read-only; writing it accesses the registers mapped at the same address. The counter in this register is reset when writing a 1 to either the monobit fail acknowledge (TRNG_INTACK.MBITFAIL), run fail acknowledge (TRNG_INTACK.RUNFAIL) or the poker fail acknowledge (TRNG_INTACK.PKRFAIL) bits.

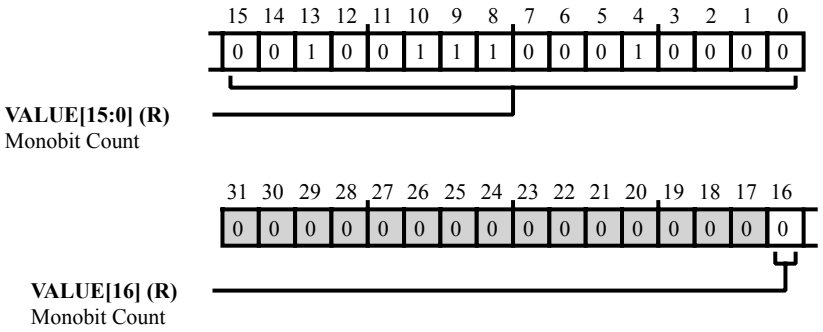

Figure 18-17: TRNG_MONOBITCNT Register Diagram

Table 18-20: TRNG_MONOBITCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16:0 (R/NW)	VALUE	Monobit Count. The TRNG_MONOBITCNT.VALUE bit field is the up/down counter which monitors 1 and 0 bits. After 20,000 bits, this counter should have a value in the range 9310 through 10690 (inclusive) to pass the Monobit Test. This counter is protected against overflow and underflow.

TRNG Output Registers

The TRNG_OUTPUT[n] registers provide read access to the 128-bit random number output. A subset of these registers are also used as output for post-processor testing (if available). They share their addresses with the TRNG_INPUT0 through TRNG_INPUT3 registers. The least significant word is contained in the TRNG_OUTPUT0 register.

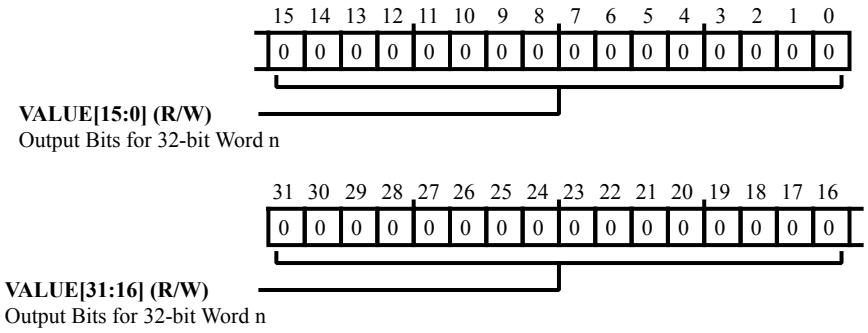

Figure 18-18: TRNG_OUTPUT[n] Register Diagram

Table 18-21: TRNG_OUTPUT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Output Bits for 32-bit Word n. The TRNG_OUTPUT[n].VALUE bit field is used to holds 32 bits of the 128-bit word of random data. Only valid when the TRNG_STAT.RDY bit =1. Alternatively, this register holds the 32-bits of the 42-bit word of result data for 3-DES post-processing testing. Only valid when the TRNG_STAT.TSTRDY bit =1.

TRNG Poker Test Result Registers

The TRNG_POKER[n] registers are used to access the 16 counters used perform a poker test on blocks of 20,000 bits (in parallel to the monobit and run tests).

Poker test result registers are read-only; writing them accesses the registers mapped at these same addresses. All counters in these registers are reset when writing a 1 to either the monobit fail acknowledge (TRNG_INTACK.MBITFAIL), run fail acknowledge (TRNG_INTACK.RUNFAIL) or the poker fail acknowledge (TRNG_INTACK.PKRFAIL) bits.

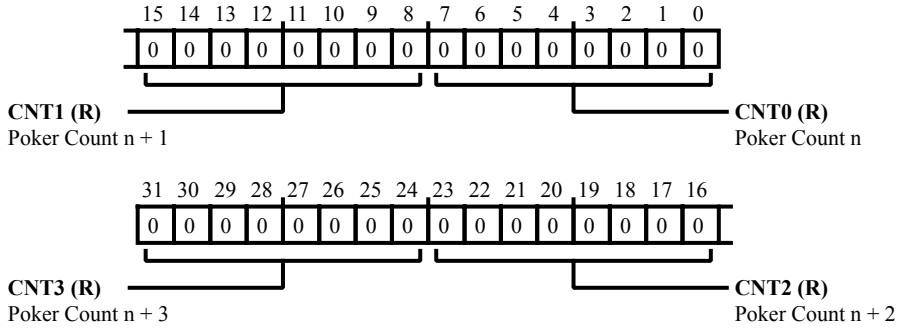

Figure 18-19: TRNG_POKER[n] Register Diagram

Table 18-22: TRNG_POKER[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/NW)	CNT3	Poker Count n + 3. The TRNG_POKER[n].CNT3 bit field provides the counter for 4-bit value 0x3, 0x7, 0xB, 0xF in TRNG_POKER0, TRNG_POKER1, TRNG_POKER2, TRNG_POKER3, respectively.
23:16 (R/NW)	CNT2	Poker Count n + 2. The TRNG_POKER[n].CNT2 bit field provides the counter for 4-bit value 0x2, 0x6, 0xA, 0xE in TRNG_POKER0, TRNG_POKER1, TRNG_POKER2, TRNG_POKER3, respectively.
15:8 (R/NW)	CNT1	Poker Count n + 1. The TRNG_POKER[n].CNT1 bit field provides the counter for 4-bit value 0x1, 0x5, 0x9, 0xD in TRNG_POKER0, TRNG_POKER1, TRNG_POKER2, TRNG_POKER3, respectively.
7:0 (R/NW)	CNT0	Poker Count n. The TRNG_POKER[n].CNT0 bit field provides the counter for 4-bit value 0x0, 0x4, 0x8, 0xC in the TRNG_POKER0, TRNG_POKER1, TRNG_POKER2, TRNG_POKER3, respectively.

TRNG Run Count Registers

The TRNG_RUNCNT registers are used to access the 10 counters that perform a run test and long run test as specified by the AIS-31 standard (tests T3 and T4, ref 4). They are also used to perform the noise source failure test proposed in section E.5 of that same standard.

The TRNG_RUNCNT registers are read-only; writing them accesses the other registers which are mapped at the same addresses. Unless otherwise indicated, all counters and state bits in these registers are reset when writing a 1 to either the Monobit Fail acknowledge (TRNG_INTACK.MBITFAIL), Run Fail acknowledge (TRNG_INTACK.RUNFAIL) or the Poker Fail acknowledge (TRNG_INTACK.PKRFAIL) bits.

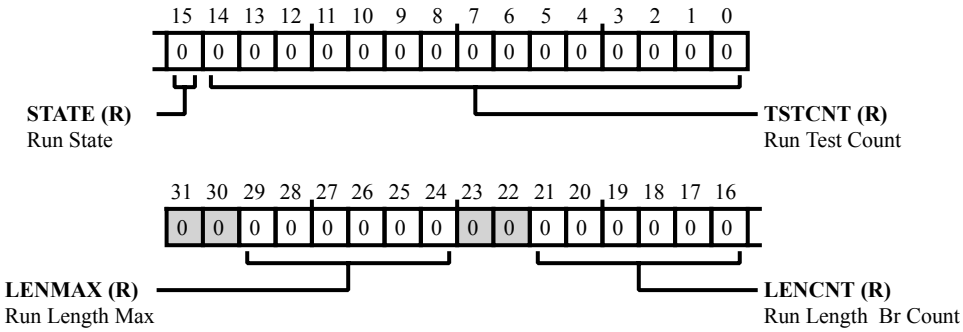

Figure 18-20: TRNG_RUNCNT Register Diagram

Table 18-23: TRNG_RUNCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
29:24 (R/NW)	LENMAX	Run Length Max. The TRNG_RUNCNT.LENMAX bit field configures the maximum run length count value encountered since start of test. This value is reset back to zero when writing a 1 to either the Noise Fail acknowledge (TRNG_INTACK.NOISEFAIL) or the Long Run Fail acknowledge (TRNG_INTACK.LRUNFAIL) bits.
21:16 (R/NW)	LENCNT	Run Length Br Count. The TRNG_RUNCNT.LENCNT bit field configures the counter for the current run of consecutive 0/1 bits; cannot increment past its maximum value of 63.
15 (R/NW)	STATE	Run State. The TRNG_RUNCNT.STATE bit field provides th state of bits in the current run.
14:0 (R/NW)	TSTCNT	Run Test Count. The TRNG_RUNCNT.TSTCNT bit field configures the block length counter for the run and poker tests - counts up for 20,000 tested bits and then controls testing of the run_X_count_... and poker_count_X counters to contain expected values, after which they - and this counter - are reset for the next block.

TRNG Run Test State and Result Registers

The TRNG_RUN[n] registers holds the counts for the associated run bucket for 1's and 0's.

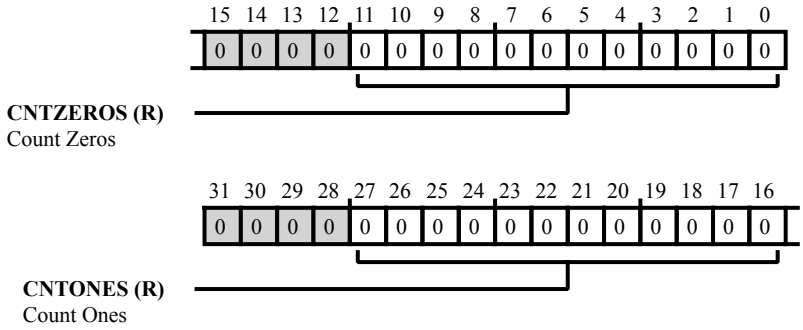

Figure 18-21: TRNG_RUN[n] Register Diagram

Table 18-24: TRNG_RUN[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
27:16 (R/NW)	CNTONES	Count Ones. In TRNG_RUN1, this counter is for single bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 2267 to 2733 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 4095. In TRNG_RUN2, this counter is for two bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 1079 to 1421 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 2047. In TRNG_RUN3, this counter is for three bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 502 to 748 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 1023. In TRNG_RUN4, this counter for four bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 233 to 402 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 511. In TRNG_RUN5, this counter is for five bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 90 to 223 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255. In TRNG_RUN6, this counter for six and higher bit runs of value one bits. After 20,000 bits, this counter should have a value in the range 90 to 233 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255.
11:0 (R/NW)	CNTZEROS	Count Zeros. In TRNG_RUN1, this counter is for single bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 2267 to 2733 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 4095.

Table 18-24: TRNG_RUN[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		In TRNG_RUN2, this counter is for two bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 1079 to 1421 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 2047. In TRNG_RUN3, this counter is for three bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 502 to 748 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 1023. In TRNG_RUN4, this counter is for four bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 233 to 402 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 511. In TRNG_RUN5, this counter is for five bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 90 to 223 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255. In TRNG_RUN6, this counter is for six and higher bit runs of value zero bits. After 20,000 bits, this counter should have a value in the range 90 to 233 (inclusive) to pass the run test. This counter cannot increment past its maximum value of 255.

TRNG Status Register

The TRNG_STAT register provides status results. This register shares the same address as the Interrupt Acknowledge (TRNG_INTACK) register.

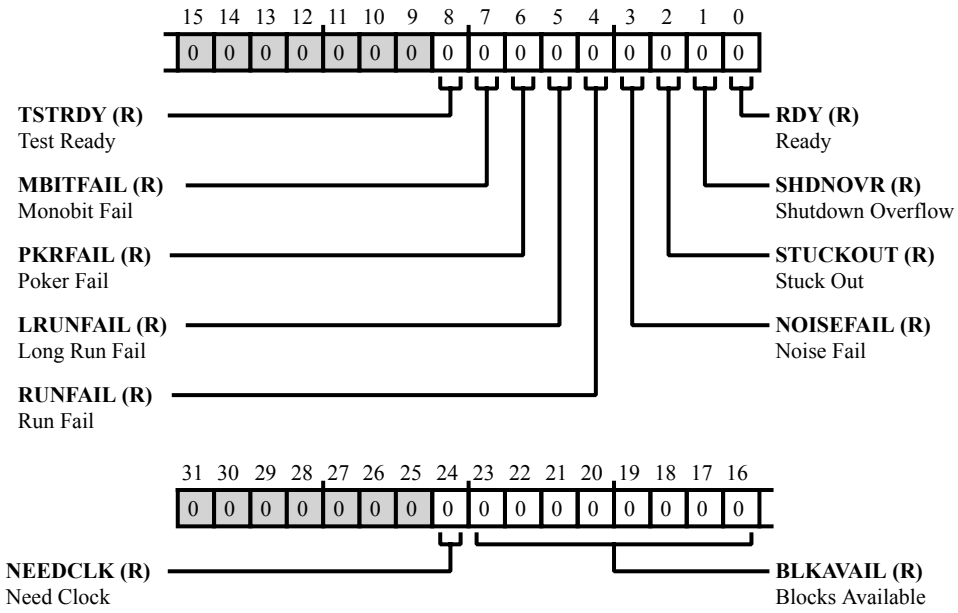

Figure 18-22: TRNG_STAT Register Diagram

Table 18-25: TRNG_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
24 (R/NW)	NEEDCLK	Need Clock. When the TRNG_STAT.NEEDCLK bit is set, it indicates that the TRNG is busy generating entropy or is in one of its test modes the module clock may not be turned off.
23:16 (R/NW)	BLKAVAIL	Blocks Available. This field indicates the number of 128 bits blocks of random data that are available in the random data buffer. If this value is non-zero, the output registers will be re-filled from the random data buffer immediately after acknowledging the TRNG_STAT.RDY by writing a '1' to TRNG_INTACK.RDY.
8 (R/NW)	TSTRDY	Test Ready. When the TRNG_STAT.TSTRDY bit is set, it indicates that data for known-answer tests on the Monobit Test, Run Test, Poker Test and post-processor functions can be written to the TRNG_INPUT[n] registers. When testing the post-processor, result data can be read from those same registers when this bit has become 1 again (after dropping to 0).
7 (R/NW)	MBITFAIL	Monobit Fail.

Table 18-25: TRNG_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		When the TRNG_STAT.MBITFAIL bit is set, the Monobit Test logic monitoring data, shifted into the main LFSR, detected an out-of-bounds number of 1s after checking 20,000 bits (test T1 as specified in the AIS-31 standard).
6 (R/NW)	PKRFAIL	Poker Fail. When the TRNG_STAT.PKRFAIL bit is set, the Poker Test logic monitoring data shifted into the main LFSR detected an out-of-bounds value in at least one of the 16 Poker Counters or an out of bounds sum of squares value after checking 20,000 bits (test T2 as specified in the AIS-31 standard).
5 (R/NW)	LRUNFAIL	Long Run Fail. When the TRNG_STAT.LRUNFAIL bit is set, the Run Test logic monitoring data shifted into the main LFSR detected a sequence of 34 identical bits (test T4 as specified in the AIS-31 standard).
4 (R/NW)	RUNFAIL	Run Fail. When the TRNG_STAT.RUNFAIL bit is set, the Run Test logic monitoring data shifted into the main LFSR detected an out-of-bounds value for at least one of the TRNG_RUN[n].CNTZEROS or TRNG_RUN[n].CNTONES counters after checking 20,000 bits (test T3 as specified in the AIS-31 standard).
3 (R/NW)	NOISEFAIL	Noise Fail. When the TRNG_STAT.NOISEFAIL bit is set, the Run Test logic monitoring data shifted into the main LFSR detected a sequence of 48 identical bits, which is considered a noise source failure as proposed in section E.5 of the AIS-31 standard.
2 (R/NW)	STUCKOUT	Stuck Out. When the TRNG_STAT.STUCKOUT bit is set, the logic around the output data registers detected that the TRNG generates the same value twice in a row.
1 (R/NW)	SHDNOVR	Shutdown Overflow. When the TRNG_STAT.SHDNOVR bit is set, the number of FROs shut down after a second error event (the number of 1 bits in the TRNG_ALMSTP register) has exceeded the threshold set by the TRNG_ALMCNT.SHDNTHRESH bit field.
0 (R/NW)	RDY	Ready. When the TRNG_STAT.RDY bit is set, data is available in the TRNG_OUTPUT0 to TRNG_OUTPUT3 registers. If a new number is already available in the random data buffer, that number is directly moved into the result register. In this case the ready status bit is asserted again, after at most six module clock cycles.

TRNG Test Register

The TRNG_TEST register can be used by the host processor to perform a number of tests on the TRNG logic including:

- Register controlled characterization by connecting the tst_fro_clk_out output to a selected FRO clock output
- FRO logic connectivity and error event detection checking by feeding known patterns through the FRO delay line and error event detection circuits
- Direct XOR-ed FRO outputs capture by disabling the main LFSR feedback logic
- Extend the Monobit Test and Poker Test by not resetting the Monobit count and Poker Test X counters after each 20,000 bits block
- Perform known answer tests on the Run Test, Poker Test and post-processor functions.

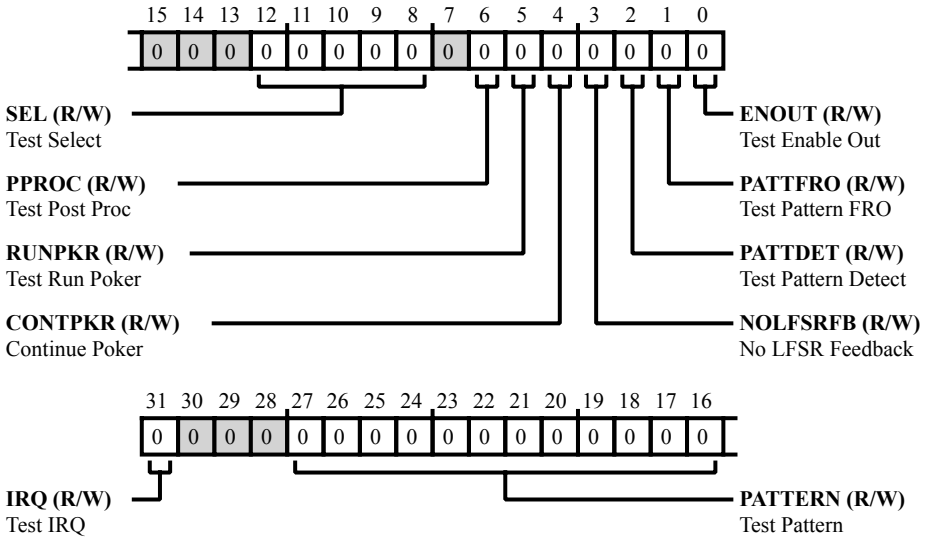

Figure 18-23: TRNG_TEST Register Diagram

Table 18-26: TRNG_TEST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	IRQ	Test IRQ. When the TRNG_TEST.IRQ bit is set force irq output HIGH for interrupt signal connectivity testing.	
		0	Do not force IRQ high
		1	Force IRQ output high
27:16	PATTERN	Test Pattern.	

Table 18-26: TRNG_TEST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
(R/W)		The TRNG_TEST.PATTERN bit field sets up a repeating sequence of bits to be fed into the selected FRO delay chain TRNG_TEST.PATTFRO =1 and/or the selected FRO error detection circuit TRNG_TEST.PATTDET =1. This field is rotated right over one bit, once every sample period, when either of these control bits is 1. Therefore, bit [16] is the actual pattern bit fed into the test target.	
12:8 (R/W)	SEL	Test Select. The TRNG_TEST.SEL bit field configures the number of the FRO to be tested, the value should be in the range of 0 to 7.	
6 (R/W)	PPROC	Test Post Proc. When the TRNG_TEST.PPROC bit is set, it provides direct access to the post-processor for known-answer tests (writing input data to the TRNG_INPUT[n] registers). While this bit is set, the TRNG can continue to generate entropy in the main LFSR and any buffered random data is preserved, to be loaded into the output registers as soon as this bit is reset to 0 again. The need clock output is forced active while this bit is set. For X9.31 post-processors, it is advisable to re-seed the post-processor after running known-answer tests as the original key and V values are modified to known values. This bit is only present when post-processing is available and can only be set to 1 when the TRNG_CTL.PPROCEN bit =1 and the test run poker (TRNG_TEST.RUNPKR) bit in this register is 0.	
5 (R/W)	RUNPKR	Test Run Poker. When the TRNG_TEST.RUNPKR bit is set, it provides direct access to the inputs of the Monobit, Run and Poker Test circuits (writing input data in chunks of 32 bits to the TRNG_INPUT0 register). While this bit is 1, the TRNG is not allowed to generate entropy but any buffered random data is preserved, to be loaded into the output registers as soon as this bit is reset to 0 again. The TRNG_STAT.NEEDCLK bit is forced active while this bit is 1. The Monobit, Run and Poker Test circuits are reset to their initial states on any change of this bit.	
4 (R/W)	CONTPKR	Continue Poker. When the TRNG_TEST.CONTPKR bit is set, Monobit Test and Poker Test keep running continuously by not resetting the Monobit count (TRNG_MONOBITCNT) and poker counters (TRNG_POKER[n] register) at the end of each 20,000 bits test block. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.	
		0	Do not continue poker test
		1	Continue poker test

Table 18-26: TRNG_TEST Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	NOLFSRFB	No LFSR Feedback. When the TRNG_TEST.NOLFSRFB bit is set, it removes XNOR feedback from the main LFSR, converting it into a normal shift register for the XOR-ed outputs of the FROs (shifting data in on the LSB side). A 1 also forces the LFSR to sample continuously. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.	
		0	Keep XNOR feedback
		1	Remove XNOR feedback
2 (R/W)	PATTDET	Test Pattern Detect. When the TRNG_TEST.PATTDET bit is set, it repeatedly feeds test pattern (PATTERN) into the error detection circuit of the FRO selected by the test select (TRNG_TEST.SEL) field. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.	
		0	Do not repeat feed test pattern
		1	Repeat feed test pattern
1 (R/W)	PATTFRO	Test Pattern FRO. When the TRNG_TEST.PATTFRO bit is set, it repeatedly feeds test pattern (PATTERN) into the delay chain of the FRO selected by the test select (TRNG_TEST.SEL) field by forcing the corresponding FRO enable (FROEN) output LOW. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.	
		0	Do not repeat feed test pattern
		1	Repeat feed test pattern
0 (R/W)	ENOUT	Test Enable Out. When the TRNG_TEST.ENOUT bit is set, it enables the tst_fro_clk_out output, connecting to the FRO selected by the test select (TRNG_TEST.SEL) field. This bit can only be set to 1 when TRNG_CTL.TSTMODE =1.	
		0	Disable tst_fro_clk_out
		1	Enable tst_fro_clk_out

TRNG Post-Process "V" Value Registers

The TRNG_V[n] registers are used to load the V value used for post-processing (if available). These registers are write-only. Reads return the values of the other registers mapped at the same addresses.

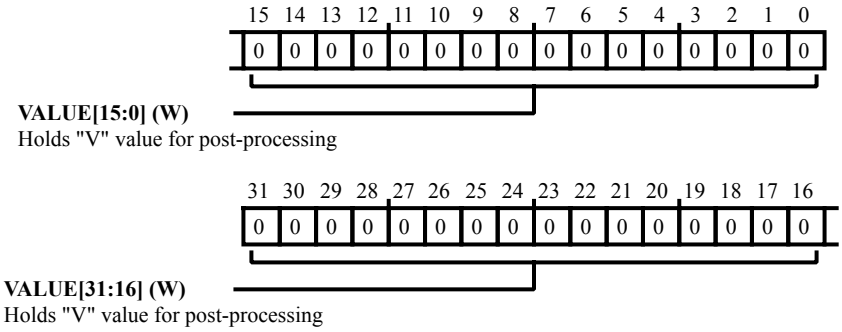

Figure 18-24: TRNG_V[n] Register Diagram

Table 18-27: TRNG_V[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Holds "V" value for post-processing. Bits of the post-processing 'V' value.

19 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
- High Speed Descriptor Array (HSDA) feature for MDMA streams

DMA Channel Functional Description

This section provides a functional description of the DMA channel interface.

DMA Channel List for ADSP-BF70x

The following tables provide DMA channel assignment and channel parametric information for the ADSP-BF70x processor.

Table 19-1: SPORT DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/Monitor Support	Memory Bus Width (DMA_STAT.MBWID)	Peripheral Bus Width (DMA_STAT.PBWID)	Max Outstanding Reads	Max Outstanding Writes
DMA0	SPORT0 Channel A	64	No	32-bit	32-bit	4	4
DMA1	SPORT0 Channel B	64	No	32-bit	32-bit	4	4
DMA2	SPORT1 Channel A	64	No	32-bit	32-bit	4	4
DMA3	SPORT1 Channel B	64	No	32-bit	32-bit	4	4

Table 19-2: SPI DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/Monitor Support	Memory Bus Width (DMA_STAT.MBWID)	Peripheral Bus Width (DMA_STAT.PBWID)	Max Outstanding Reads	Max Outstanding Writes
DMA4	SPI0 Transmit	64	No	32 bit	32-bit	4	4
DMA5	SPI0 Receive	64	No	32 bit	32-bit	4	4
DMA6	SPI1 Transmit	64	No	32 bit	32-bit	4	4
DMA7	SPI1 Receive	64	No	32 bit	32-bit	4	4
DMA8	SPI2 Transmit	64	No	32 bit	32-bit	4	4
DMA9	SPI2 Receive	64	No	32 bit	32-bit	4	4

Table 19-3: UART DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/Monitor Support	Memory Bus Width (DMA_STAT.MBWID)	Peripheral Bus Width (DMA_STAT.PBWID)	Max Outstanding Reads	Max Outstanding Writes
DMA10	UART0 Transmit	64	No	32-bit	32-bit	4	4

Table 19-3: UART DMA Channels (Continued)

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/Monitor Support	Memory Bus Width (DMA_STAT.MBWID)	Peripheral Bus Width (DMA_STAT.PBWID)	Max Outstanding Reads	Max Outstanding Writes
DMA11	UART0 Receive	64	No	32-bit	32-bit	4	4
DMA12	UART1 Transmit	64	No	32-bit	32-bit	4	4
DMA13	UART1 Receive	64	No	32-bit	32-bit	4	4

Table 19-4: EPPI DMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/Monitor Support	Memory Bus Width (DMA_STAT.MBWID)	Peripheral Bus Width (DMA_STAT.PBWID)	Max Outstanding Reads	Max Outstanding Writes
DMA14	EPPI0 Channel 0	128	No	32-bit	32-bit	8	8
DMA15	EPPI0 Channel 1	128	No	32-bit	32-bit	8	8

Table 19-5: MDMA Channels

DMA Channel	Peripheral	FIFO Depth (Bytes)	Bandwidth Limit/Monitor Support	Memory Bus Width (DMA_STAT.MBWID)	Peripheral Bus Width (DMA_STAT.PBWID)	Max Outstanding Reads	Max Outstanding Writes
DMA16	MDMA0 Source/ Destination	128	Yes	32-bit	32-bit	4	4
DMA17	MDMA0 Destination/Source	64	Yes	32-bit	32-bit	4	4
DMA18	MDMA1 Source/ Destination	64	Yes	32-bit	32-bit	8	2
DMA19	MDMA1 Destination/Source	64	Yes	32-bit	32-bit	4	8
DMA20	MDMA2 Source/ Destination	128	Yes	32-bit	32-bit	8	2
DMA21	MDMA2 Destination/Source	64	Yes	32-bit	32-bit	4	8

ADSP-BF70x 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 19-6: ADSP-BF70x 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

ADSP-BF70x DMA Channel List

Table 19-7: ADSP-BF70x DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPORT0_A_DMA	SPORT0
DMA1	SPORT0_B_DMA	SPORT0
DMA2	SPORT1_A_DMA	SPORT1
DMA3	SPORT1_B_DMA	SPORT1
DMA4	SPI0_TXDMA	SPI0

Table 19-7: ADSP-BF70x DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA5	SPI0_RXDMA	SPI0
DMA6	SPI1_TXDMA	SPI1
DMA7	SPI1_RXDMA	SPI1
DMA8	SPI2_TXDMA	SPI2
DMA9	SPI2_RXDMA	SPI2
DMA10	UART0_TXDMA	UART0
DMA11	UART0_RXDMA	UART0
DMA12	UART1_TXDMA	UART1
DMA13	UART1_RXDMA	UART1
DMA14	EPPI0_CH0_DMA	EPPI0
DMA15	EPPI0_CH1_DMA	EPPI0
DMA16	SYS_MDMA0_SRC	MDMA
DMA17	SYS_MDMA0_DST	MDMA
DMA18	SYS_MDMA1_SRC	MDMA
DMA19	SYS_MDMA1_DST	MDMA
DMA20	SYS_MDMA2_SRC	MDMA
DMA21	SYS_MDMA2_DST	MDMA

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.

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 or 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.

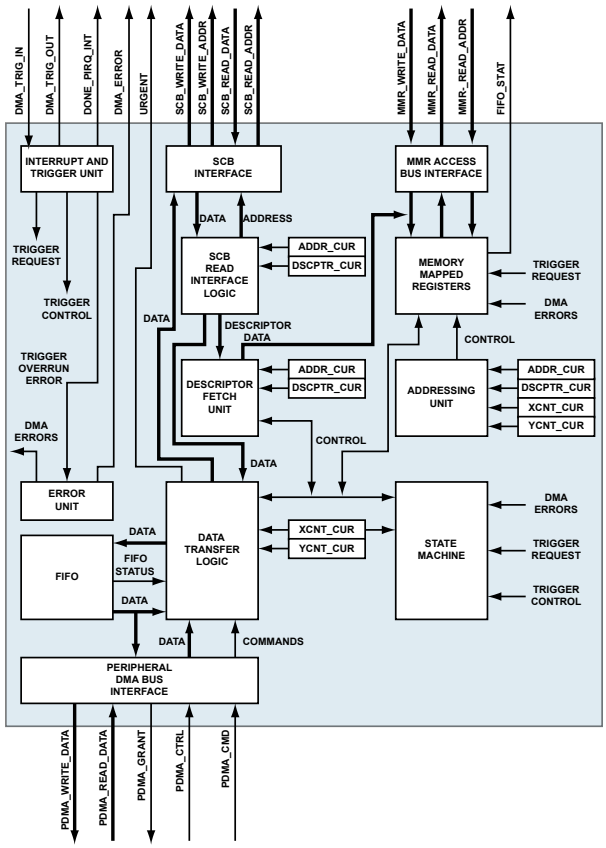

Figure 19-1: DMA Channel Block Diagram

For more information on the interfaces shown in the block diagram, see:

- DMA Channel Peripheral DMA Bus

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 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 4 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 as shown in the ***SCB Interface Signals*** table.

Table 19-8: SCB Interface Signals

Signal	Width (bits)	Description
SCB_WRITE_DATA	16, 32, 64, or 128	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.MBWID`.
SCB_WRITE_ADDRESS	32	Write address bus. Provides the address of the first transfer in a burst transaction.
SCB_READ_DATA	16, 32, 64, or 128	Data bus used for read operations. The width of the bus can be determined from `DMA_STAT.MBWID`.
SCB_READ_ADDRESS	32	Read address bus. Provides the address of the first transfer in a burst transaction.

SCB Burst Transfers

The SCB interface supports burst transfers for memory read and write operations. The burst length is a function of the configurable memory size of the DMA channel for the work unit and the fixed bus width of the SCB data bus of the DMA channel.

- If the DMA channel configuration selects a memory transfer size less than or equal to the DMA channels bus width, the burst length is always 1.
- If the configured memory size is greater than the SCB interface bus width, the burst length is sufficient to transfer a transaction as specified by the configured memory size.

Table 19-9: DMA Channel SCB Burst Lengths

Configured Memory Size	Burst Length			
	16-bit Bus	*32-bit Bus*	*64-Bit Bus*	*128-bit Bus*
1 Byte	1	1	1	1
2 Bytes	1	1	1	1
4 Bytes	2	1	1	1
8 Bytes	4	2	1	1
16 Bytes	8	4	2	1
32 Bytes	16	8	4	2

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 19-10: DMA Channel Address Alignment Requirements

Configured Memory Size	Address Restriction
1 Byte	No restriction

Table 19-10: DMA Channel Address Alignment Requirements (Continued)

Configured Memory Size	Address Restriction
2 Bytes	ADDR[0] == 0
4 Bytes	ADDR[1:0] == 0
8 Bytes	ADDR[2:0] == 0
16 Bytes	ADDR[3:0] == 0
32 Bytes	ADDR[4:0] == 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 supporting data bus widths of 8, 16, 32, or 64 bits. 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 19-11: Peripheral DMA Bus Signals

Signal	Width (bits)	Description
PDMA_WRITE_DATA	8, 16, 32, or 64	Data bus used for write operations. The width of the bus can be determined from `DMA_STAT.PBWID`.
PDMA_READ_DATA	8, 16, 32, or 64	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 19-12: 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
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 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 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. 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 interrupts 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 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 channel configurations that do not involve an additional peripheral impose no restrictions on which of the DMA channels is used for the read operation or the write operation. But, the configuration of both channels cannot have the same transfer direction. For MDMA channel configurations that enable a peripheral between the read and write channels, be aware of possible restrictions imposed on which channel can be used for a given transfer direction.

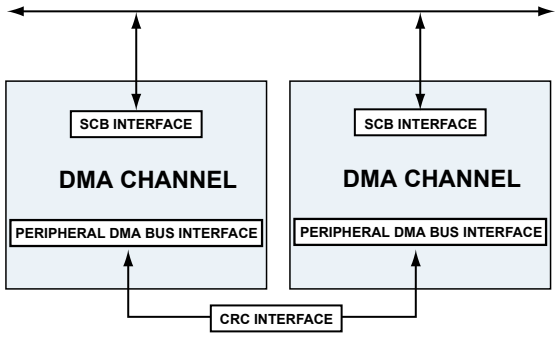

Figure 19-2: MDMA Channel Dedicated Pair

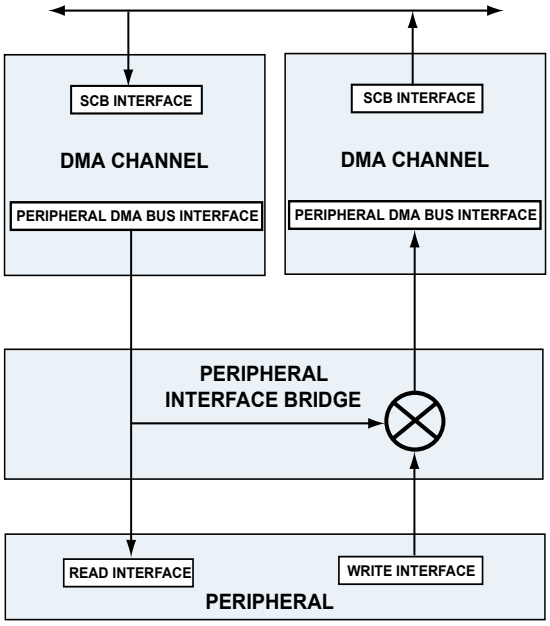

Figure 19-3: MDMA Channel Pair with Peripheral

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 `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. Enabling `DMA_CFG.TWAIT` 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.

Low-Speed MDMA. MDMA0 belongs to this category. It runs in the SCLK0 domain and support 32-bit memory and peripheral bus width. The maximum theoretical bandwidth of this MDMA stream is 400 MB/s.

High-Speed MDMA. MDMA1 and MDMA2 belong to this category. They run in the SCLK1 domain and support 32-bit memory and peripheral bus width. The maximum theoretical bandwidth of these MDMA streams is 800 MB/s.

NOTE: BF70x processors have three MDMA streams available.

- The MDMA0 stream operates in the *SCLK0* domain; whereas the MDMA1 and MDMA2 streams operate in the *SCLK1* domain. These streams are capable of providing double the bandwidth of MDMA0 stream.
- Only MDMA0 streams allows access to system MMR space.
- The CRC block can be enabled on MDMA1 and MDMA2 streams only.
- HSDA mode should only be used on MDMA1 and/or MDMA2.

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 19-13: 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 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 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 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 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

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 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 interrupts are combined into a single shared interrupt. Combined error signals require reading the DMA_STAT register of each DMA channel associated with a combined error interrupt 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 19-14: 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 19-15: 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 19-16: 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

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 19-17: 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

Table 19-17: Offsets for Descriptor-List Mode Parameters and Descriptors (Continued)

Descriptor Offset	Parameter Register
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.

High-Speed Descriptor-Array Mode (HSDA)

HSDA mode can significantly improve the throughput of descriptor-array chains with one data word per work unit and no change to the DMA_CFG register settings between work units. High-speed descriptor-array mode (enabled with DMA_CFG.HSDA bit) is useful in the following cases.

- ***Gather DMA***. This case applies for scattered source data that spreads across random memory locations, such that the locations are not fetch-able using fixed address increments starting from one base address. In other words, the channel cannot fetch the data using a single work unit programmed to use 1D DMA or 2D DMA. For this reason, this case uses a descriptor-array chain, with each work unit taking in one data for the source stream of MDMA. The start addresses present in the descriptor-array indicate each of the random addresses at which data is present. The destination DMA can be a single work unit which places the collected data into a single chunk as required.
- ***Scatter DMA***. This case applies for arranged source data that is fetch-able using a single work unit. However, for the destination DMA, the channel requires a descriptor-array chain to scatter the data to multiple addresses. Each descriptor work unit writes one data word for the destination channel of MDMA.

Effects of Using HSDA

When the configuration enables high-speed descriptor-array mode for a channel, DMA controller operations adjust as follows:

- The controller pipelines the address requests of consecutive work units for data transfer. This operation change implies that the DMA controller can give out the address requests (for data transfer) of the next work unit without waiting for memory to complete the data transfer corresponding to the previous work unit. The DMA controller issues the address requests of the next work unit even before the read data or the write response corresponding to the previous work unit has arrived.
- The controller prefetches the descriptors for future work units. This operation change uses an 8-deep FIFO (the descriptor-prefetch FIFO) within the controller, which stores prefetched descriptors through SCB reads. The controller uses these prefetched descriptors as required. When the DMA_CFG.DBURST bit enables data burst, the descriptor fetches occur in bursts of 4 descriptors at a time. The channel makes new requests for descriptors only if the descriptor–prefetch FIFO has at least 4 free locations. If the DMA_CFG.DBURST bit does not enable data burst, the descriptor requests occur in varying burst sizes, depending on the number of free locations in the descriptor prefetch FIFO.

HSDA Mode Restrictions

High-speed descriptor-array mode operations have the following restrictions:

1. Only enable HSDA when DMA controller programming selects descriptor-array mode with the DMA_CFG.FLOW bit field =0x5.
2. Disable interrupt and trigger output generation in HSDA mode. In other words, configure the DMA_CFG.TRIG bit =0x0 and configure the DMA_CFG.INT bit field =0x0.
3. Only use HSDA for MDMA, not peripheral DMA. HSDA can be used either in the source MDMA channel or in the destination MDMA channel, but not on both channels.
4. The DMA_CFG.NDSIZE bit field must be 0x0 or 0x1. The entire descriptor chain must use the same DMA_CFG.NDSIZE value. If an operation needs a different next descriptor set size, use a new descriptor chain. Use the following methods to stop HSDA:
   - When DMA_CFG.NDSIZE =0x0, stopping HSDA requires a write to the DMA_CFG.EN bit.
   - When DMA_CFG.NDSIZE =0x1, stopping HSDA requires a write to DMA_CFG.EN bit. Alternatively, the final DMA_CFG descriptor can stop a mode configuration with an interrupt enabled (if required) and HSDA disabled.
5. When the DMA_CFG.NDSIZE bit field =0x1, the DMA_CFG register must not change between successive descriptors except for the last work unit mentioned in the DMA_CFG.NDSIZE restriction. Enabling or disabling the DMA_CFG.TWAIT bit applies in any work unit.

   For a description of the function of the DMA_CFG.TWAIT bit in HSDA mode, see Trigger Wait (TWAIT) Function with HSDA Enabled.
6. If one work unit in a descriptor chain has HSDA enabled, the next work unit cannot have HSDA disabled, unless it is a stop mode work unit.
7. Do not set the DMA_CFG.SYNC bit in read mode while in HSDA mode. If the configuration enables DMA_CFG.SYNC, the next work unit starts only after completing the data transfer for one work unit, which slows HSDA.
8. When the DMA_CFG.NDSIZE = 0x0, the descriptors fetched by the DMA channel (until it gets disabled) should be aligned as per the DMA_CFG.MSIZE field. This implies that the descriptor array may need to extend beyond the number of work units required.
   - If HSDA is enabled on the source channel, the number of extra source descriptors needed are : 2*FIFO_DEPTH (128 bytes) /work_unit_size (bytes) + 10.
   - If HSDA is enabled on the destination channel, the number of extra destination descriptors needed is 10.
9. When DMA_CFG.NDSIZE = 0x1, the descriptors should not be placed near boundary of reserved memory space, because HSDA fetches the descriptors in advance. Or, it is safe to add extra 10 descriptors at the end.

Trigger Wait (TWAIT) Function with HSDA Enabled

In high-speed descriptor-array mode, if a write to a memory-mapped register sets the DMA_CFG.TWAIT bit in the first DMA configuration word, the descriptor fetches of the first work unit only start after the reception of the trigger input. If any of the configuration words fetched as part of descriptor fetch (DMA_CFG.NDSIZE =0x1 case) enable DMA_CFG.TWAIT HSDA, the channel delays the data transfer of the next work unit until the channel receives the next trigger input. As part of descriptor prefetch functionality in HSDA, the descriptor fetch of the next work unit can already be complete. In this case, the channel only delays the data transfer the next work unit until an incoming trigger.

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.

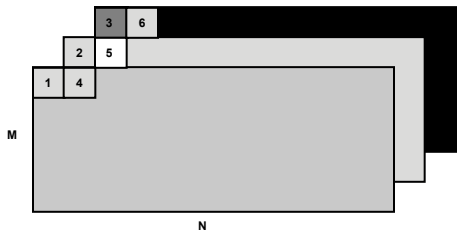

Figure 19-4: Capturing a Video Data Stream 2D DMA Example

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 19-18: 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, 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 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 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 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 interrupts based on the work unit state and, instead, results in generating an interrupt when the DMA channel receives the command from the peripheral.

The channel only generates the interrupt 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 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 interrupts 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.

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.

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.

On DMA completion, the DMA channel enters the idle state until either disabled or reconfigured for a new transfer.

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.

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.

On DMA completion, the DMA channel restarts the DMA operation, creating an endless circular buffer transfer.

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 must be set for stop mode. The DMA_CFG.WNR bit must be configured 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 must be set for stop mode. The DMA_CFG.WNR bit must be configured 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 and trigger events and using a poll of the status bits of these events within the DMA channel registers, or combining these techniques. Polling for address or count can only provide synchronization within loose tolerances comparable to pipeline lengths.

Interrupt and Trigger Event-Based Synchronization

Interrupt and trigger based synchronization methods must avoid interrupt or trigger 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 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 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 interrupts 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 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.

HSDA Mode Programming Guidelines

Use the following guidelines for HSDA mode configurations DMA_CFG.NDSIZE=0 and DMA_CFG.NDSIZE=1.

1. When DMA_CFG.NDSIZE=0 and HSDA mode is enabled on a source channel:
    - Keep extra source descriptors in the memory: 2*FIFO_DEPTH (128 bytes) /work_unit_size (bytes) + 10.
    - Enable the interrupt of the destination MDMA channel. The interrupt of the source MDMA channel can be masked.
    - Enable the destination MDMA channel first and then the source DMA channel.
    - After the destination MDMA interrupt occurs, disable the MDMA channels. Disable the source DMA channel first, but make sure to maintain the DMA_CFG.HSDA bit set while disabling this DMA channel. Then, disable the destination DMA channel.
    - Poll the DMA_STAT.RUN field of the source channel for the idle or stop state.
2. When DMA_CFG.NDSIZE=0 and HSDA mode is enabled on a destination channel:
    - Keep ten extra destination descriptors in the memory.
    - Set the DMA_CFG.SYNC bit of the source MDMA channel.

- Enable the interrupt of the source MDMA channel. The interrupt of the destination MDMA channel can be masked.
- Enable the source MDMA channel first and then the destination MDMA channel.
- After the source MDMA interrupt occurs, poll the DMA_STAT.FIFOFILL status of destination MDMA channel for the empty state.
- Disable the MDMA channels. Disable the destination DMA channel first, but make sure to maintain the DMA_CFG.HSDA bit set while disabling this DMA channel. Then, disable the source DMA channel.
- Poll the DMA_STAT.RUN field of the destination channel for the idle or stop state.

3. When DMA_CFG.NDSIZE=1, make sure that last work unit is configured in STOP mode with DMA_CFG.HSDA= 0.

ADSP-BF70x DMA Register Descriptions

DMA Channel (DMA) contains the following registers.

Table 19-19: ADSP-BF70x 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.

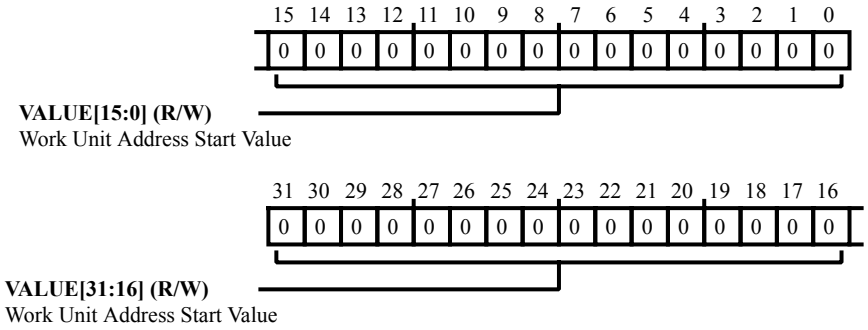

Figure 19-5: DMA_ADDRSTART Register Diagram

Table 19-20: 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.

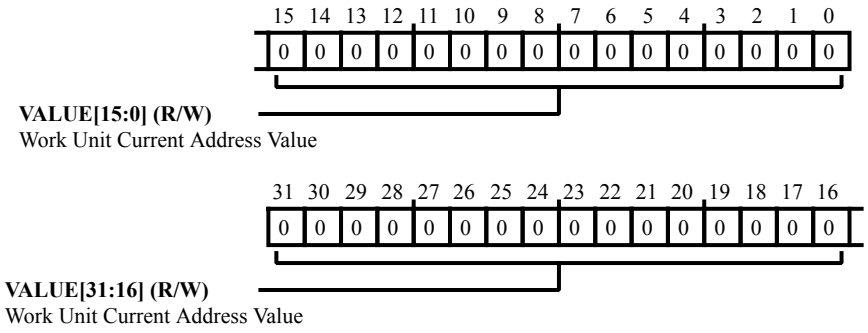

Figure 19-6: DMA_ADDR_CUR Register Diagram

Table 19-21: 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 DMA_BWLCNT register into DMA_BWLCNT_CUR 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.

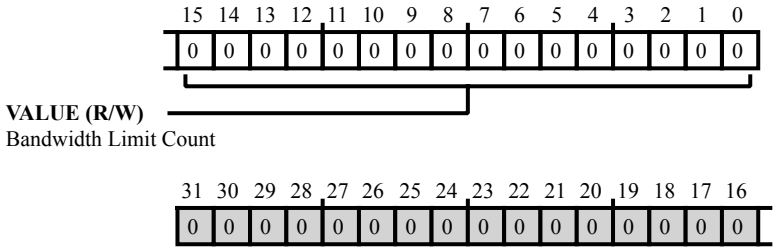

Figure 19-7: DMA_BWLCNT Register Diagram

Table 19-22: 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.

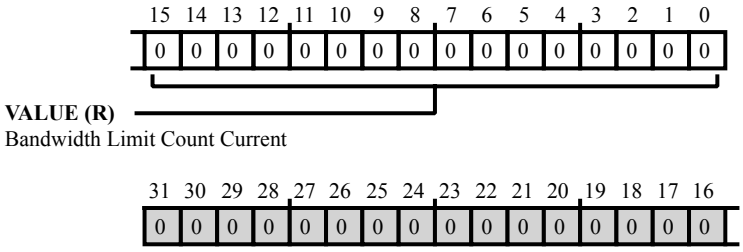

Figure 19-8: DMA_BWLCNT_CUR Register Diagram

Table 19-23: 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.

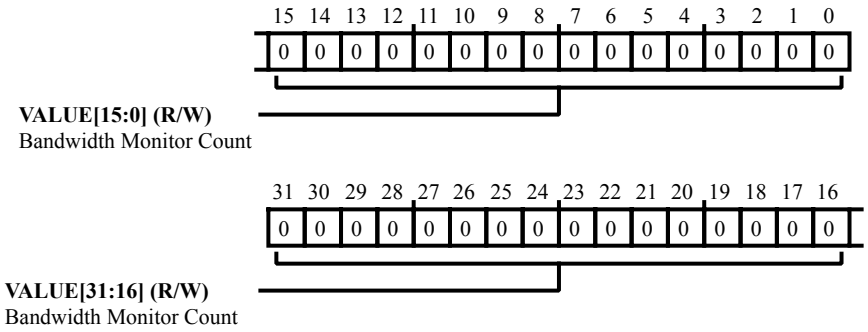

Figure 19-9: DMA_BWMCNT Register Diagram

Table 19-24: 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.

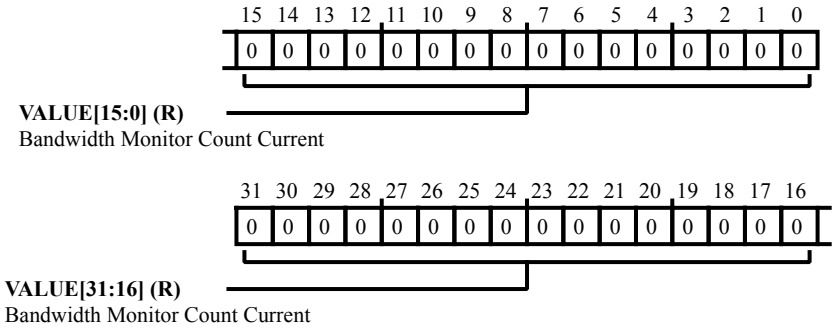

Figure 19-10: DMA_BWMCNT_CUR Register Diagram

Table 19-25: 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 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).

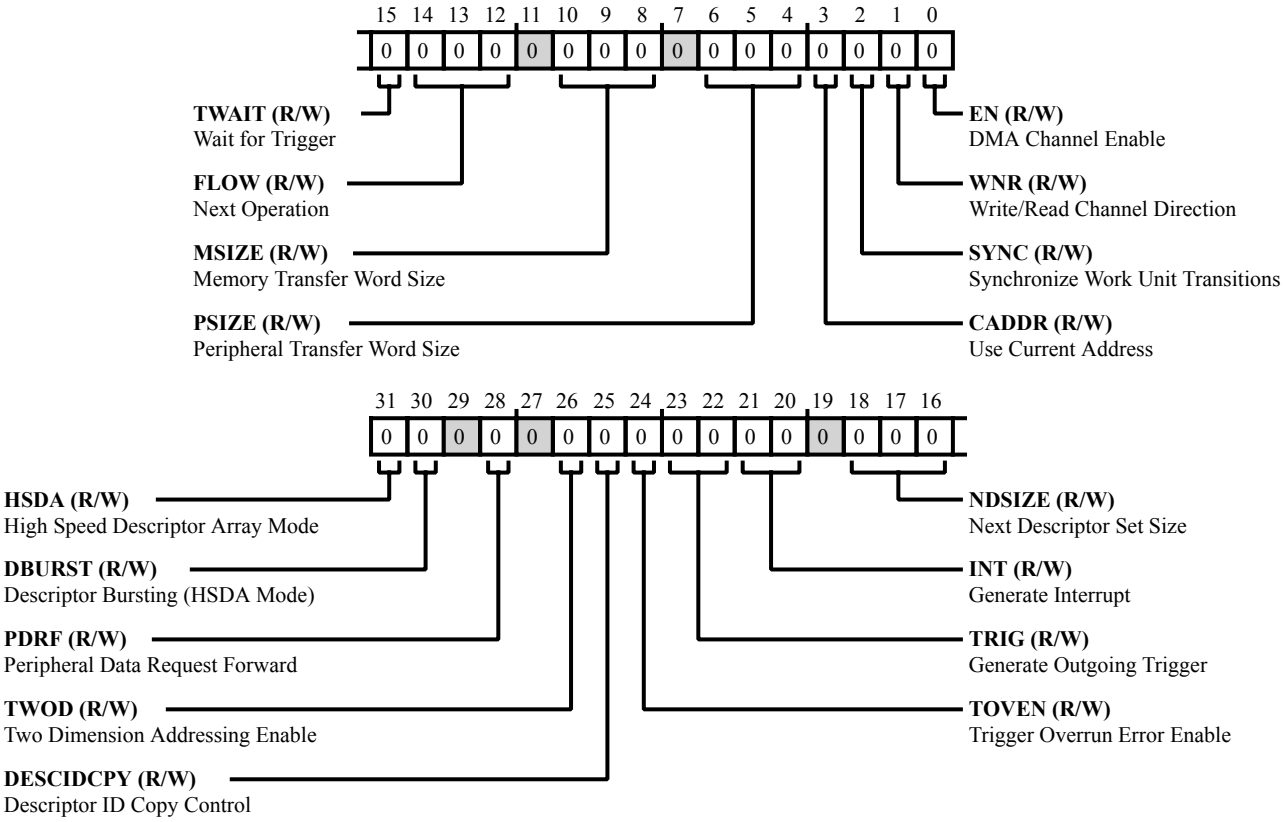

Figure 19-11: DMA_CFG Register Diagram

Table 19-26: DMA_CFG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	HSDA	High Speed Descriptor Array Mode. The DMA_CFG.HSDA defines whether the DMA incorporates features of descriptor prefetch and data pipelining across work units when working in descriptor array mode. This feature is intended to be used only in MDMAs.
30 (R/W)	DBURST	Descriptor Bursting (HSDA Mode). The DMA_CFG.DBURST defines how the DMA makes descriptor requests in High Speed Descriptor Array mode. If set =1, only bursts of 4 descriptors together are issued. If cleared =0, burst length may vary and depends on the number of locations available for descriptors in the descriptor prefetch FIFO.

Table 19-26: DMA_CFG Register Fields (Continued)

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
26 (R/W)	TWOD	Two Dimension Addressing Enable. The DMA_CFG.TWOD 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 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 19-26: 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 19-26: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21:20 (R/W)	INT	Generate Interrupt. The DMA_CFG.INT bit field selects whether an interrupt 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 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 on count completion (DMA_XCNT_CUR or DMA_YCNT_CUR reach 0) and the peripheral issues a finish command, the DMA issues an interrupt 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
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 19-26: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	TWAIT	Wait for Trigger. The DMA_CFG.TWAIT controls whether the DMA waits for an incoming trigger from another channel or user. If DMA_CFG.TWAIT 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 selects descriptor handling options.	
		0	STOP - Stop See the Stop Flow Mode section
		1	AUTO - Autobuffer See the Autobuffer Flow Mode section
		2	Reserved
		3	Reserved
		4	DSCL - Descriptor List See the Descriptor List Mode section
		5	DSCA - Descriptor Array 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 19-26: 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 19-26: 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 selects receive (write to memory) or transmit (read from memory) channel direction.	
		0	Transmit (Read from memory)
		1	Receive (Write to memory)

Table 19-26: DMA_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Channel Enable. The DMA_CFG.EN 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 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.

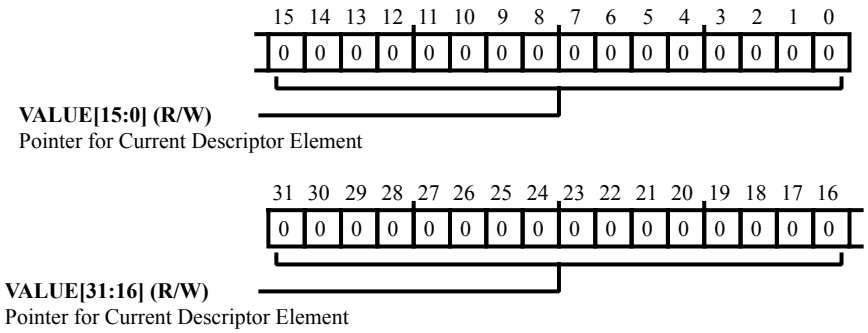

Figure 19-12: DMA_DSCPTR_CUR Register Diagram

Table 19-27: 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.

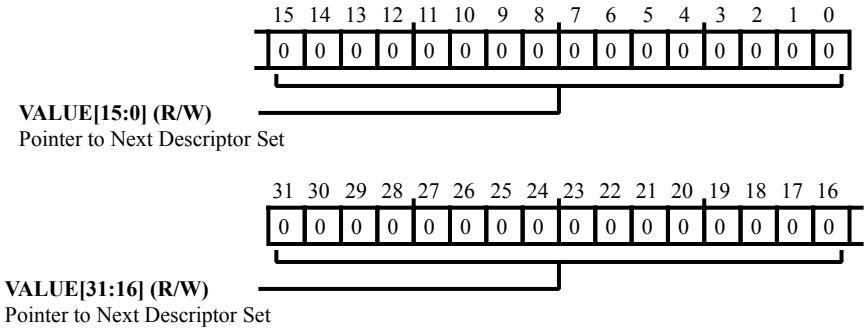

Figure 19-13: DMA_DSCPTR_NXT Register Diagram

Table 19-28: 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.DESCIDCPY is 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.

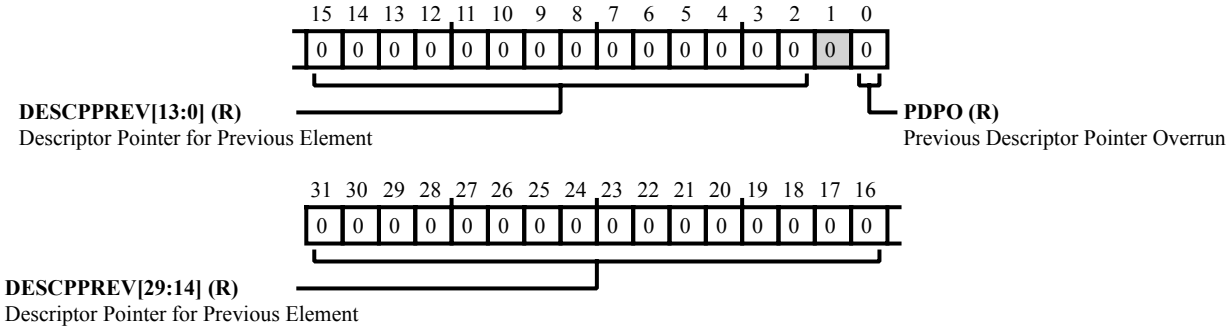

Figure 19-14: DMA_DSCPTR_PRV Register Diagram

Table 19-29: 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.

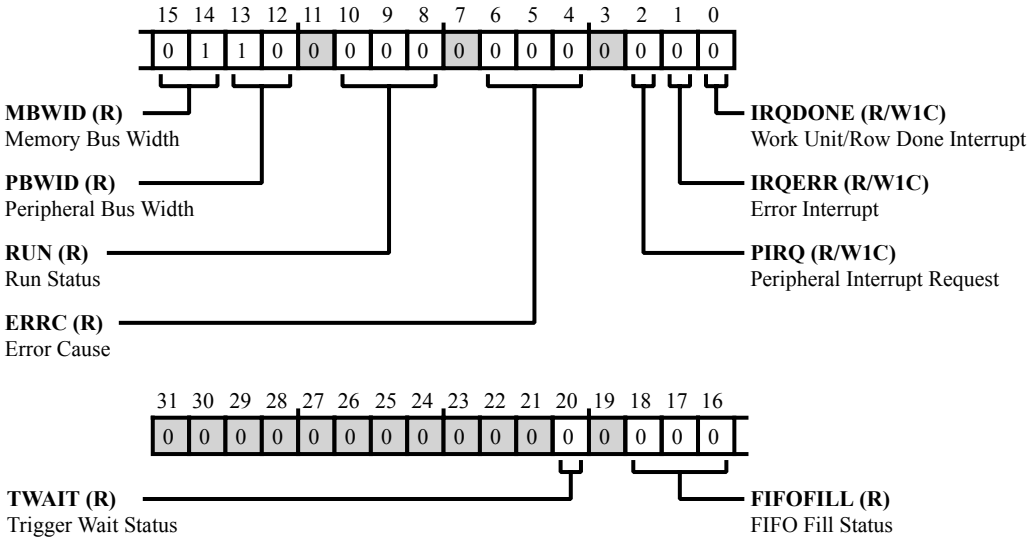

Figure 19-15: DMA_STAT Register Diagram

Table 19-30: DMA_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
20 (R/NW)	TWAIT	Trigger Wait Status. The DMA_STAT.TWAIT 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 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 19-30: 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 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 DMA_STAT.ERRC to identify the type of error. For more information, see the errors section of the DMA functional description.	

Table 19-30: 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. Programs can use the DMA_STAT.PIRQ bit status to help determine which DMA asserted the interrupt. This bit also helps to distinguish between an interrupt caused by the state of the work unit and an interrupt caused by the peripheral.	
		0	No Interrupt
		1	Interrupt Signaled by Peripheral
1 (R/W1C)	IRQERR	Error Interrupt. The DMA_STAT.IRQERR 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 DMA_STAT.IRQERR to help determine which DMA issued the error interrupt. The DMA_STAT.IRQERR does not clear a transition of the DMA_CFG.EN bit from 0 to 1. Clear the DMA_STAT.IRQERR 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 indicates the DMA has detected the completion of a work unit or row (inner loop count) and has issued an interrupt. Programs can use the DMA_STAT.IRQDONE status to help determine which DMA asserted the interrupt. Programs can also use these bits to help distinguish between an interrupt based on the state of the work unit and an interrupt 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.

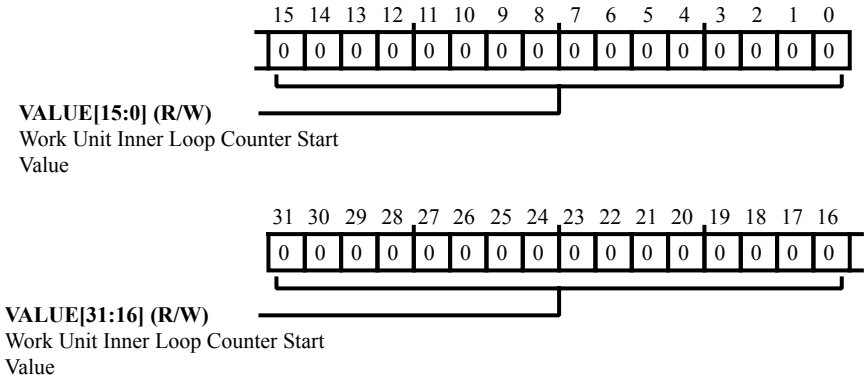

Figure 19-16: DMA_XCNT Register Diagram

Table 19-31: 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.

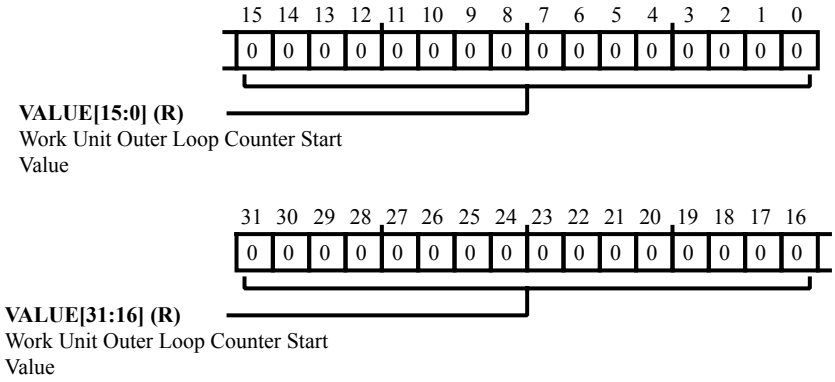

Figure 19-17: DMA_XCNT_CUR Register Diagram

Table 19-32: 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.

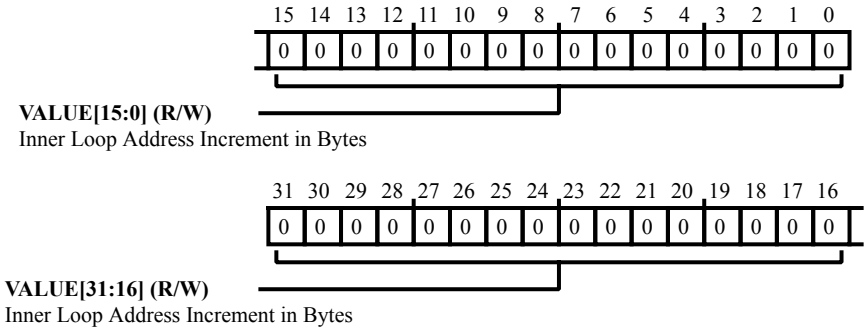

Figure 19-18: DMA_XMOD Register Diagram

Table 19-33: 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_YCNTregister 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.

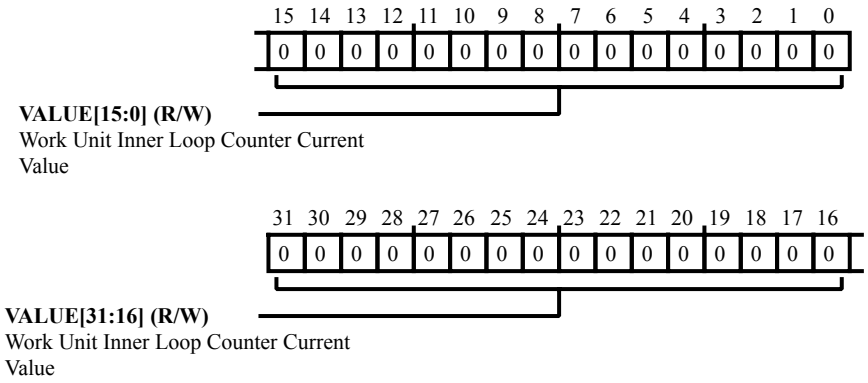

Figure 19-19: DMA_YCNT Register Diagram

Table 19-34: 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.

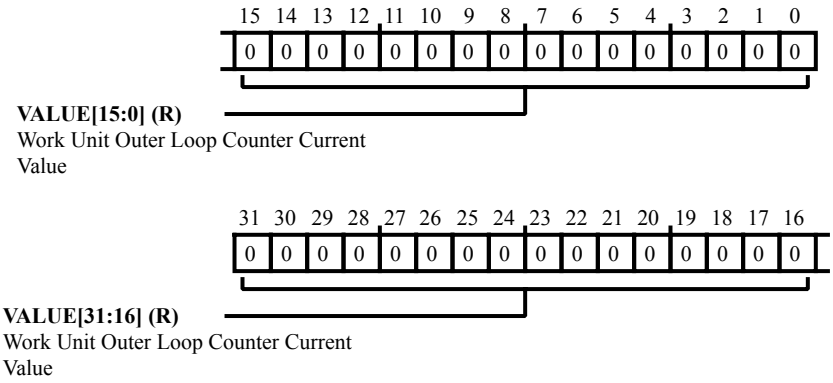

Figure 19-20: DMA_YCNT_CUR Register Diagram

Table 19-35: 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.

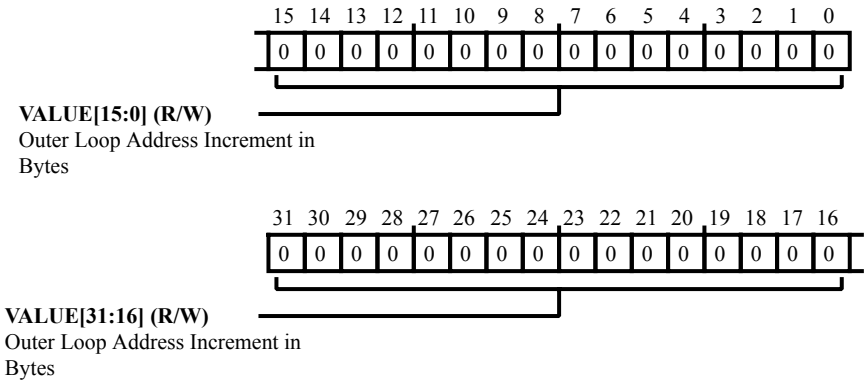

Figure 19-21: DMA_YMOD Register Diagram

Table 19-36: 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.

20 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 interrupts

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.

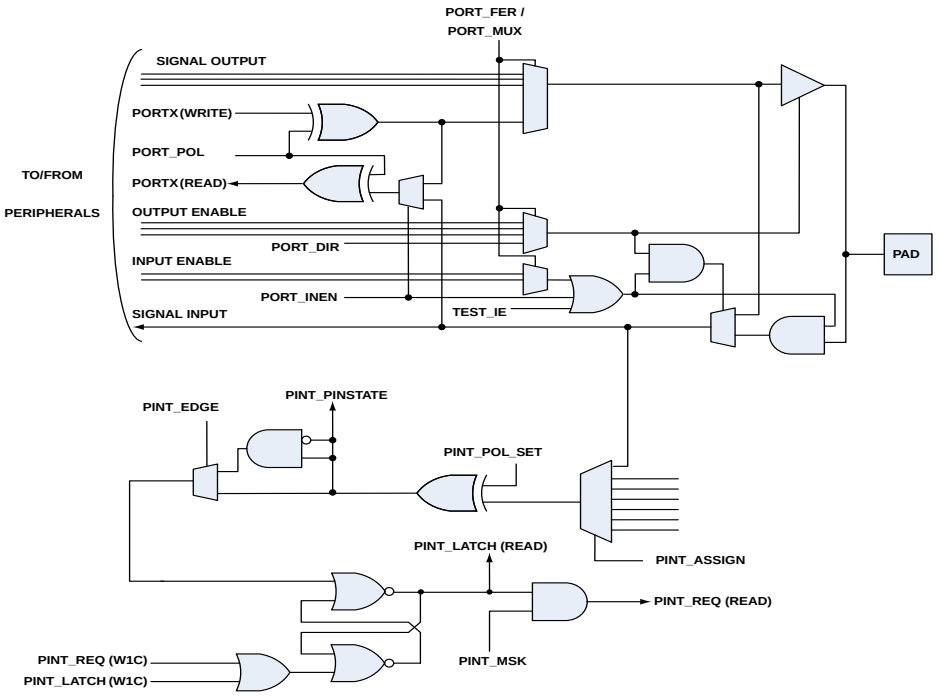

Figure 20-1: Simplified GPIO and Pin Interrupt Signal Flow

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 functionality
- Byte-wide pin-to-interrupt 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.

ADSP-BF70x 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 20-1: ADSP-BF70x 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

ADSP-BF70x 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 20-2: ADSP-BF70x 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

ADSP-BF70x PINT Interrupt List

Table 20-3: ADSP-BF70x PINT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
20	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level	
21	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level	
22	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level	

ADSP-BF70x PINT Trigger List

Table 20-4: ADSP-BF70x PINT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
10	PINT0_BLOCK	PINT0 Pin Interrupt Block	Level
11	PINT1_BLOCK	PINT1 Pin Interrupt Block	Level
12	PINT2_BLOCK	PINT2 Pin Interrupt Block	Level

Table 20-5: ADSP-BF70x PINT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

ADSP-BF70x PADS Register List

The PADS controls signal hysteresis and other system interface signal features for a number of module interfaces.

Table 20-6: ADSP-BF70x PADS Register List

Name	Description
PADS_PCFG0	Peripheral PAD Configuration0 Register

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 Functionality
- Port Multiplexing Control

Internal Interfaces

All of the pin multiplexing, GPIO, and pin interrupt control block MMRs can be accessed through the MMR bus. There is no DMA support. Each of the pin interrupt (PINTx) modules has its own dedicated interrupt request output signal that connects directly to the system event controller (SEC).

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 Functionality

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.
- Interrupts 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 and PORTB
- PINT1 can sense pin activity on PORTB and PORTC
- PINT2 can sense pin activity on PORTC and PORTA

The processor supports both 32-bit and 16-bit peripheral bus accesses to PINTx registers.

Pins connect to the PINTx module and then to the system event controller (SEC), as shown in the ***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 generation is level-sensitive, and an interrupt occurs when the enabled pin is sensed as active high. Use the PINT_EDGE_SET register to change the interrupt genearion scheme to instead be edge-sensitive (rising edge generates the interrupt). Use the PINT_INV_SET register to invert the polarity such that the PINTx block generates the interrupt on active-low signals or falling edges.

The PINTx modules also assist when both signal edges must generate unique interrupts. 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 for each signal edge. When both signal edges can be serviced by the same interrupt, 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. 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, an interrupt is forwarded to the SEC.

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 appicable, to effect changes on a pin-by-pin basis rather than on the full port.

For more detailed descriptions of the configurations, see PORT Register Descriptions.

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.

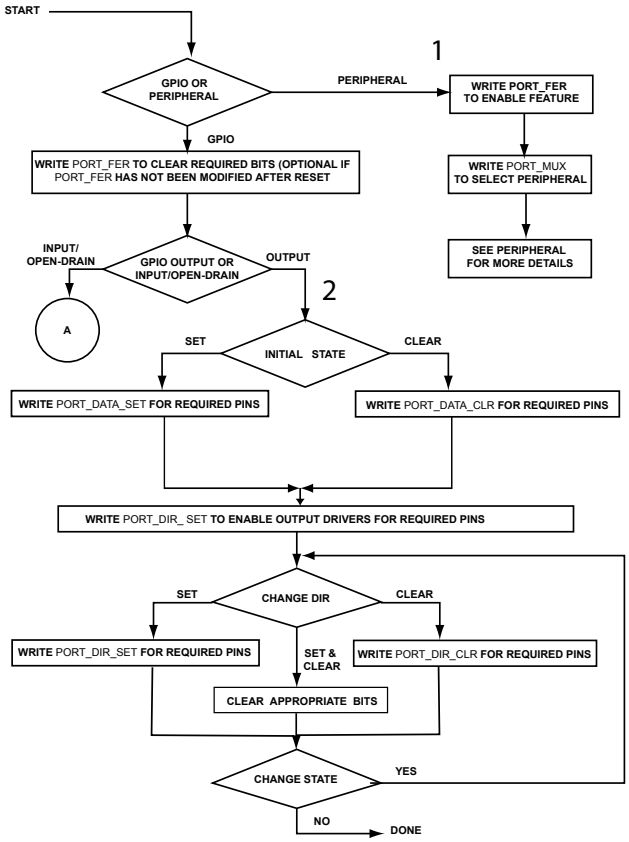

Figure 20-2: GPIO Programming Model Flow (Part 1)

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 interrupts 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.

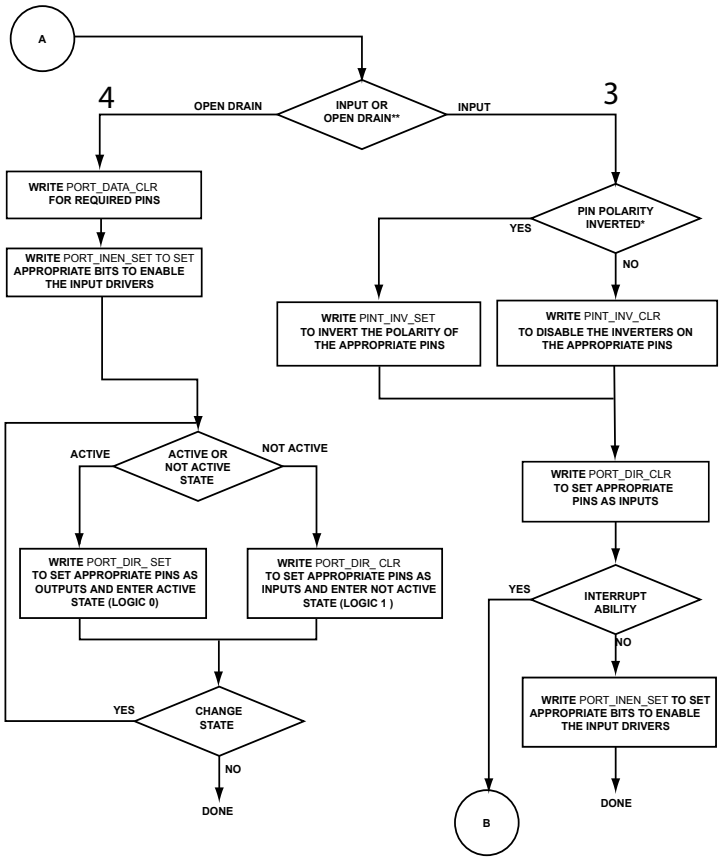

Figure 20-3: GPIO Programming Model Flow (Part 2)

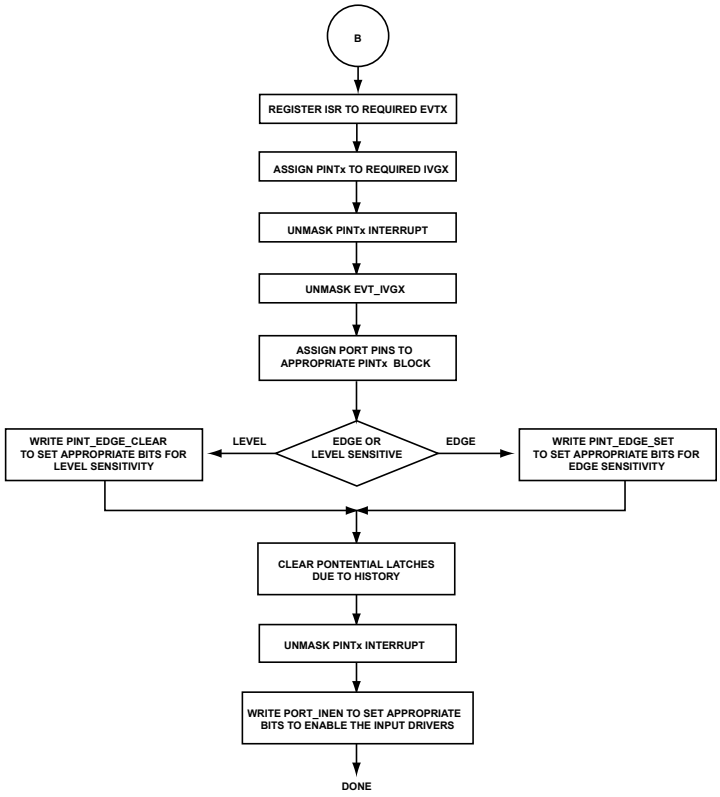

Figure 20-4: GPIO Programming Model Flow (Part 3)

ADSP-BF70x PORT Register Descriptions

The General-Purpose Input/Output Port (PORT) contains the following registers.

Table 20-7: ADSP-BF70x 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

Table 20-7: ADSP-BF70x PORT Register List (Continued)

Name	Description
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.

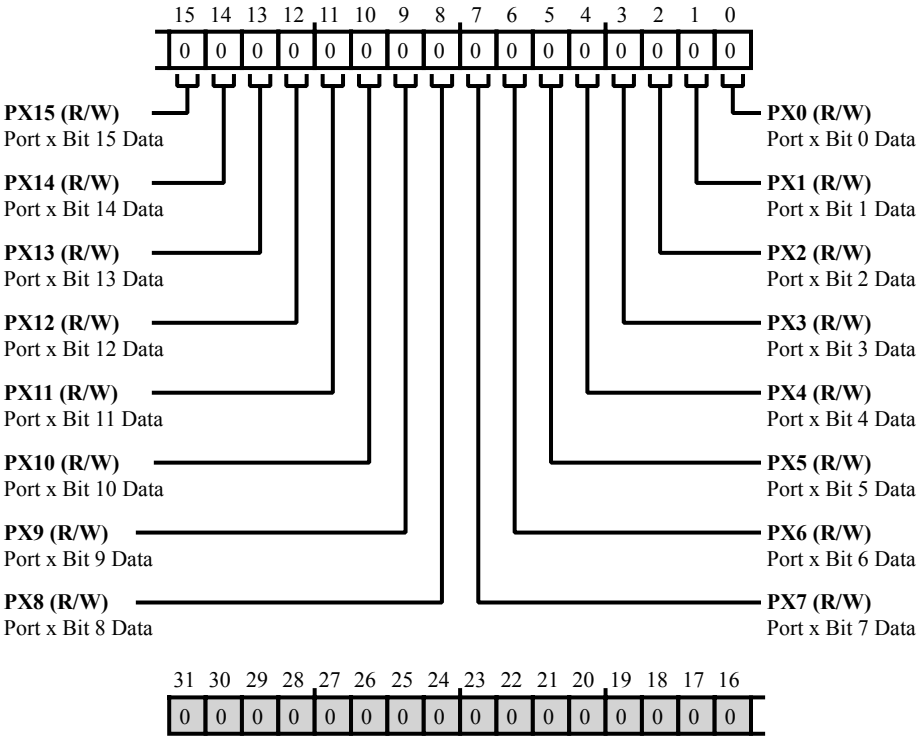

Figure 20-5: PORT_DATA Register Diagram

Table 20-8: 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 20-8: 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 20-8: 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.

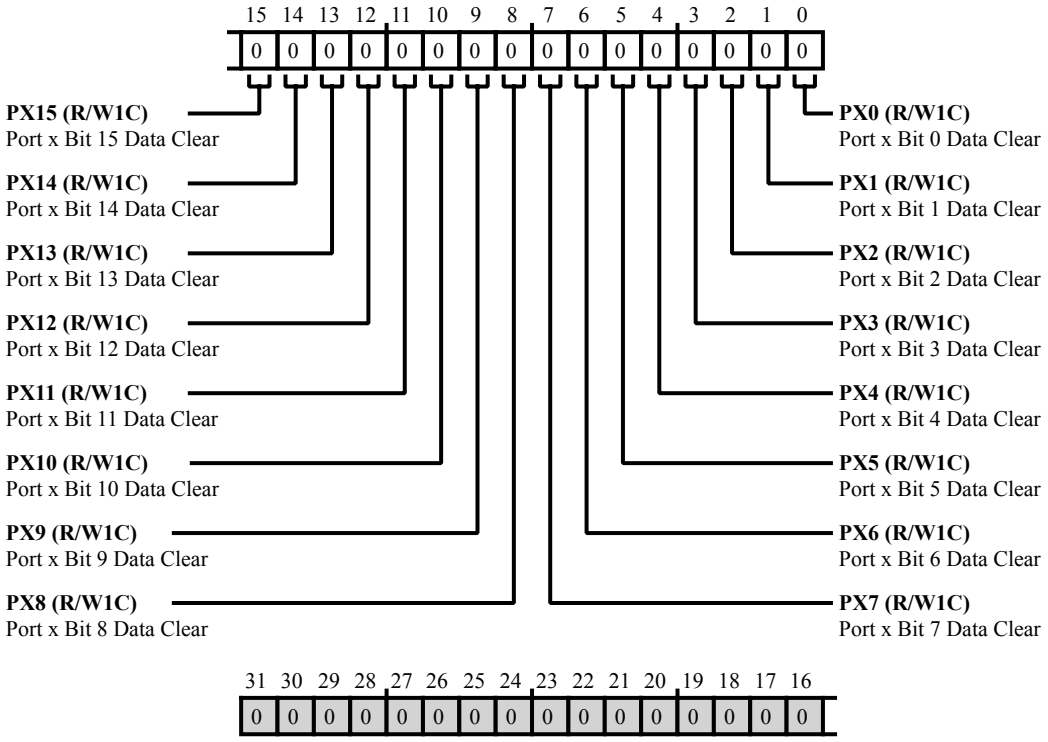

Figure 20-6: PORT_DATA_CLR Register Diagram

Table 20-9: PORT_DATA_CLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W1C)	PX15	Port x Bit 15 Data Clear.	
		0	No Effect
		1	Clear Bit. Write 1 for signal low in output mode.
14 (R/W1C)	PX14	Port x Bit 14 Data Clear.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
13 (R/W1C)	PX13	Port x Bit 13 Data Clear.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.

Table 20-9: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W1C)	PX12	Port x Bit 12 Data Clear.	
		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.	
		0	No Effect
		1	Clear Bit. Write 1 for signal low in output mode.
10 (R/W1C)	PX10	Port x Bit 10 Data Clear.	
		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.	
		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.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.
7 (R/W1C)	PX7	Port x Bit 7 Data Clear.	
		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.	
		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.	
		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.	
		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.	
		0	No Effect. Write 0 has no effect in output mode.
		1	Clear Bit. Write 1 for signal low in output mode.

Table 20-9: PORT_DATA_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W1C)	PX2	Port x Bit 2 Data Clear.	
		0	No Effect Write 0 has no effect in output mode.
		1	Clear Bit Write 1 for signal low in output mode.
1 (R/W1C)	PX1	Port x Bit 1 Data Clear.	
		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.

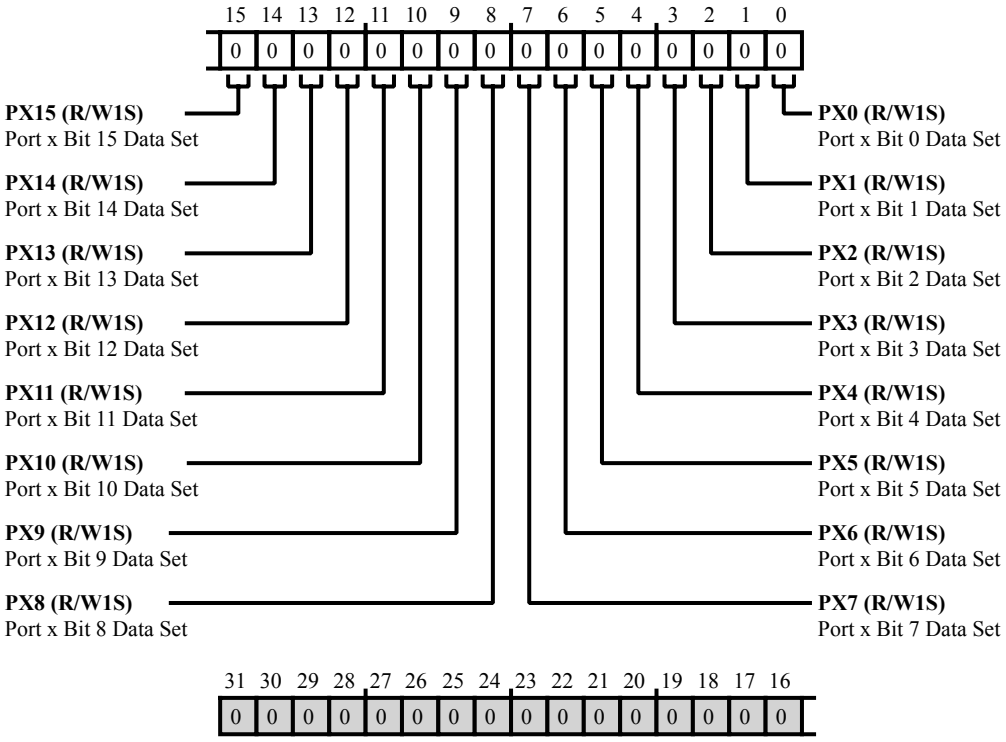

Figure 20-7: PORT_DATA_SET Register Diagram

Table 20-10: 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 20-10: 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 20-10: 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.

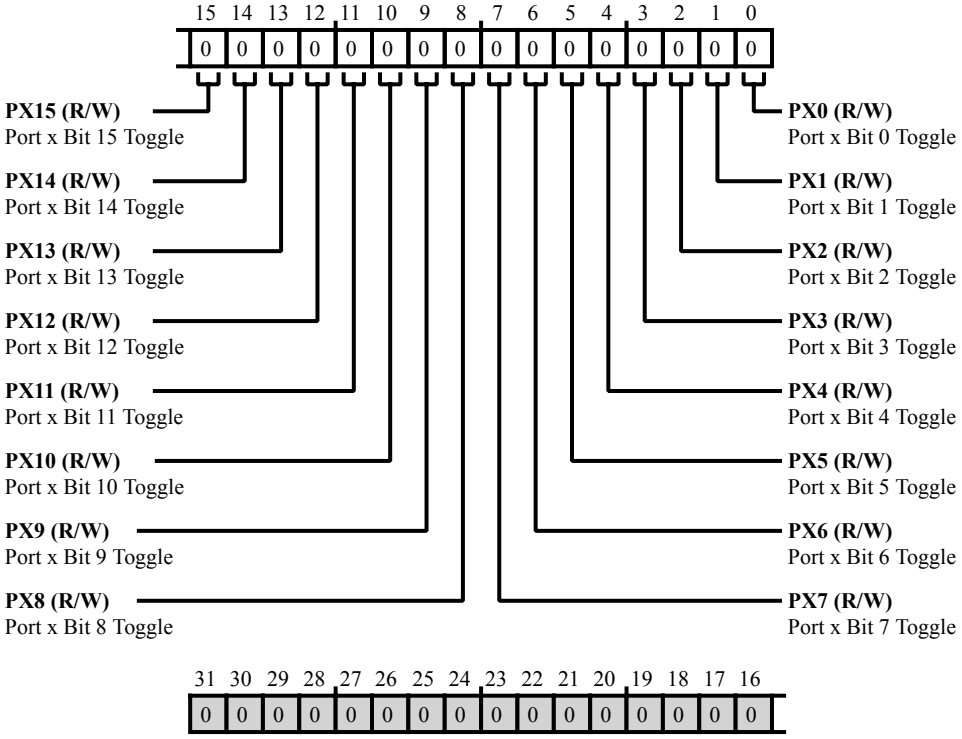

Figure 20-8: PORT_DATA_TGL Register Diagram

Table 20-11: 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 20-11: 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 20-11: 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.

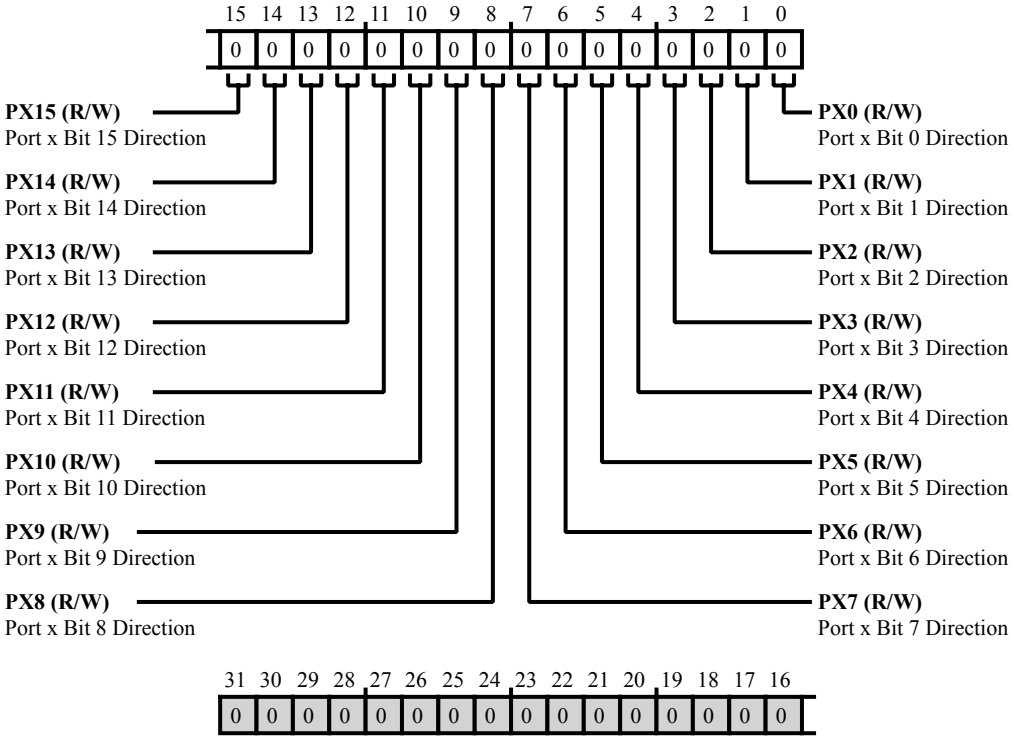

Figure 20-9: PORT_DIR Register Diagram

Table 20-12: 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 20-12: 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 20-12: 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.

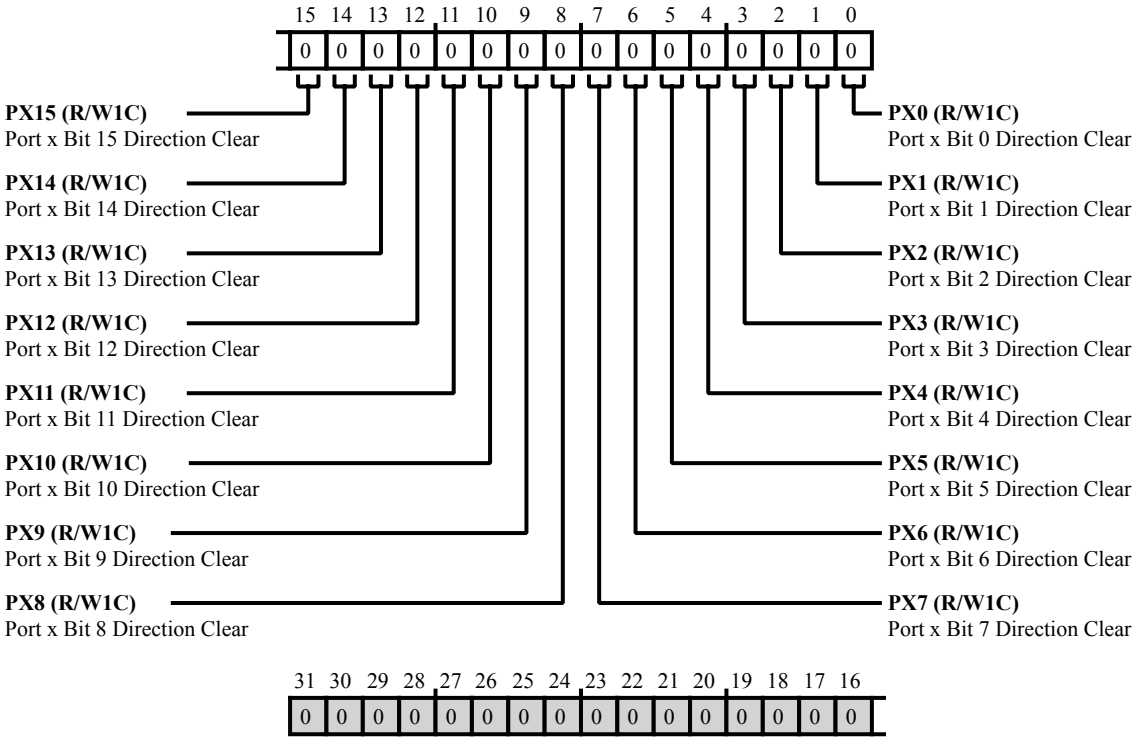

Figure 20-10: PORT_DIR_CLR Register Diagram

Table 20-13: 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 20-13: 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
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

Table 20-13: PORT_DIR_CLR Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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.

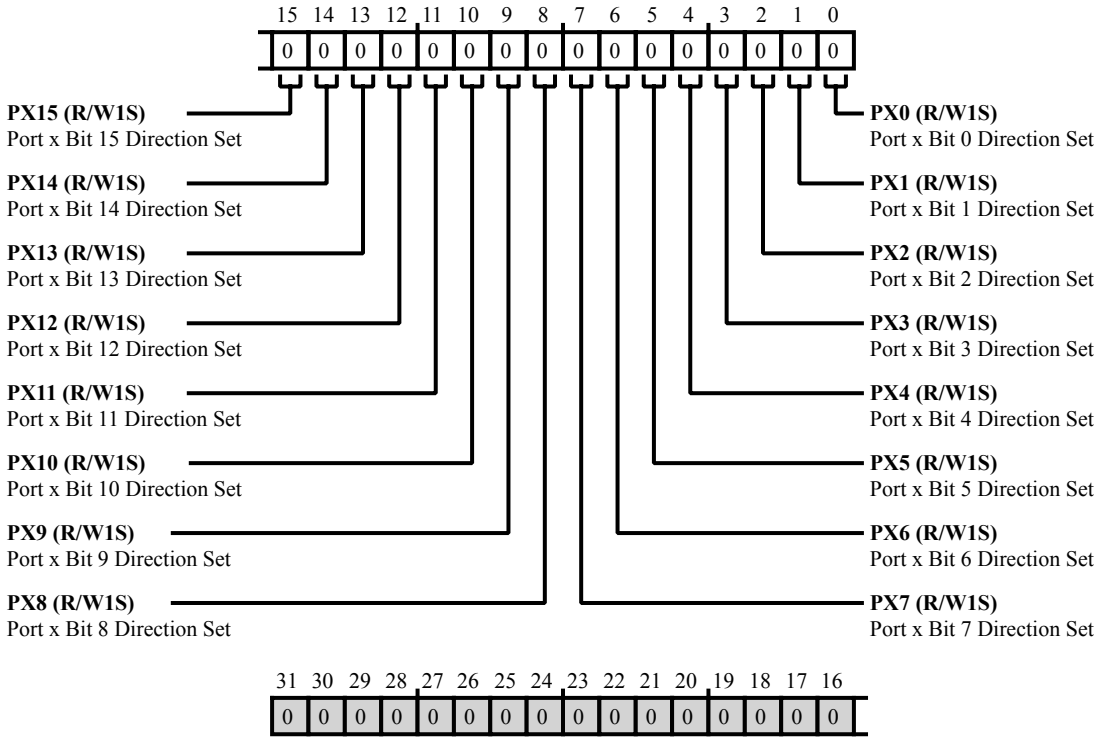

Figure 20-11: PORT_DIR_SET Register Diagram

Table 20-14: 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 20-14: 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 20-14: 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.

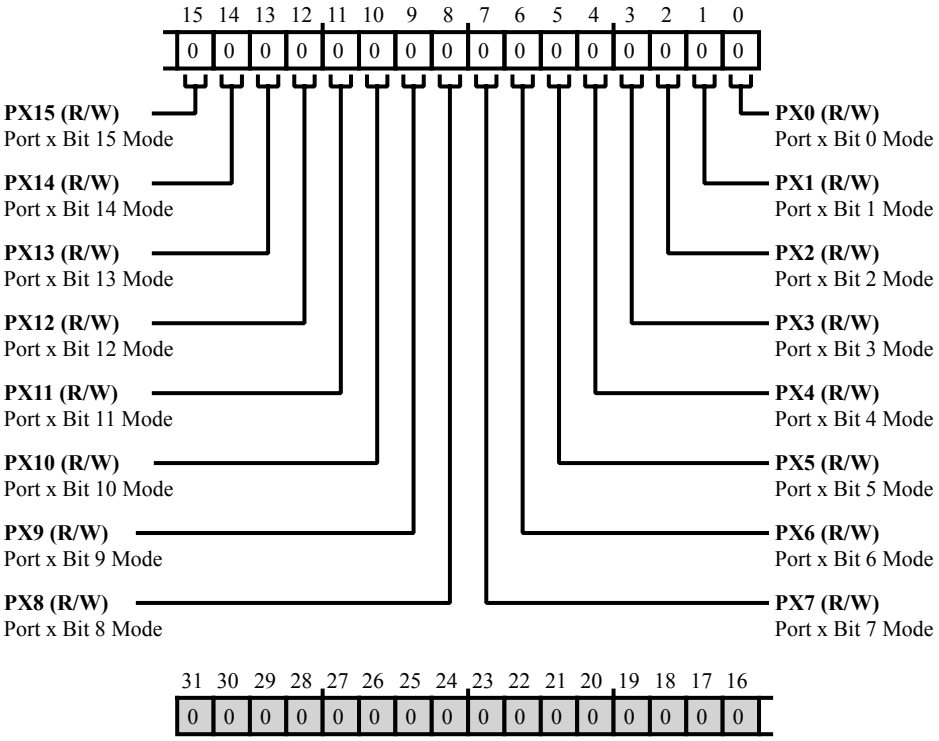

Figure 20-12: PORT_FER Register Diagram

Table 20-15: 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 20-15: 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 20-15: 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.

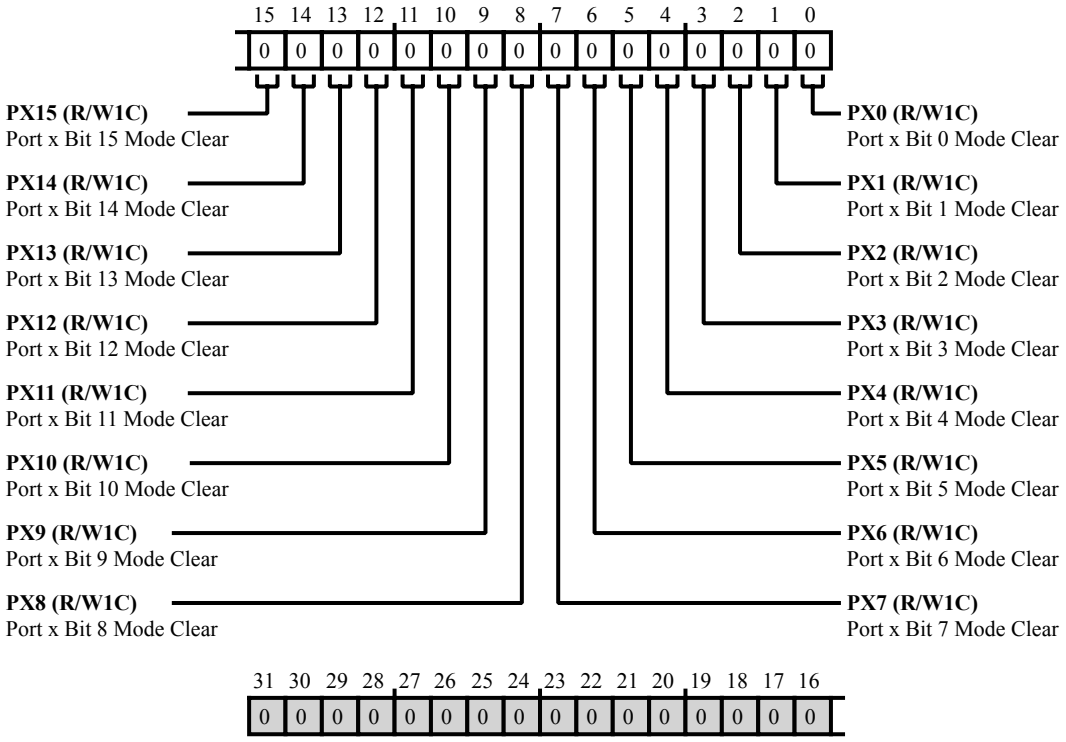

Figure 20-13: PORT_FER_CLR Register Diagram

Table 20-16: 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 20-16: 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 20-16: 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.

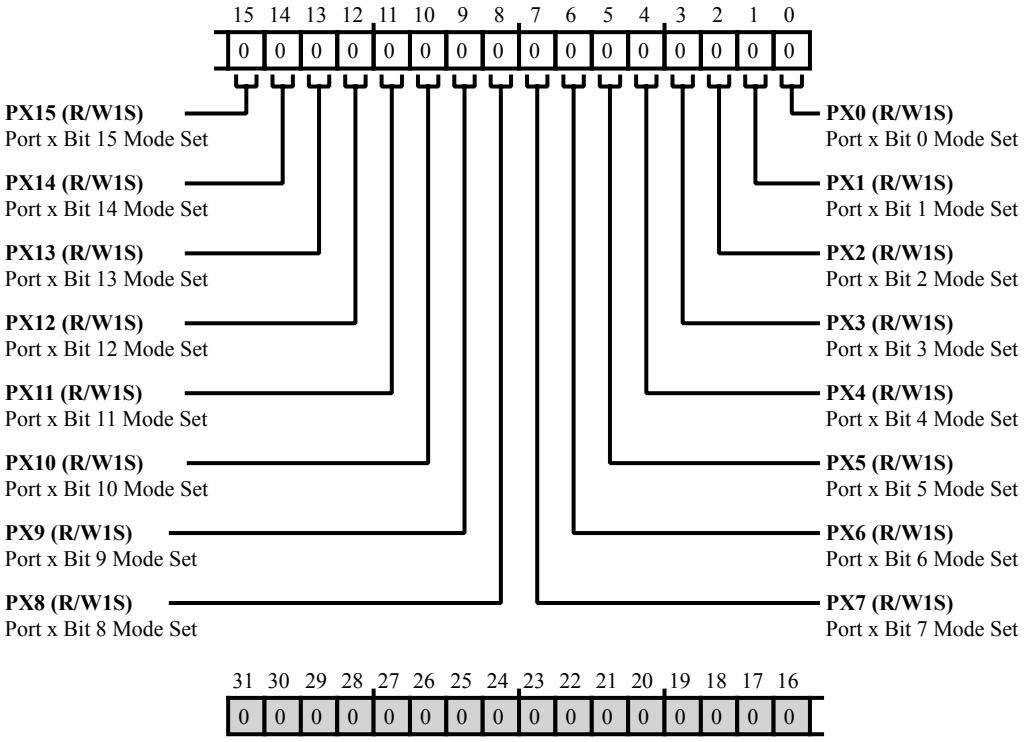

Figure 20-14: PORT_FER_SET Register Diagram

Table 20-17: 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 20-17: 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 20-17: 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.

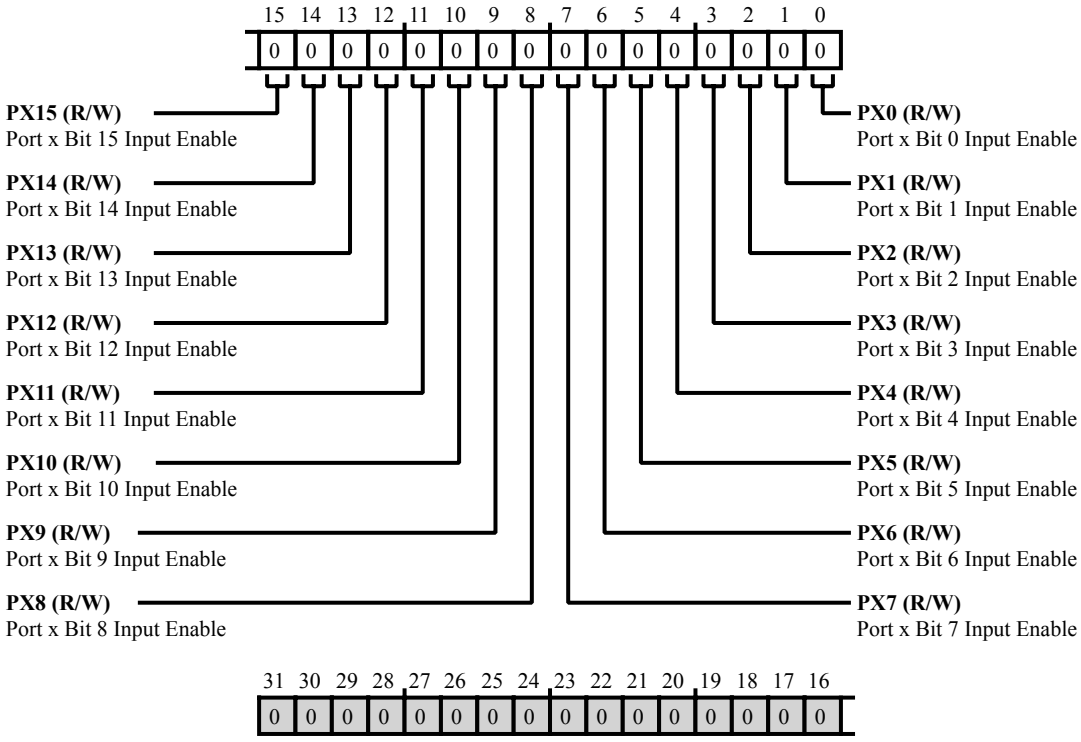

Figure 20-15: PORT_INEN Register Diagram

Table 20-18: 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 20-18: 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 20-18: 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.

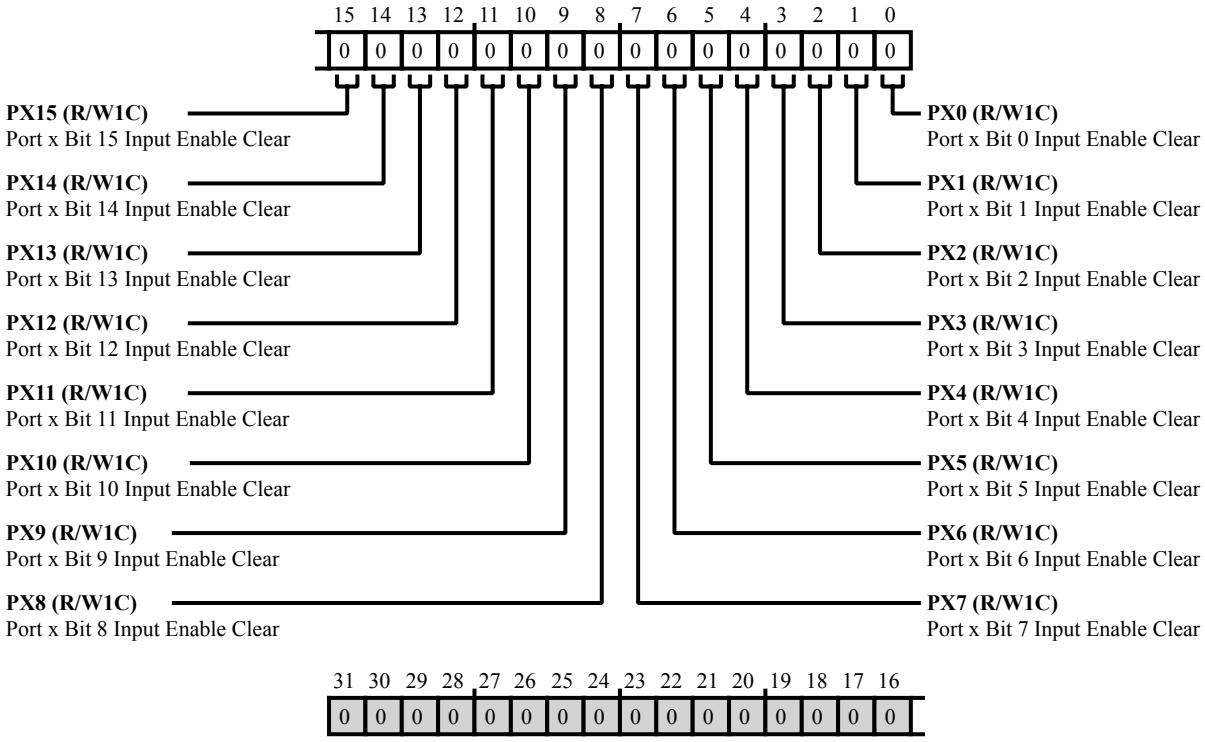

Figure 20-16: PORT_INEN_CLR Register Diagram

Table 20-19: 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 20-19: 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 20-19: 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.

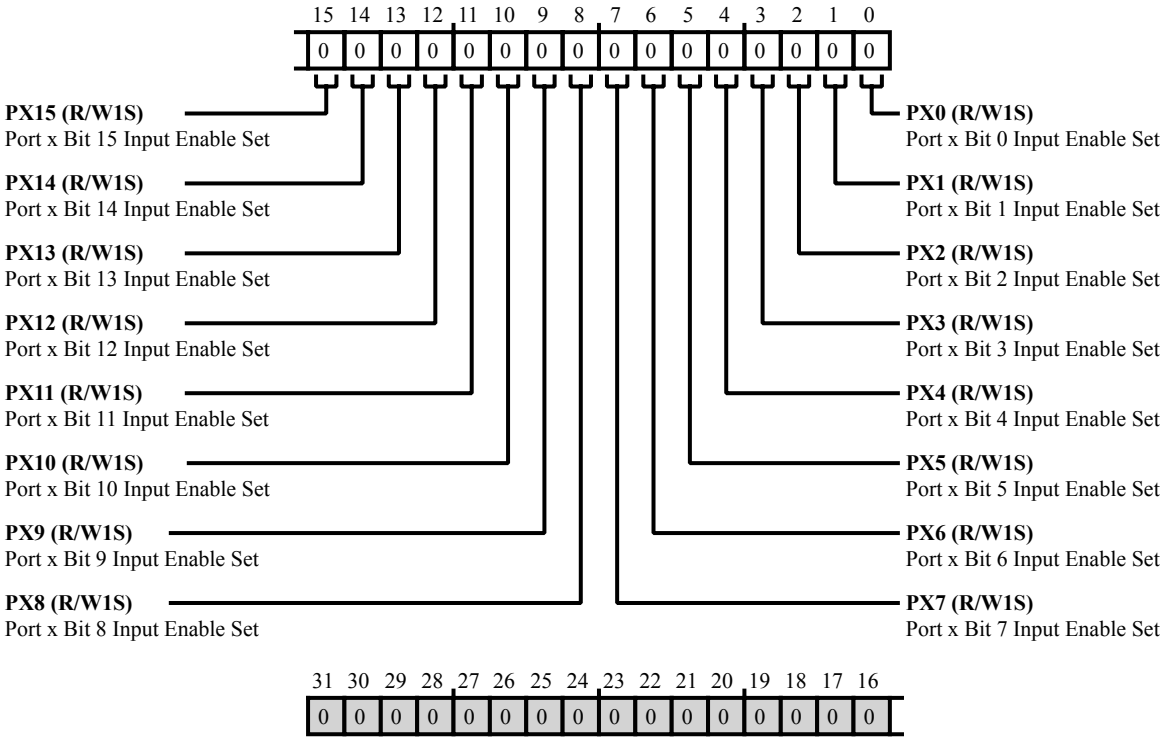

Figure 20-17: PORT_INEN_SET Register Diagram

Table 20-20: 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 20-20: 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 20-20: 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.

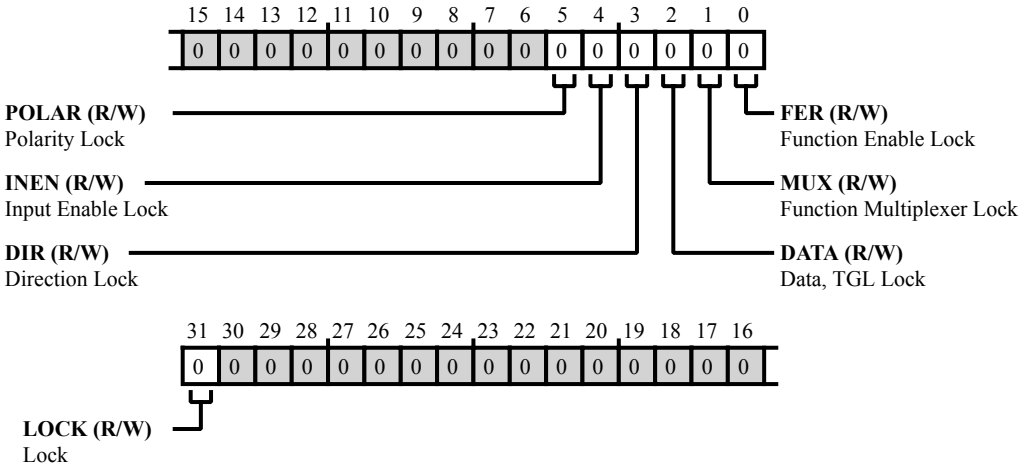

Figure 20-18: PORT_LOCK Register Diagram

Table 20-21: 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 20-21: 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	Lock DIR
		1	Unlock 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.

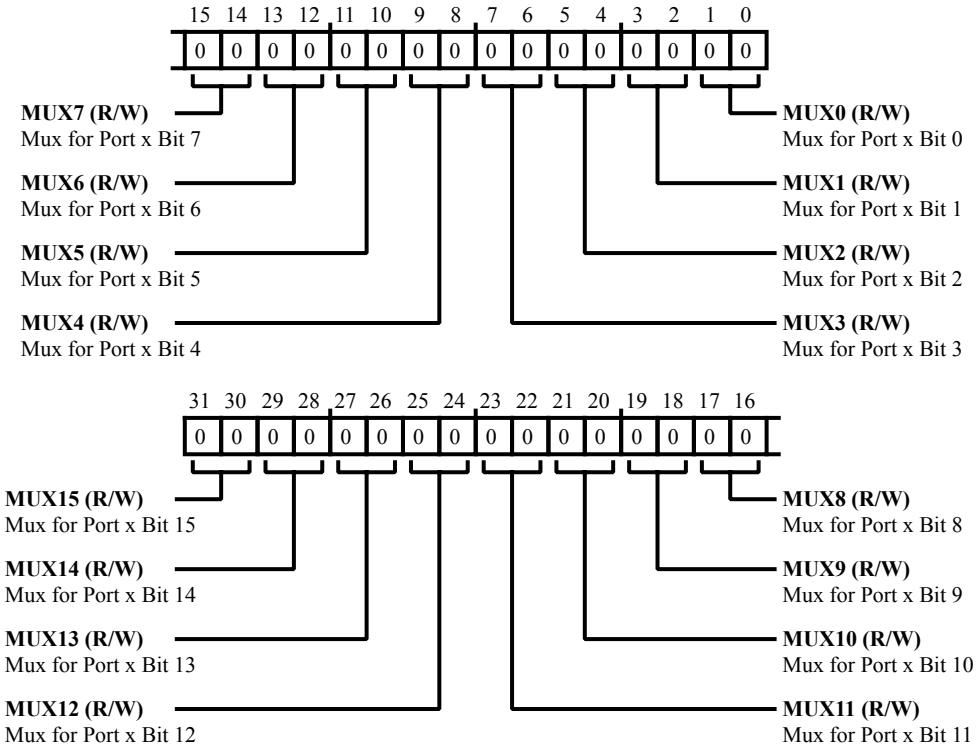

Figure 20-19: PORT_MUX Register Diagram

Table 20-22: 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.
29:28	MUX14	Mux for Port x Bit 14.

Table 20-22: PORT_MUX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		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.

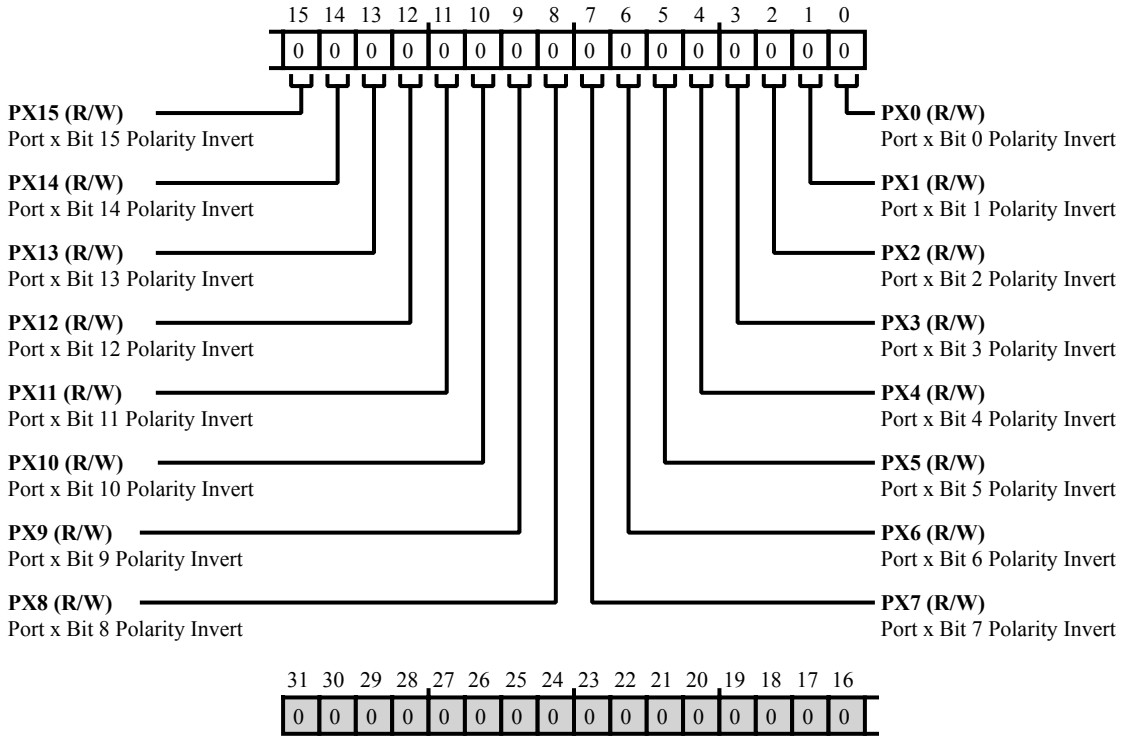

Figure 20-20: PORT_POL Register Diagram

Table 20-23: 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 20-23: 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 20-23: 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 20-23: 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.

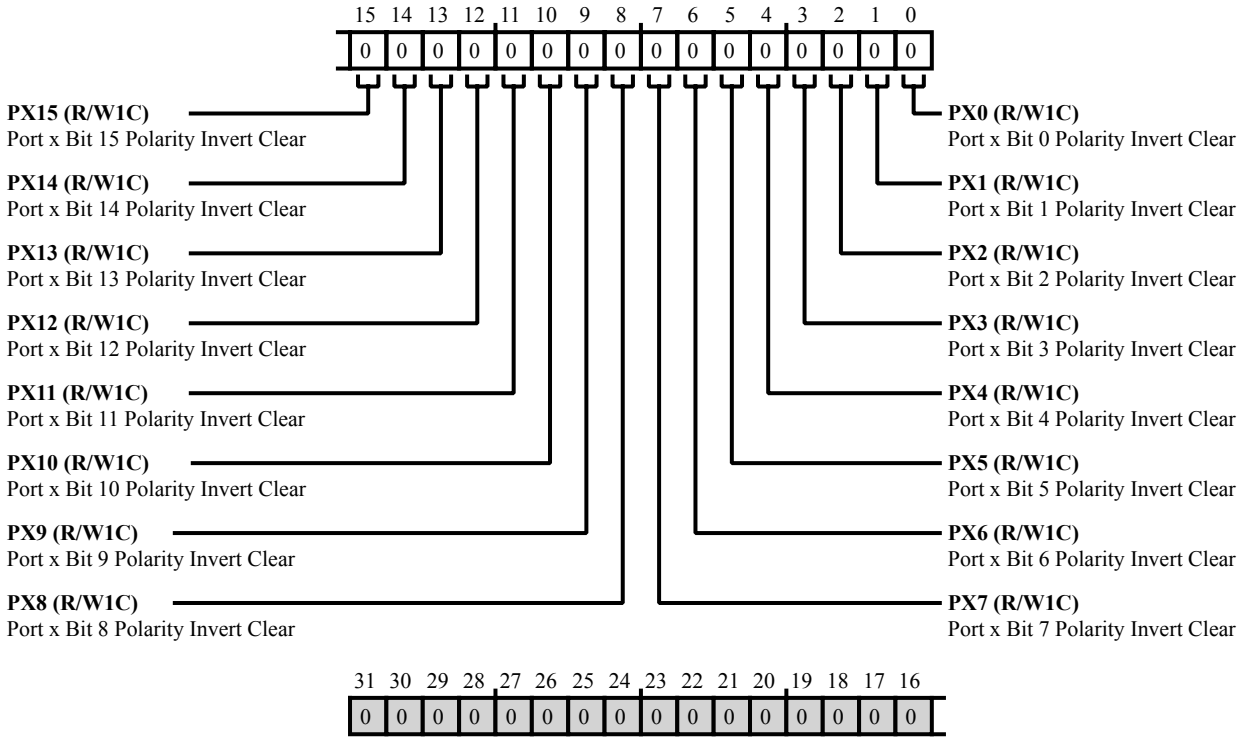

Figure 20-21: PORT_POL_CLR Register Diagram

Table 20-24: 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 20-24: 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 20-24: 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.

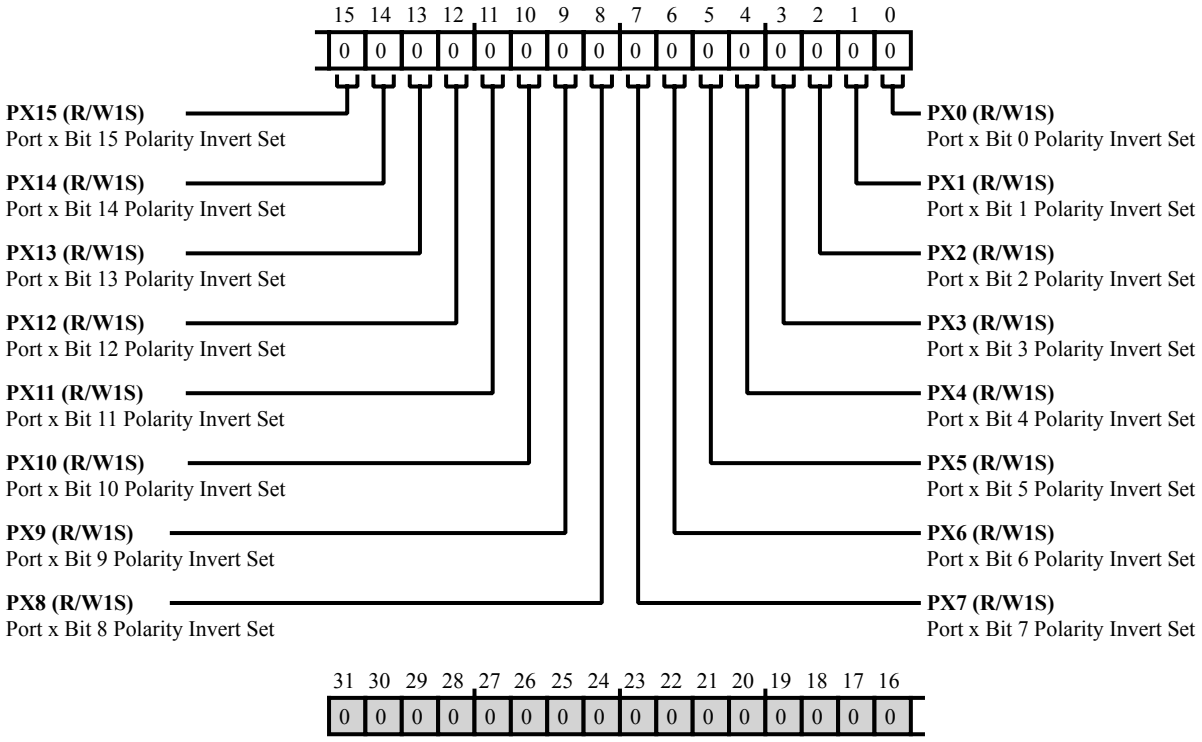

Figure 20-22: PORT_POL_SET Register Diagram

Table 20-25: 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 20-25: 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 20-25: 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.

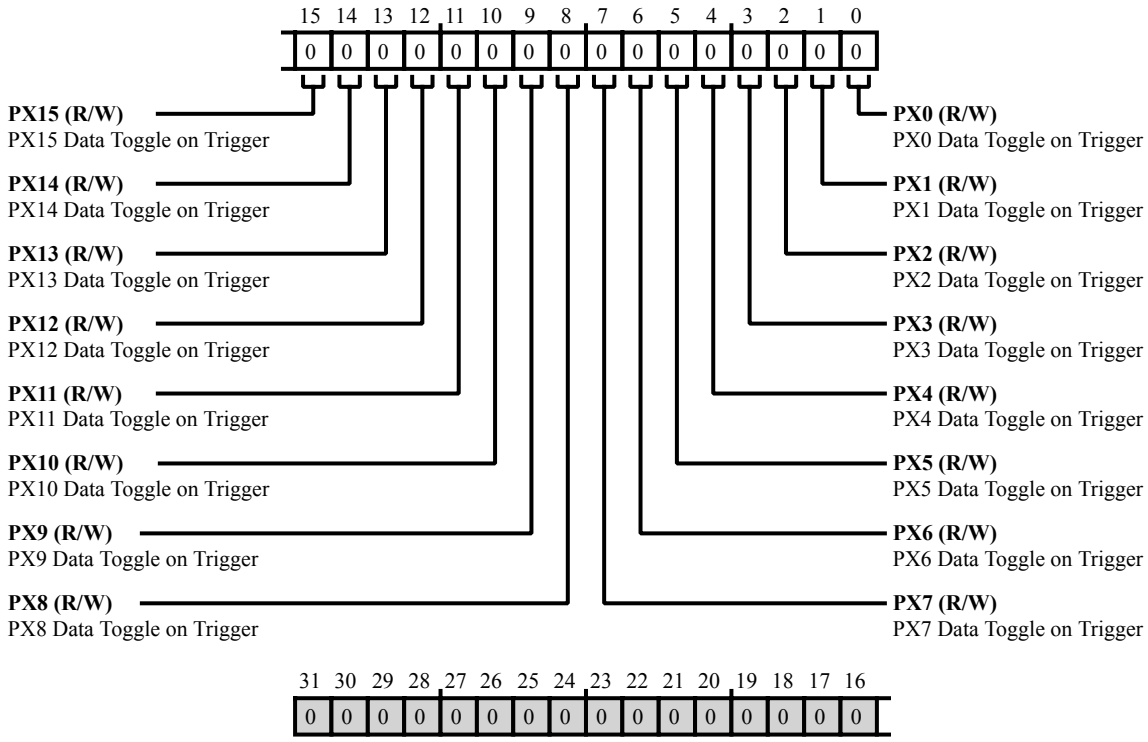

Figure 20-23: PORT_TRIG_TGL Register Diagram

Table 20-26: 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 20-26: 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.

ADSP-BF70x PINT Register Descriptions

The Pin Interrupt module (PINT) contains the following registers.

Table 20-27: ADSP-BF70x 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 20-27: ADSP-BF70x 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 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.

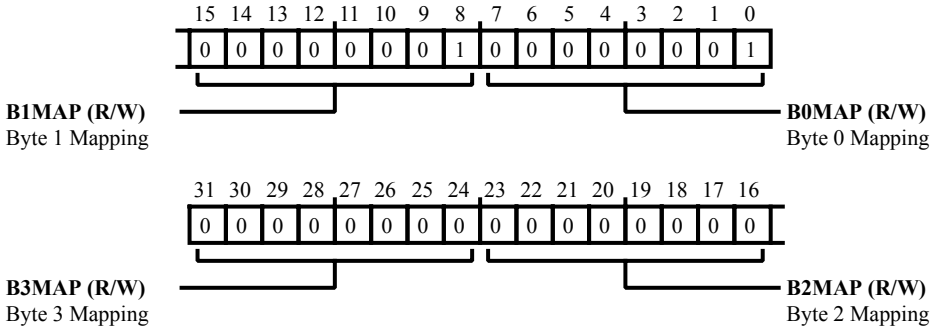

Figure 20-24: PINT_ASSIGN Register Diagram

Table 20-28: 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.

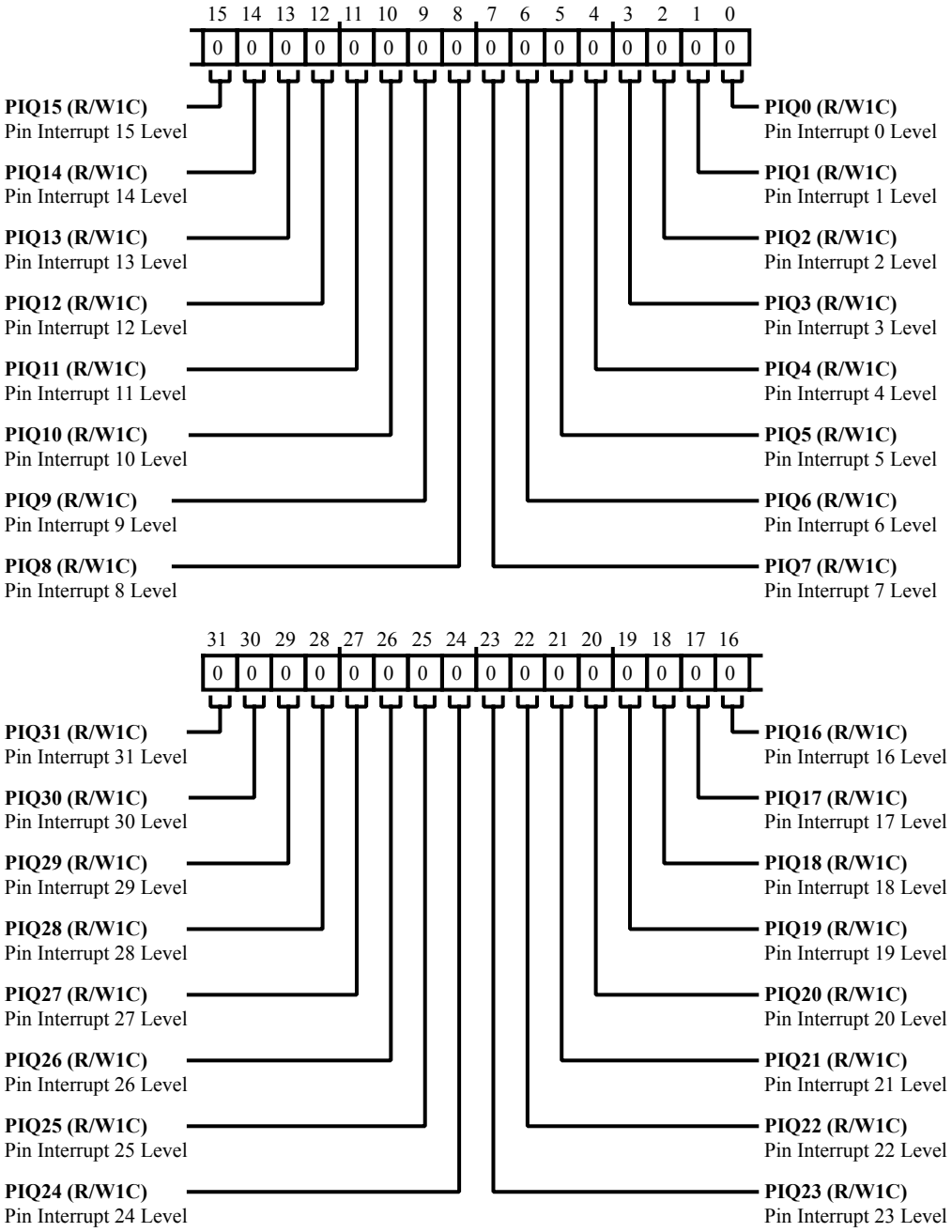

Figure 20-25: PINT_EDGE_CLR Register Diagram

Table 20-29: 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 20-29: 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.

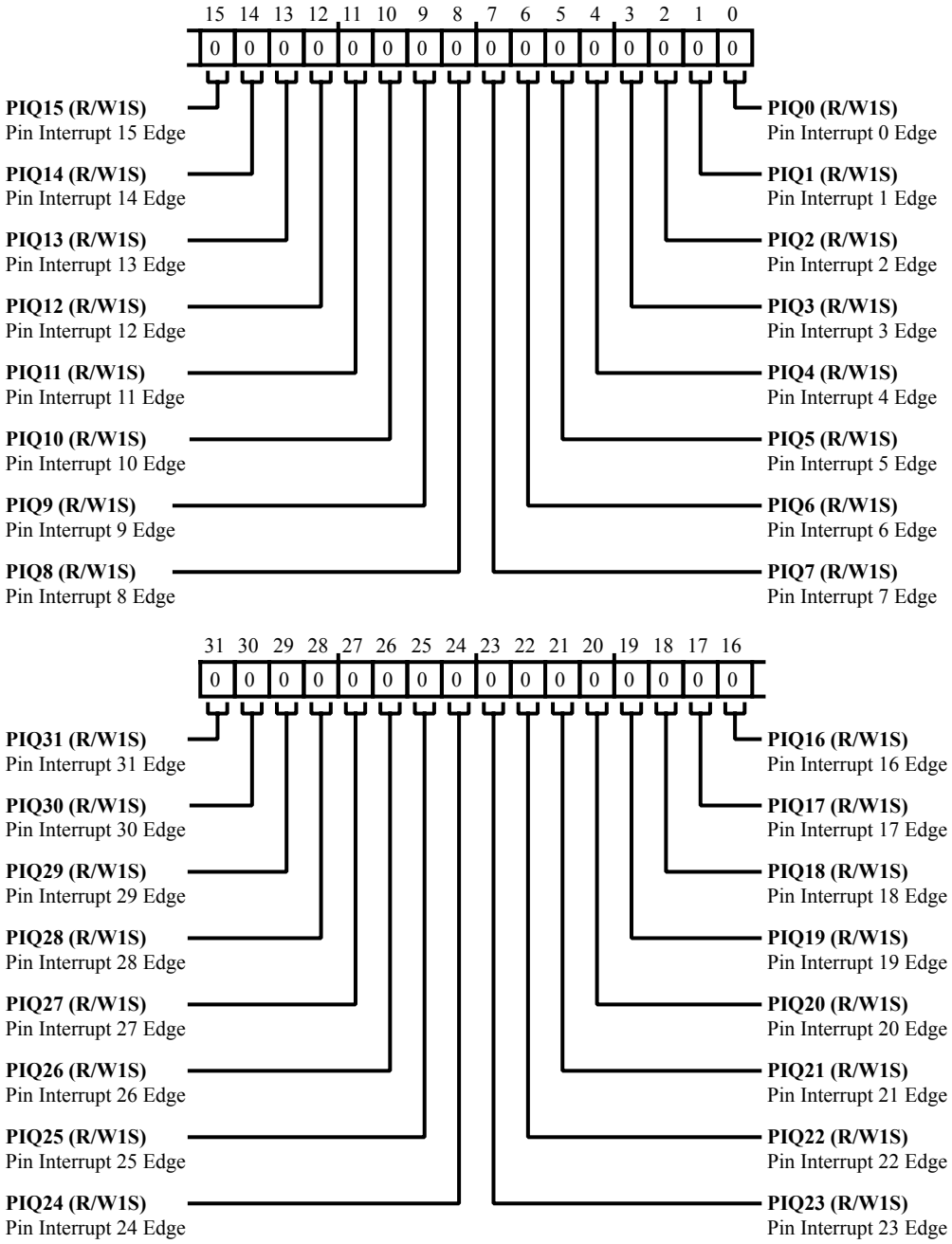

Figure 20-26: PINT_EDGE_SET Register Diagram

Table 20-30: 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 20-30: 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.

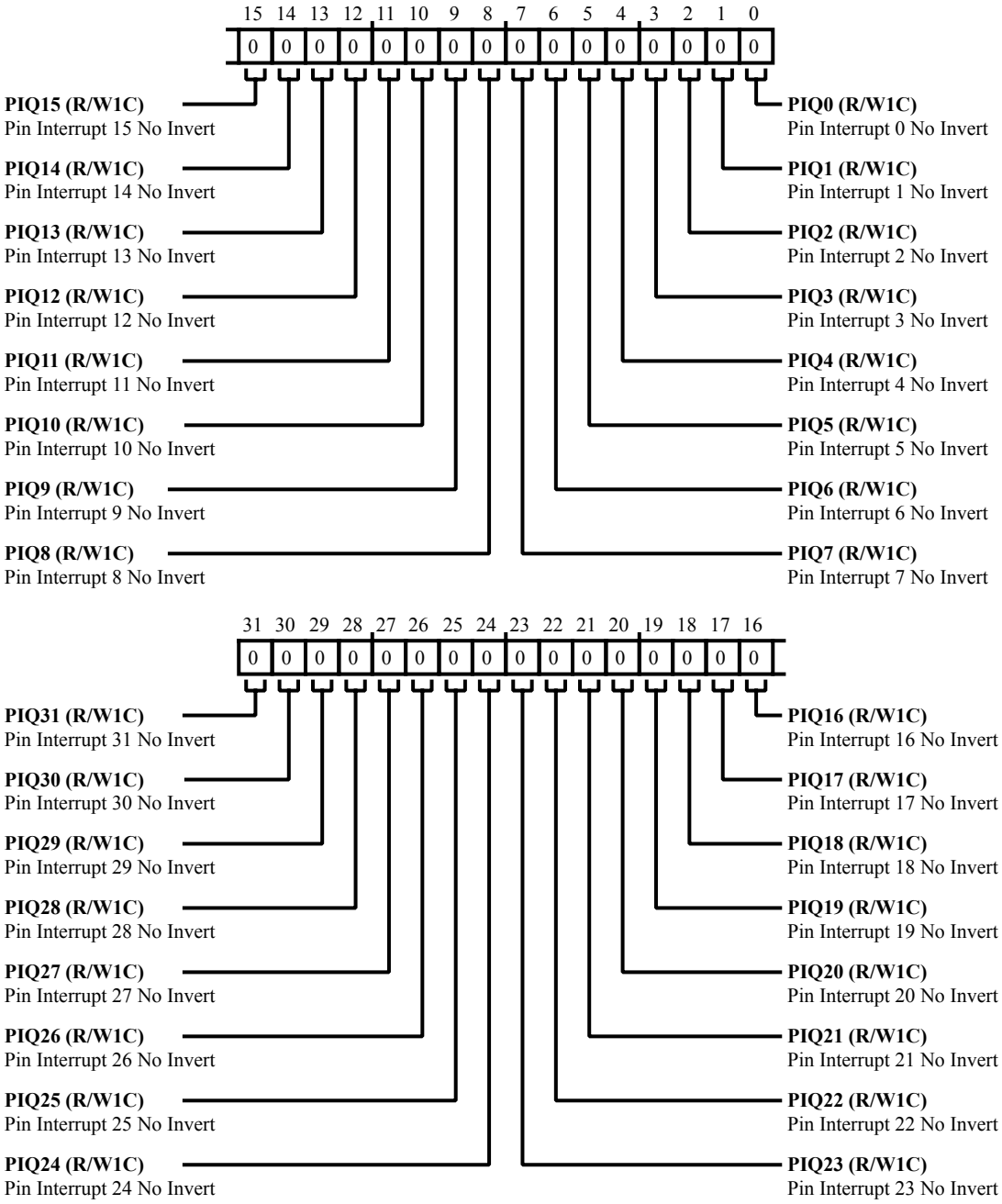

Figure 20-27: PINT_INV_CLR Register Diagram

Table 20-31: 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 20-31: 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.

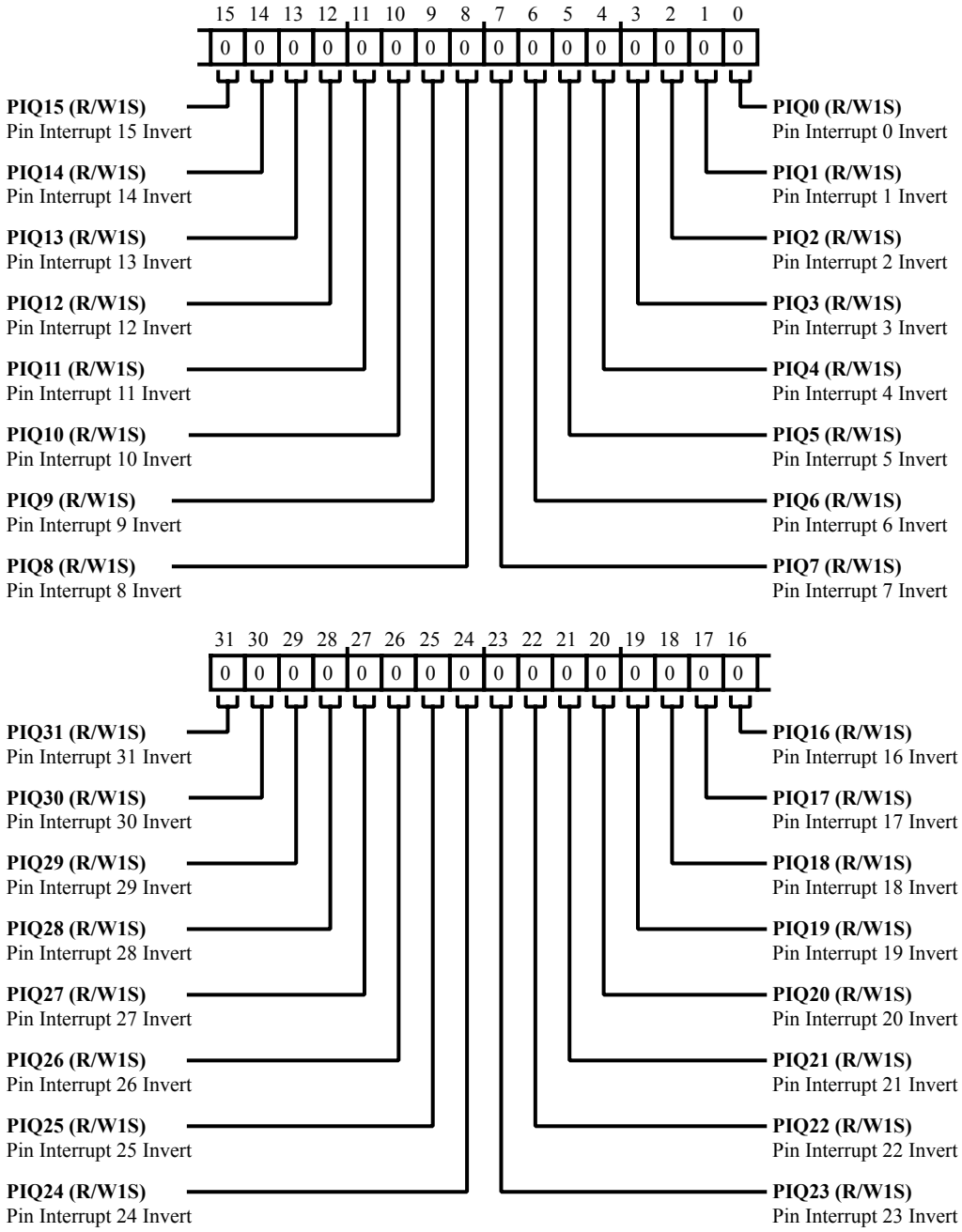

Figure 20-28: PINT_INV_SET Register Diagram

Table 20-32: 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 20-32: 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.

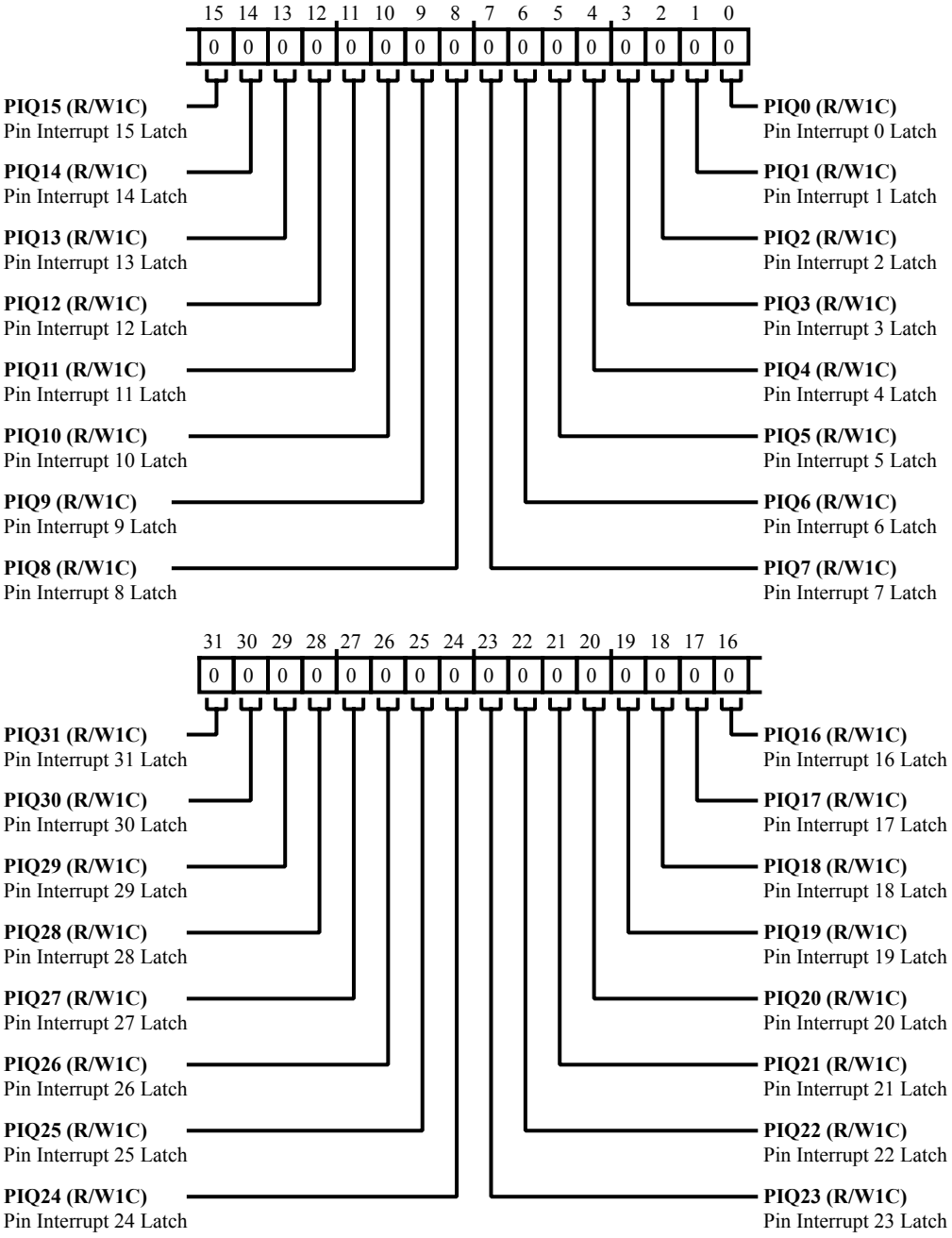

Figure 20-29: PINT_LATCH Register Diagram

Table 20-33: 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 20-33: 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 20-33: 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 interrupts. Writing 1 to a bit in PINT_MSK_CLR masks the corresponding pin interrupt.

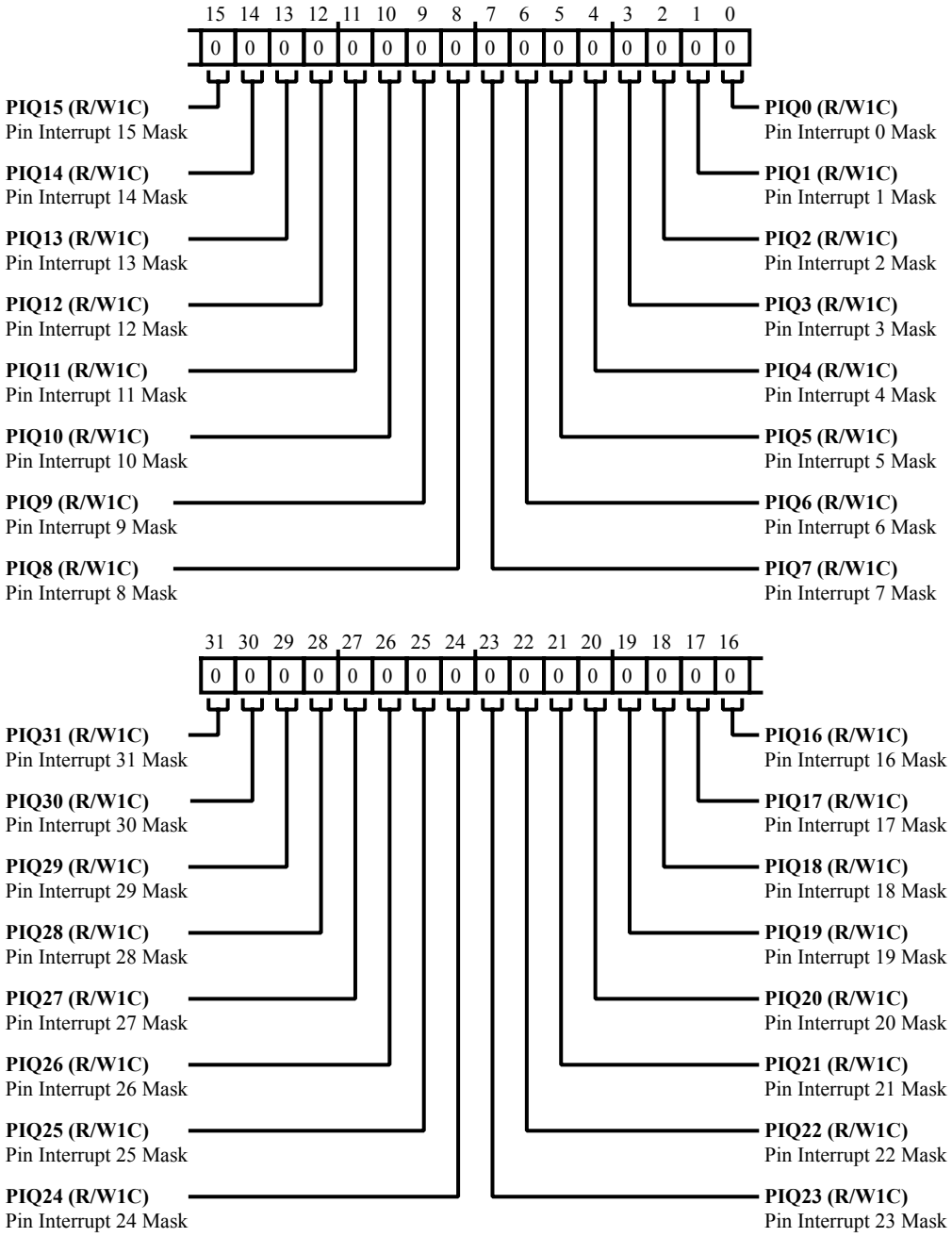

Figure 20-30: PINT_MSK_CLR Register Diagram

Table 20-34: 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 20-34: 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 interrupts. Writing 1 to a bit in PINT_MSK_SET unmasks the corresponding pin interrupt.

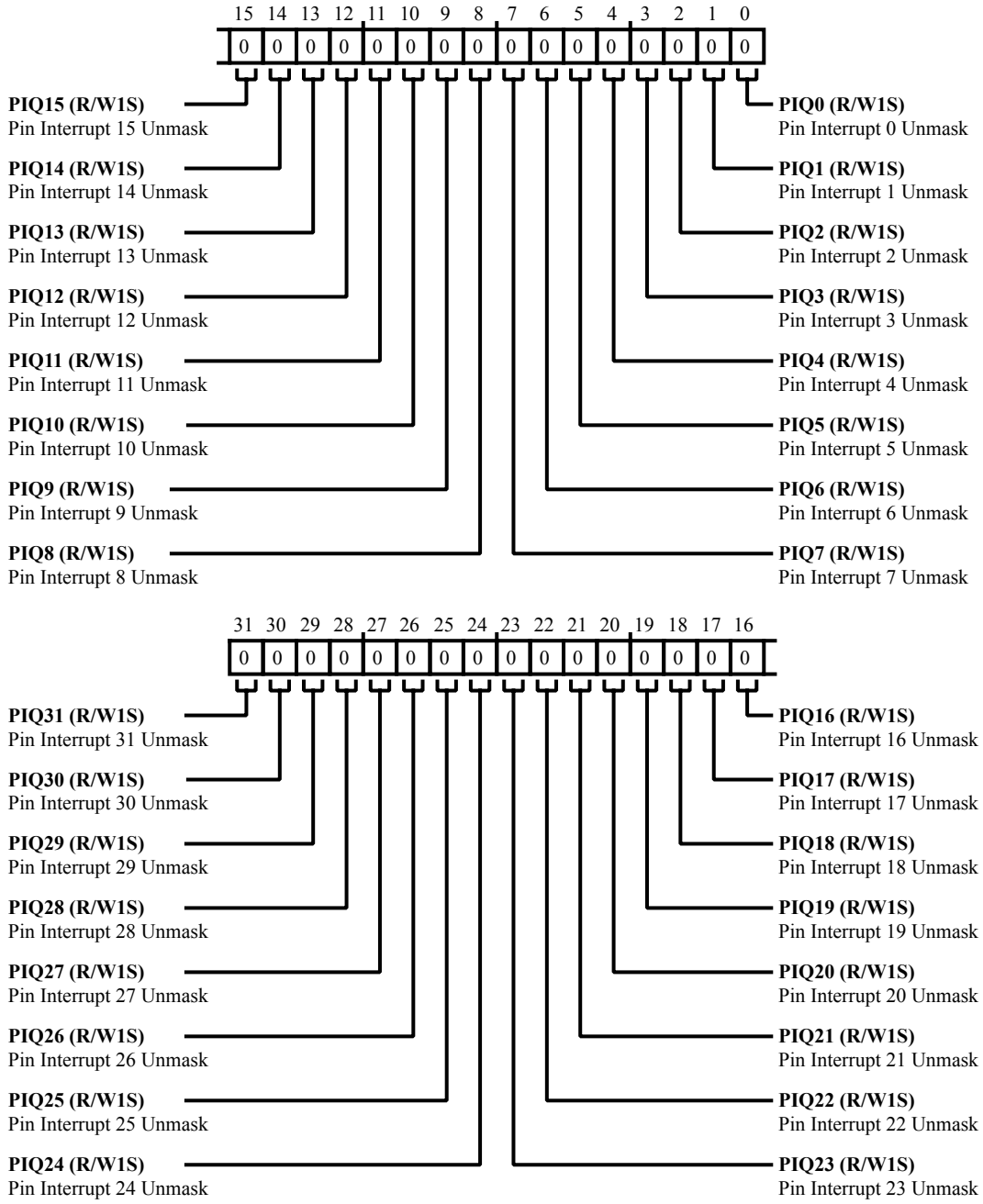

Figure 20-31: PINT_MSK_SET Register Diagram

Table 20-35: 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 20-35: 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.

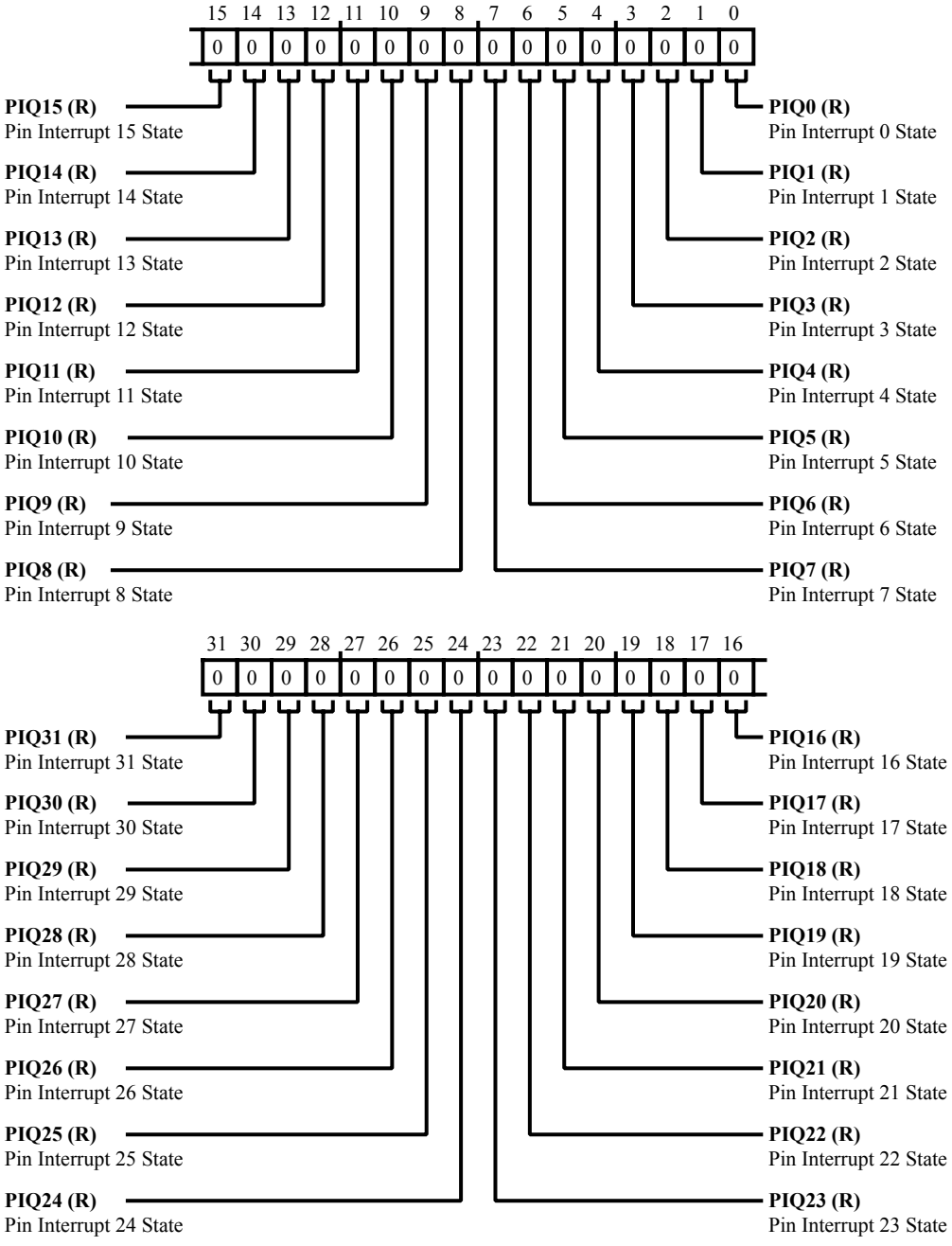

Figure 20-32: PINT_PINSTATE Register Diagram

Table 20-36: 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 20-36: 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 20-36: 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.

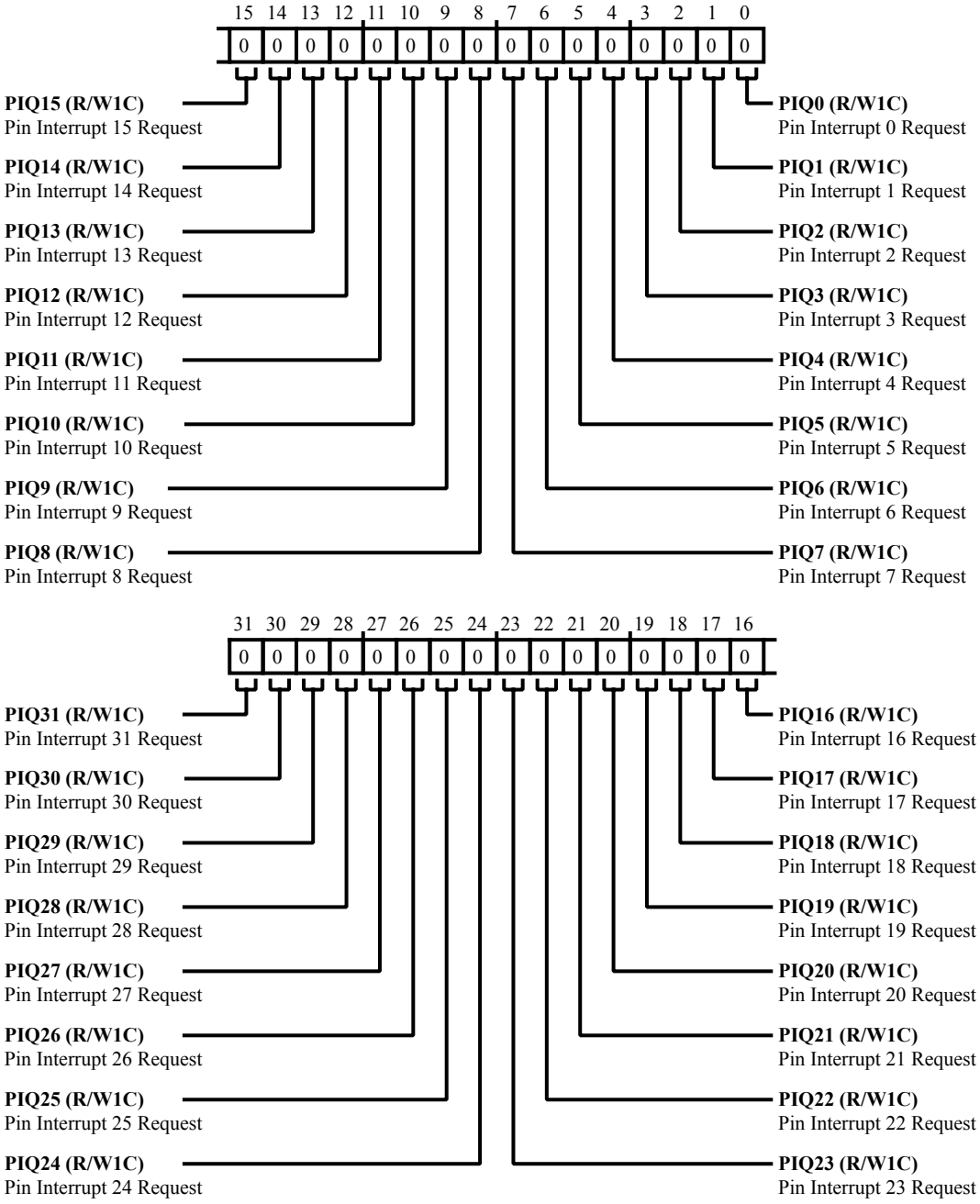

Figure 20-33: PINT_REQ Register Diagram

Table 20-37: 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 20-37: 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 20-37: 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.

ADSP-BF70x PADS Register Descriptions

Pads Controller (PADS) contains the following registers.

Table 20-38: ADSP-BF70x PADS Register List

Name	Description
PADS_PCFG0	Peripheral PAD Configuration0 Register

Peripheral PAD Configuration0 Register

The PADS_PCFG0 register provides several configuration options for the pads and multiplexing for peripherals.

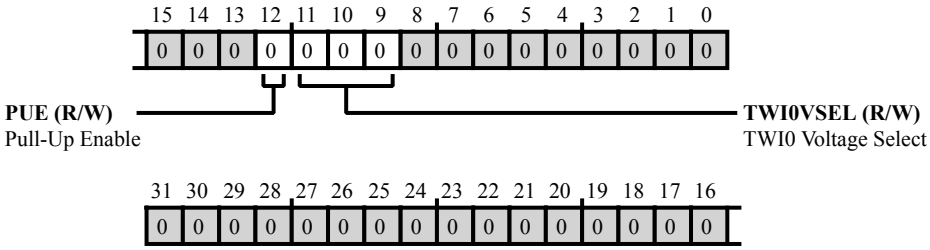

Figure 20-34: PADS_PCFG0 Register Diagram

Table 20-39: PADS_PCFG0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	PUE	Pull-Up Enable. The PADS_PCFG0.PUE bit overrides the input enable and enables the pull-up on the GPIO, SPI2 and AFC pads (Housekeeping ADC).	
11:9 (R/W)	TWI0VSEL	TWI0 Voltage Select. The PADS_PCFG0.TWI0VSEL bit selects the drive/tolerate voltage for the TWI_SCL and TWI_SDA pins for TWI0. By default this bit is cleared (=0, 3.3V).	
		256	VDD_EXT=3.3V, VBUS_TWI=5V
		0	VDD_EXT=3.3V, VBUS_TWI=3.3V
		16	Reserved
		1	VDD_EXT=1.8V, VBUS_TWI=1.8V
		17	VDD_EXT=1.8V, VBUS_TWI=3.3V
		257-273	Reserved

21 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, various interrupts 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

ADSP-BF70x 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 21-1: ADSP-BF70x 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

ADSP-BF70x TIMER Interrupt List

Table 21-2: ADSP-BF70x TIMER Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
11	TIMER0_TMR0	TIMER0 Timer 0	Level	
12	TIMER0_TMR1	TIMER0 Timer 1	Level	
13	TIMER0_TMR2	TIMER0 Timer 2	Level	
14	TIMER0_TMR3	TIMER0 Timer 3	Level	
15	TIMER0_TMR4	TIMER0 Timer 4	Level	
16	TIMER0_TMR5	TIMER0 Timer 5	Level	
17	TIMER0_TMR6	TIMER0 Timer 6	Level	
18	TIMER0_TMR7	TIMER0 Timer 7	Level	
19	TIMER0_STAT	TIMER0 Status	Level	

ADSP-BF70x TIMER Trigger List

Table 21-3: ADSP-BF70x TIMER Trigger List Masters

Trigger ID	Name	Description	Sensitivity
2	TIMER0_TMR0_MST	TIMER0 Timer 0	Edge
3	TIMER0_TMR1_MST	TIMER0 Timer 1	Edge
4	TIMER0_TMR2_MST	TIMER0 Timer 2	Edge
5	TIMER0_TMR3_MST	TIMER0 Timer 3	Edge
6	TIMER0_TMR4_MST	TIMER0 Timer 4	Edge
7	TIMER0_TMR5_MST	TIMER0 Timer 5	Edge
8	TIMER0_TMR6_MST	TIMER0 Timer 6	Edge
9	TIMER0_TMR7_MST	TIMER0 Timer 7	Edge

Table 21-4: ADSP-BF70x TIMER Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
2	TIMER0_TMR0_SLV0	TIMER0 Timer 0 Slave 0	Pulse
3	TIMER0_TMR1_SLV0	TIMER0 Timer 1 Slave 0	Pulse
4	TIMER0_TMR2_SLV0	TIMER0 Timer 2 Slave 0	Pulse
5	TIMER0_TMR3_SLV0	TIMER0 Timer 3 Slave 0	Pulse
6	TIMER0_TMR4_SLV0	TIMER0 Timer 4 Slave 0	Pulse

Table 21-4: ADSP-BF70x TIMER Trigger List Slaves (Continued)

Trigger ID	Name	Description	Sensitivity
7	TIMER0_TMR5_SLV0	TIMER0 Timer 5 Slave 0	Pulse
8	TIMER0_TMR6_SLV0	TIMER0 Timer 6 Slave 0	Pulse
9	TIMER0_TMR7_SLV0	TIMER0 Timer 7 Slave 0	Pulse
10	TIMER0_TMR0_SLV1	TIMER0 Timer 0 Slave 1	Pulse
11	TIMER0_TMR1_SLV1	TIMER0 Timer 1 Slave 1	Pulse
12	TIMER0_TMR2_SLV1	TIMER0 Timer 2 Slave 1	Pulse
13	TIMER0_TMR3_SLV1	TIMER0 Timer 3 Slave 1	Pulse
14	TIMER0_TMR4_SLV1	TIMER0 Timer 4 Slave 1	Pulse
15	TIMER0_TMR5_SLV1	TIMER0 Timer 5 Slave 1	Pulse
16	TIMER0_TMR6_SLV1	TIMER0 Timer 6 Slave 1	Pulse
17	TIMER0_TMR7_SLV1	TIMER0 Timer 7 Slave 1	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 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 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 SCLK0 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.

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 21-5: 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 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 21-6: Usage of the Period, Width, and Delay Registers in Different Timer Modes

Timer Mode	Period	Width	DELAY
IDLE	Not writable	Not writable	Not writable

Table 21-6: Usage of the Period, Width, and Delay Registers in Different Timer Modes (Continued)

Timer Mode	Period	Width	DELAY
WATCHDOG	Can be updated on-the-fly. New value takes effect either upon 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.	Can be updated on-the-fly. New value takes effect either upon timer start or when an asserting edge on the input signal is sensed.
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	Can be updated on-the-fly. New value takes effect either upon timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Can be updated on-the-fly. New value takes effect either upon timer start or at the end of the current period. A write followed by immediate read returns the current operating values.	Can be updated on-the-fly. New value takes effect either upon 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 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 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 SCLK0 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).

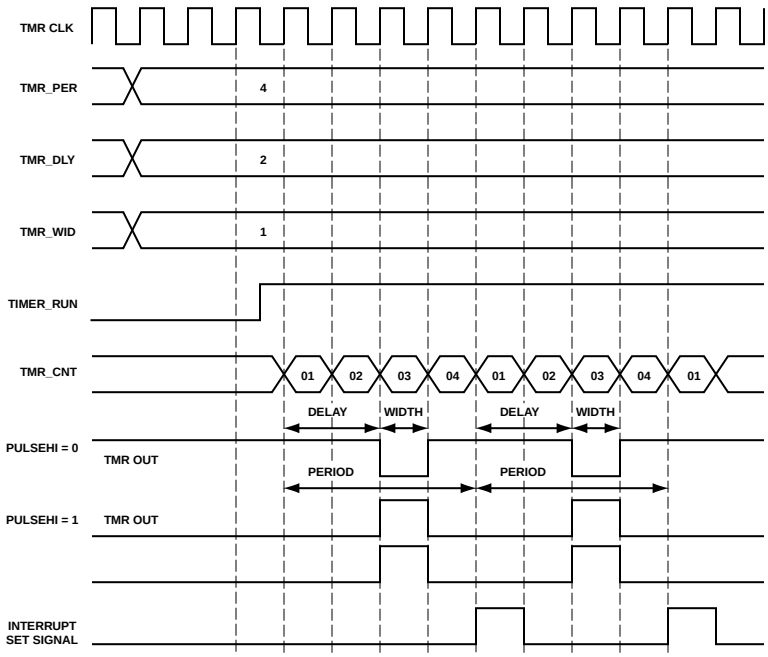

Figure 21-1: Signal Generation in Continuous PWMOUT Mode

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.

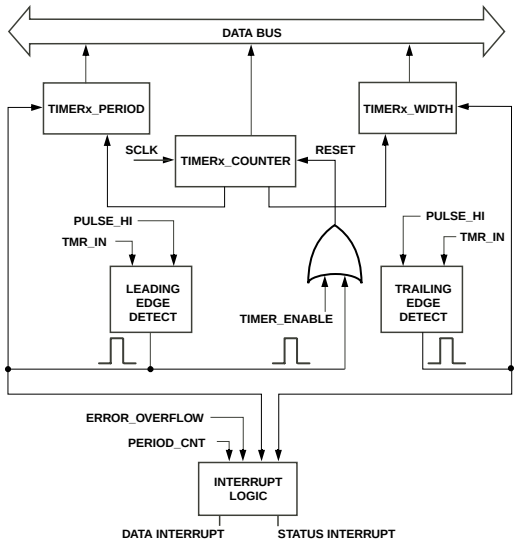

Figure 21-2: Timer Signal Flow in Width Capture Mode

NOTE: SCLK in the ***Timer Signal Flow in Width Capture Mode*** figure is SCLK0.

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.

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 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.

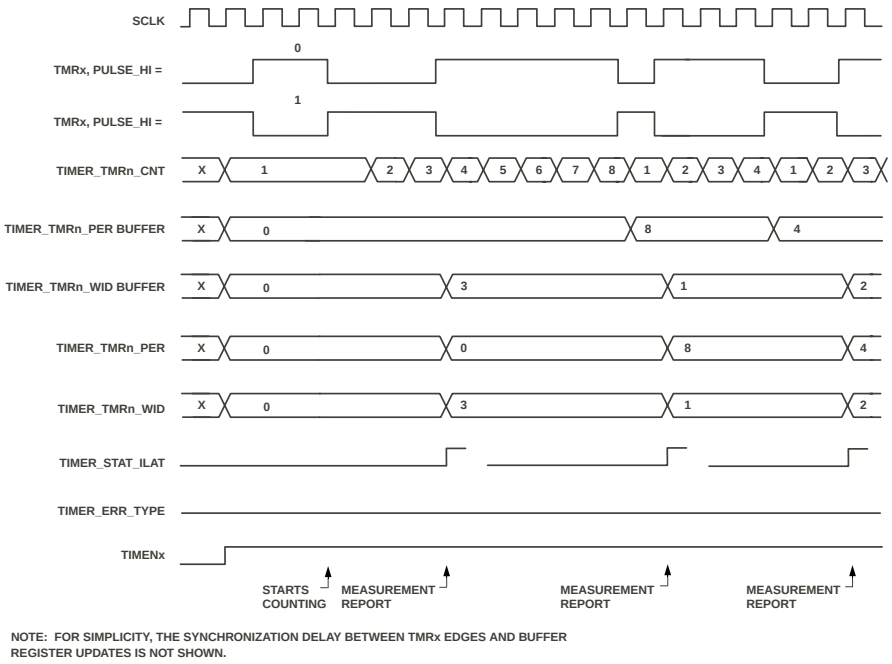

Figure 21-3: Example of Width Capture Deasserted Mode (TMODE=b#1011)

NOTE: SCLK in the ***Example of Width Capture Deasserted Mode (TMODE=b#1011)*** figure is SCLK0.

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.

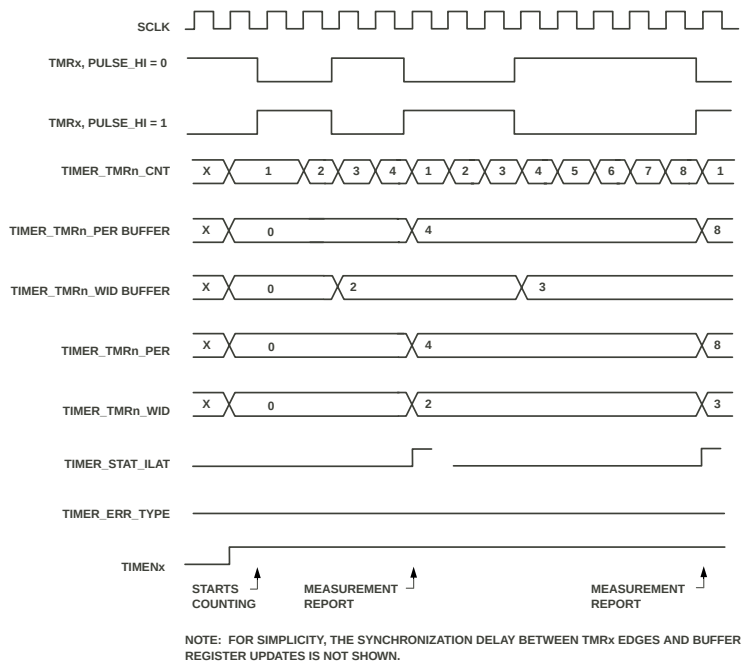

Figure 21-4: Example of Width Capture Asserted Mode (TMODE=b#1010)

NOTE: SCLK in the ***Example of Width Capture Asserted Mode (TMODE=b#1010)*** figure is SCLK0.

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 (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 (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 in WIDCAP mode indicates that a new measurement is ready to be read (a measurement report). Similarly, an interrupt on the timer status interrupt line (shared interrupt 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 interrupts to measure input signal periods longer than 0xFFFF FFFF. Each error report interrupt adds a full 2^{32}SCLK0 counts to the total for the period, but the width is ambiguous. Ensure that if software monitors only the status interrupt, then status interrupts 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.

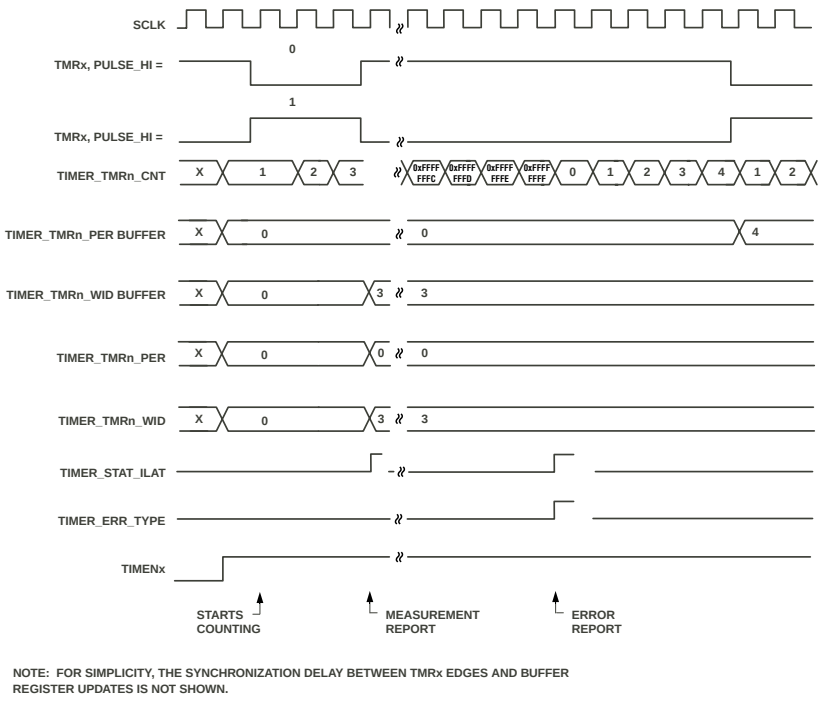

Figure 21-5: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1010)

NOTE: SCLK in the ***Example Timing for Width Capture Followed by Period Overflow*** figure is SCLK0.

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 SCLK0 period. The minimum input high time is a little more than one SCLK0 period. (Refer to the product data sheet for details). The maximum TIMER_TMR[n] input frequency is less than SCLK0/2, with a 50% duty cycle. Under these conditions, the WIDCAP mode timer measures: period =2 and pulse width =1.

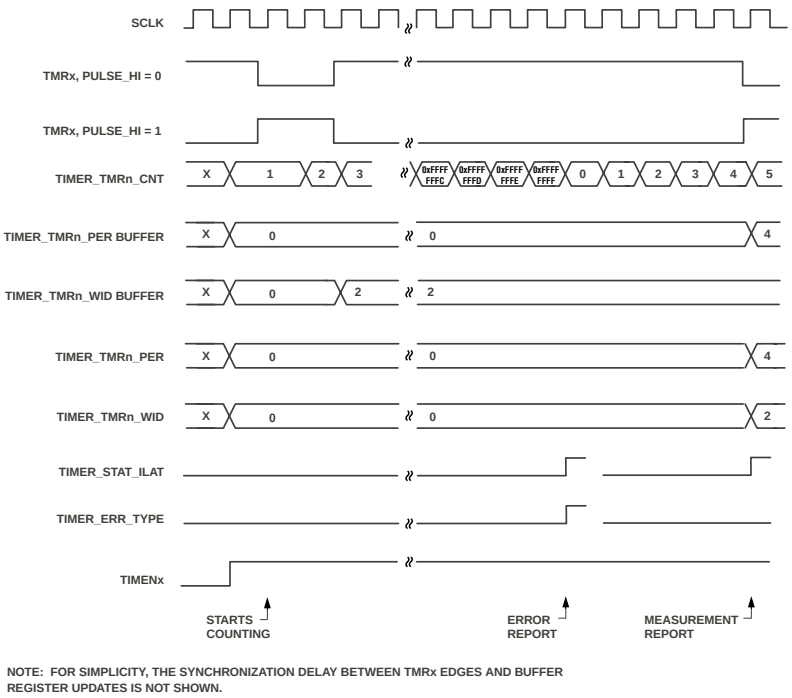

Figure 21-6: Example Timing for Width Capture Followed by Period Overflow (TMR_CFG.TMODE=b#1011)

NOTE: SCLK in the ***Example Timing for Width Capture Followed by Period Overflow*** figure is SCLK0.

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 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 (SCLK0 or alternate clock). The timer can generate an interrupt if the de-asserting 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 de-asserting 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.

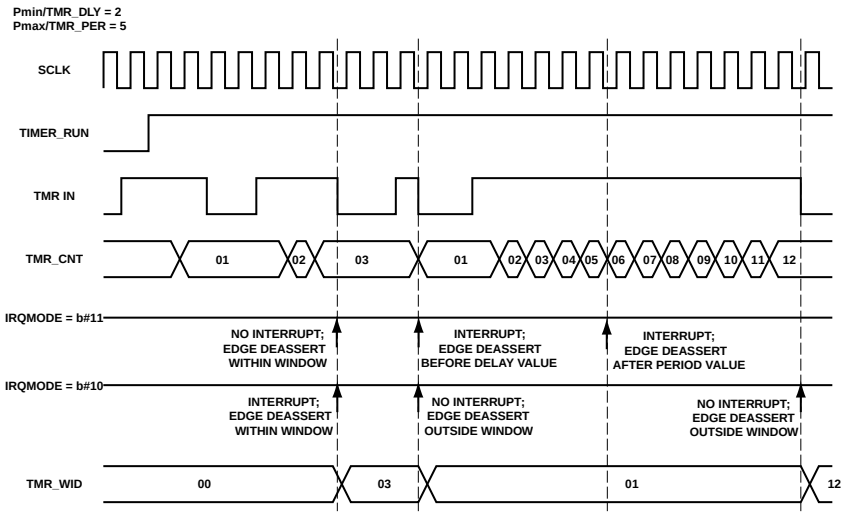

Figure 21-7: Watchdog Width Mode Timing

NOTE: SCLK in the ***Watchdog Width Mode Timing*** figure is SCLK0.

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 21-7: 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 de-asserting edge of input signal	No Interrupt	No Error Interrupt
		PW ≤ TMR_PER	Interrupt at de-asserting 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 De-asserting edge of input signal	No Error Interrupt
		TMR_DLY < PW≤ TMR_PER	Interrupt at de-asserting 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 (SCLK0) 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.

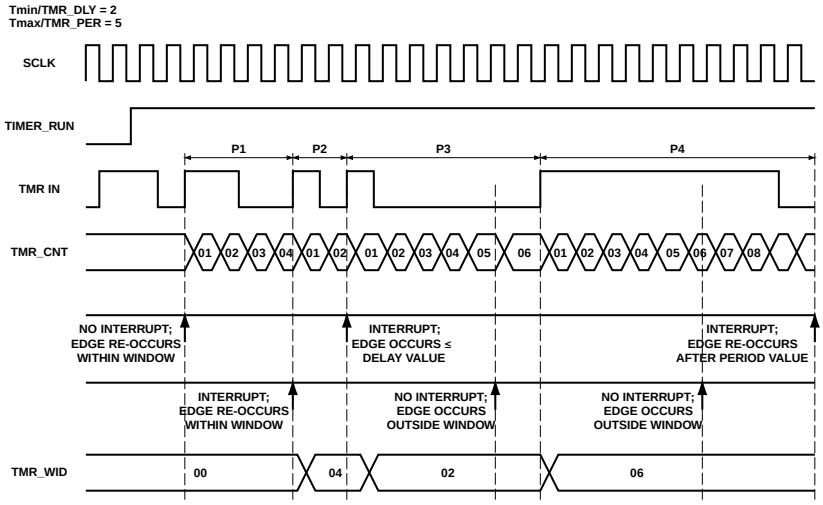

Figure 21-8: Watchdog Period Mode Timing

NOTE: SCLK in the ***Watchdog Period Mode Timing*** figure is SCLK0.

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 21-8: 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 de-asserting 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 de-asserting edge of input signal	No Error Interrupt
		TMR_DLY < Pulse Period ≤TMR_PER	Interrupt at de-asserting 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 21-8: 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 is generated in the SCLK0 clock domain, the width of the input signal must be more than one SCLK0 period. Along with generating the interrupt, 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 at the input. To avoid this problem, programs must ensure that interrupts 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 SCLK0. 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 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.

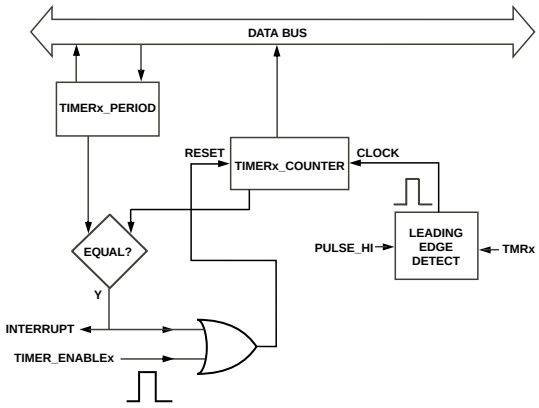

Figure 21-9: EXTCLK Mode Control Flow

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.
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 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 at the system event controller.
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 at the system event controller.
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 21-9: 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 21-10: 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 SCLK0] b#11	Set

Table 21-10: Rollover Event (Continued)

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)
				[timer running at ALT_CLKx]	
≥ 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 21-11: 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 21-12: 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 21-13: 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 21-14: 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 21-15: 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 21-16: 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 21-17: 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 21-18: 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 21-19: 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 21-20: 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

ADSP-BF70x TIMER Register Descriptions

General-Purpose Timer Block (TIMER) contains the following registers.

Table 21-21: ADSP-BF70x 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

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.

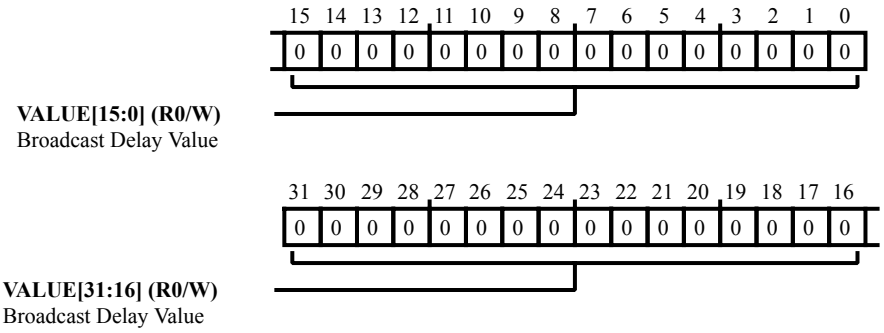

Figure 21-10: TIMER_BCAST_DLY Register Diagram

Table 21-22: 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.

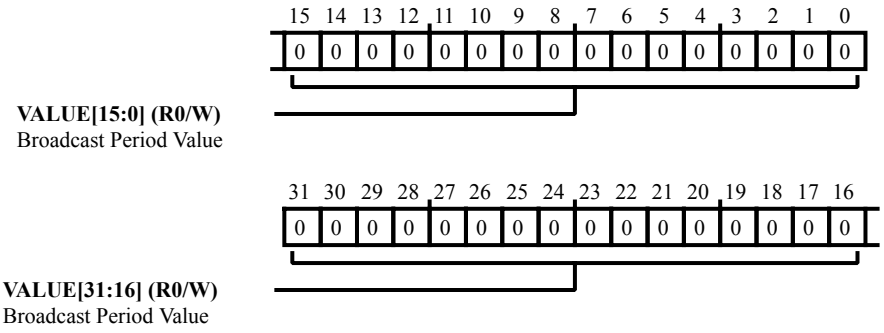

Figure 21-11: TIMER_BCAST_PER Register Diagram

Table 21-23: 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.

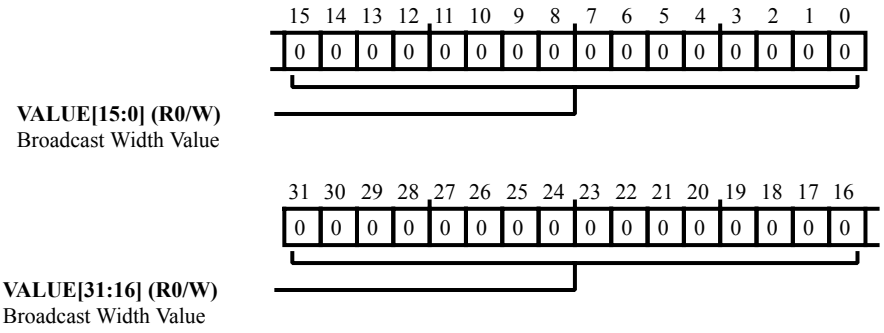

Figure 21-12: TIMER_BCAST_WID Register Diagram

Table 21-24: 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 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.

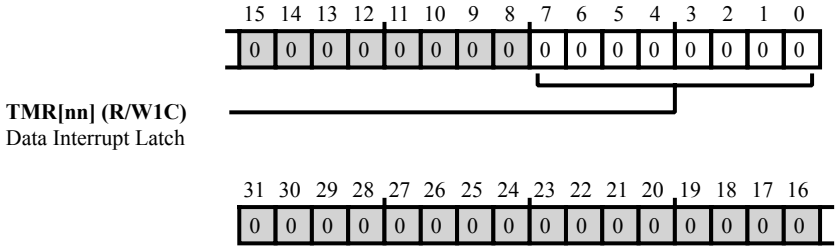

Figure 21-13: TIMER_DATA_ILAT Register Diagram

Table 21-25: 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.

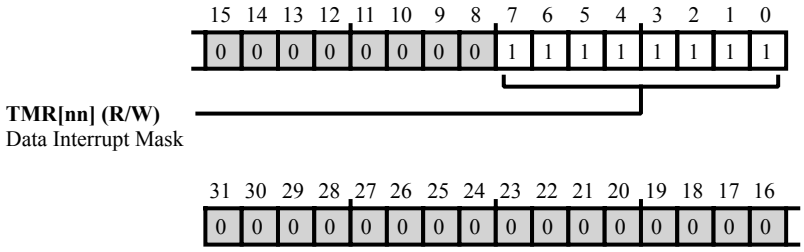

Figure 21-14: TIMER_DATA_IMSK Register Diagram

Table 21-26: 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 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.

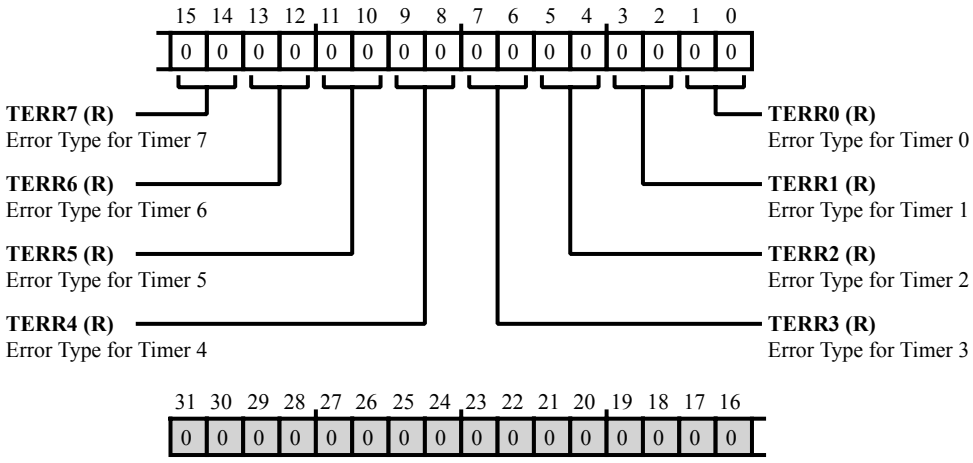

Figure 21-15: TIMER_ERR_TYPE Register Diagram

Table 21-27: 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 21-27: 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.

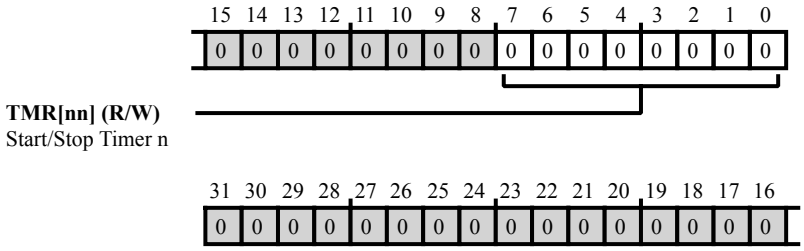

Figure 21-16: TIMER_RUN Register Diagram

Table 21-28: 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.

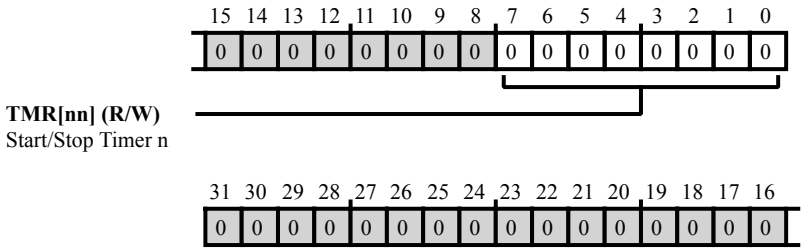

Figure 21-17: TIMER_RUN_CLR Register Diagram

Table 21-29: 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.

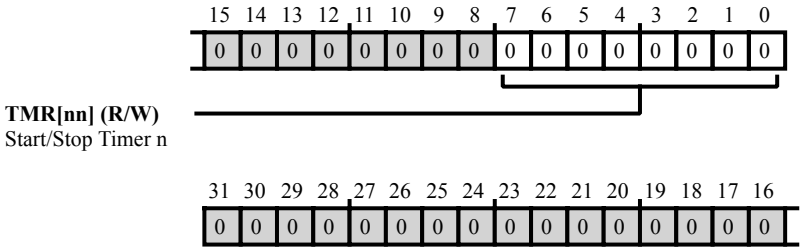

Figure 21-18: TIMER_RUN_SET Register Diagram

Table 21-30: 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 interrupts, 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 source.

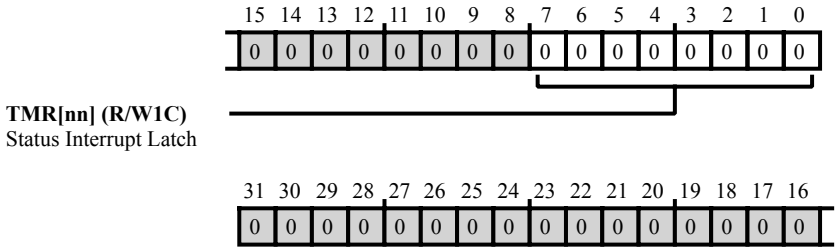

Figure 21-19: TIMER_STAT_ILAT Register Diagram

Table 21-31: 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 is latched, and status of 1 indicates a timer counter overflow or programming error interrupt 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.

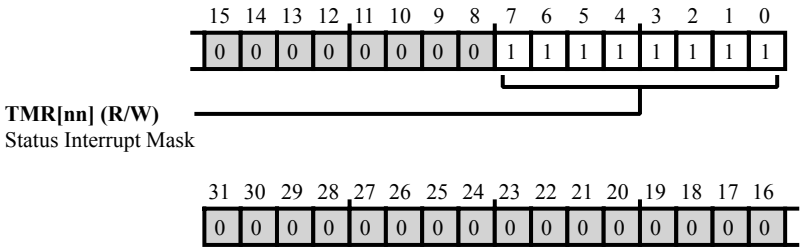

Figure 21-20: TIMER_STAT_IMSK Register Diagram

Table 21-32: 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.

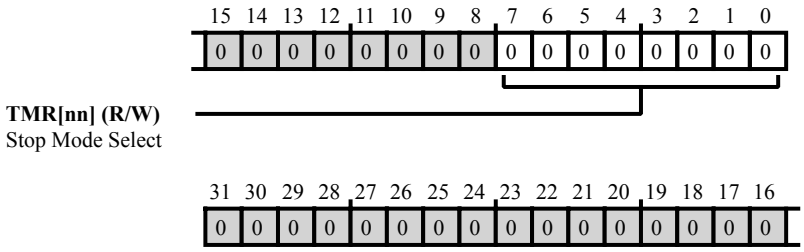

Figure 21-21: TIMER_STOP_CFG Register Diagram

Table 21-33: 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.

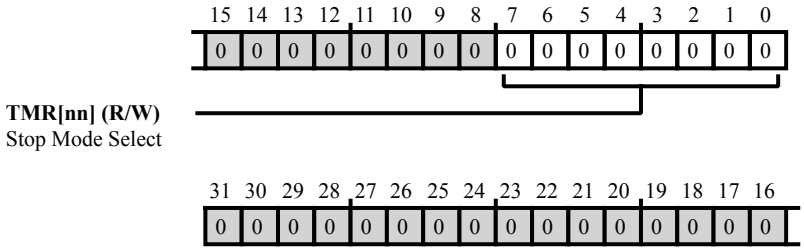

Figure 21-22: TIMER_STOP_CFG_CLR Register Diagram

Table 21-34: 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.

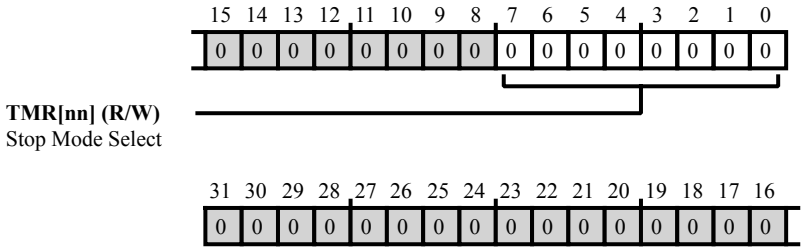

Figure 21-23: TIMER_STOP_CFG_SET Register Diagram

Table 21-35: 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.

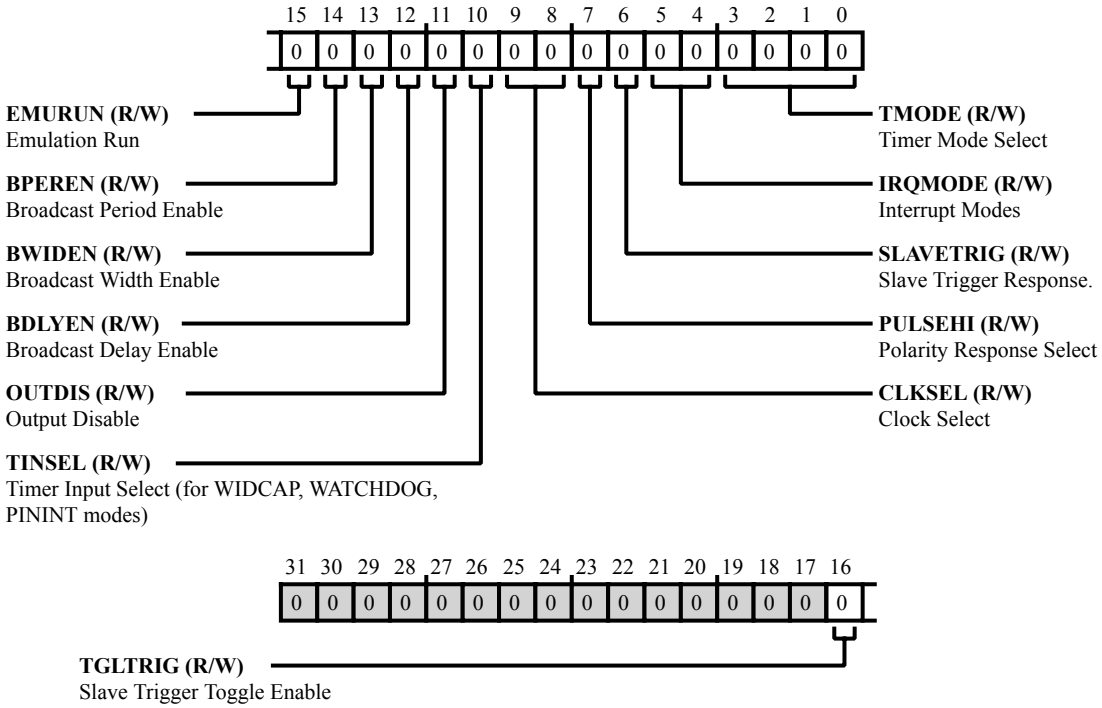

Figure 21-24: TIMER_TMR[n]_CFG Register Diagram

Table 21-36: 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 21-36: 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 SCLK0
		1	Use TMR_ALT_CLK0 as TMR Clock
		3	Use TMR_ALT_CLK1 as TMR Clock

Table 21-36: 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 21-36: 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
		10	Measurement Report at Asserting Edge of Waveform

Table 21-36: TIMER_TMR[n]_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		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 SCLK0, the TMR or alternate capture input pins, TIMER_ACLK[n]. The counter retains its value after the timer is disabled.

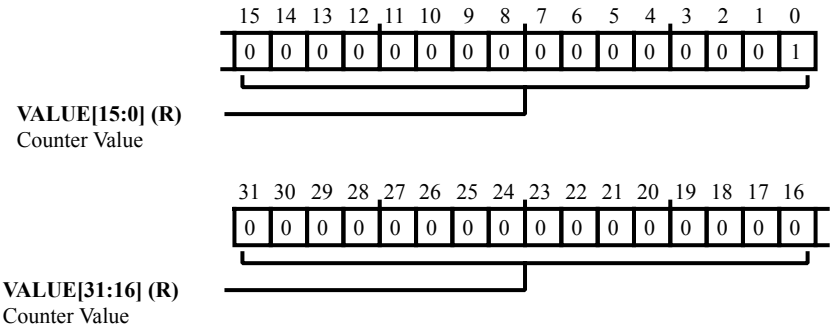

Figure 21-25: TIMER_TMR[n]_CNT Register Diagram

Table 21-37: 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.

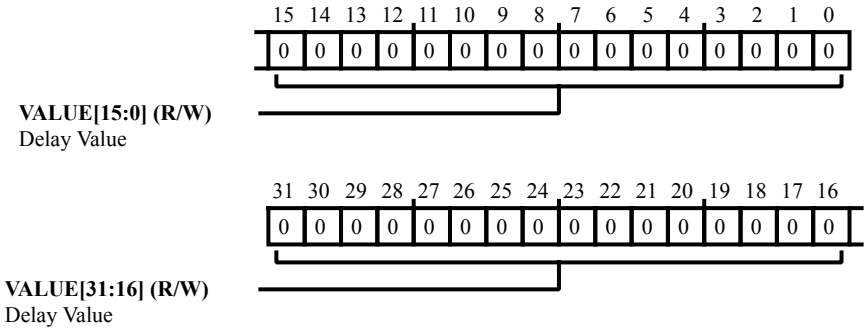

Figure 21-26: TIMER_TMR[n]_DLY Register Diagram

Table 21-38: 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.

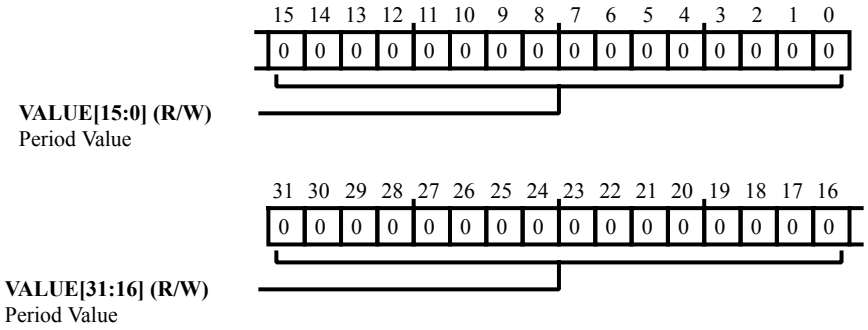

Figure 21-27: TIMER_TMR[n]_PER Register Diagram

Table 21-39: 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.

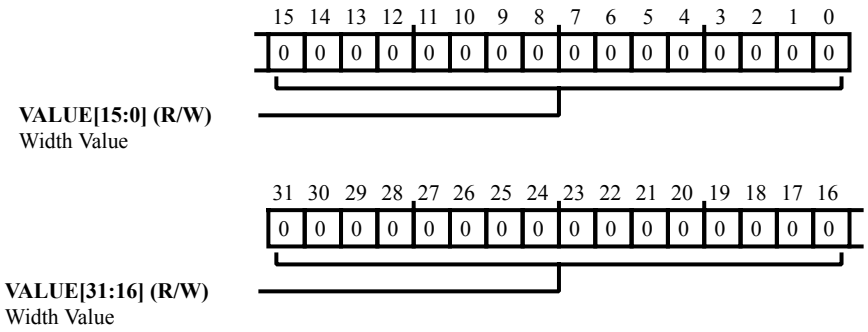

Figure 21-28: TIMER_TMR[n]_WID Register Diagram

Table 21-40: 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.

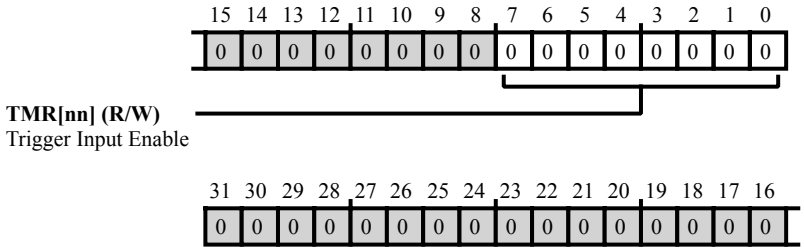

Figure 21-29: TIMER_TRG_IE Register Diagram

Table 21-41: 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.

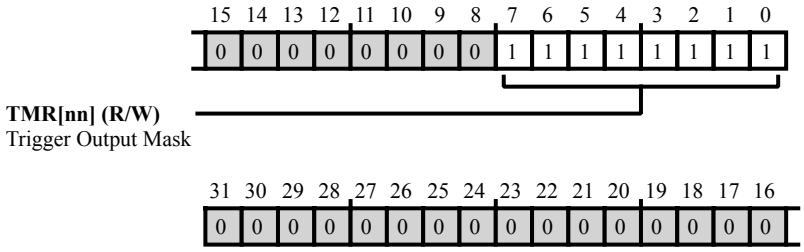

Figure 21-30: TIMER_TRG_MSK Register Diagram

Table 21-42: 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.

22 Watchdog Timer (WDOG)

The processor has a 32-bit watchdog timer that can be used to improve system reliability by generating an event to the processor core when the watchdog expires before software updates it. The system clock (SCLK0) clocks the watchdog timers.

WDOG Features

The watchdog timer has the following features:

- Programmable 32-bit watchdog count value
- 8-bit disable bit pattern
- General-purpose core event generation

Applications typically use the watchdog timer to 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 would be indicative of the system software 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 via the System Event Controller (SEC) to the core to be serviced by a handler function, providing full software control of device resources (GPIO management, reset control, etc.)
- as a fault condition via 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 (e.g., GPIO control).

To facilitate debugging, the watchdog timer does not decrement (even if enabled) when the processor is in emulation mode.

WDOG Functional Description

When enabled, the 32-bit watchdog timer counts downward every SCLK0 cycle. If it reaches zero, the watchdog expiration event is generated, which can be used in a variety of ways.

To start the watchdog timer:

1. Program the watchdog timeout period (in SCLK0cycles) in the WDOG_CNT register. With the watchdog disabled, this write also pre-loads 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 will continue decrementing until it reaches zero, at which point the expiration event is generated, and the WDOG_CTL.WDROrollover bit is set.

Watchdog operation will continue in this manner unless disabled by explicitly writing 0xAD to the WDOG_CTL.WDEN field.

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 will execute based on interrupt priority or to initiate automated hardware response through the SEC Fault Interface (SFI), which allows for automatic management of external fault pins, automatic system resets, and trigger outputs to numerous potential trigger slaves. See the SEC and TRU chapters for details.

ADSP-BF70x 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 22-1: ADSP-BF70x WDOG Register List

Name	Description
WDOG_CNT	Count Register
WDOG_CTL	Control Register
WDOG_STAT	Watchdog Timer Status Register

ADSP-BF70x WDOG Interrupt List

Table 22-2: ADSP-BF70x WDOG Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
3	WDOG0_EXP	WDOG0 Expiration	Level	

WDOG Block Diagram

The *Watchdog Timer Block Diagram* figure shows the detailed block diagram for the watchdog timer.

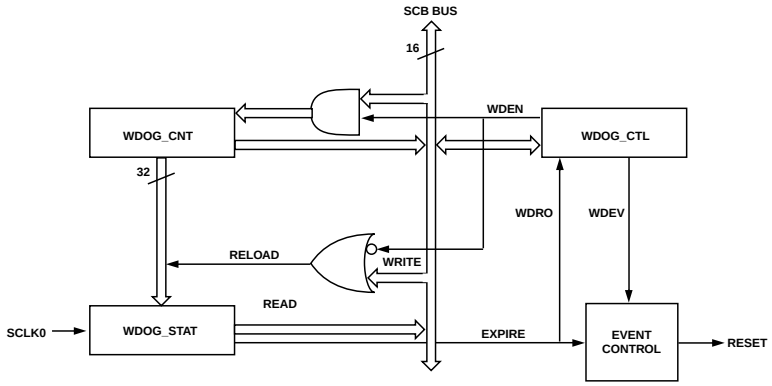

Figure 22-1: Watchdog Timer Block Diagram

Internal Interface

The system clock (SCLK0) clocks the watchdog timer. The registers are accessed through the 16-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.

ADSP-BF70x WDOG Register Descriptions

Watchdog Timer Unit (WDOG) contains the following registers.

Table 22-3: ADSP-BF70x 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.

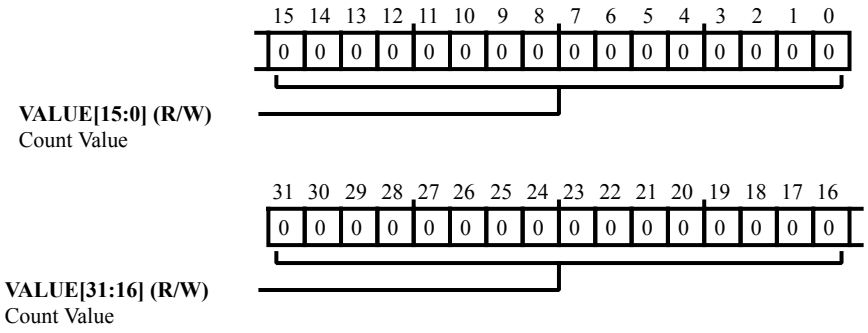

Figure 22-2: WDOG_CNT Register Diagram

Table 22-4: 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.

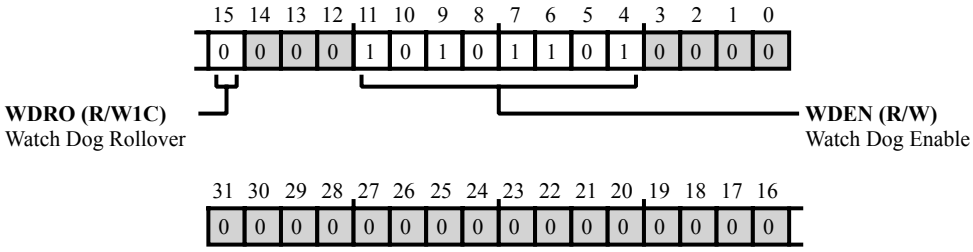

Figure 22-3: WDOG_CTL Register Diagram

Table 22-5: 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 SCLK0 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 SCLK0 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.

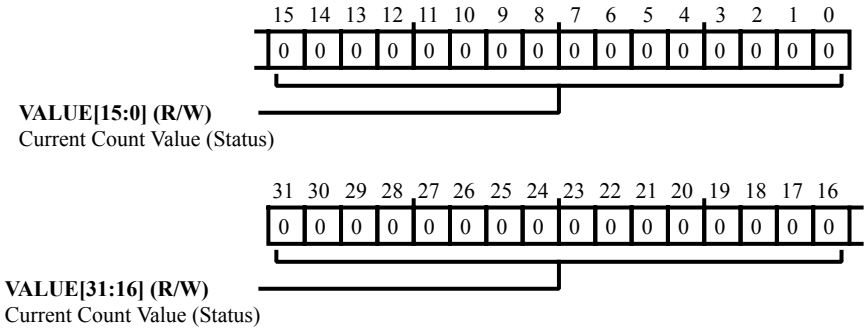

Figure 22-4: WDOG_STAT Register Diagram

Table 22-6: 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.

23 Real Time Clock (RTC)

The processor contains a Real Time Clock (RTC) which provides a set of digital watch features, including time of day, alarm, and stopwatch countdown. It is typically used to implement either a real-time watch or a life counter which counts the elapsed time since the last system reset. The RTC uses dedicated power supply pins and is independent of any reset, which enables it to maintain functionality even when the rest of the processor is powered down.

RTC Features

The RTC interface has the following features.

- Provides a 1 Hz clock with a second, minute, hour, and day counter (0–32767 days)
- Alarm with a time-of-day interrupt
- Operates on a dedicated supply from an external 3.3 V battery
- Stopwatch function
- Interrupt generation for different events
- Standard two-pin interface with an external 32.768 kHz crystal, 6 pF capacitor on each pin, and a 100 MΩ resistor between the pins
- Calibration corrects time once a day; application can use the RTCXTALIN pin to determine calibration settings
- Power down and bus disable
- Provides a wake-up signal (I/O supply domain) to the processor to come out of hibernate and deep sleep modes (see the DPM section)

RTC Functional Description

The RTC provides a set of digital watch features to the processor. The RTC external interface consists of two clock pins, which together with the external components form the reference clock circuit for the RTC. It uses an external 32.768 kHz crystal with external capacitor and operates on a dedicated external 3.3 V Lithium coin cell which is never powered off.

The following are functional characteristics of the RTC.

Power supply partitioning

The RTC logic is partitioned between the processor core supply voltage and RTC supply voltage. The core of RTC unit functions on a RTC power supply.

Battery life

To increase the battery life of the external 3.3V cell, most functions reside in the RTC core voltage domain which runs off the processor supply. Only basic clock circuitry resides in the RTC IO voltage.

Reads/writes to 1Hz registers

There is no latency when reading 1 Hz registers, as the values come from the shadow registers.

Writes to the RTC 1 Hz registers are synchronized to the 1 Hz RTC clock.

ADSP-BF70x RTC Register List

The Real-Time Clock (RTC) provides clock-related services, including a set of processor events that can be counted during program execution. A set of registers governs RTC operations. For more information on RTC functionality, see the RTC register descriptions.

Table 23-1: ADSP-BF70x RTC Register List

Name	Description
RTC_ALM	RTC Alarm Register
RTC_CLK	RTC Clock Register
RTC_IEN	Interrupt Enable Register
RTC_INIT	RTC Initialization Register
RTC_INITSTAT	RTC Initialization Status Register
RTC_STAT	RTC Status Register
RTC_STPWTCH	RTC Stop Watch Register

ADSP-BF70x RTC Interrupt List

Table 23-2: ADSP-BF70x RTC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
61	RTC0_EVT	RTC0 Event	Level	

ADSP-BF70x RTC Trigger List

Table 23-3: ADSP-BF70x RTC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
45	RTC0_EVT	RTC0 Event	Level

Table 23-4: ADSP-BF70x RTC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

RTC Definitions

The following definitions are helpful when using the RTC module.

RTC HV

The RTC HV is the portion of the RTC which runs on 3.3 V domain and contains the logic for RTC counting/clock functionality

RTC LV

The RTC LV operates on the core voltage supply and provides the control/access functionality and user interface to the application

Oscillator

The oscillator is the heart of the RTC. It works with the dedicated external crystal to generate a 32768 Hz signal which is divided down to a 1Hz clock used to operate the rest of the RTC

RTC Signal Descriptions

The RTC signals are listed in the following table.

Table 23-5: RTC Signal Descriptions

Signal Name	Signal Description
XTALIN	Pin of the dedicated 32.768KHz external crystal
XTALOUT	Pin of the dedicated 32.768KHz external crystal
VDDRTC	Power supply pin (from Lithium Coin Cell) for RTC module

RTC Architectural Concepts

The key use of the RTC is to provide the time keeping function and maintain the time and date in an accurate and reliable manner with minimal power consumption. In addition to time keeping it also provides the stopwatch and alarm features. The RTC uses the internal counters to keep the time of the day in terms of seconds, minutes, hours and days. This data is enough for the user application to extract the date and time information from the RTC.

Interrupts can be issued periodically, either every second, every minute, every hour, or every day. Each of these interrupts can be independently controlled. It is the responsibility of the program to set the correct time by a software write into the RTC_CLK register. Once set, the counters maintain time as long as the RTC supply is valid. The RTC provides two alarm features, programmed with the RTC alarm register (RTC_ALM). The first is a time of day alarm (hour, minute, and second). When the alarm interrupt is enabled, the RTC generates an interrupt each day at the time specified. The second alarm feature allows the application to specify a day as well as a time.

The RTC also provides a stopwatch function that acts as a countdown timer. The application can program a second count into the RTC stopwatch count register (RTC_STPWTCH). When the stopwatch interrupt is enabled and the specified number of seconds has elapsed, the RTC generates an interrupt.

RTC Block Diagram

The RTC block diagram provides a top level block diagram of the elements used. The main blocks of the RTC are the individual counters, the alarm register, and the event control.

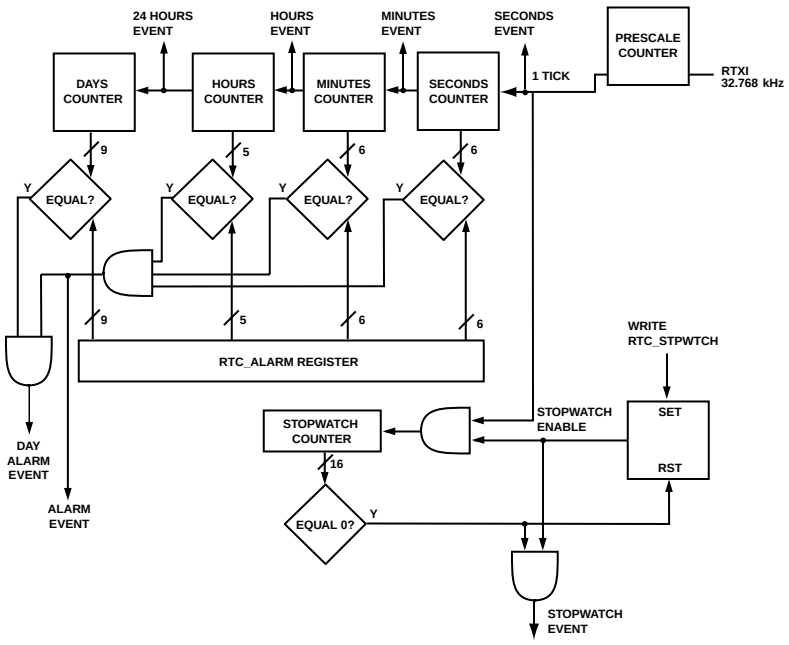

Figure 23-1: RTC Block Diagram

Power Supply Partitioning

The RTC logic is partitioned between the processor core supply voltage and RTC supply voltage. The core of RTC unit functions on a RTC power supply so that the RTC can maintain the time even if the processor power is off. When the core supply voltage is absent, interrupts are ignored.

NOTE: Battery power supply can operate the RTC when the I/O voltage is turned off.

The RTC is partitioned into two blocks. The counting and clock function is provided in the I/O voltage domain (VDD_EXT), while the control and access function and the user interface are provided in the core voltage domain. The RTC I/O operates on a dedicated power supply provided by the external 3.3 V (nominal) lithium coin cell. The RTC LV operates on the nominal core voltage supply (VDD_INT). The interface between both blocks is provided by a set of level shifters. The partitioning at chip level is shown in the ***Power Supply Partition*** figure.

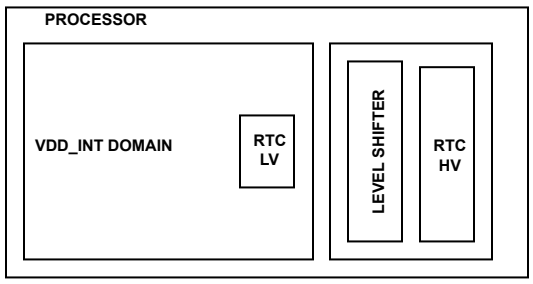

Figure 23-2: Power Supply Partition

Battery Life

To increase the battery life of the external 3.3V cell, most functions reside in the RTC core voltage domain which runs off the processor supply. Only basic clock circuitry resides in the RTC IO voltage. The RTC module provides the following features to extend battery life.

- The seconds, minutes, hours and days counters reside inside the RTC IO voltage.
- The alarm register and comparators reside inside the RTC IO voltage. This allows programs to power down the rest of the chip without the alarm being reset.
- The programmable interface registers, through which the application reads or writes the current time and alarm settings, are part of the RTC core voltage. In order to set the current time and/or alarm, software writes into the shadow registers in the RTC core voltage. The data is then transferred into the corresponding register in the RTC IO voltage by hardware.
- The RTC core voltage runs primarily on the processor's peripheral clock while the RTC IO voltage runs primarily on a self generated 1 Hz clock. The synchronization circuitry sits inside the RTC core voltage.
- The stopwatch circuitry is inside the RTC core voltage and operates on a 1 Hz clock, level shifted from the RTC IO voltage.

Writes to the 1 Hz Registers

Writes to the RTC 1 Hz registers are synchronized to the 1 Hz RTC clock. When setting the time of day, do not factor in the delay when writing to the RTC 1 Hz registers. The most accurate method of setting the RTC is to monitor the seconds (1 Hz) event flag or to program an interrupt for this event and then write the current time to the RTC status register (`RTC_STAT`) in the interrupt service routine (ISR). The new value is inserted ahead of the

incremented value. Hardware adds one second to the written value (with appropriate carries into minutes, hours and days) and loads the incremented value at the next 1 Hz tick, when it represents the then-current time.

Writes posted at any time are properly synchronized to the 1 Hz clock. Writes complete at the rising edge of the 1 Hz clock. A write posted just before the 1 Hz tick may not be completed until the 1 Hz tick one second later.

Reads From the 1 Hz Registers

There is no latency when reading 1 Hz registers, as the values come from the shadow registers. The shadows are updated and ready for reading by the time any RTC interrupts or event flags for that second are asserted. Once the internal core logic completes its initialization sequence after SCLK0 starts, there is no point in time when it is unsafe to read the 1 Hz registers (for synchronization reasons). The registers always return coherent values, although the values may be unknown.

RTC Operating Modes

The following sections provide information on the operating modes available to the real-time clock.

Alarm

The RTC provides two alarm features, programmed with the RTC alarm register (RTC_ALM). The first is a time of day alarm (hour, minute, and second). When the alarm interrupt is enabled, the RTC generates an interrupt each day at the time specified.

Day Alarm

The second alarm feature allows the application to specify a day as well as a time. When the day alarm interrupt is enabled, the RTC generates an interrupt on the day and time specified. The alarm interrupt and day alarm interrupt can be enabled or disabled independently.

Stopwatch

The RTC stopwatch count register (RTC_STPWTCH) contains the countdown value for the stopwatch. The stopwatch counts down seconds from the programmed value and generates an interrupt (if enabled) when the count reaches 0. The counter stops counting at this point and does not resume counting until a new value is written to the RTC_STPWTCH register. Once running, the counter may be overwritten with a new value. This allows the stopwatch to be used as a watchdog timer with a precision of one second.

The stopwatch can be programmed to any value between 0 and $(2^{16} – 1)$ seconds, which is a range of 18 hours, 12 minutes, and 15 seconds. Typically, software should wait for a 1 Hz tick, then write the RTC_STPWTCH register. One second later, the RTC_STPWTCH value changes to the new value and begins decrementing. Because the register write occupies nearly one second, the time from writing a value of N until the stopwatch interrupt is nearly N + 1 seconds. To produce an exact delay, software can compensate by writing N – 1 to get a delay of nearly N seconds. This implies that a delay of 1 second with the stopwatch cannot be achieved. Writing a value of 1 immediately after a 1 Hz tick results in a stopwatch interrupt nearly two seconds later. To wait one second, software should just wait for the next 1 Hz tick

Digital Watch Mode

The primary function of the RTC is to maintain an accurate day count and time of day. The RTC accomplishes this by means of four counters:

- 60-second counter
- 60-minute counter
- 24-hour counter
- 32768-day counter

The RTC provides a set of digital watch features. The internal oscillator generates a 32768 Hz signal using the crystal which is scaled down to 1 Hz and used to clock the second, minute, hour and day counters. The 32768 day counter increments each day at midnight (during the change from 23:59:59). The counter operates on the RTC supply (either the external battery or I/O supply) and is active irrespective of the status of the processor core supply (VDD_INT). When the processor core and I/O supply are valid then the following occurs.

- The current time is updated every second into the RTC clock register (RTC_CLK)
- Interrupts can be issued periodically every second, every minute, every hour or every day

Each of the interrupts can be independently controlled, and are described in the RTC Event Control section.

It is the responsibility of the program to set the correct time by a software write into the RTC_CLK register. Once set, the counters maintain time as long as the RTC supply is valid.

Calibration for Accuracy

To guard against the possibility of long term (> 1 day) errors, the RTC provides a calibration feature using 4 bits of the RTC_INIT register (not available for the stopwatch function).

This is a simple a time correction at the end of every day (when the clock register changes from a Day:Hour:Min:Sec value of XXX:23:59:59 to YYY:00:00:00). It functions by adding or subtracting an integer number of seconds (to a maximum of 7) from the start of the next day, to correct accumulated time error over the course of the previous day. The number of seconds that are added or subtracted is defined in the RTC_INIT.CAL bit field.

As an example, if there is a –50 ppm error in the 1 Hz frequency, this translates into a 86400 ´ 50 ppm seconds (+4.32 seconds) error. That is at time 00:00:00, RTC time is 4.32 seconds ahead of actual time. The RTC can correct this by adding 4 seconds (if 4 is the value written into the calibration register) to the time at 00:00:00. Therefore, from 23:59:59, the timer counter jumps directly to 00:00:04, (there is no 00:00:00 to 00:00:03 time occurrences). At the instant it jumps to 00:00:04, the error reduces to +0.32 seconds over the course of the day, which is only 3.7 ppm.

As a second example, if there is a +50 ppm error in the 1 Hz frequency, this also translates into 86400 ´ 50 ppm seconds (–4.32 seconds) error. In this case the time must be subtracted. This is corrected in the RTC by counting 00:00:00 to 00:00:03 twice, so that the time is effectively subtracted. As soon as the RTC reaches 00:00:04, the error reduces to –0.32 seconds over the course of the previous day and accumulated error is minimized.

When the RTC is powered up for the first time, the calibration values are written once to ensure proper function (if they are not to be used, write 0000). These register bits are sticky, which means that once set, they retain their value irrespective of the status of the core power supply.

The addition or subtraction of time can only be in integer multiples of seconds. Zero to seven seconds can be added or subtracted using 4 bits. The MSB indicates addition (0) or subtraction (1). The three LSBs indicate number of seconds (0 – 7 represented by their binary 3 bit equivalents). Because the clock runs at a time period of one second (@ 1 Hz frequency) 0.25 or 0.5 second resolution is not possible.

The calibration technique introduces a guard band for alarm by definition. In case the alarm is set within the duration of the time (Day:00:00:00 to Day:00:00:06) corrected by the calibration register, then it occurs at the nearest corrected time. This is shown in the following two examples.

If the `RTC_INIT.CAL` is 0101, the RTC clock jumps from 23:59:59 to 00:00:05 due to calibration. If the alarm is set to 00:00:01, it occurs at the RTC time 00:00:05.

If the `RTC_INIT.CAL` is 1101, then the RTC clock counts from 00:00:00 to 00:00:04 twice and then moves to 00:00:05. If the alarm is set to 00:00:01, it occurs at the RTC time 00:00:05.

Accuracy

In order to perform calibration on the bench, use the RTXI pin and check the ppm deviation from 32.768 kHz. This ppm error is the same as in the internal 1Hz clock and the calibration register should be updated with the corresponding values as explained in Calibration for Accuracy above.

Note that total accuracy is ≤ ±35 ppm, < ±1.5 minutes per month of error, inclusive of any inaccuracies of the RTC input crystal at room temperature. This is achieved with a crystal of ±10 ppm error at 25°C. A crystal error of ±20 ppm translates into a maximum inaccuracy of ±45 ppm.

RTC Event Control

The RTC generates multiple events depending on the value on `RTC_CLK`, `RTC_ALM` and `RTC_STPWTCH` registers. It generates an event on each second, minute, hour, day. It also generates an event when an alarm or day alarm or stopwatch expiry occurs.

RTC Events

The RTC can provide interrupts at several programmable intervals

- Per Second
- Per minute (clock counter x:y:z:59)
- Per hour (clock counter x:y:59:59)
- Per day (clock counter x:23:59:59)
- Everyday at a particular time (day alarm)
- On a particular day and time (time of the day alarm)

- Expiry of Stop Watch counter
- When the 1 Hz clock from RTC HV fails

The RTC can also be programmed to provide an interrupt at the completion of all pending writes to any of the RTC 1Hz registers (RTC_ALM, RTC_CLK, RTC_INIT and RTC_STPWTCH). Any of these interrupts can be individually enabled/disabled through the bits in RTC_IEN register.

Programs can disable all the RTC interrupts when the processor enters emulation space by setting the RTC_IEN.EMUDIS bit. Interrupts are not generated even if the individual interrupt enable bits in the RTC_IEN register are set.

In the service routine, the RTC_STAT register should be read to identify the cause of the interrupt. While reading the status register the RTC automatically clears the respective status bit, ensuring that the cause has been cleared before ending the routine. Note that the pending RTC interrupt is cleared whenever all enabled and set (=1) bits in the RTC_STAT register are read, or when all bits in the RTC_IEN register corresponding to pending events are cleared (=0).

RTC Status and Error Signals

The RTC contains a status register (RTC_STAT) that provides the status on the errors and other events generated by the module. The module captures the 1 Hz clock failures in this register as well. The RTC_STAT register contains the RTC event flags and RTC interrupt status. Once set by the event, each bit remains set until cleared by a software read of this register. These sticky bits are independent of the interrupt enable bits in the RTC_IEN register. Values are cleared by reading RTC_STAT register, except for the RTC_STAT.WRPEND, RTC_STAT.ALM and RTC_STAT.DAYALM bits. Writes to the RTC_STAT register have no effect.

RTC Programming Model

The following sections provide basic programming steps for the RTC interface.

Power-Up, Power-Down and Reset

The RTC_INIT.PWDN programmable bit is provided to power down the RTC. Power-down is interpreted as a crystal oscillator disable, which reduces power dissipation to only leakage current. Once set or reset, this bit retains its value unless changed, irrespective of the status of core supply.

The inclusion of the power-down bit (RTC_INIT.PWDN) as well as the possibility that the RTC may not be used in certain applications introduces specific constraints on the power-up and reset behavior of the RTC. These are described below.

1. When the RTC is powered-up for the first time, it remains in an undefined state until the core powers-up and the corresponding RTC_INIT.PWDN bit is written by software. Programs should clear (=0) the RTC_INIT.PWDN bit if the RTC function is desired and set it (=1) if it is not.

2. After clearing the RTC_INIT.PWDN bit the application must wait at least until the first seconds' event before it writes the timer and alarm registers. This is because the oscillator has a startup time before the clock is generated.

   This sequence applies only to the first time the RTC supply (battery or I/O) is connected. Once the RTC_INIT.PWDN bit is set or reset, its value is retained as long as RTC supply (battery or I/O) is valid.

3. After the RTC supply is connected for the first time and the RTC_INIT.PWDN bit =0, the application is free to power-up and power-down the core supply any number of times without loss of RTC function (provided the RTC supply, battery or I/O, is valid). Conversely, if the RTC_INIT.PWDN bit =1, then the RTC oscillator remains disabled irrespective of the status of the core supply.

4. The current status of the RTC power-down is updated by hardware into the initialization status register (RTC_INITSTAT.PWDN). This is useful when the rest of the processor wakes up from power-down and needs to know the status of the RTC.

5. Whenever the processor core wakes up from power-down, the values of the RTC_CLK, RTC_ALM and RTC_STPWTCH registers is zero until the first seconds' event after power-up. At the first seconds' event, an arbitrary value is uploaded into these registers. To put them in a defined state software must write the desired value into these registers. If the RTC_CLK and RTC_ALM registers have been set before core power-down and subsequent power-up, their values are valid throughout, but can be read by the program only after the first seconds' event after power-up.

Register Access

The interface to the RTC is through a set of memory-mapped registers. The RTC is configured through software and the current state of the RTC is also determined through reads and writes to these registers. Writes of the alarm, clock, stopwatch and initialization registers is performed in a two step sequence.

1. The desired values are programmed into a shadow register in the processor's core VDD_INT domain and operating on the processor's peripheral clock.

2. The contents of the shadow register are synchronized onto the contents of the RTC's internal clock register which operates on the 1 Hz clock in the RTC power domain.

To ensure that writes between the core voltage and RTC voltage domain are properly synchronized, all write commands should be issued immediately after a seconds' event in the RTC status register (RTC_STAT). This two step sequence results in a write latency of up to 1 second.

While the write sequence is ongoing, the write pending (RTC_STAT.WRPEND) bit is set and is cleared by hardware when the process is complete. Resetting or powering down the peripherals while a write is in progress, (that is when this bit is set) is forbidden. Subsequent writes to the same register before completion of the previous write are ignored.

- Do not attempt write to any of RTC_CLK, RTC_ALM or RTC_STPWTCH registers when the RTC oscillator is powered down or when the RTC_INIT.RDEN bit is set.

- During initialization, after a write of the RTC_INIT register, make sure that the RTC_STAT.WRPEND bit is cleared before attempting writes to other registers.

The RTC_INIT register can be written any time. However, programs must ensure that the interrupt enable bits corresponding to Clock, Alarm and Stopwatch features are set only after initializing the RTC_CLK, RTC_ALM and RTC_STPWTCH registers. The RTC_STAT and RTC_INITSTAT registers can be read any time.

ADSP-BF70x RTC Register Descriptions

Real Time Clock (RTC) contains the following registers.

Table 23-6: ADSP-BF70x RTC Register List

Name	Description
RTC_ALM	RTC Alarm Register
RTC_CLK	RTC Clock Register
RTC_IEN	Interrupt Enable Register
RTC_INIT	RTC Initialization Register
RTC_INITSTAT	RTC Initialization Status Register
RTC_STAT	RTC Status Register
RTC_STPWTCH	RTC Stop Watch Register

RTC Alarm Register

The RTC_ALM register is programmed by software for the time (in hours, minutes, and seconds) the alarm interrupt occurs. Reads and writes can occur at any time. The alarm interrupt occurs whenever the hour, minute, and second fields first match those of the RTC_CLK register. The day interrupt occurs whenever the day, hour, minute, and second fields first match those of the RTC_CLK status register.

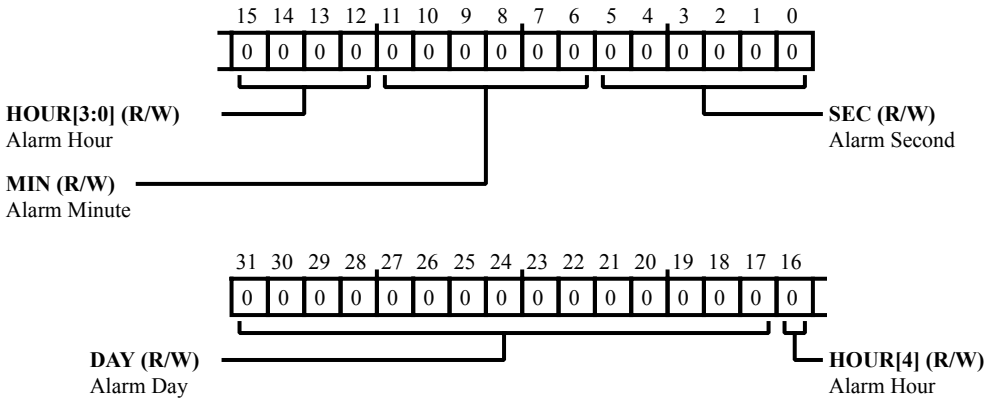

Figure 23-3: RTC_ALM Register Diagram

Table 23-7: RTC_ALM Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:17 (R/W)	DAY	Alarm Day. The RTC_ALM.DAY bit provides the alarm day from 0 to 32767. The day interrupt occurs whenever the day, hour, minute, and second fields first match those of the RTC status register.
16:12 (R/W)	HOUR	Alarm Hour. The RTC_ALM.HOUR bit field configures the alarm to occur whenever the alarm hour (0 to 23) first matches that of the RTC status register.
11:6 (R/W)	MIN	Alarm Minute. The RTC_ALM.MIN bit field configures the alarm to occur whenever the alarm minute (0 to 59) first matches that of the RTC status register.
5:0 (R/W)	SEC	Alarm Second. The RTC_ALM.SEC bit field configures the alarm to occur whenever the seconds (0 to 59) first match those of the RTC status register.

RTC Clock Register

The RTC_CLK register is used to read or write the current time. It has no reset and an undefined value when the RTC VDD is first powered up. Writing invalid time values is forbidden (for example, an hour value more than 23 and a minute value more than 59). The RTC_CLK register is updated every second. If the RTC is already running when the core starts up, the values read from RTC_CLK are zero until the first second event comes. In this case, programs must wait for the second event and then read this register.

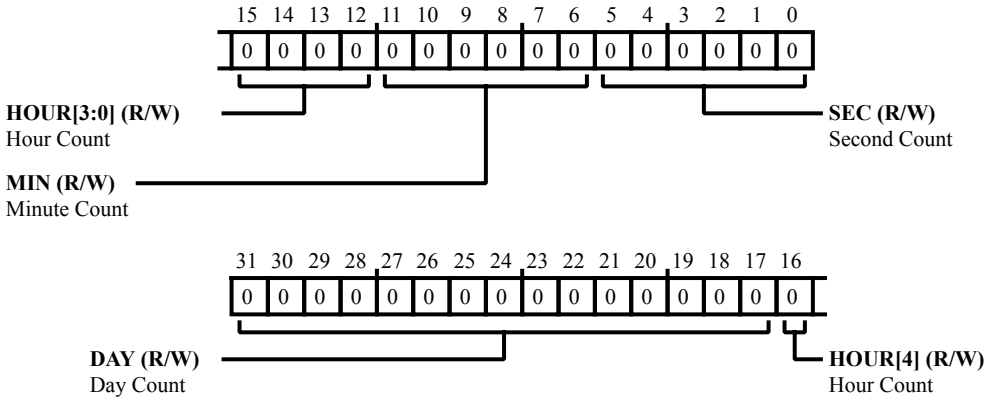

Figure 23-4: RTC_CLK Register Diagram

Table 23-8: RTC_CLK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:17 (R/W)	DAY	Day Count. The RTC_CLK.DAY bit provides the day count from 0 to 32767.
16:12 (R/W)	HOUR	Hour Count. The RTC_CLK.HOUR bit provides the hour count from 0 to 23.
11:6 (R/W)	MIN	Minute Count. The RTC_CLK.MIN bit provides the minute count from 0 to 59.
5:0 (R/W)	SEC	Second Count. The RTC_CLK.SEC bit provides the second count from 0 to 59.

Interrupt Enable Register

The RTC_IEN register enables interrupts (when the bits are set) or disables interrupts (when bits are cleared).

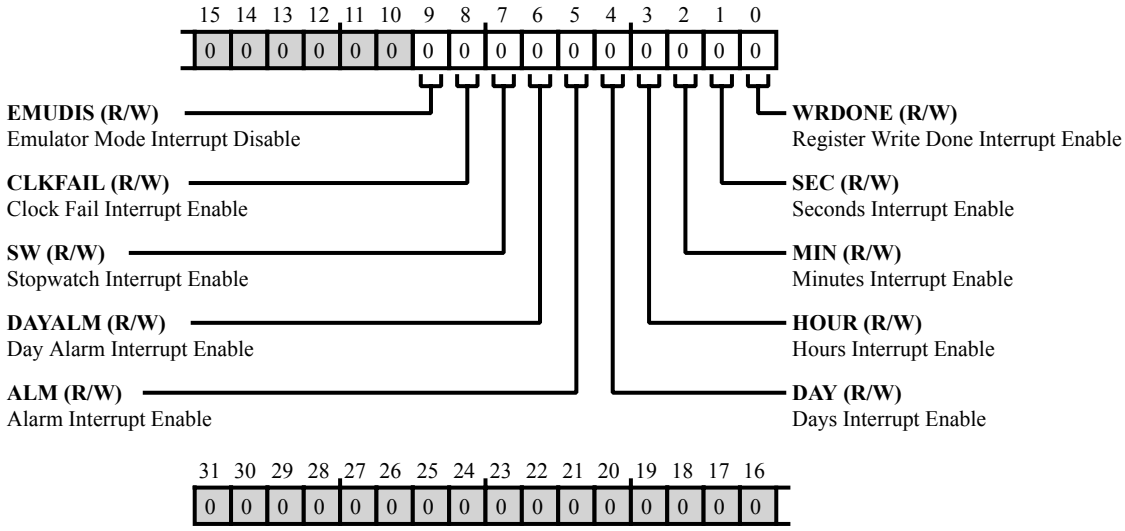

Figure 23-5: RTC_IEN Register Diagram

Table 23-9: RTC_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	EMUDIS	Emulator Mode Interrupt Disable. The RTC_IEN register disables or enables RTC interrupts in emulation mode. Interrupts are not generated even if individual interrupt enable bits in the RTC_IEN register are set.	
		0	Enable interrupt
		1	Disable interrupt
8 (R/W)	CLKFAIL	Clock Fail Interrupt Enable. The RTC_IEN.CLKFAIL bit enables the RTC 1Hz clock fail interrupt.	
		0	Disable interrupt
		1	Enable interrupt
7 (R/W)	SW	Stopwatch Interrupt Enable. The RTC_IEN.SW bit enables the stopwatch interrupt.	
		0	Disable interrupt
		1	Enable interrupt

Table 23-9: RTC_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	DAYALM	Day Alarm Interrupt Enable. The RTC_IEN.DAYALM bit enables the alarm (day, hour, minute, second) interrupt.	
		0	Disable interrupt
		1	Enable interrupt
5 (R/W)	ALM	Alarm Interrupt Enable. The RTC_IEN.ALM bit enables the alarm (hour, minute, second) interrupt.	
		0	Disable interrupt
		1	Enable interrupt
4 (R/W)	DAY	Days Interrupt Enable. The RTC_IEN.DAY bit enables the days interrupt.	
		0	Disable interrupt
		1	Enable interrupt
3 (R/W)	HOUR	Hours Interrupt Enable. The RTC_IEN.HOUR bit enables the hours interrupt.	
		0	Disable interrupt
		1	Enable interrupt
2 (R/W)	MIN	Minutes Interrupt Enable. The RTC_IEN.MIN bit enables the minutes interrupt.	
		0	Minute interrupt disabled
		1	Enable interrupt
1 (R/W)	SEC	Seconds Interrupt Enable. The RTC_IEN.SEC bit enables the seconds interrupt.	
		0	Disable interrupt
		1	Enable interrupt
0 (R/W)	WRDONE	Register Write Done Interrupt Enable. The RTC_IEN.WRDONE bit enables the interrupt for register write completion. The RTC_IEN.WRDONE bit is applicable only for the alarm, clock, and stopwatch registers.	
		0	Disable interrupt
		1	Enable interrupt

RTC Initialization Register

The RTC_INIT register provides the calibration function, powers down the unit, and disables the output buses.

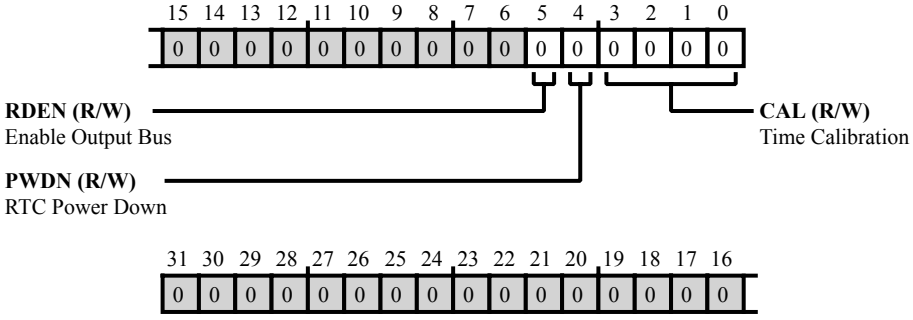

Figure 23-6: RTC_INIT Register Diagram

Table 23-10: RTC_INIT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	RDEN	Enable Output Bus. The RTC_INIT.RDEN bit disables the output bus.	
		0	Enable output bus
		1	Disable output bus
4 (R/W)	PWDN	RTC Power Down. The RTC_INIT.PWDN bit powers down the RTC module. For complete information on this function, see the "Power-Up, Power-Down and Reset" section of this chapter.	
		0	Power up
		1	Power down
3:0 (R/W)	CAL	Time Calibration. The RTC_INIT.CAL bit performs calibration the RTC module. For complete information on this function, see the "Calibration for Accuracy" section of this chapter.	

RTC Initialization Status Register

The RTC_INITSTAT register contains values of various status bits which can be used to check the status of RTC when the core comes out of reset.

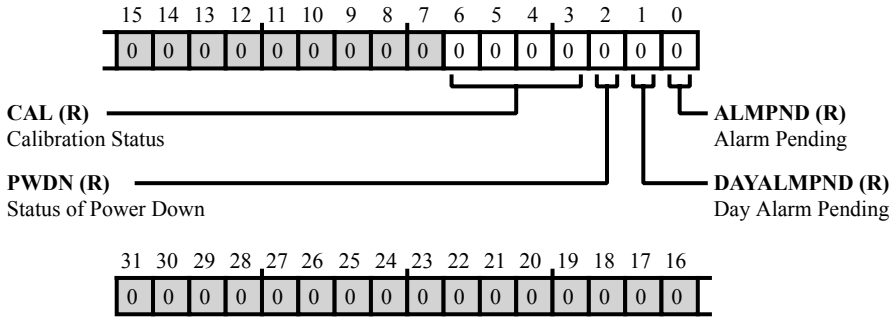

Figure 23-7: RTC_INITSTAT Register Diagram

Table 23-11: RTC_INITSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
6:3 (R/NW)	CAL	Calibration Status.	
2 (R/NW)	PWDN	Status of Power Down. The RTC_INITSTAT.PWDN bit indicates the status of the RTC.	
		0	Oscillator is powered down
		1	Oscillator is running
1 (R/NW)	DAYALMPND	Day Alarm Pending. The RTC_INITSTAT.DAYALMPND bit indicates that an alarm has occurred. This indication is useful when the core has powered down or reset in the middle of operation. This bit is cleared when the RTC_INITSTAT register is read.	
		0	Alarm occurred
		1	No alarm occurred
0 (R/NW)	ALMPND	Alarm Pending. The RTC_INITSTAT.ALMPND bit indicates that an alarm has occurred. This indication is useful when the core has powered down or reset in the middle of operation. This bit is cleared when the RTC_INITSTAT register is read.	
		0	No alarm occurred
		1	Alarm occurred

RTC Status Register

The RTC_STAT register contains the RTC event flags and RTC interrupt status. These bits are sticky. Once set by the event, each bit remains set until cleared by a software read. These sticky bits are independent of the interrupt enable bits in the RTC_IEN register. Values are cleared by reading the RTC_STAT register, except for the RTC_STAT.WRPEND bit, which is read-only. Writes of 0 or 1 to any bit of this register has no effect. This register is cleared at reset.

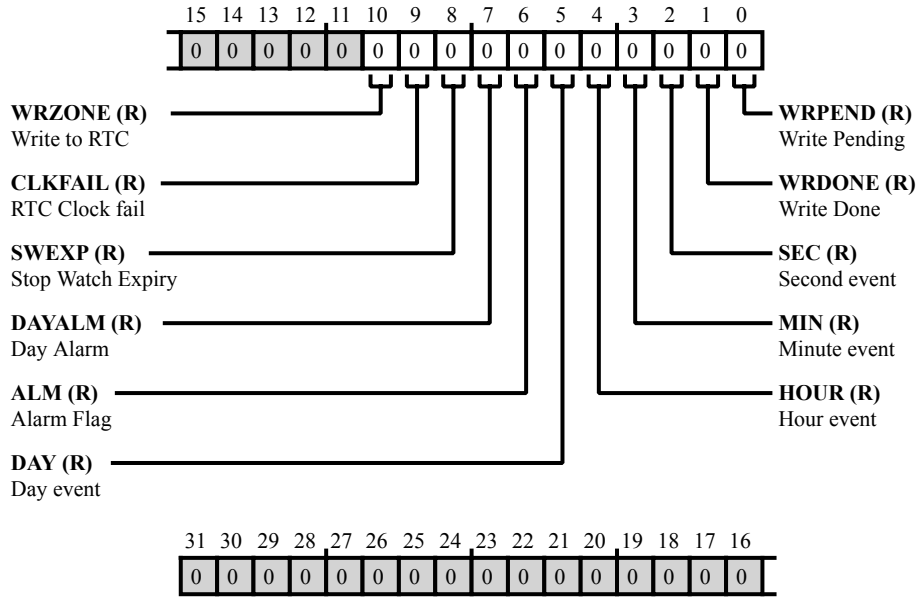

Figure 23-8: RTC_STAT Register Diagram

Table 23-12: RTC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	WRZONE	Write to RTC.	
9 (R/NW)	CLKFAIL	RTC Clock fail. The RTC_STAT.CLKFAIL bit indicates whether the RTC 1 Hz clock is functional. 0 = RTC 1 Hz clock is functional 1 = RTC 1 Hz clock failed	
		0	Clock functioning
		1	Clock failed

Table 23-12: RTC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/NW)	SWEXP	Stop Watch Expiry. 1 = stopwatch counter expired 0 = stopwatch counter running	
		0	Counter running
		1	Counter expired
7 (R/NW)	DAYALM	Day Alarm. The RTC_STAT.DAYALM bit indicates that a time of day alarm has occurred (=1).	
		0	Alarm did not occur
		1	Alarm occurred
6 (R/NW)	ALM	Alarm Flag. The RTC_STAT.ALM bit indicates that an alarm has occurred (=1).	
		0	Alarm did not occur
		1	Alarm occurred
5 (R/NW)	DAY	Day event. The RTC_STAT.DAY bit indicates that a day event (clock counter value x:23:59:59) has occurred (=1).	
		0	Event did not occur
		1	Event occurred
4 (R/NW)	HOUR	Hour event. The RTC_STAT.HOUR bit indicates that a hour event (clock counter value x:y:59:59) has occurred (=1).	
		0	Event did not occur
		1	Event occurred
3 (R/NW)	MIN	Minute event. The RTC_STAT.MIN bit indicates that a minute event (clock counter value x:y:z:59) has occurred (=1).	
		0	Event did not occur
		1	Event occurred
2 (R/NW)	SEC	Second event. The RTC_STAT.SEC bit indicates that a second event has occurred (=1).	
		0	Event did not occur
		1	Event occurred

Table 23-12: RTC_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	WRDONE	Write Done. The RTC_STAT.WRDONE bit shows that the register write is complete. The RTC_STAT.WRDONE bit is applicable only for the alarm, clock and stopwatch registers.	
		0	Write is done
		1	Write is not done
0 (R/NW)	WRPEND	Write Pending. The RTC_STAT.WRPEND bit shows that a write to the RTC_CLK, RTC_ALM, RTC_STPWTCH, or RTC_INIT register is pending. This bit is automatically cleared and set by the hardware.	
		0	No write pending
		1	Write pending

RTC Stop Watch Register

The RTC_STPWTCH register contains the countdown value for the stop watch. The stopwatch counts down seconds from the programmed value and generates an interrupt (if SW_INTEN=1) when the count reaches 0. The counter stops counting at this point and does not resume counting until a new nonzero value is written to the RTC_STPWTCH register. Writing a value of 0 to the running stopwatch forces it to stop; no interrupt is generated in this case. The register can be programmed to any value between 0 and (2^{16-1}) seconds (that is, a range of 18 hours, 12 minutes and 15 seconds).

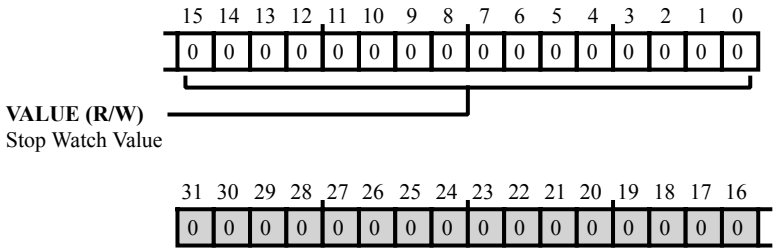

Figure 23-9: RTC_STPWTCH Register Diagram

Table 23-13: RTC_STPWTCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Stop Watch Value. The RTC_STPWTCH register contains the countdown value for the stop watch.

24 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 SCLK0 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 also 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.

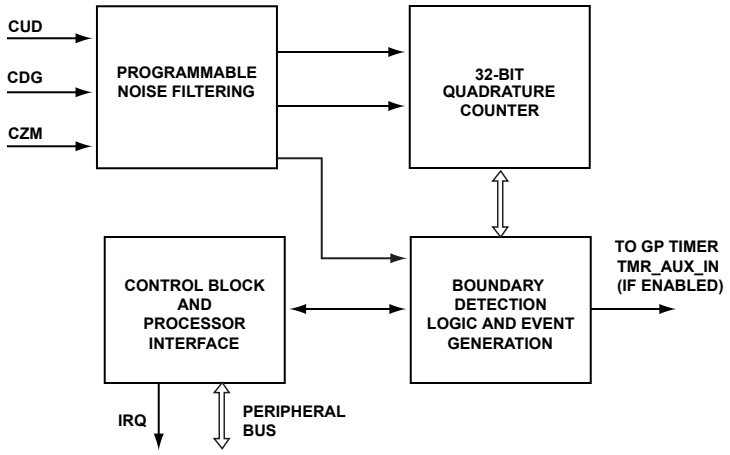

Figure 24-1: GP Counter Block Diagram

ADSP-BF70x 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 24-1: ADSP-BF70x 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_MIN	Minimum Count Register
CNT_STAT	Status Register

ADSP-BF70x CNT Interrupt List

Table 24-2: ADSP-BF70x CNT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
27	CNT0_STAT	CNT0 Status	Level	

ADSP-BF70x CNT Trigger List

Table 24-3: ADSP-BF70x CNT Trigger List Masters

Trigger ID	Name	Description	Sensitivity
13	CNT0_STAT	CNT0 Status	Level

Table 24-4: ADSP-BF70x CNT Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

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 SCLK0 cycle. Gray coding prohibits such transitions. Therefore, the CNT_CNTR register remains unchanged, and an error condition is signaled.

Table 24-5: 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 SCLK0 cycle. Such transitions are erroneous. Therefore, the CNT_CNTR register remains unchanged, and an error condition is signaled.

Table 24-6: 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 SCLK0 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.

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 event controller.
4. If timing information is required, set up the relevant GP Timer in width capture mode.
5. Finally, enable interrupts and the GP Counter itself using the CNT_IMSK and CNT_CFG registers, respectively.

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.

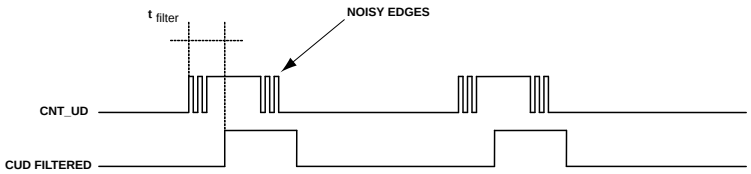

Figure 24-2: Programmable Noise Filtering

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 SCLK0 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 SCLK0 and the `CNT_DEBNCE.DPRESCALE` value, where lower values of `CNT_DEBNCE.DPRESCALE` result in shorter debounce delays:

$t_{filter} = 128 \times (2^{DPRESCALE} \times SCLK0)$

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.

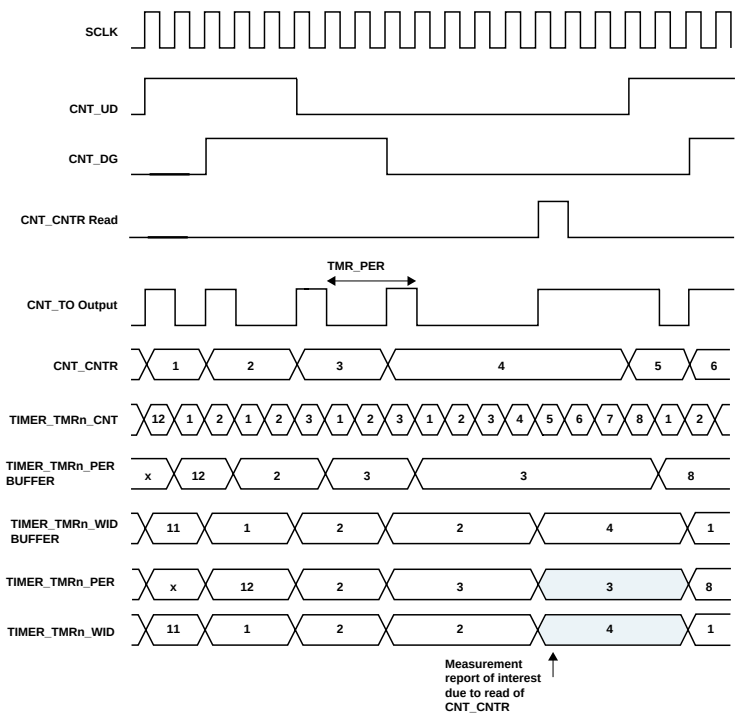

Figure 24-3: Capture Intervals

NOTE: SCLK in the ***Capture Intervals*** figure is SCLK0.

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 SCLK0 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 SCLK0 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.

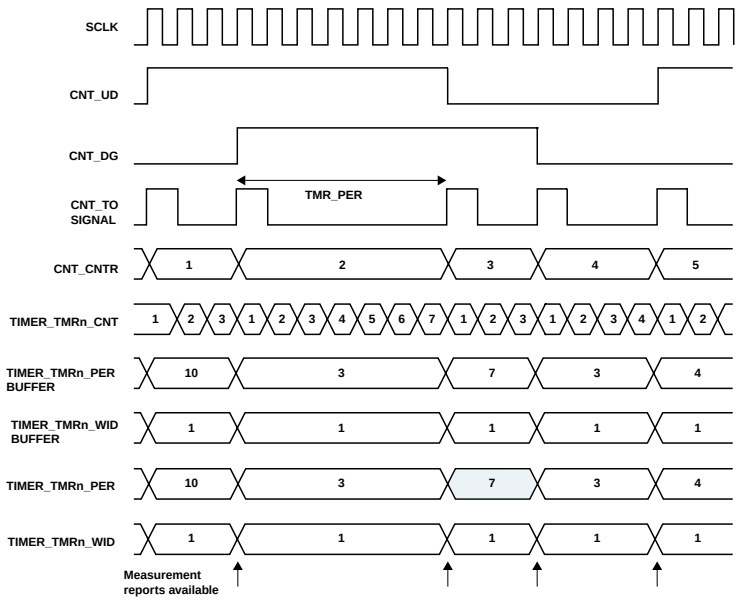

Figure 24-4: Period Register Timing

NOTE: SCLK in the ***Period Register Timing*** figure is SCLK0.

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 SCLK0 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 SCLK0 cycle later.

GP Counter Event Control

11 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 interrupts. It uses dedicated bits in the CNT_STAT register to report events. When an interrupt from the GP counter is acknowledged, 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 interrupts.

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 SCLK0 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 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.

ADSP-BF70x CNT Register Descriptions

CNT (CNT) contains the following registers.

Table 24-7: ADSP-BF70x 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_MIN	Minimum Count Register
CNT_STAT	Status Register

Configuration Register

The CNT_CFG register configures counter modes, configures input pins, and enables the CNT.

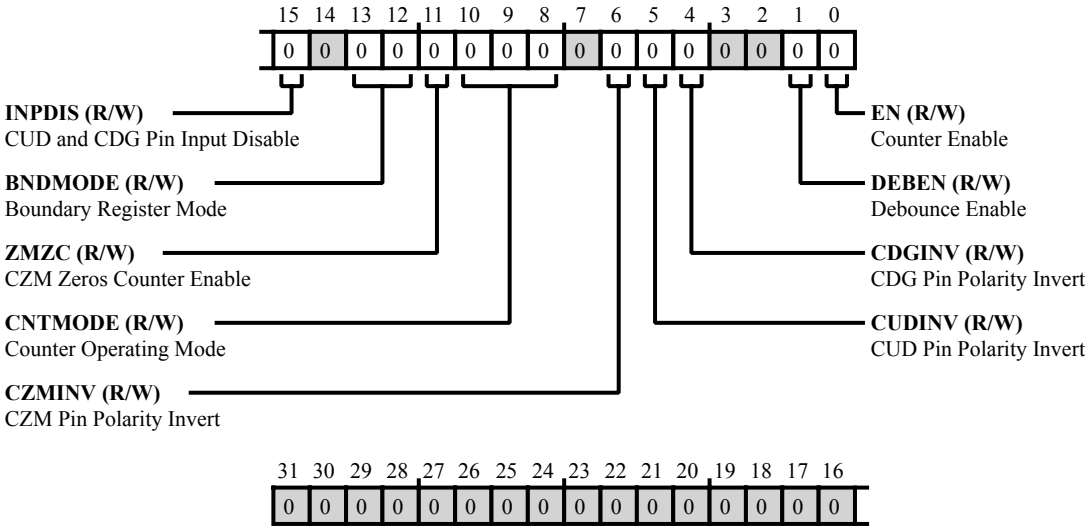

Figure 24-5: CNT_CFG Register Diagram

Table 24-8: 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 24-8: 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
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
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

Table 24-8: CNT_CFG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.

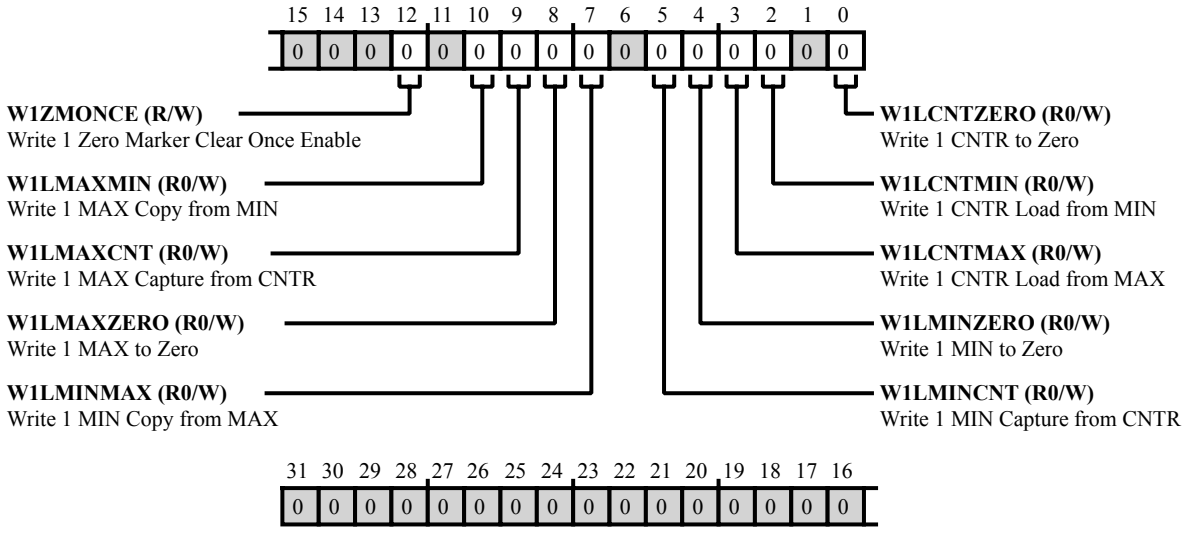

Figure 24-6: CNT_CMD Register Diagram

Table 24-9: 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.
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.

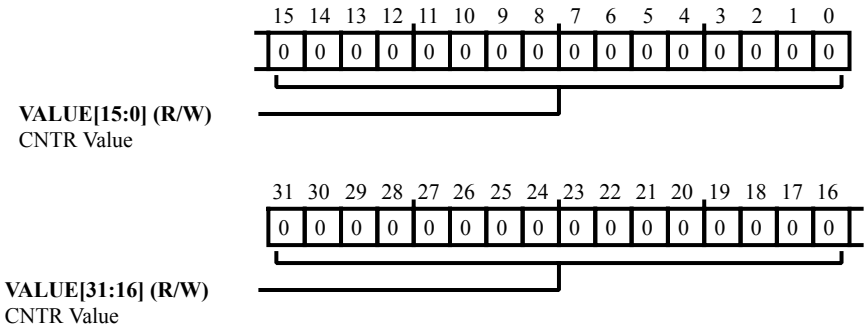

Figure 24-7: CNT_CNTR Register Diagram

Table 24-10: 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 ($2^{DPRESCALE}$/ SCLK0)

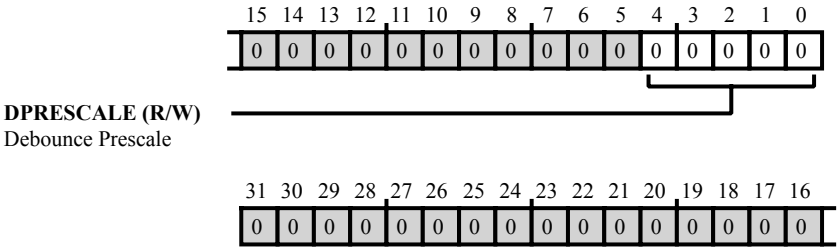

Figure 24-8: CNT_DEBNCE Register Diagram

Table 24-11: 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 SCLK0.	
		0	1x cycles = 128 SCLK0 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 24-11: 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 (if bit cleared) or enable/unmask an interrupt (if bit set).

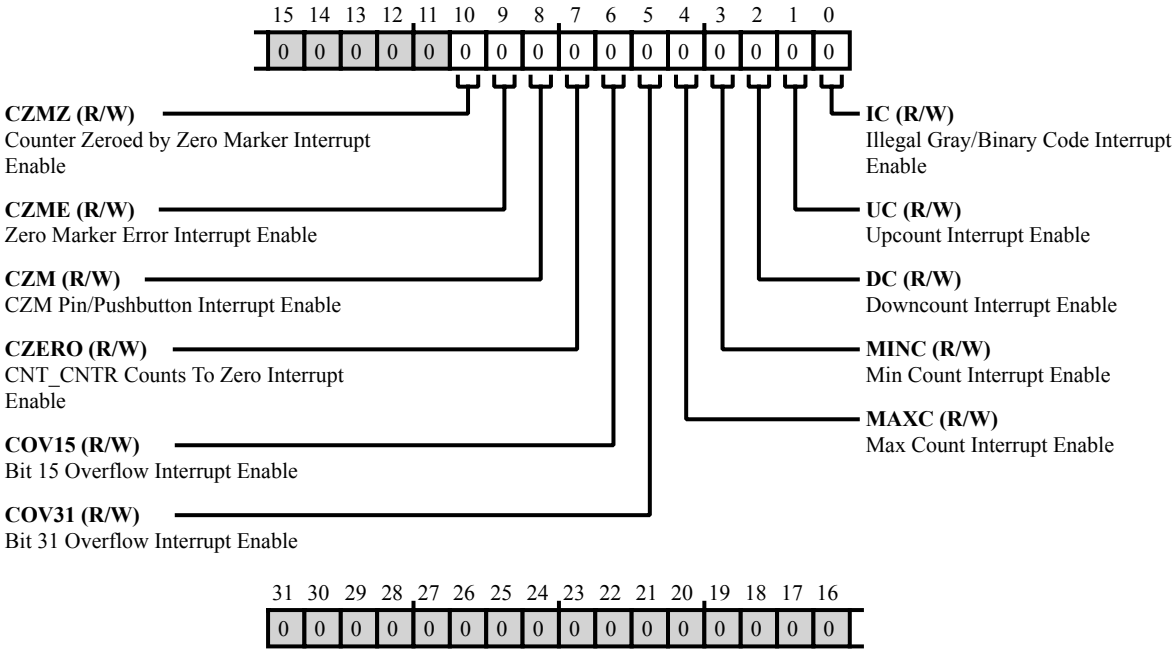

Figure 24-9: CNT_IMSK Register Diagram

Table 24-12: CNT_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
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.	
		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.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 24-12: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	CZM	CZM Pin/Pushbutton Interrupt Enable. The CNT_IMSK.CZM bit enables (unmasks) the CZM pin/pushbutton interrupt.	
		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.	
		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.	
		0	Mask Interrupt
		1	Unmask Interrupt
5 (R/W)	COV31	Bit 31 Overflow Interrupt Enable. The CNT_IMSK.COV31 bit enables (unmasks) the bit 31 overflow interrupt.	
		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.	
		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.	
		0	Mask Interrupt
		1	Unmask Interrupt
2 (R/W)	DC	Downcount Interrupt Enable. The CNT_IMSK.DC bit enables (unmasks) the down count interrupt.	
		0	Mask Interrupt
		1	Unmask Interrupt

Table 24-12: CNT_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	UC	Upcount Interrupt Enable. The `CNT_IMSK.UC` bit enables (unmasks) the up count interrupt.	
		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 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.

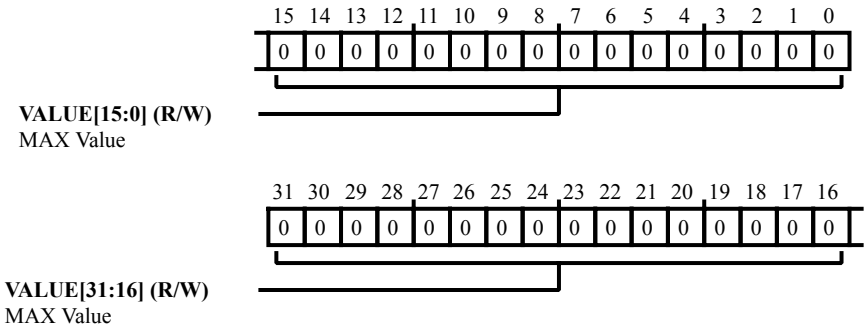

Figure 24-10: CNT_MAX Register Diagram

Table 24-13: 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.

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.

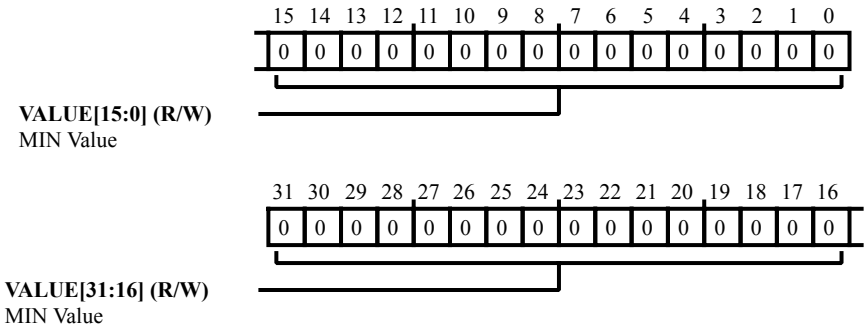

Figure 24-11: CNT_MIN Register Diagram

Table 24-14: 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.

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 pending (if bit cleared) or an interrupt pending (if bit set).

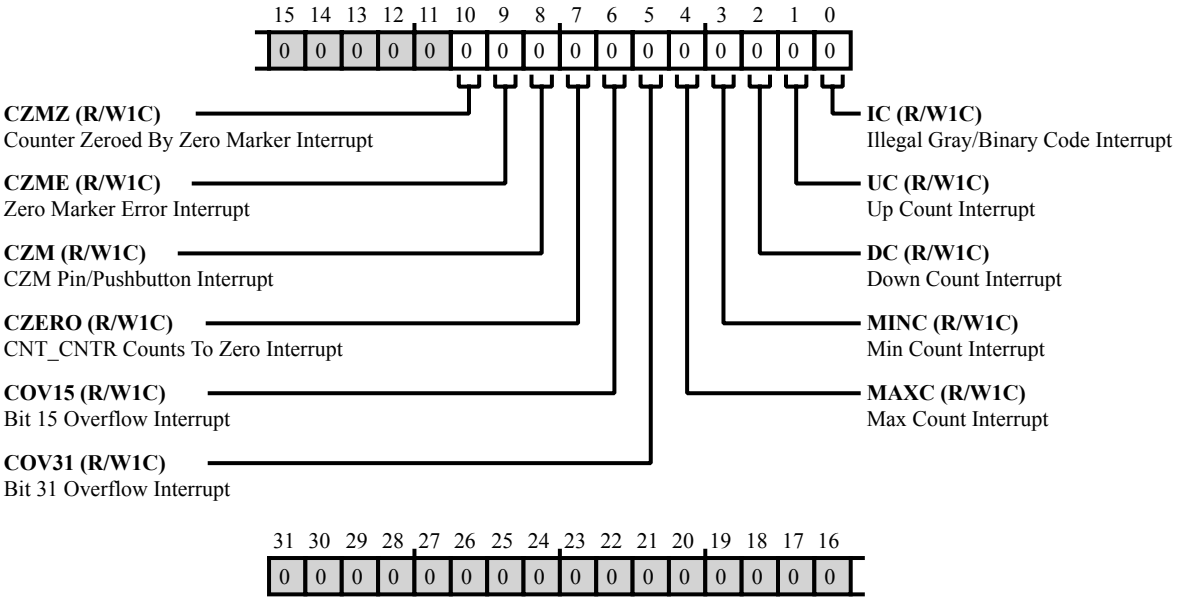

Figure 24-12: CNT_STAT Register Diagram

Table 24-15: CNT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
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 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

Table 24-15: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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 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
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

Table 24-15: CNT_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

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. 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.

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
- Flexible transmit and receive interrupt timing
- 3 interrupt outputs for receive, transmit, and status

- 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	SCLK0/16

UART Functional Description

The following sections provide details on the UARTs functionality.

ADSP-BF70x 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: ADSP-BF70x 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

ADSP-BF70x UART Interrupt List

Table 25-3: ADSP-BF70x UART Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
48	UART0_STAT	UART0 Status	Level	
49	UART0_TXDMA	UART0 Transmit DMA	Level	10
50	UART0_RXDMA	UART0 Receive DMA	Level	11
51	UART1_STAT	UART1 Status	Level	
52	UART1_TXDMA	UART1 Transmit DMA	Level	12
53	UART1_RXDMA	UART1 Receive DMA	Level	13

ADSP-BF70x UART DMA Channel List

Table 25-4: ADSP-BF70x UART DMA Channel List

DMA ID	DMA Channel Name	Description
	UART0_TXDMA	UART0

Table 25-4: ADSP-BF70x UART DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA10		
DMA11	UART0_RXDMA	UART0
DMA12	UART1_TXDMA	UART1
DMA13	UART1_RXDMA	UART1

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.

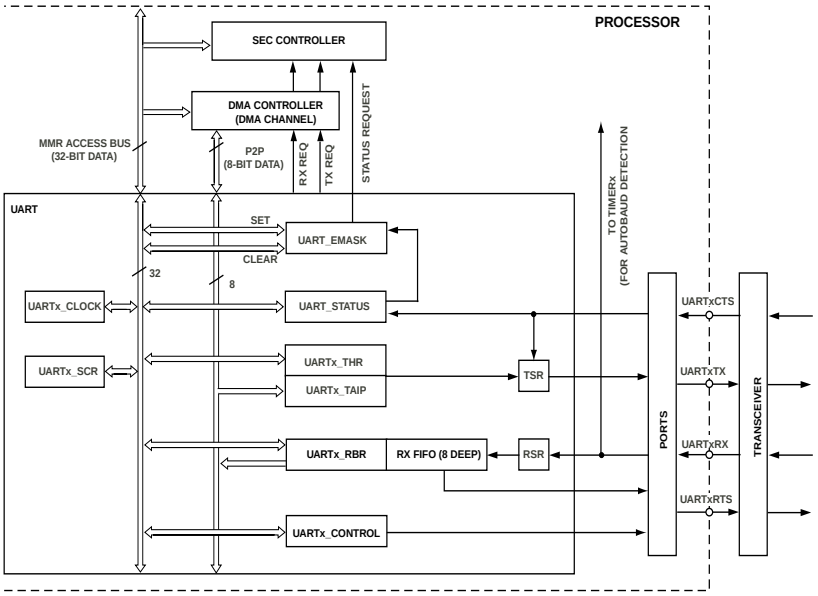

Figure 25-1: UART Block Diagram

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 (8-bit data width).

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 the SEC. 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.

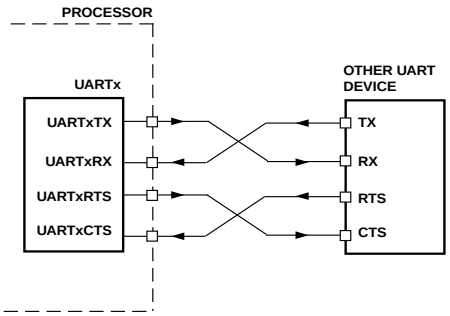

Figure 25-2: UART Hardware Flow Control

In both DMA and core mode, the receiver can de-assert 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 de-asserted 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 (SCLK0) 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:

Bit Rate = SCLK0/16 $^{(1\text{-EDBO})}$ × Divisor

ADSP-BF70x Processor Example

The UART Bit Rate Examples with 100 MHz SCLK0 table provides example divide factors required to support standard baud rates at an SCLK0 of 100 MHz.

Table 25-5: UART Bit Rate Examples with 100 MHz SCLK0

	D Factor = 16			D Factor = 1		
Bit Rate	**DL**	**Actual**	**% Error**	**DL**	**Actual**	**% Error**
2400	2604	2400.15	0.006	41667	2399.98	0.001
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 SCLK0 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 are not always as accurate. Therefore, 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 $\overline{\text{UART\_RX}}$ 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 $\overline{\text{UART\_RX}}$ 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 SCLK0). 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.

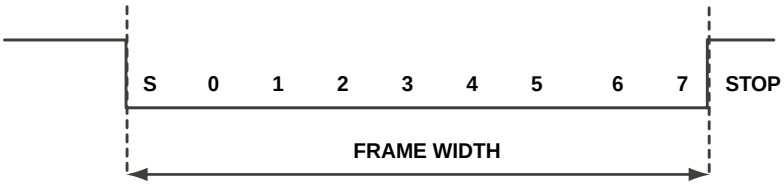

Figure 25-3: Autobaud Detection

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 do not always return adequate results without extra conditioning of the analog signal. Measure signal periods to work around this issue.

For example, predefine 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.

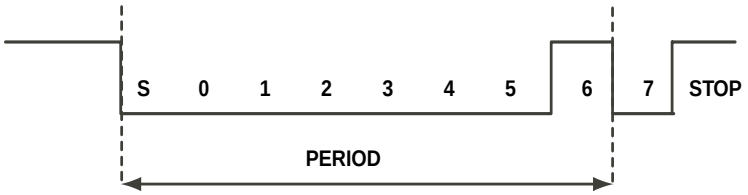

Figure 25-4: Autobaud Detection Character 0x40

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 25-6: 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.

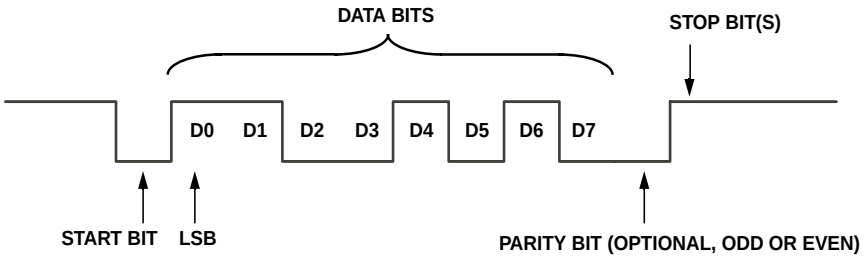

Figure 25-5: Bit Stream on a UART TX Pin Transmitting an "S" Character (0x53)

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.

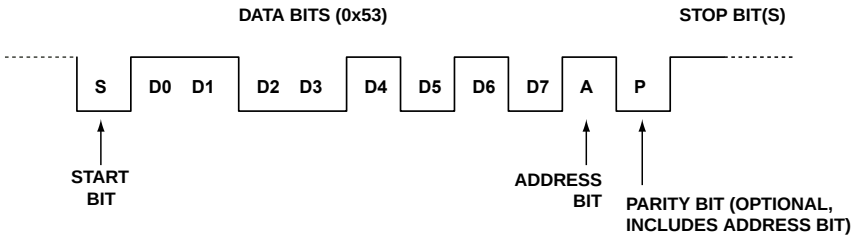

Figure 25-6: UART Frame with Address Bit

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.

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 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_CTL 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 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 0 is transmitted as a high pulse of 16 UART clock periods and a 1 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 0 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.

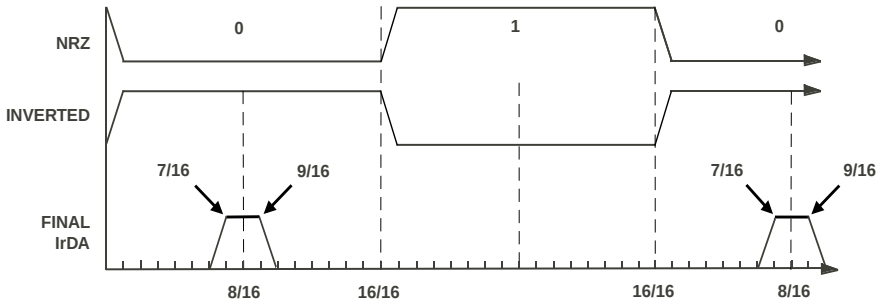

Figure 25-7: IrDA Transmit Pulse

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 0. 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 (SCLK0) 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 SCLK0 can change across systems, the longest glitch tolerated is inversely proportional to the SCLK0 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.

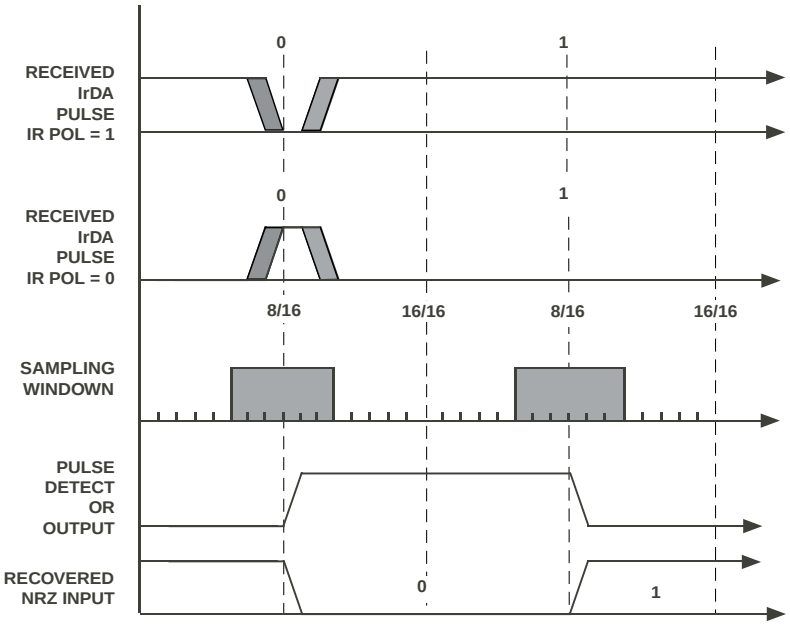

Figure 25-8: IrDA Receiver Pulse Detection

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.

NOTE: The status interrupt for the UART goes directly to the system event controller (SEC), 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 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.DI_EN 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 DI_EN 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 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

Use the following steps 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 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.

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 interrupts 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 event controller (SEC).

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 output. The status interrupt goes directly to the SEC 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 interrupts.

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.

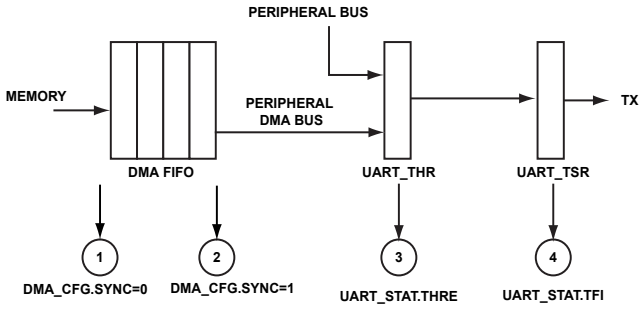

Figure 25-9: Transmit Interrupts

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 SEC. 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.

Polling the SEC_SSTAT[n] register without enabling the interrupts by the SEC_CCTL[n] register is an alternate method of operation to consider. Software can write up to two words into the UART_THR register before enabling the UART clock. As soon as the UART_CTL.EN bit is set, the UART module sends those two words.

Receive Interrupts

The UART module uses the UART_IMSK_SET.ERBFI bit to enable receive-interrupts. 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 SEC. 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.

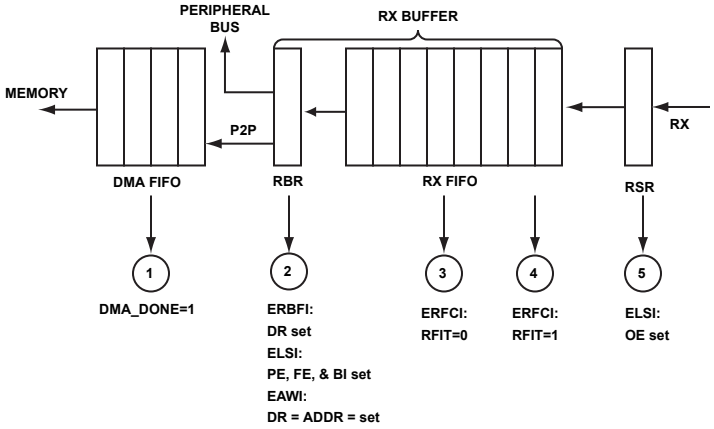

Figure 25-10: Receive Interrupts

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 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 SEC. 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 interrupts
- Flow control interrupts
- Receive FIFO threshold interrupts
- Transmission finished interrupt

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 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 and the UART_STAT.RFCS bit clear 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 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, address sticky (UART_STAT.ASTKY) bit and 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 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. (Words with mode bit =0 are not moved from the UART_RSR to the receive FIFO). This bit does not affect reception in non-MDB modes.

UART Programming Model

The following sections provide basic procedures for configuring various UART operations.

Detecting Autobaud

Refer to Autobaud Detection for more information. The required steps are:

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 PORTs chapter.
2. Program the UART_CLK register. Refer to Bit Rate Generation.
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 system event controller (SEC).
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.

ADSP-BF70x UART Register Descriptions

UART (UART) contains the following registers.

Table 25-7: ADSP-BF70x 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 (SCLK0) down to the bit clock.

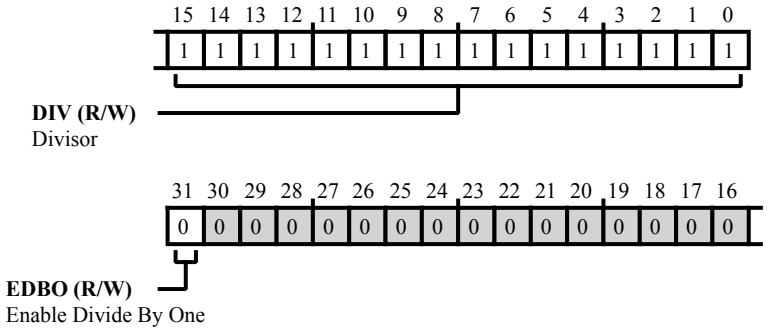

Figure 25-11: UART_CLK Register Diagram

Table 25-8: 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 = SCLK0 / ($16^{(1-EDBo)}$ 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.

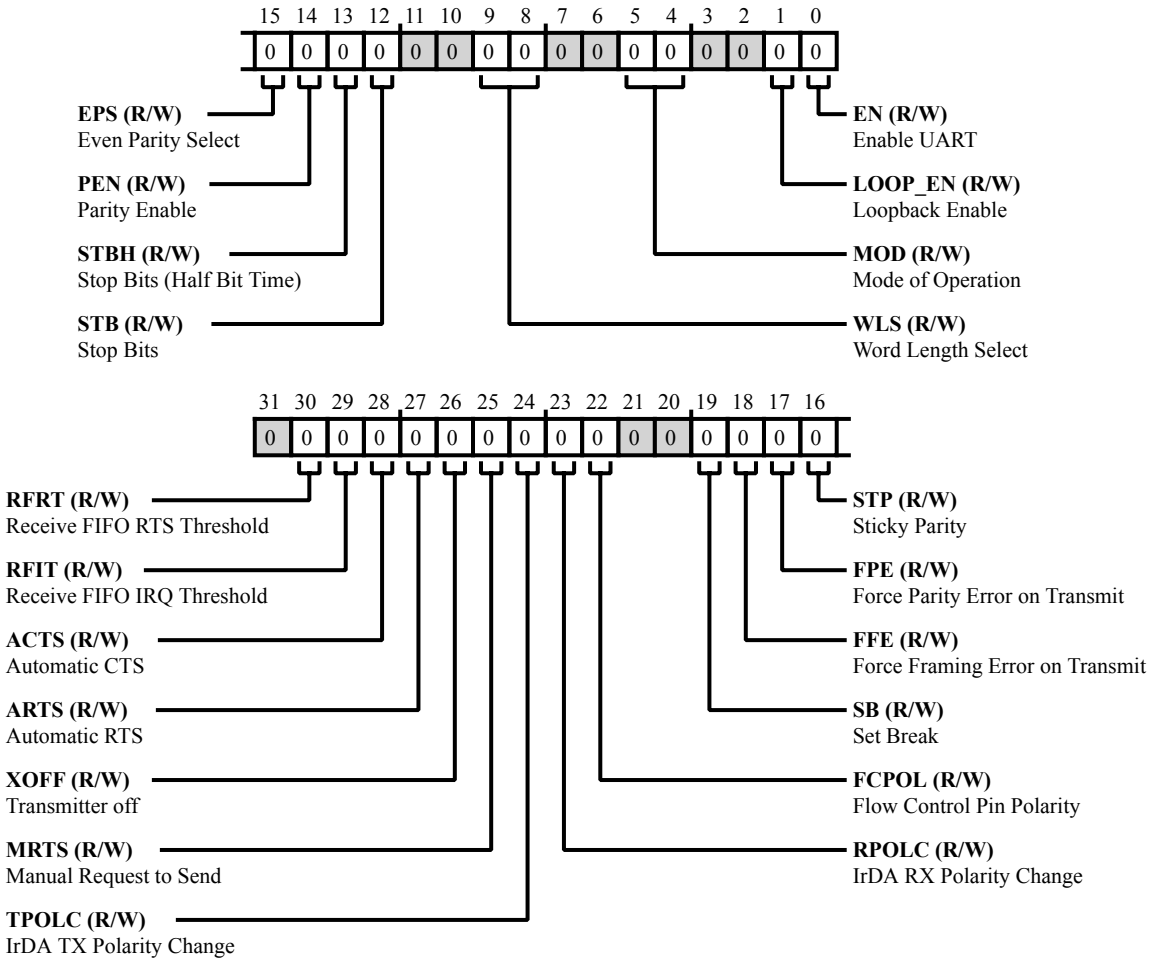

Figure 25-12: UART_CTL Register Diagram

Table 25-9: 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 25-9: 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 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 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 25-9: 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 UART_CTS and UART_RTS pins. When the UART_CTL.FCPOL bit is cleared, the UART_RTS and UART_CTS pins are active low, and the UART is halted when the UART_RTS and UART_CTS pin state is high. When UART_CTL.FCPOL bit is set, the UART_RTS and 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 25-9: 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 25-9: 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 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 the UART_IMSK_SET register enables (unmasks) interrupts, 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 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. However, the error interrupt for the UART goes directly to the system interrupt handling unit, bypassing the DMA unit completely.

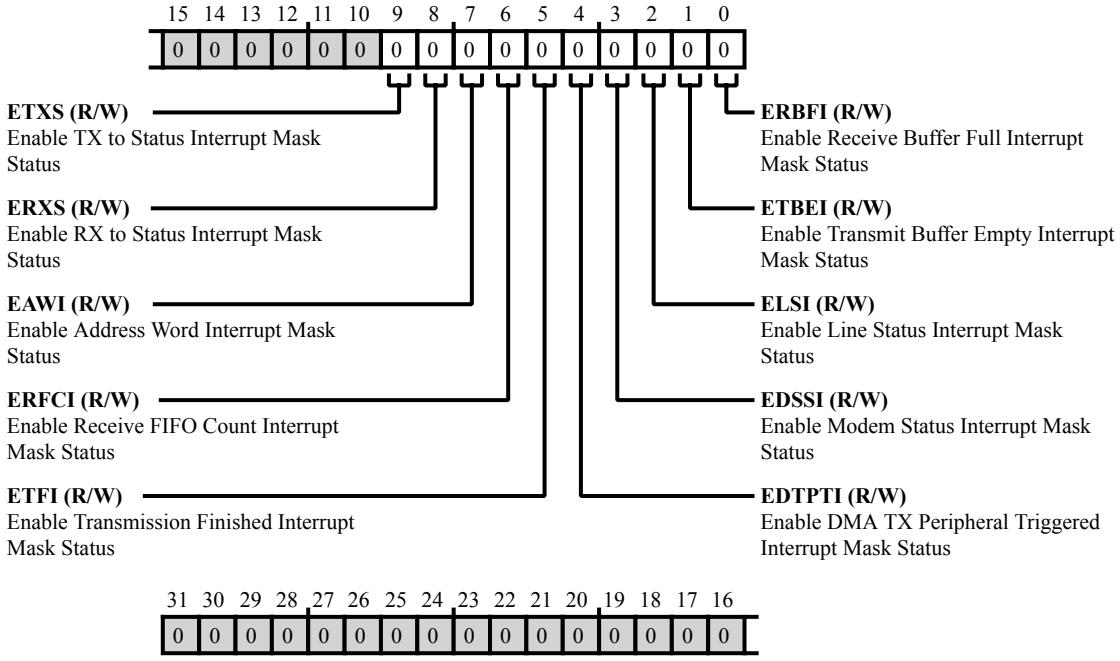

Figure 25-13: UART_IMSK Register Diagram

Table 25-10: 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 interrupts to status interrupt output. If cleared, TX interrupts 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 interrupts to status interrupt output. If cleared, RX interrupts are routed to normal interrupt outputs.	
		0	Interrupt is masked
		1	Interrupt is unmasked

Table 25-10: 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 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 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 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 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, 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 25-10: 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 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 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 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) interrupts, 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.

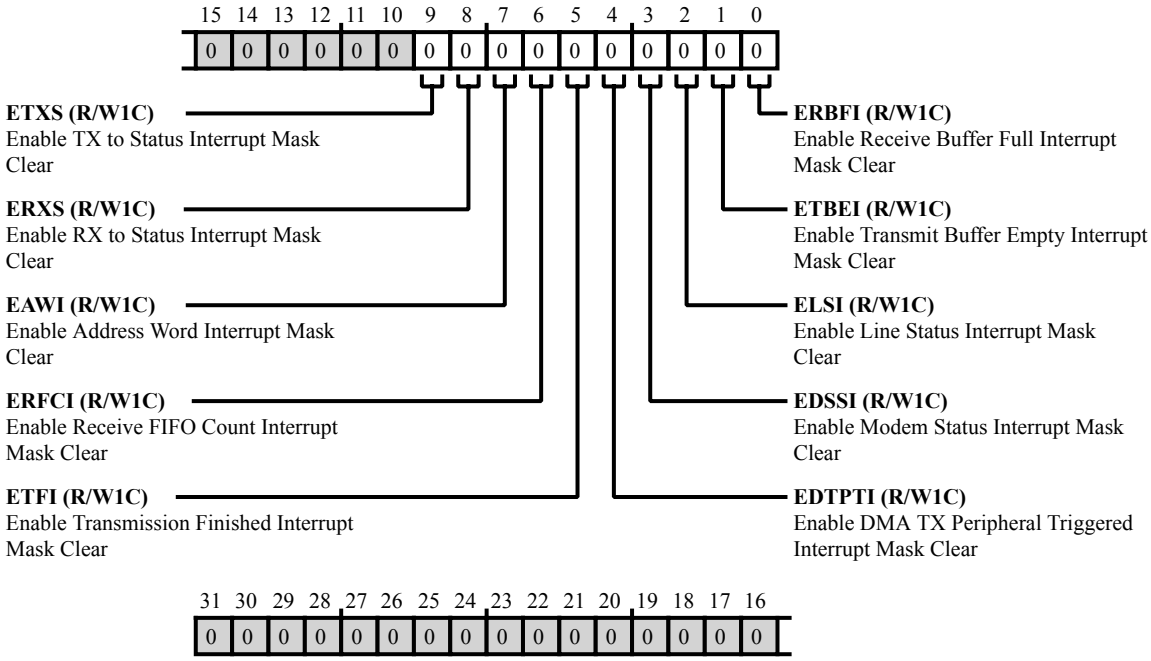

Figure 25-14: UART_IMSK_CLR Register Diagram

Table 25-11: 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 25-11: 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 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) interrupts, 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.

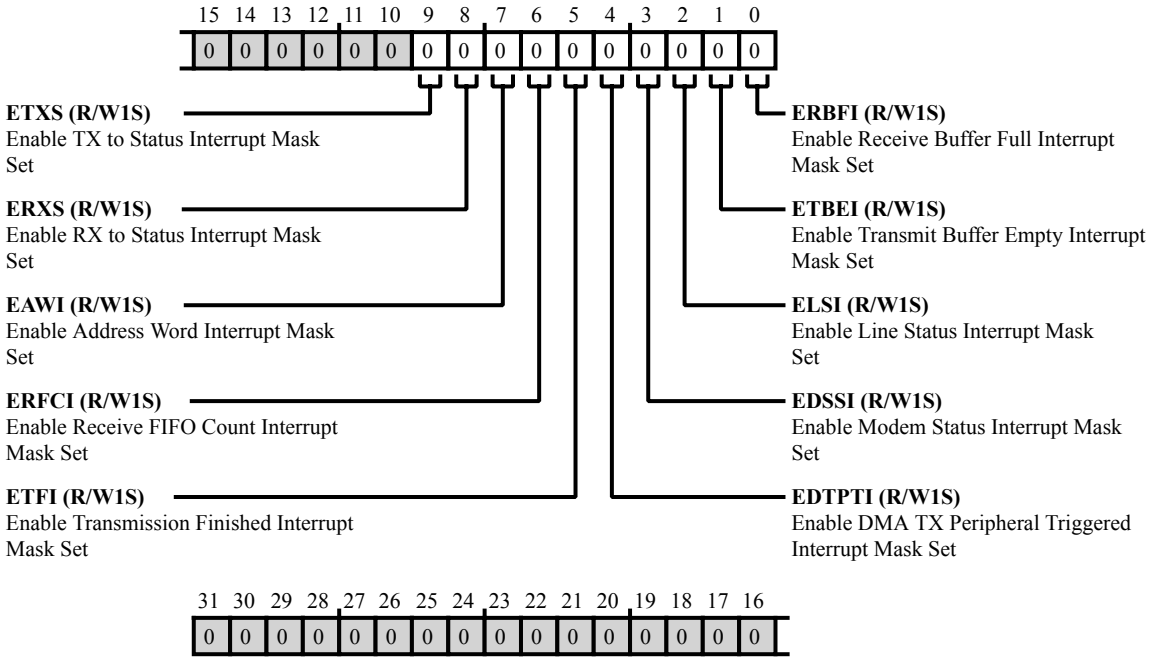

Figure 25-15: UART_IMSK_SET Register Diagram

Table 25-12: 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 25-12: 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.

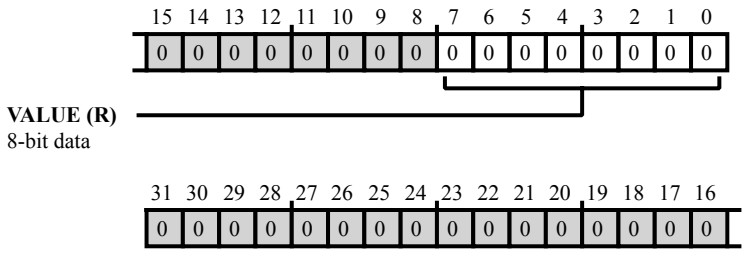

Figure 25-16: UART_RBR Register Diagram

Table 25-13: 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).

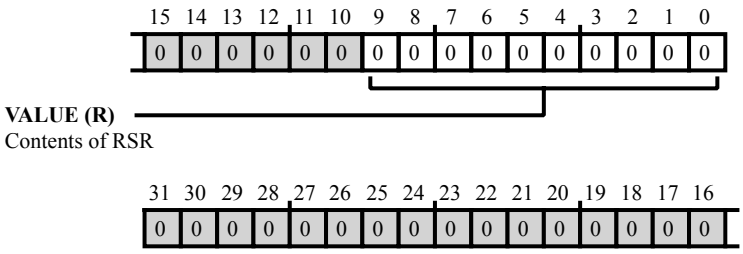

Figure 25-17: UART_RSR Register Diagram

Table 25-14: 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).

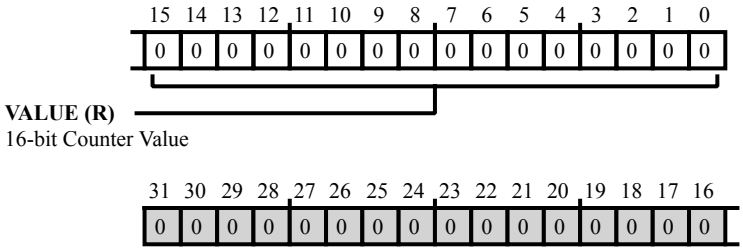

Figure 25-18: UART_RXCNT Register Diagram

Table 25-15: 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.

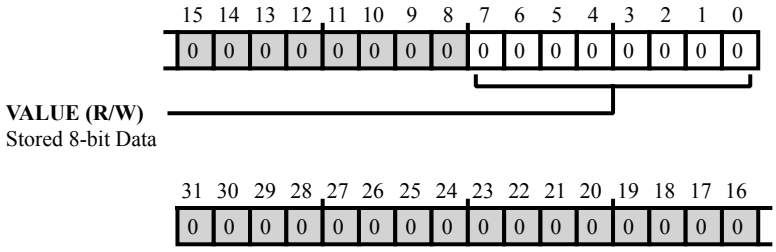

Figure 25-19: UART_SCR Register Diagram

Table 25-16: 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.

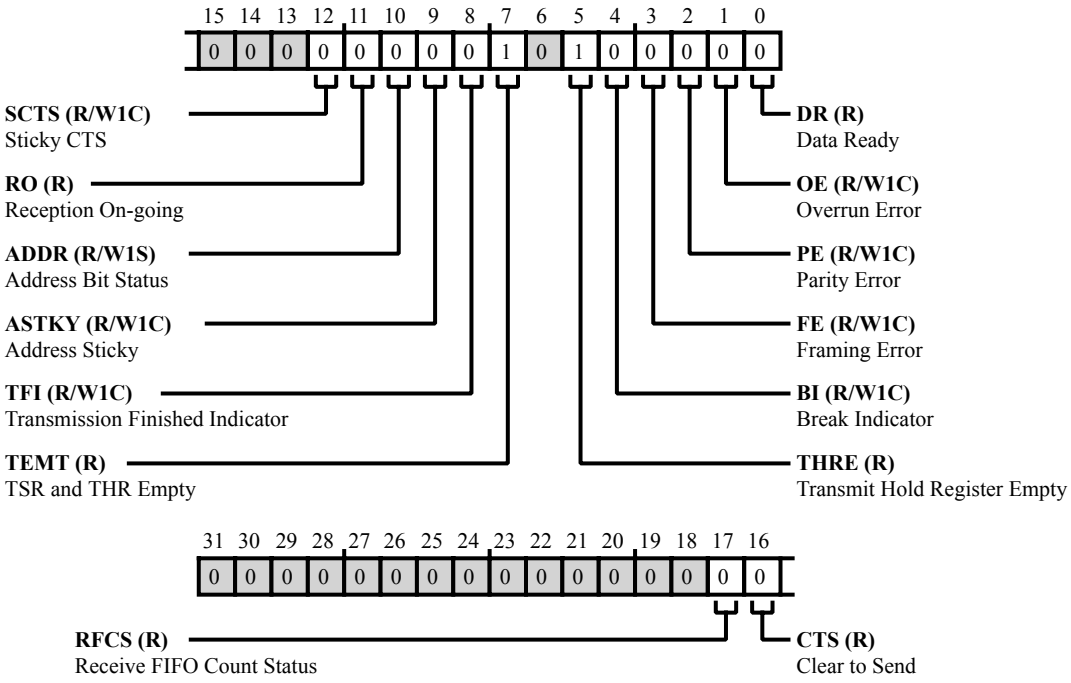

Figure 25-20: UART_STAT Register Diagram

Table 25-17: 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 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 25-17: 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 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 25-17: 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 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
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

Table 25-17: UART_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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 25-17: 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.

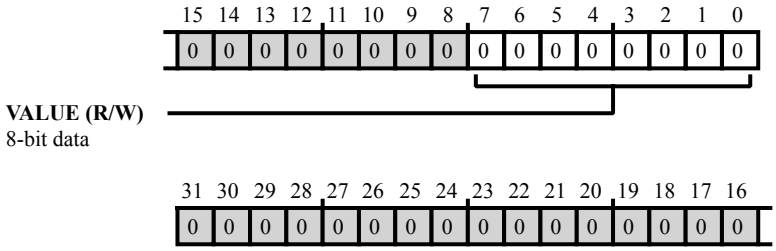

Figure 25-21: UART_TAIP Register Diagram

Table 25-18: 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.

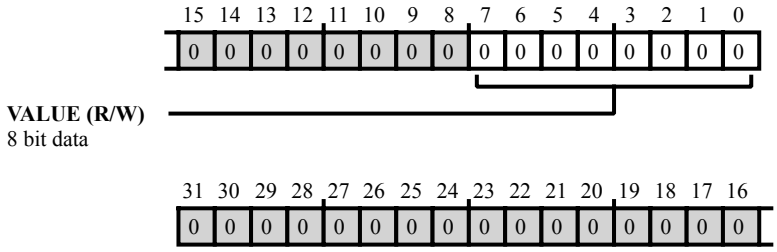

Figure 25-22: UART_THR Register Diagram

Table 25-19: 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.

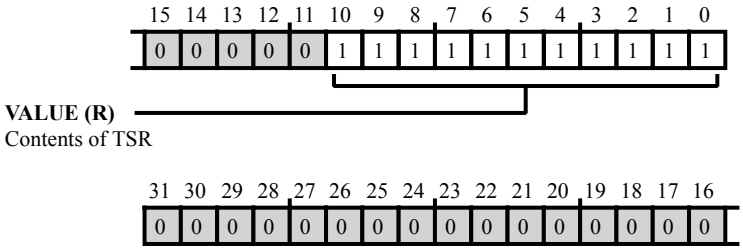

Figure 25-23: UART_TSR Register Diagram

Table 25-20: 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).

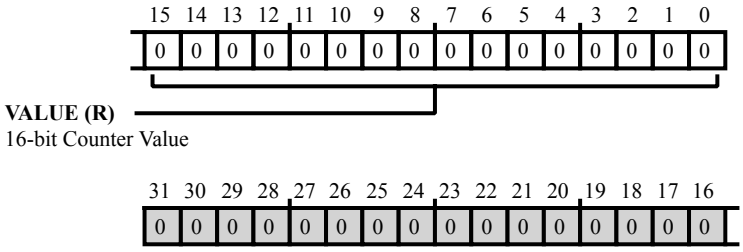

Figure 25-24: UART_TXCNT Register Diagram

Table 25-21: 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.

ADSP-BF70x 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: ADSP-BF70x 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

ADSP-BF70x TWI Interrupt List

Table 26-2: ADSP-BF70x TWI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
62	TWI0_DATA	TWI0 Data Interrupt	Level	

TWI Block Diagram

The *TWI Block Diagram* figure shows the basic blocks of the TWI interface.

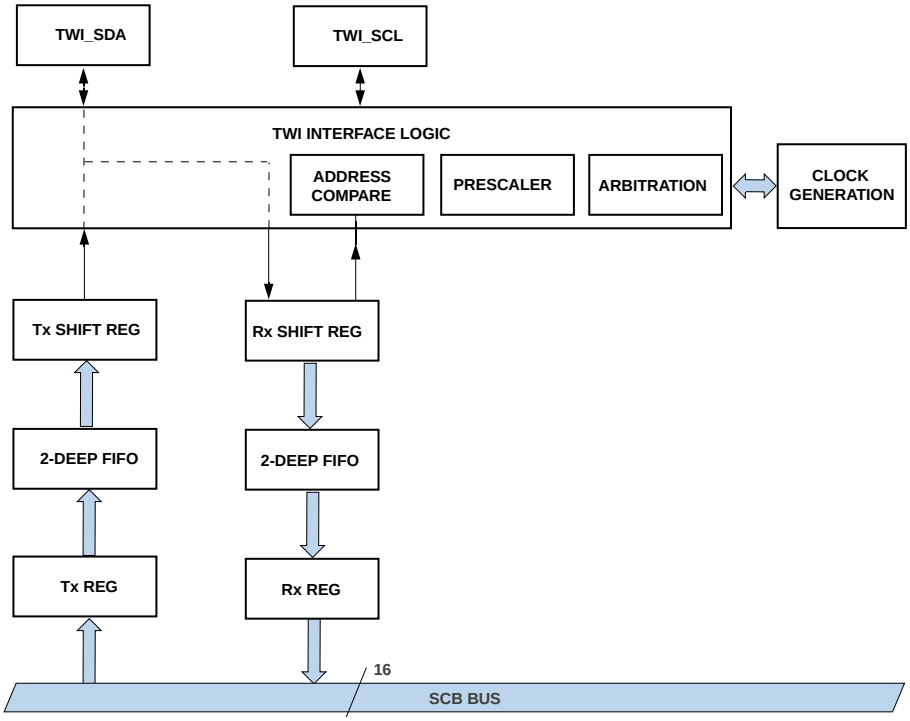

Figure 26-1: TWI Block Diagram

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 (SCLK0) and the internally timed events of the TWI controller. The internal time reference is derived from SCLK0 using a prescaled value. The prescale value is the number of SCLK0 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_{SCLK0}/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 SCLK0 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.

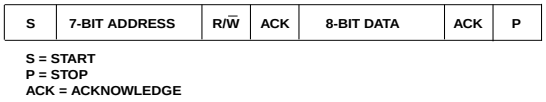

Figure 26-2: Data Transfer

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.

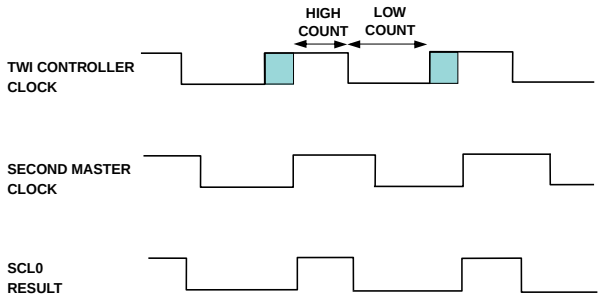

Figure 26-4: Clock Synchronization

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.

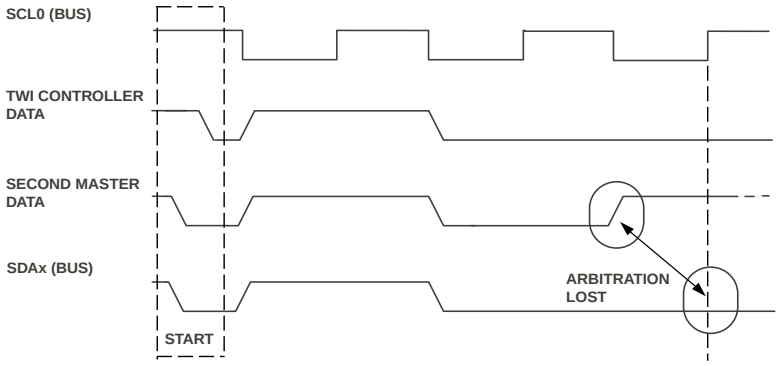

Figure 26-5: Bus Arbitration

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.

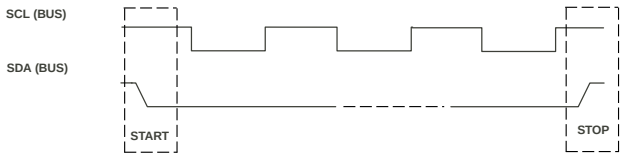

Figure 26-6: Start and Stop Conditions

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.

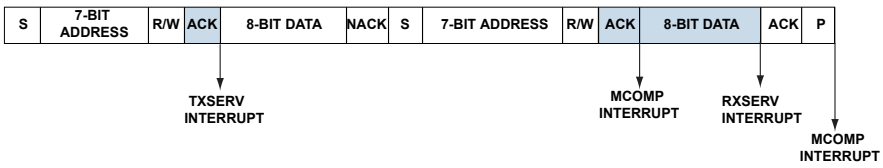

Figure 26-7: Repeated Start Followed by Data Receive

The tasks performed at each interrupt are:

- Transmit FIFO service (`TWI_ISTAT.TXSERV`) interrupt. 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 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 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. 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.

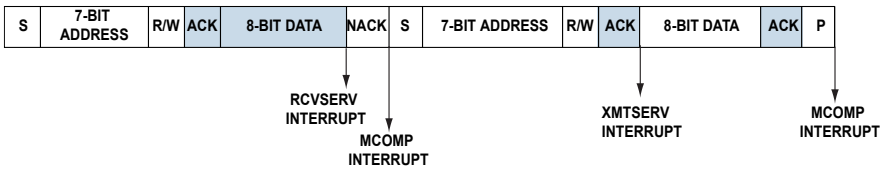

Figure 26-8: Repeated Start Data Receive Followed by Data Transmit

The tasks performed at each interrupt are:

- Receive FIFO service (TWI_ISTAT.RXSERV) interrupt. This interrupt 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 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 is generated due to a FIFO access. Simple data handling is the only requirement.
- Master transfer complete (TWI_ISTAT.MCOMP) interrupt. The transfer is complete.

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 occurs the instant the transmit FIFO becomes empty. The most recent byte begins transmission. If the TWI_ISTAT.TXSERV interrupt 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.

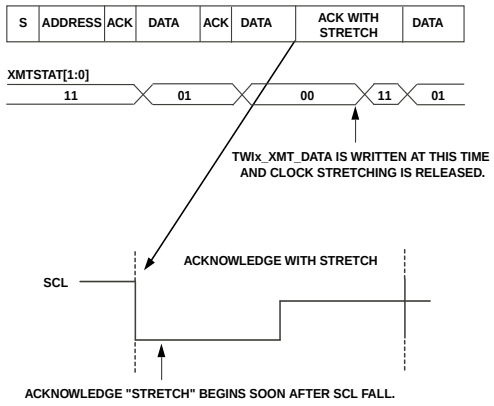

Figure 26-9: Clock Stretching during FIFO Underflow

TWI Controller	Processor
Interrupt: XMTSERV – Transmit FIFO buffer is empty.	Acknowledge: Clear the interrupt source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transmit complete (DCNT= 0x00).	Acknowledge: Clear the interrupt 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.

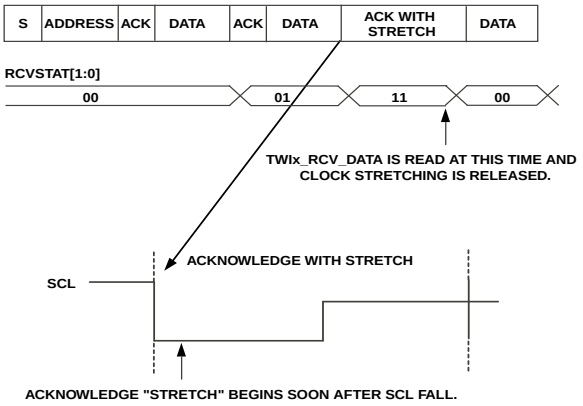

Figure 26-10: Clock Stretching During FIFO Overflow

TWI Controller	Processor
Interrupt: RCVSERV – Receive FIFO buffer is full.	Acknowledge: Clear the interrupt 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.

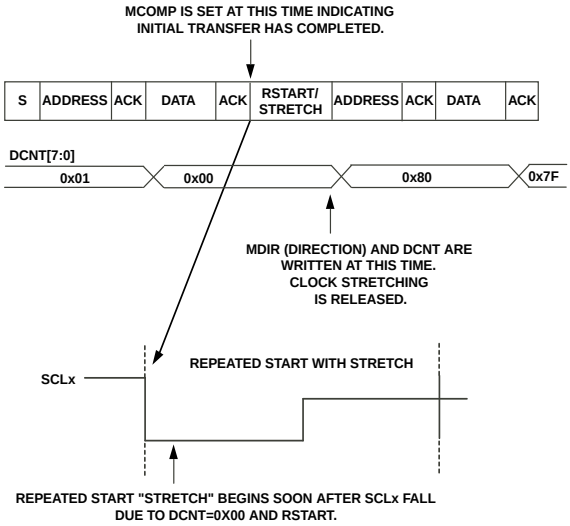

Figure 26-11: Clock Stretching during Repeated Start Condition

TWI Controller	Processor
Interrupt: MCOMP – Initial transmit has completed and DCNT = 0x00. Note: transfer in progress, RSTART previously set.	Acknowledge: Clear the interrupt 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 source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master receive complete	Acknowledge: Clear the interrupt 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.

The following sections provide programming steps for general setup, and master and slave modes.

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 output to the processor, when the TWI module detects a valid address match. An interrupt 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.

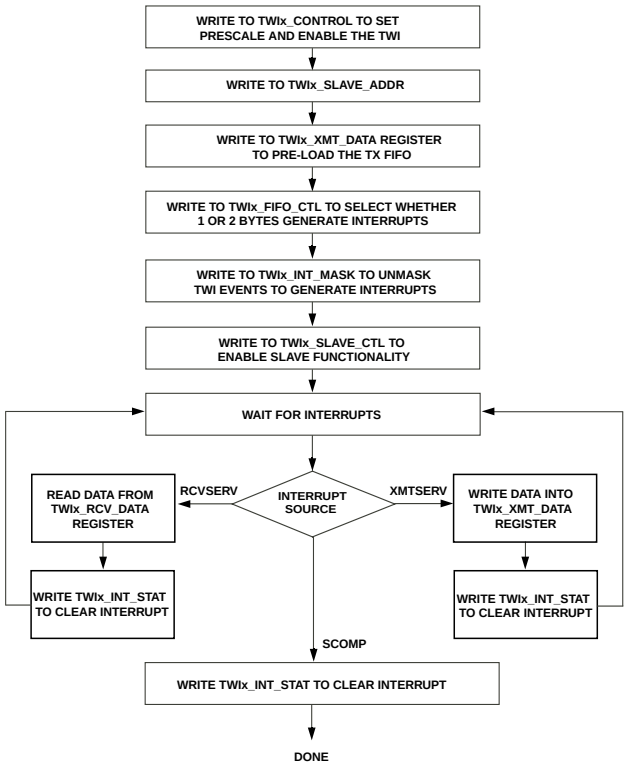

Figure 26-12: TWI Slave Mode Program Flow

Table 26-3: 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.

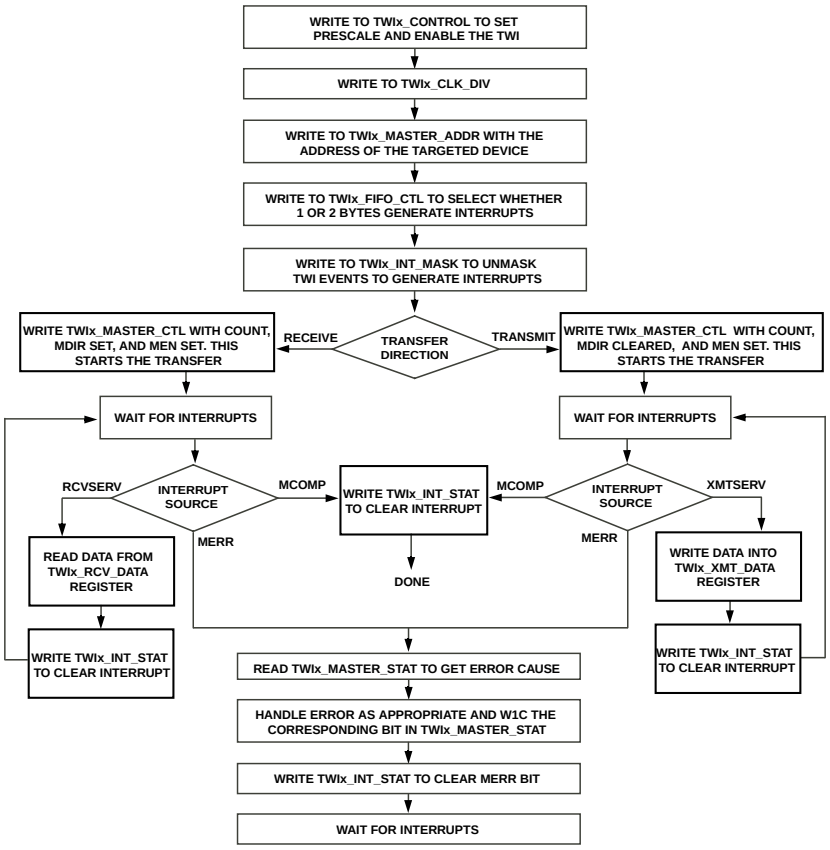

Figure 26-13: Master Mode Program Flow

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 interrupts 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 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 26-4: Master Mode Transmit Setup Interaction

TWI Controller	Processor
Interrupt: XMTSERV – Transmit buffer is empty.	Acknowledge: Clear the interrupt source bits. Write to the transmit FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt 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 interrupts 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 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 26-5: Master Mode Receive Setup Interaction

TWI Controller	Processor
Interrupt: RCVSERV – Receive buffer is full.	Acknowledge: Clear the interrupt source bits. Read the receive FIFO buffer.
...	...
Interrupt: MCOMP – Master transfer complete.	Acknowledge: Clear the interrupt 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.

ADSP-BF70x TWI Register Descriptions

Two-Wire Interface (TWI) contains the following registers.

Table 26-6: ADSP-BF70x 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.

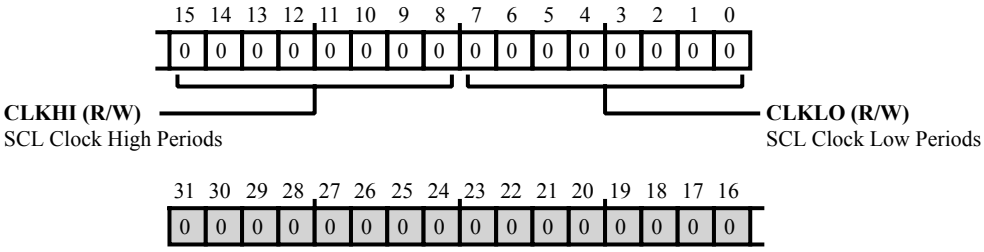

Figure 26-14: TWI_CLKDIV Register Diagram

Table 26-7: 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 (SCLK0) and the TWI controller's internally timed events, and enables SCCB compatibility.

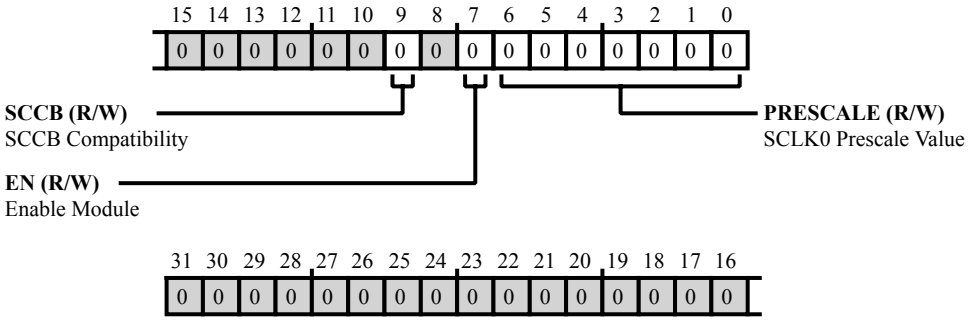

Figure 26-15: TWI_CTL Register Diagram

Table 26-8: 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
6:0 (R/W)	PRESCALE	SCLK0 Prescale Value. The TWI_CTL.PRESCALE holds the pre-scaled value for the TWI internal time reference. This reference is derived from SCLK0 according to the formula: TWI_CTL.PRESCALE = f_{SCLK0}/10MHz	

Table 26-8: TWI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		The `TWI_CTL.PRESCALE` specifies the number of system clock (SCLK0) 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.

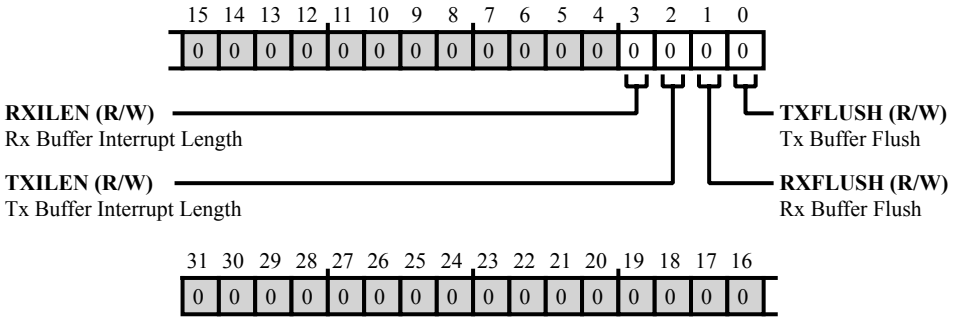

Figure 26-16: TWI_FIFOCTL Register Diagram

Table 26-9: 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 26-9: 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.

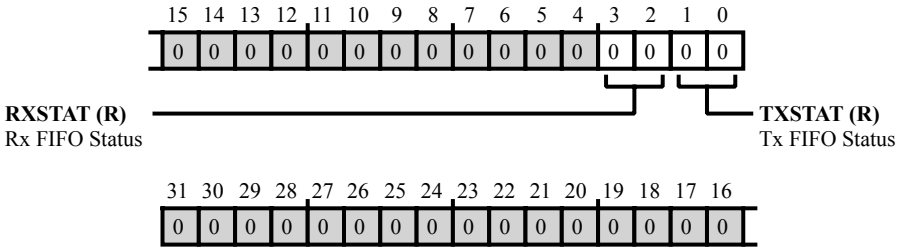

Figure 26-17: TWI_FIFOSTAT Register Diagram

Table 26-10: 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 source bit in TWI_ISTAT. Reading and writing TWI_IMSK does not affect the contents of the TWI_ISTAT.

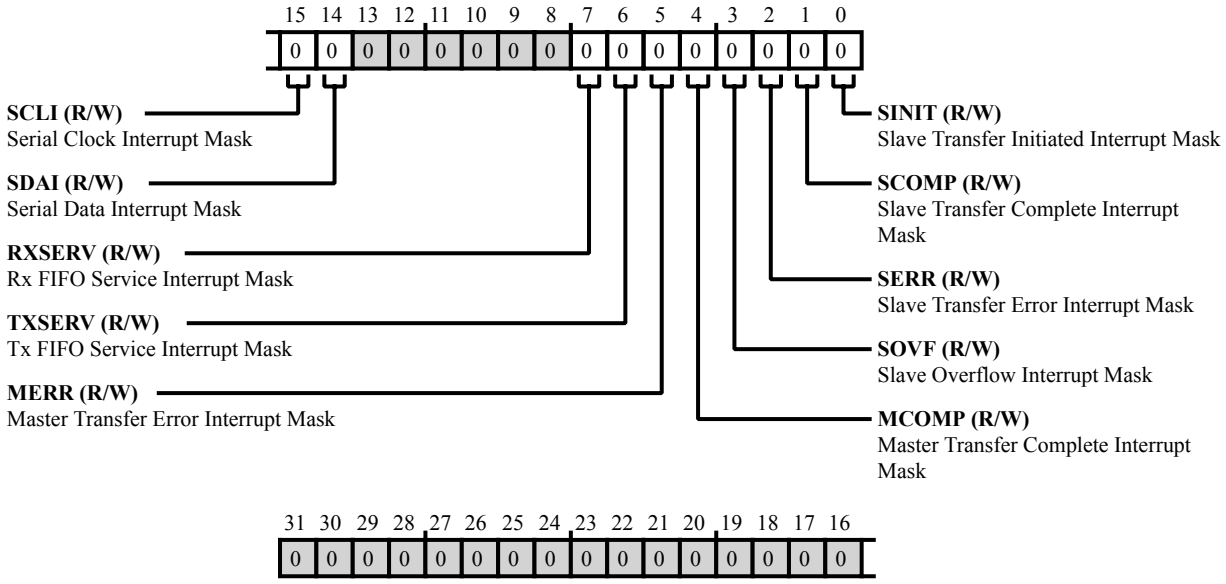

Figure 26-18: TWI_IMSK Register Diagram

Table 26-11: 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 26-11: 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.

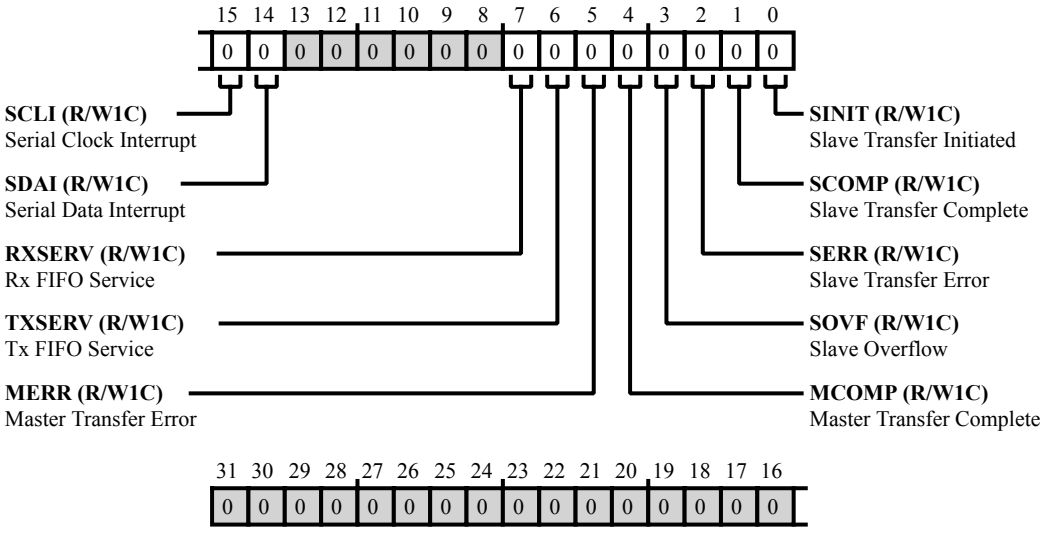

Figure 26-19: TWI_ISTAT Register Diagram

Table 26-12: 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 26-12: 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 26-12: 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.

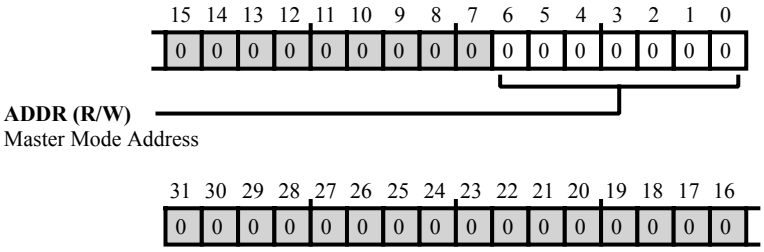

Figure 26-20: TWI_MSTRADDR Register Diagram

Table 26-13: 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.

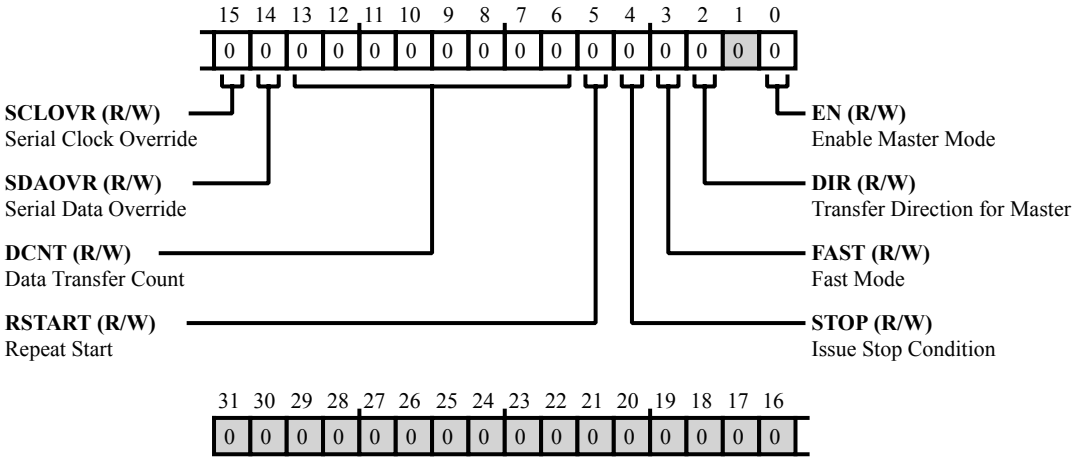

Figure 26-21: TWI_MSTRCTL Register Diagram

Table 26-14: 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 26-14: 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 26-14: 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 interrupts, 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.

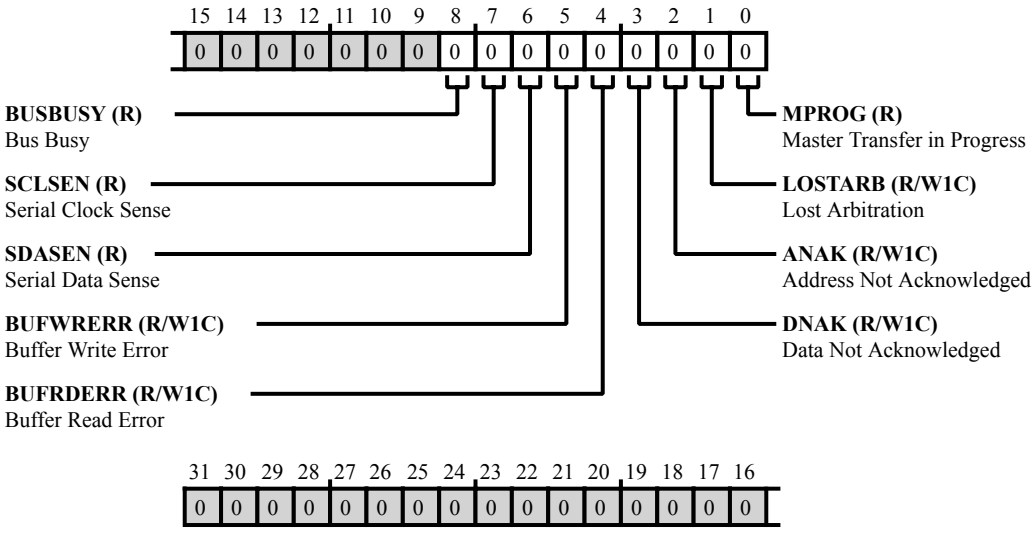

Figure 26-22: TWI_MSTRSTAT Register Diagram

Table 26-15: 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 26-15: 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 26-15: 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.

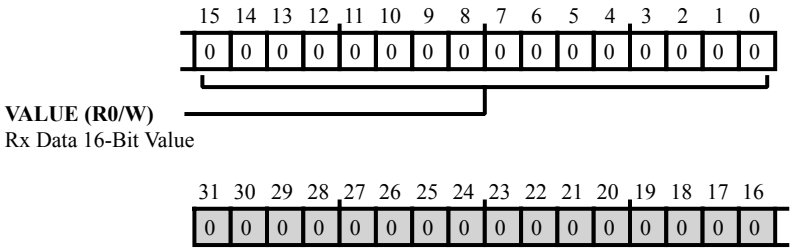

Figure 26-23: TWI_RXDATA16 Register Diagram

Table 26-16: 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.

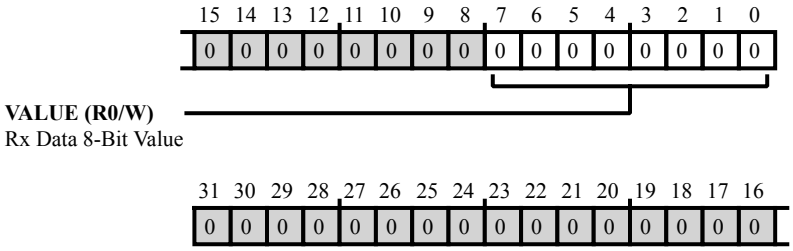

Figure 26-24: TWI_RXDATA8 Register Diagram

Table 26-17: 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.

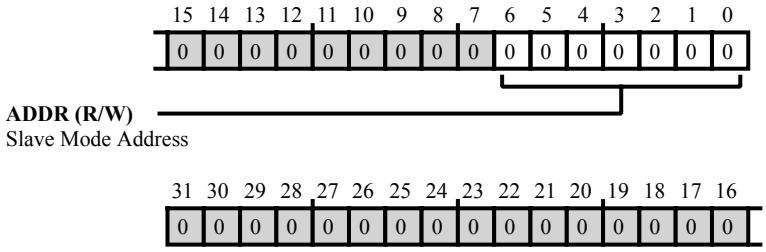

Figure 26-25: TWI_SLVADDR Register Diagram

Table 26-18: 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.

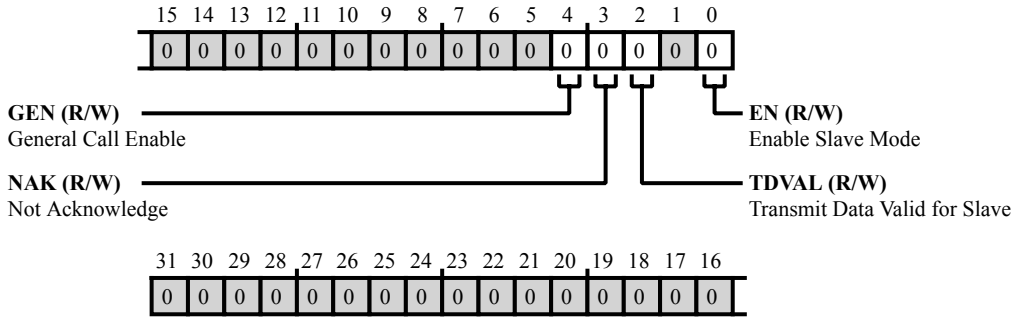

Figure 26-26: TWI_SLVCTL Register Diagram

Table 26-19: 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 26-19: 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 interrupts. Master mode operation does not affect slave mode status bits.

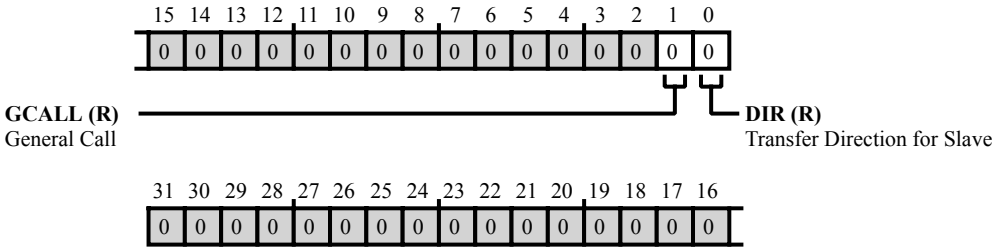

Figure 26-27: TWI_SLVSTAT Register Diagram

Table 26-20: 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 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.

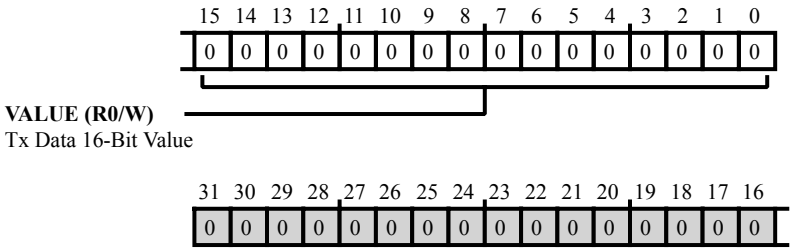

Figure 26-28: TWI_TXDATA16 Register Diagram

Table 26-21: 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.

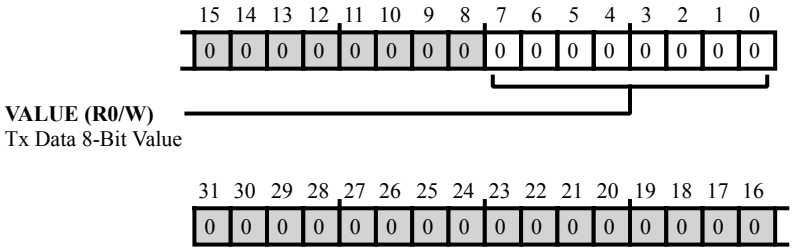

Figure 26-29: TWI_TXDATA8 Register Diagram

Table 26-22: 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^{TM} 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 SCLK0 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.

ADSP-BF70x 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: ADSP-BF70x 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: ADSP-BF70x 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

ADSP-BF70x CAN Interrupt List

Table 27-2: ADSP-BF70x CAN Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
90	CAN0_RX	CAN0 Receive	Level	
91	CAN0_TX	CAN0 Transmit	Level	
92	CAN0_STAT	CAN0 Status	Level	
93	CAN1_RX	CAN1 Recieve	Level	
94	CAN1_TX	CAN1 Transmit	Level	
95	CAN1_STAT	CAN1 Status	Level	

External Interface

The interface to the CAN bus is a simple two-wire line. The following figure 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.

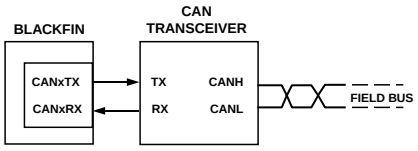

Figure 27-1: Representation of CAN Transceiver Interconnection

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.

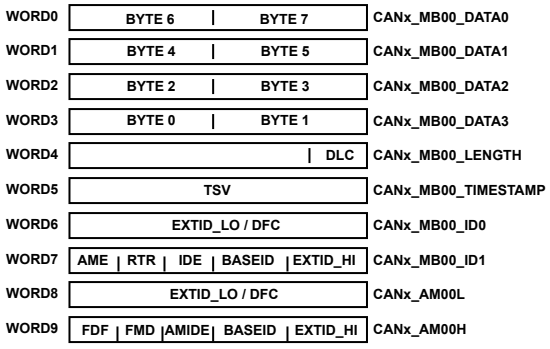

Figure 27-2: CAN Mailbox Area

The CAN mailbox identification register pair (`CAN_MB[nn]_ID0/1`) includes:

- The 29-bit identifier (base part `CAN_AM[nn]H.BASEID` plus 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 of each mailbox are marked as reserved. Always, set these 12 bits to 0.
- The mailbox word registers (`CAN_MB[nn]_DATA0/1/2/3`) supply up to 8 bytes for the data field, sent MSB first from 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.

Enable the *filtering on data field* option by setting the `CAN_CTL.DNM` and `CAN_AM[nn]H.FDF` bits. 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.

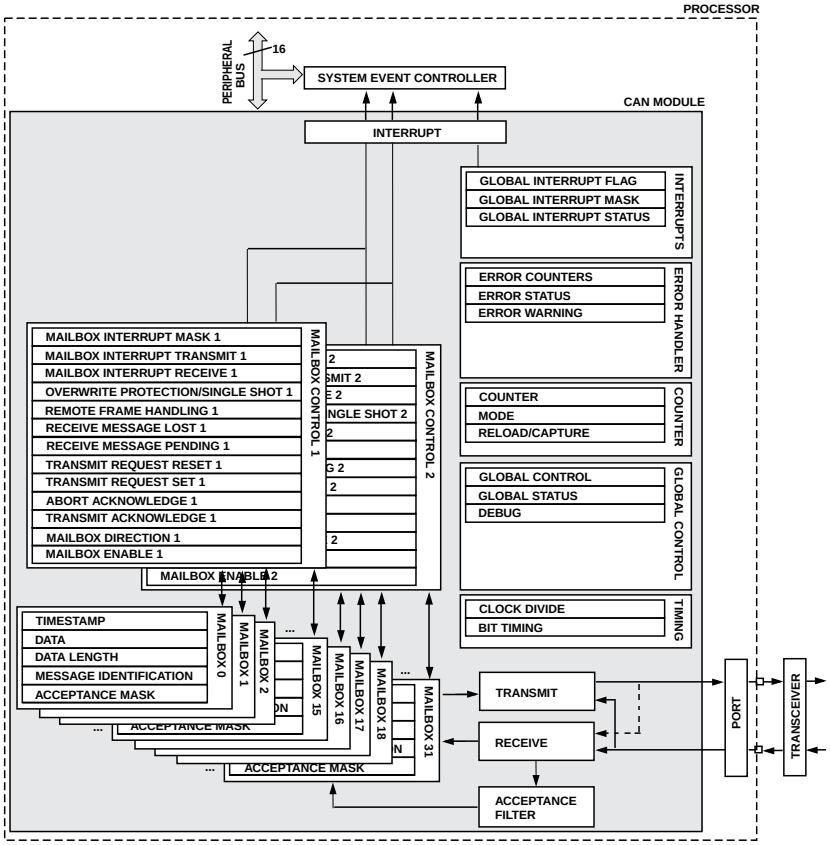

Figure 27-3: CAN Controller Block Diagram

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.

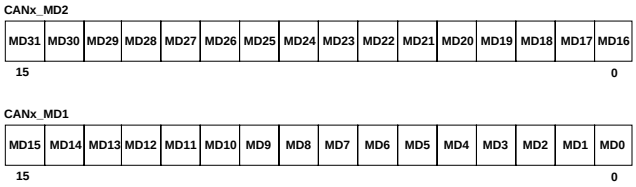

Figure 27-4: CAN Mailbox Register Pair

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 indicator (SOF) signals the beginning of a new frame. Each CAN node then begins transmitting its message starting with the message ID.

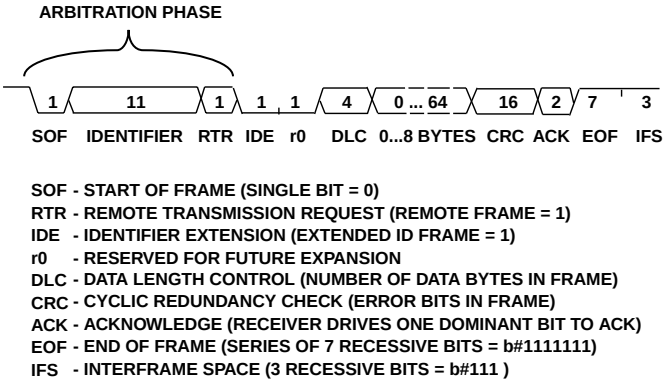

Figure 27-5: Standard CAN Frame

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.

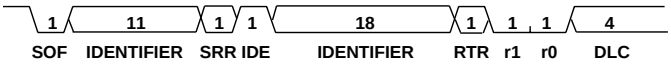

Figure 27-6: Extended CAN Frame

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.)

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).

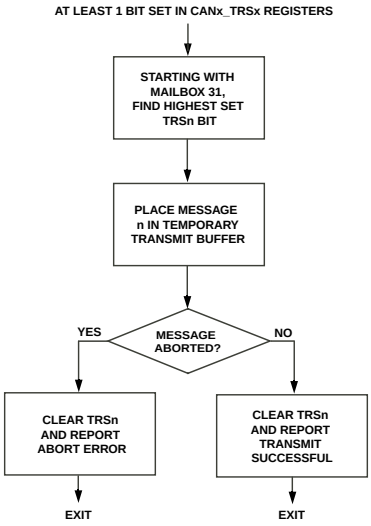

Figure 27-7: CAN Transmit Operation Flowchart

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.

The 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 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.

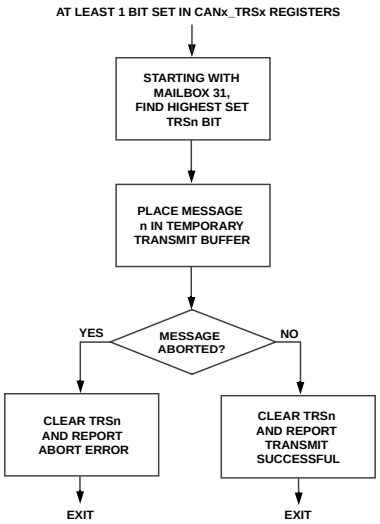

Figure 27-8: Transmit Flow

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.

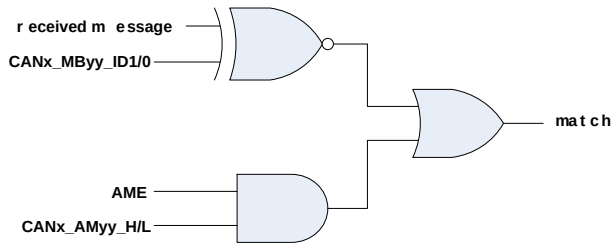

Figure 27-9: CAN Message Receive Logic

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 27-3: 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 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.

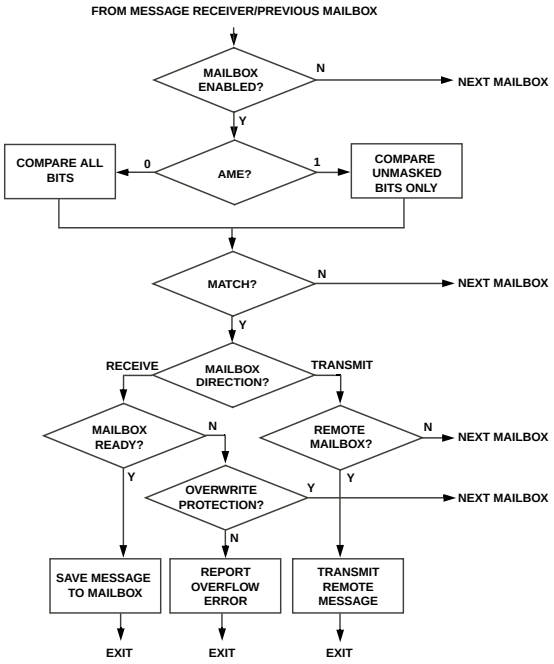

Figure 27-10: CAN Receive Operation Flowchart

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 27-4: 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 is desired for this event, set the CAN_GIM.UCEIM bit. With the mask bit set, when a watchdog interrupt 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 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 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 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

Enable automatic handling of remote frames for a transmit mailbox by setting the corresponding CAN_RFH1.MB bit.

Remote frames are data frames with 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 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.

CAN Operating Modes

The CAN controller is in configuration mode when coming out of processor reset or hibernate. 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.

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)/SCLK0

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.

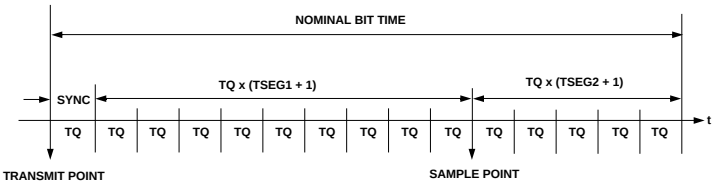

Figure 27-11: Three Phases of a CAN Bit

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 or hibernate, 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.

It also responds to the hibernate power state on processors that support that state.

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.

Wake Up From Hibernate State

Many processors provide a hibernate state, where the internal voltage regulator shuts off the internal power supply to the chip, turning off the core and system clocks in the process. In this mode, the only power drawn is that used by the regulator circuitry awaiting any of the possible hibernate wake-up events. One such event is a wake up due to CAN bus activity.

After hibernation, the CAN module must be re-initialized. For low power designs, the external CAN bus transceiver is typically put into standby mode through one of the processor's general purpose I/O pins. While in standby mode, the CAN transceiver continually drives the recessive logic 1 level onto the CAN_RX pin. If the transceiver then senses CAN bus activity, it drives the CAN_RX pin to the dominant logic 0 level. This signals the processor that CAN bus activity is detected. If the internal voltage regulator is programmed to recognize CAN bus activity as an event to exit

the hibernate state, the part responds appropriately. Otherwise, the activity on the CAN_RX pin has no effect on the processor state.

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 interrupts.

CAN Event Control

The following is a description of how CAN generates and controls events.

CAN Interrupt Signals

The CAN module provides three independent interrupts: two mailbox interrupts (mailbox receive interrupt (MBRIRQ) and mailbox transmit interrupt (MBTIRQ)) and a global CAN status interrupt (GIRQ). The values of these three interrupts 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, depending on the mailbox configuration. To enable a mailbox to generate an interrupt, 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. 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:

- Access denied 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 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 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 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 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 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 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 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 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.

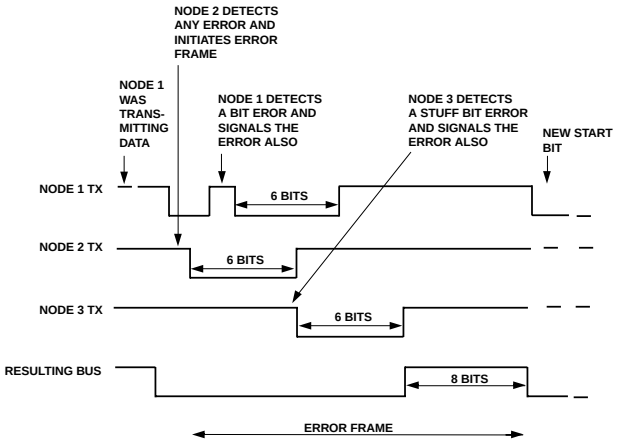

Figure 27-12: CAN Error Example

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 in 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 27-5: 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, interrupts can occur for all of these levels by unmasking them in the global CAN interrupt mask register (CAN_GIM). These interrupts 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 27-6: 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
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.

Table 27-6: Common CAN Test Mode Bit Combinations (Continued)

MRB	MAA	DIL	DTO	DRI	CDE	Functional Description
						No external acknowledge required. Nether transmit message nor acknowledge are transmitted on CAN_TX. CAN_RX input is ignored. Internal loop is enabled.

ADSP-BF70x CAN Register Descriptions

Controller Area Network (CAN) contains the following registers.

Table 27-7: ADSP-BF70x 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

Table 27-7: ADSP-BF70x CAN Register List (Continued)

Name	Description
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
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.

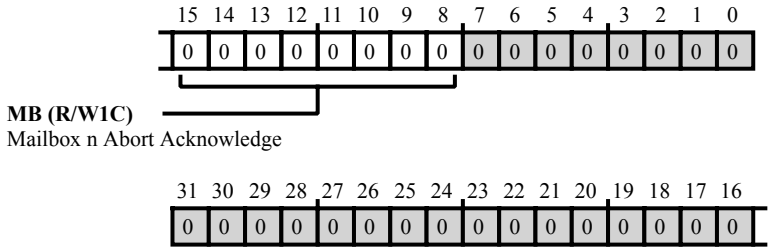

Figure 27-13: CAN_AA1 Register Diagram

Table 27-8: 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).

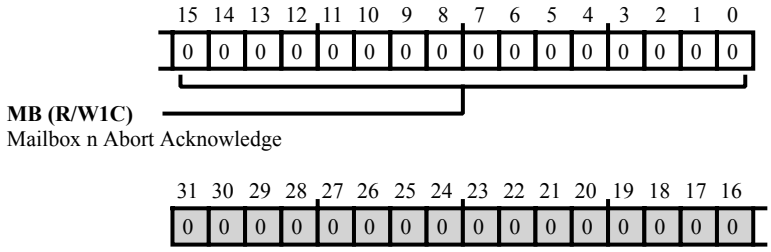

Figure 27-14: CAN_AA2 Register Diagram

Table 27-9: 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.

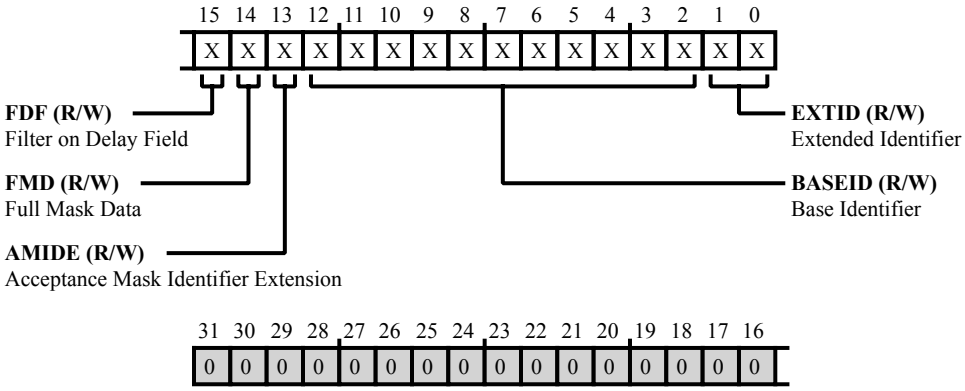

Figure 27-15: CAN_AM[nn]H Register Diagram

Table 27-10: 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.

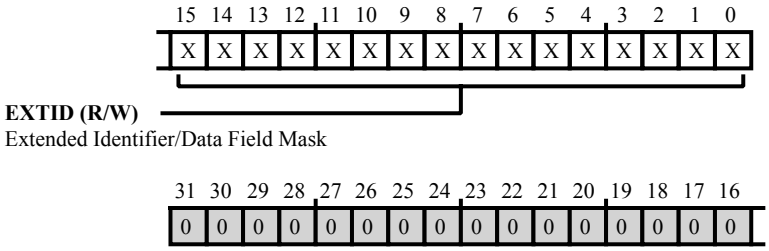

Figure 27-16: CAN_AM[nn]L Register Diagram

Table 27-11: 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.

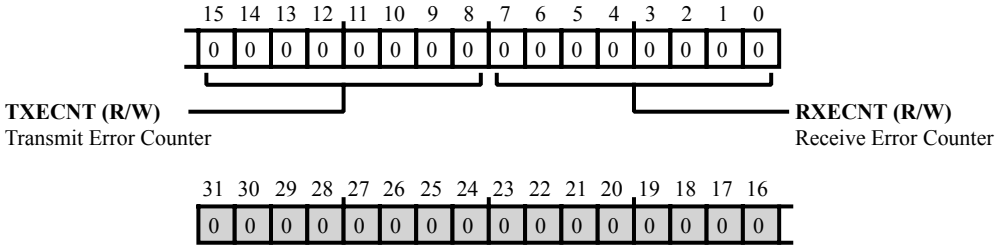

Figure 27-17: CAN_CEC Register Diagram

Table 27-12: 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 (SCLK0). For more information about bit timing and clock operation, see the CAN Operating Modes section.

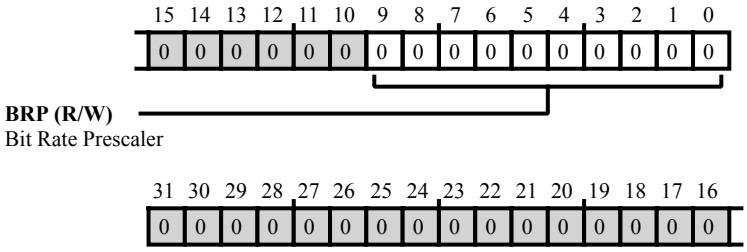

Figure 27-18: CAN_CLK Register Diagram

Table 27-13: 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) / SCLK0 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.

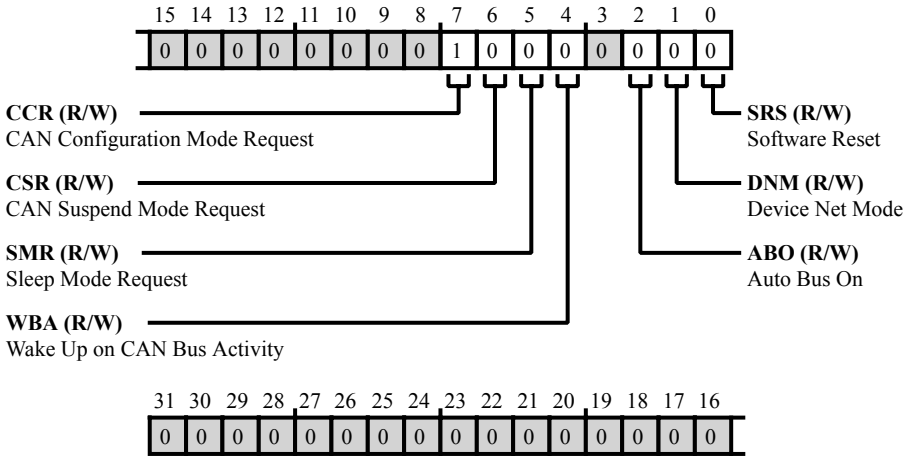

Figure 27-19: CAN_CTL Register Diagram

Table 27-14: 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 27-14: 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.

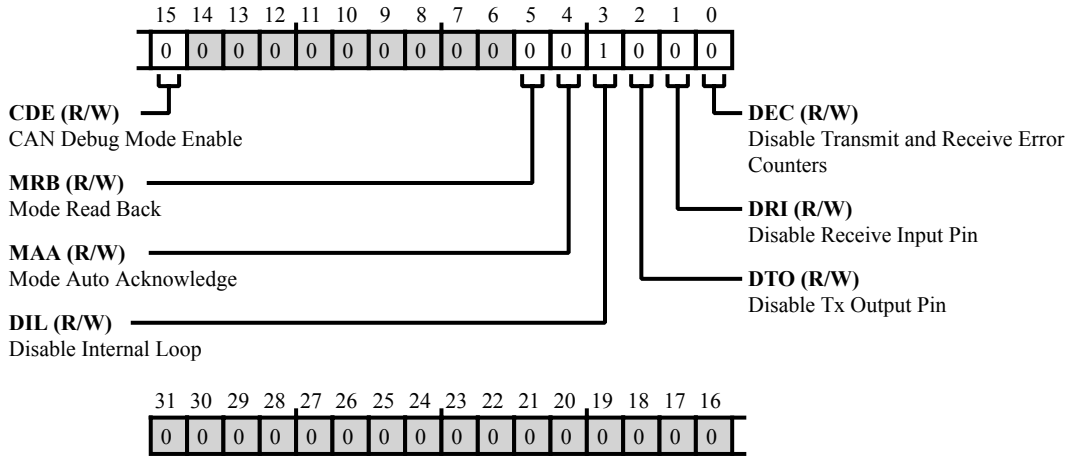

Figure 27-20: CAN_DBG Register Diagram

Table 27-15: 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 27-15: 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.

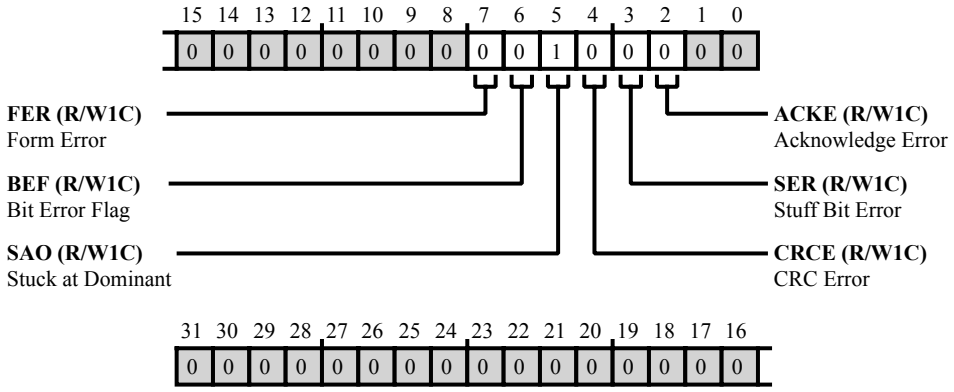

Figure 27-21: CAN_ESR Register Diagram

Table 27-16: 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 27-16: 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.

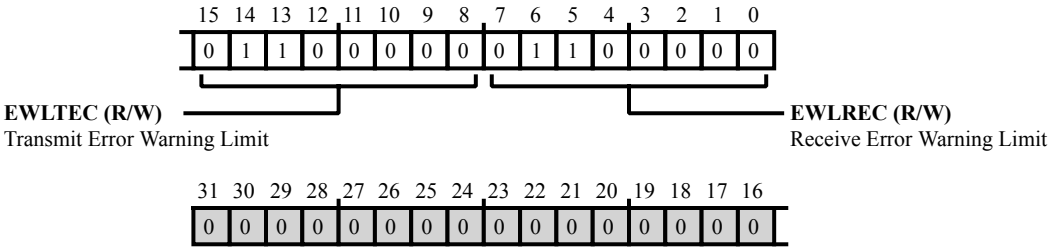

Figure 27-22: CAN_EWR Register Diagram

Table 27-17: 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 interrupts. 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.

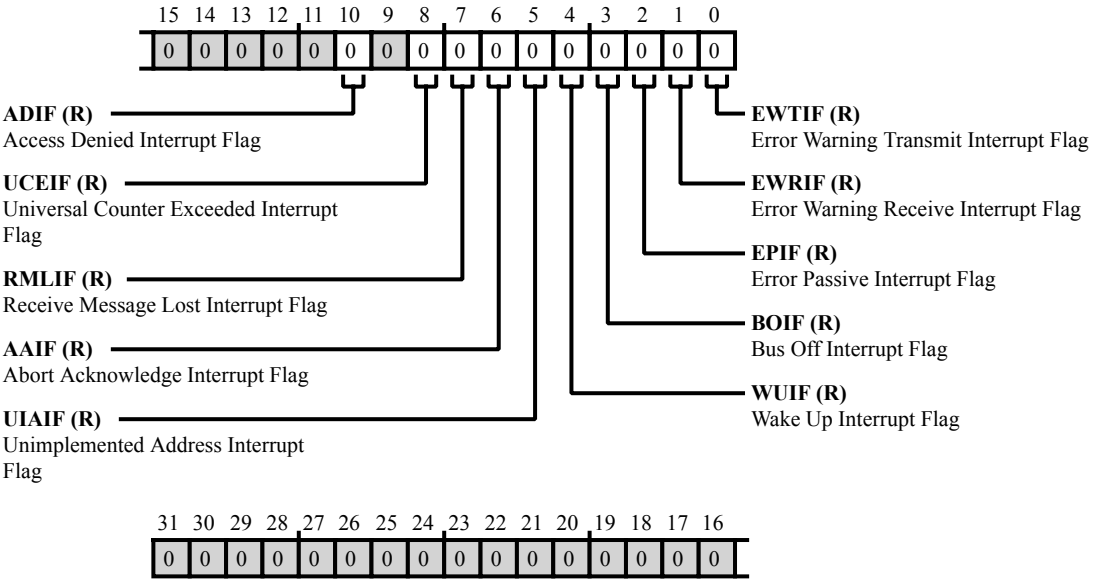

Figure 27-23: CAN_GIF Register Diagram

Table 27-18: 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 27-18: 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 27-18: 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 interrupts. 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.

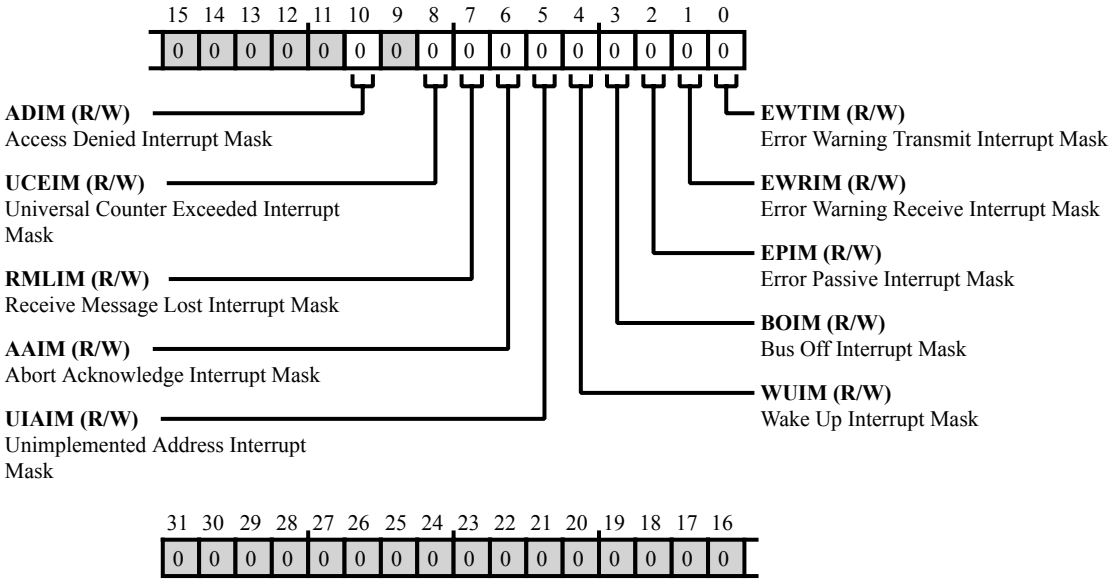

Figure 27-24: CAN_GIM Register Diagram

Table 27-19: 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.	
		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.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Table 27-19: 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.	
		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.	
		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.	
		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.	
		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.	
		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.	
		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.	
		0	Disable Interrupt (Mask)
		1	Enable Interrupt (Unmask)

Table 27-19: 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.	
		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 interrupts. 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.

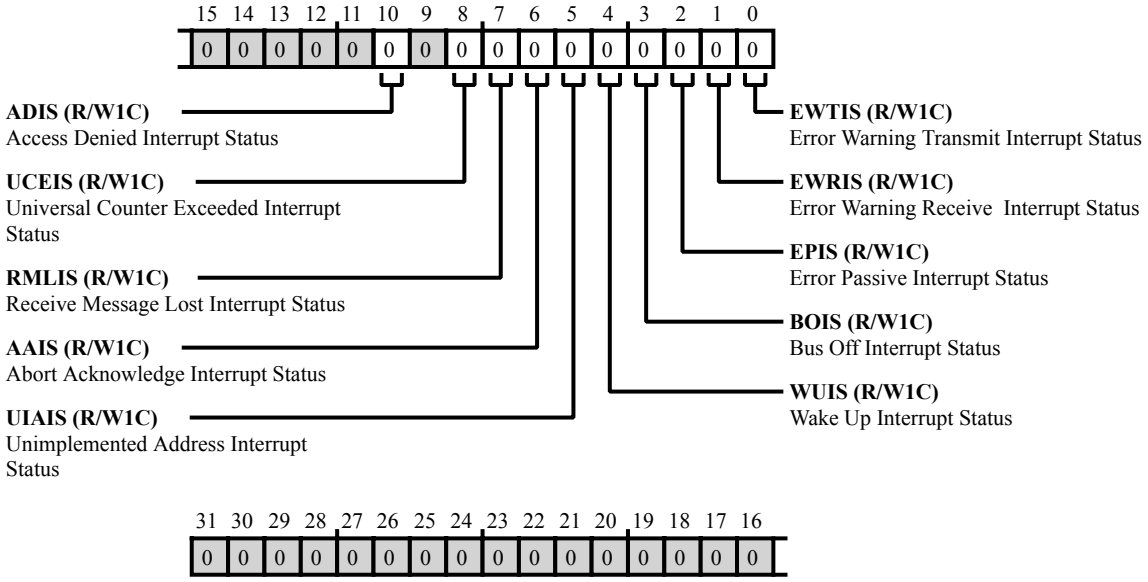

Figure 27-25: CAN_GIS Register Diagram

Table 27-20: 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 27-20: 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 27-20: 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.

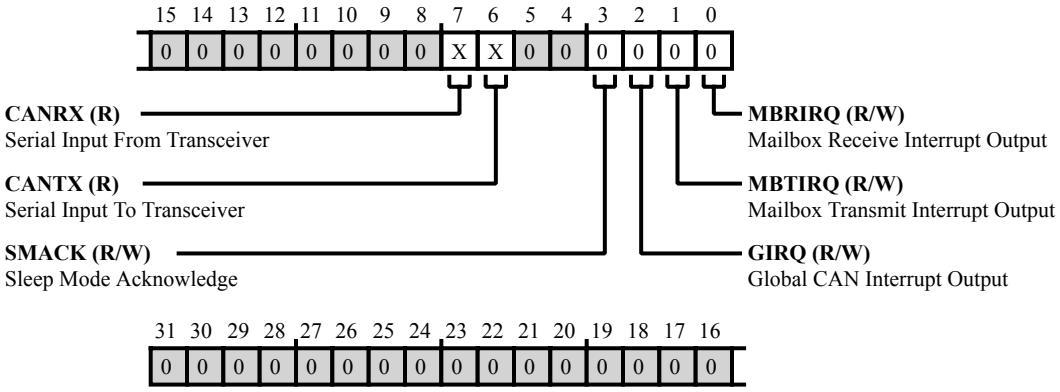

Figure 27-26: CAN_INT Register Diagram

Table 27-21: 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 27-21: 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 interrupts for mailboxes 0 through 15. Each bit in this register requests enables the transmit or receive interrupt for the corresponding mailbox when set (=1).

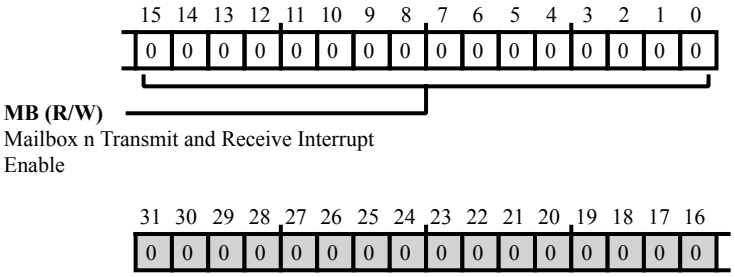

Figure 27-27: CAN_MBIM1 Register Diagram

Table 27-22: CAN_MBIM1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	MB	Mailbox n Transmit and Receive Interrupt Enable.

Mailbox Interrupt Mask 2 Register

The CAN_MBIM2 register enables transmit and receive interrupts for mailboxes 16 (bit 0) through 31 (bit 15). Each bit in this register requests enables the transmit or receive interrupt for the corresponding mailbox when set (=1).

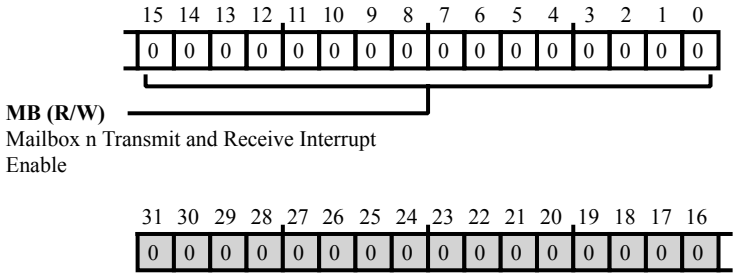

Figure 27-28: CAN_MBIM2 Register Diagram

Table 27-23: 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 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).

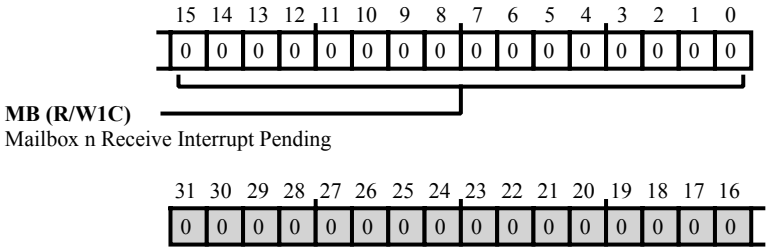

Figure 27-29: CAN_MBRIF1 Register Diagram

Table 27-24: CAN_MBRIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W1C)	MB	Mailbox n Receive Interrupt Pending.

Mailbox Receive Interrupt Flag 2 Register

The CAN_MBRIF2 register indicates when a receive interrupt 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.

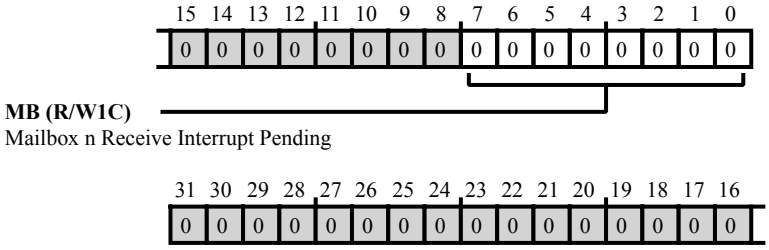

Figure 27-30: CAN_MBRIF2 Register Diagram

Table 27-25: 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.

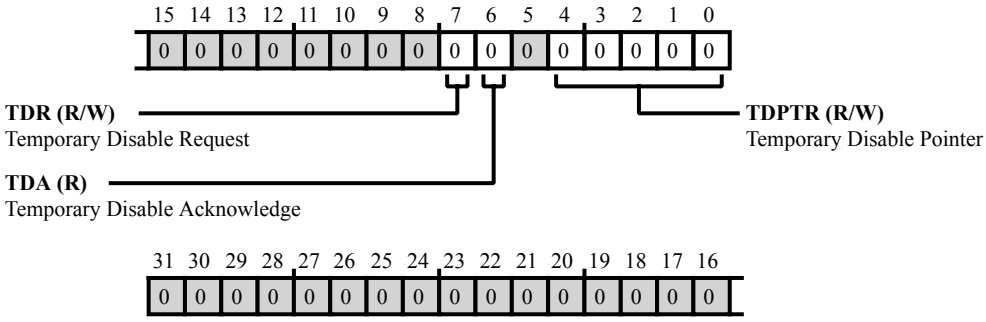

Figure 27-31: CAN_MBTD Register Diagram

Table 27-26: 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 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.

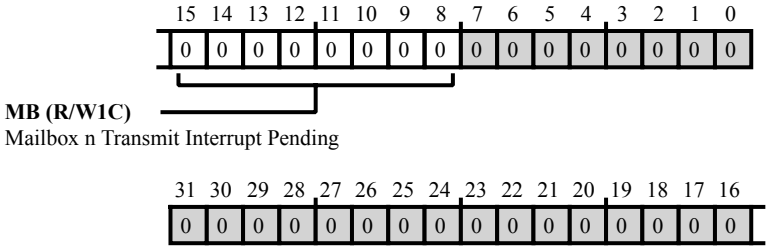

Figure 27-32: CAN_MBTIF1 Register Diagram

Table 27-27: CAN_MBTIF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:8 (R/W1C)	MB	Mailbox n Transmit Interrupt Pending.

Mailbox Transmit Interrupt Flag 2 Register

The CAN_MBTIF2 register indicates when a transmit interrupt 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.

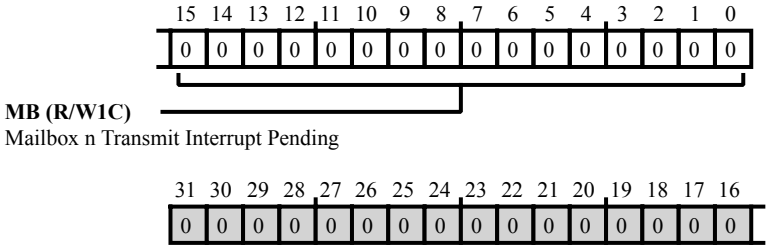

Figure 27-33: CAN_MBTIF2 Register Diagram

Table 27-28: 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.

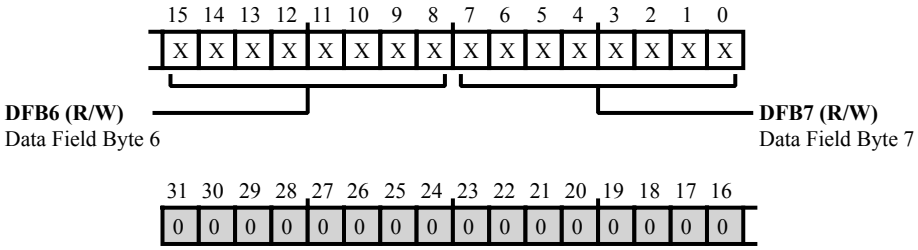

Figure 27-34: CAN_MB[nn]_DATA0 Register Diagram

Table 27-29: 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.

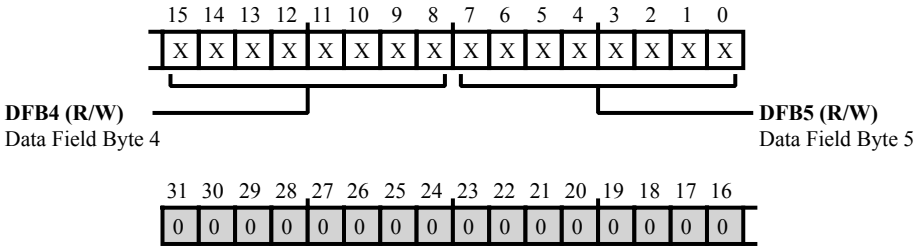

Figure 27-35: CAN_MB[nn]_DATA1 Register Diagram

Table 27-30: 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.

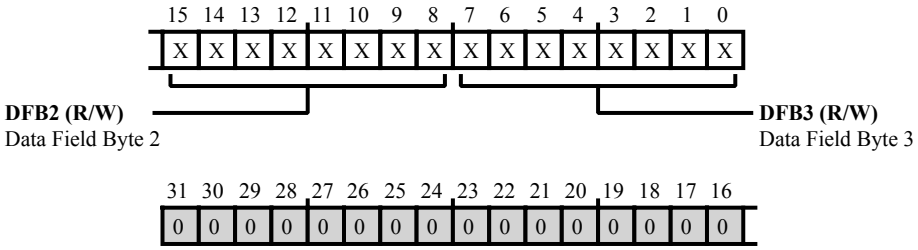

Figure 27-36: CAN_MB[nn]_DATA2 Register Diagram

Table 27-31: 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.

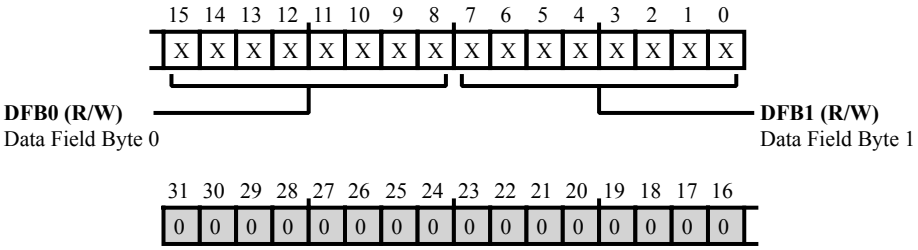

Figure 27-37: CAN_MB[nn]_DATA3 Register Diagram

Table 27-32: 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.

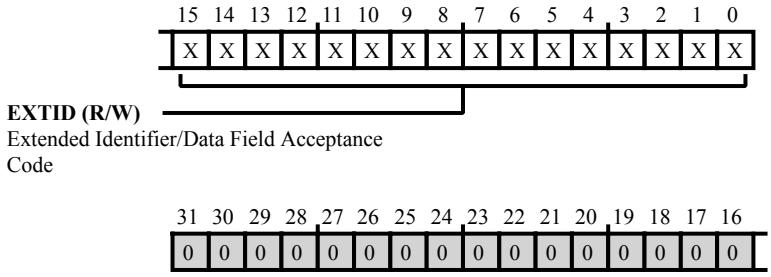

Figure 27-38: CAN_MB[nn]_ID0 Register Diagram

Table 27-33: 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.

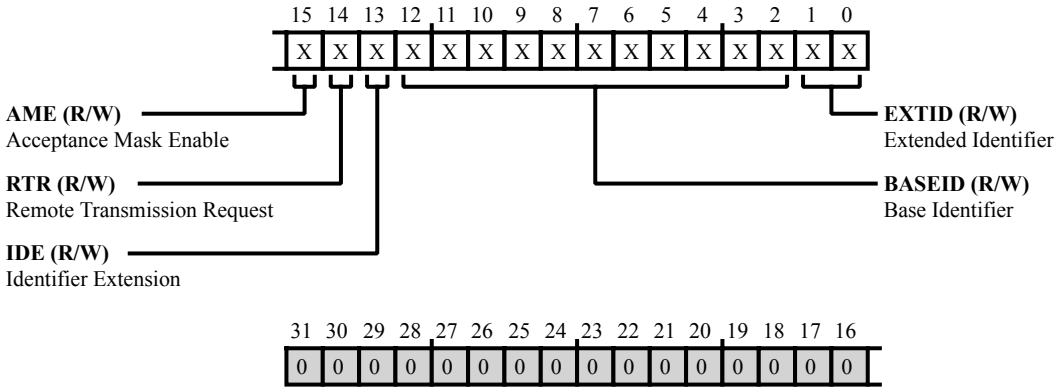

Figure 27-39: CAN_MB[nn]_ID1 Register Diagram

Table 27-34: 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.
1:0 (R/W)	EXTID	Extended Identifier.

Table 27-34: CAN_MB[nn]_ID1 Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		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.

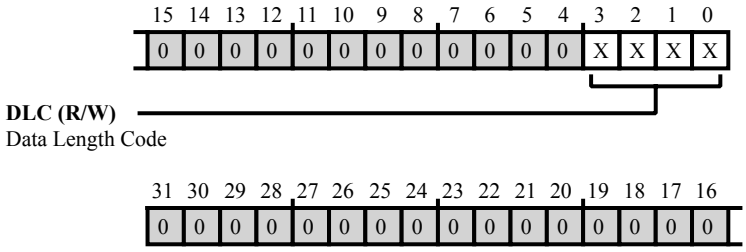

Figure 27-40: CAN_MB[nn]_LENGTH Register Diagram

Table 27-35: 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.

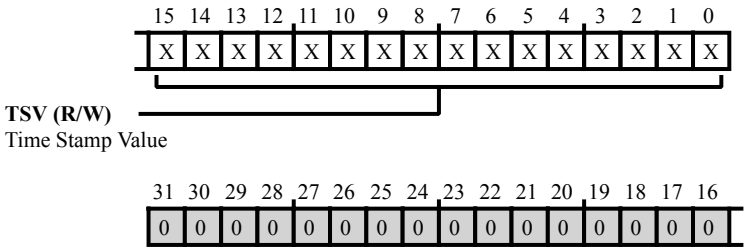

Figure 27-41: CAN_MB[nn]_TIMESTAMP Register Diagram

Table 27-36: 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.

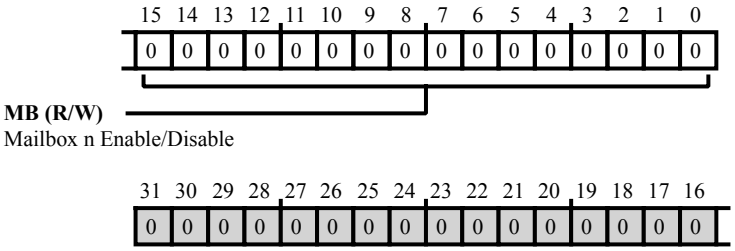

Figure 27-42: CAN_MC1 Register Diagram

Table 27-37: 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.

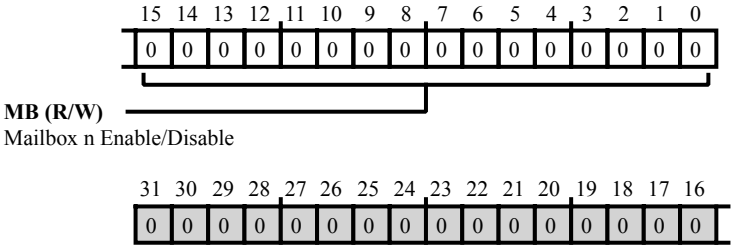

Figure 27-43: CAN_MC2 Register Diagram

Table 27-38: 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.

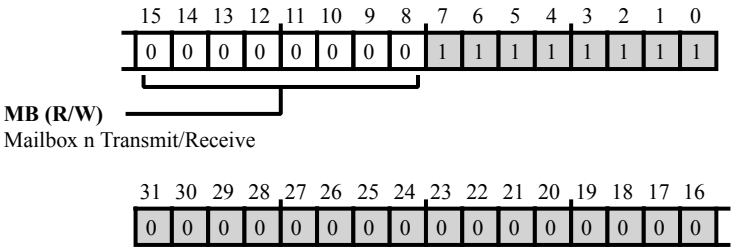

Figure 27-44: CAN_MD1 Register Diagram

Table 27-39: 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.

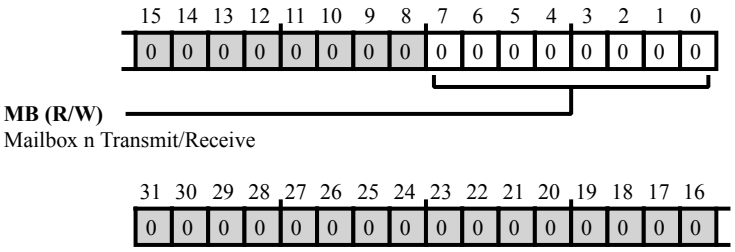

Figure 27-45: CAN_MD2 Register Diagram

Table 27-40: 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.

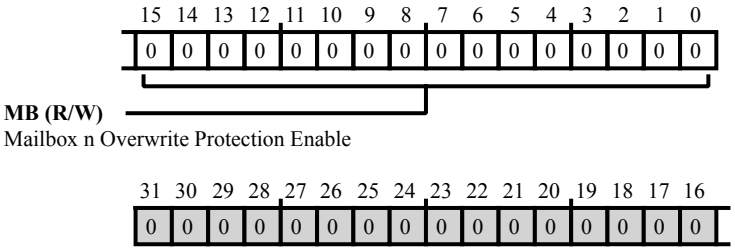

Figure 27-46: CAN_OPSS1 Register Diagram

Table 27-41: 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.

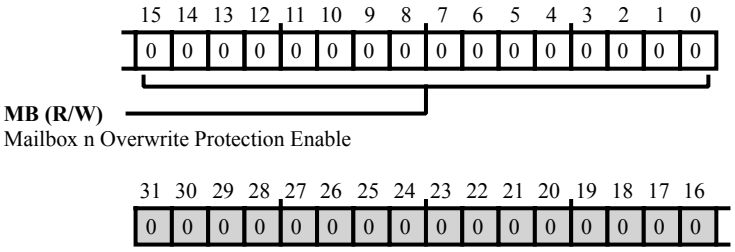

Figure 27-47: CAN_OPSS2 Register Diagram

Table 27-42: 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.

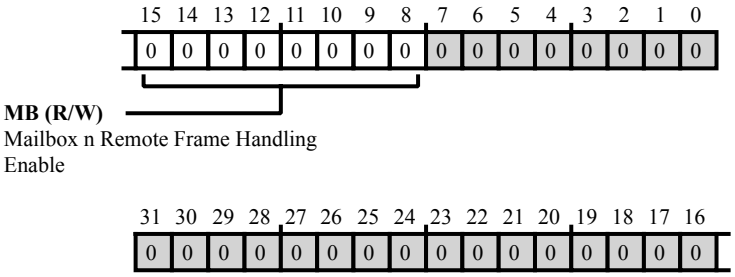

Figure 27-48: CAN_RFH1 Register Diagram

Table 27-43: 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.

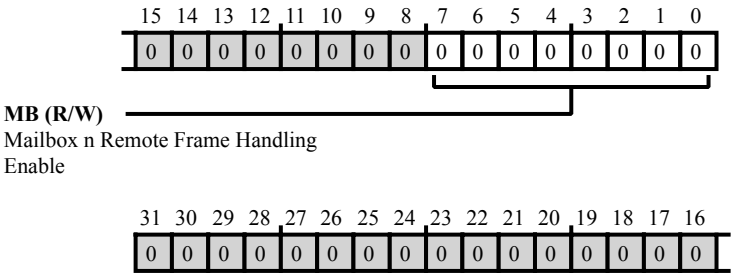

Figure 27-49: CAN_RFH2 Register Diagram

Table 27-44: 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).

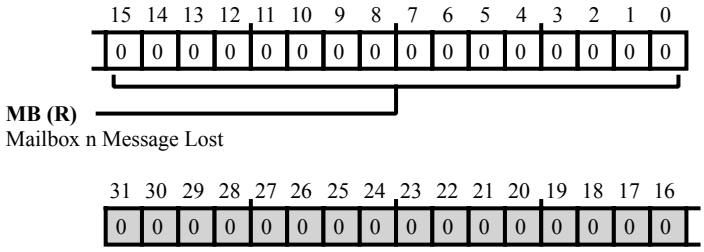

Figure 27-50: CAN_RML1 Register Diagram

Table 27-45: 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.

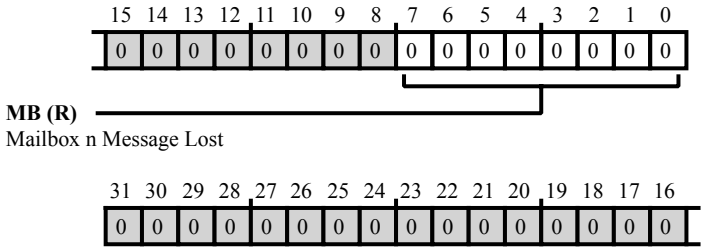

Figure 27-51: CAN_RML2 Register Diagram

Table 27-46: 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).

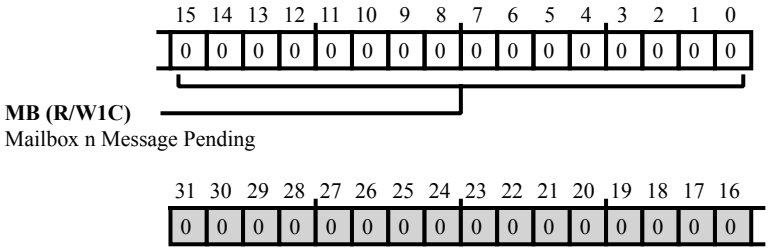

Figure 27-52: CAN_RMP1 Register Diagram

Table 27-47: 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.

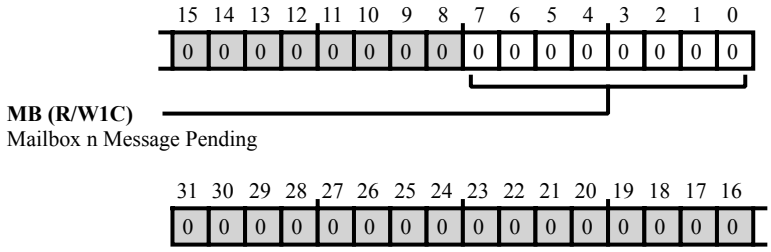

Figure 27-53: CAN_RMP2 Register Diagram

Table 27-48: 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.

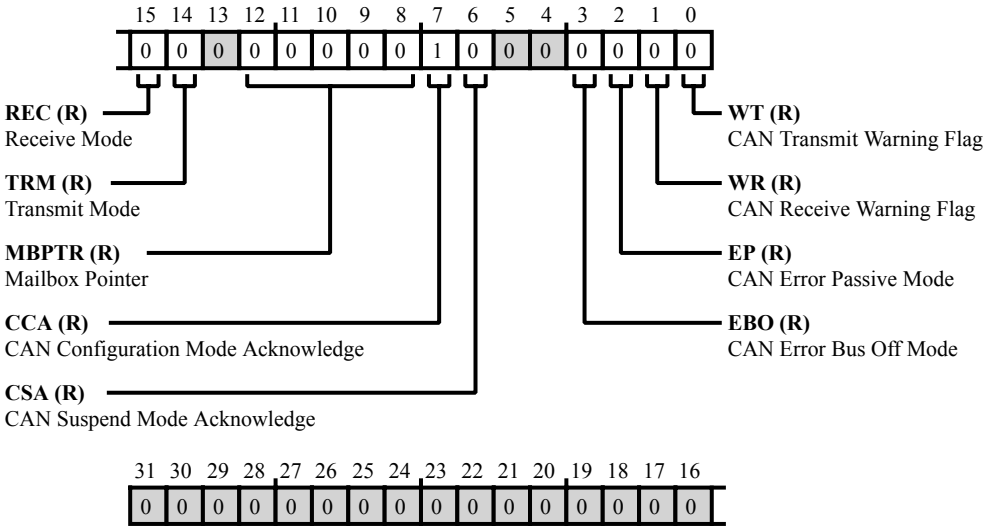

Figure 27-54: CAN_STAT Register Diagram

Table 27-49: 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 27-49: 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.

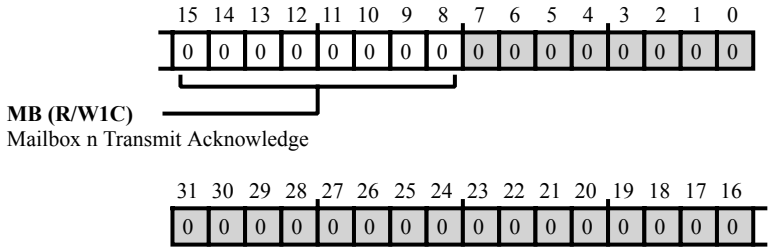

Figure 27-55: CAN_TA1 Register Diagram

Table 27-50: 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).

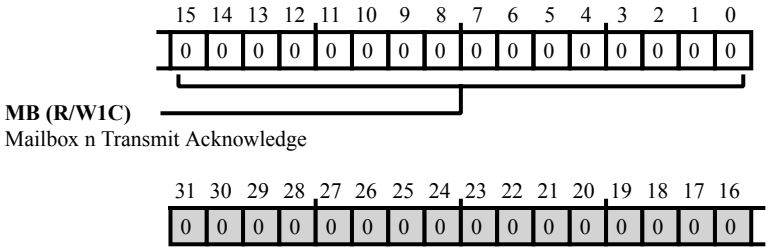

Figure 27-56: CAN_TA2 Register Diagram

Table 27-51: 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.

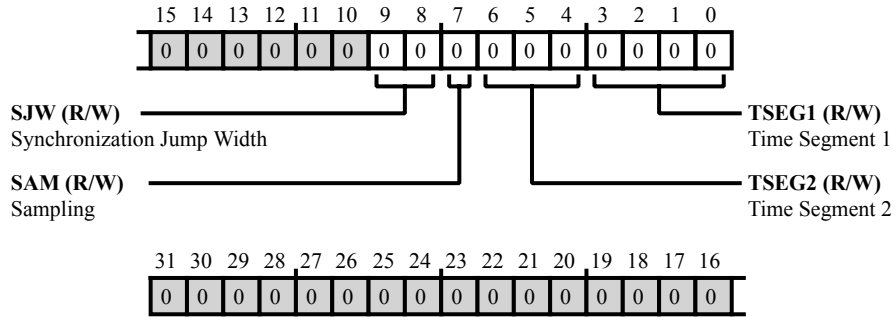

Figure 27-57: CAN_TIMING Register Diagram

Table 27-52: 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 SCLK0 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.
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

Table 27-52: CAN_TIMING Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
		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.

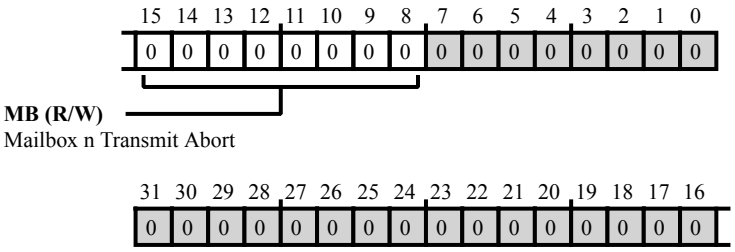

Figure 27-58: CAN_TRR1 Register Diagram

Table 27-53: 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.

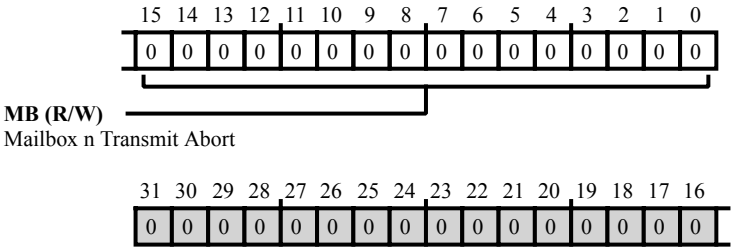

Figure 27-59: CAN_TRR2 Register Diagram

Table 27-54: 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.

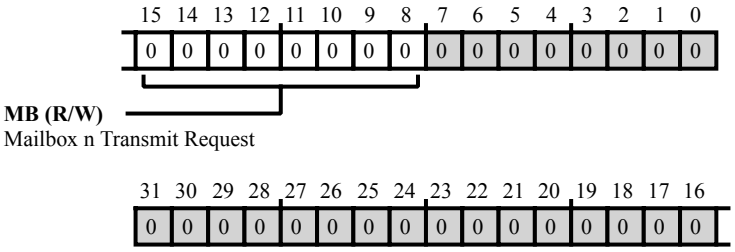

Figure 27-60: CAN_TRS1 Register Diagram

Table 27-55: 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.

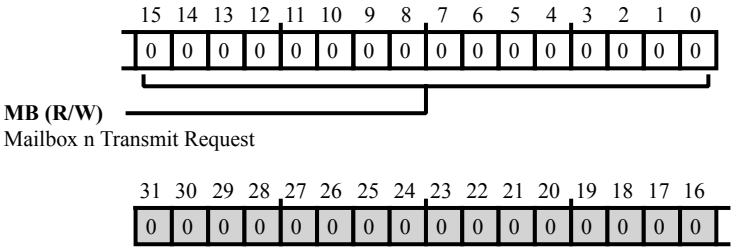

Figure 27-61: CAN_TRS2 Register Diagram

Table 27-56: 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.

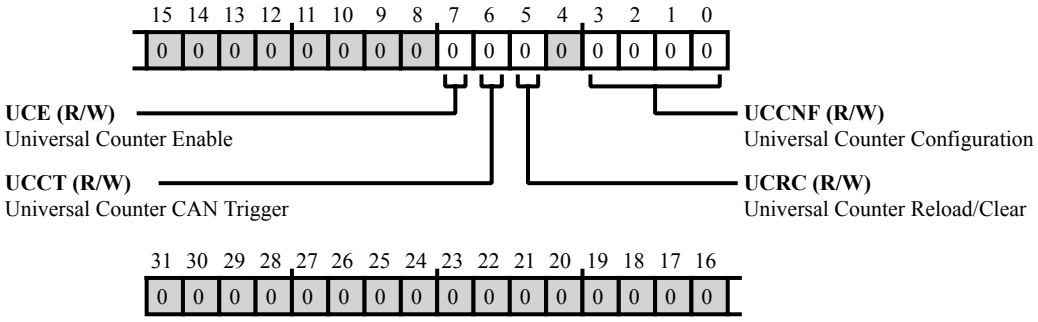

Figure 27-62: CAN_UCCNF Register Diagram

Table 27-57: 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 27-57: 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.

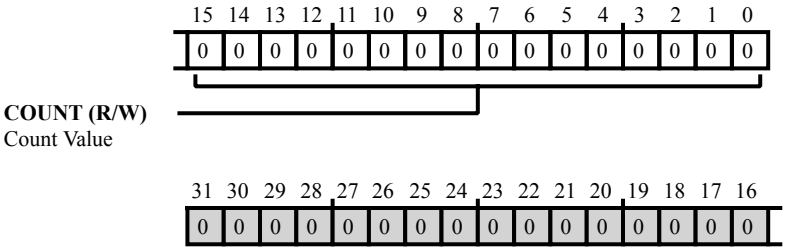

Figure 27-63: CAN_UCCNT Register Diagram

Table 27-58: 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.

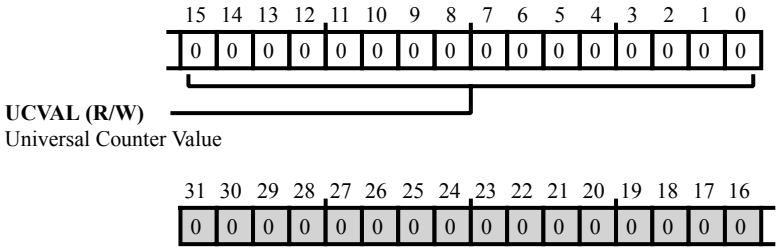

Figure 27-64: CAN_UCRC Register Diagram

Table 27-59: 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 Universal Serial Bus (USB)

The USB OTG controller provides a low-cost connectivity solution for consumer mobile devices such as cell phones, digital still cameras, and MP3 players. It allows these devices to transfer data using a point-to-point USB connection without the need for a personal computer host.

The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the On-The-Go (OTG) supplement to the USB 2.0 Specification.

NOTE: See the On-The-Go Supplement to the USB 2.0 Specification, Rev 1.0a; June 24, 2003; USB-IF and the Universal Serial Bus Specification 2.0.

The USB module supports:

- Host mode transfers at high-speed (480 Mbp/sec) rate
- Host mode transfers at full-speed (12 Mbp/sec) rate
- Host mode transfers at low-speed (1.5 Mbp/sec) rates. The connection to low-speed devices is only possible through a full-speed hub.
- Peripheral mode transfers at high-speed (480 Mbp/sec) rate
- Peripheral mode transfers at full-speed (12 Mbp/sec) rate

The USB controller uses a peripheral bus slave interface to access its control and status registers as well as read and write to the endpoint packet buffers. Data transfers to and from the USB controller through the 11 transmit and 11 receive endpoint FIFOs (EP1 – EP11), providing a total of 22 data endpoints.

USB Features

The USB controller provides the following features:

- Low-speed, full-speed, and high-speed rates supported
- One bidirectional control endpoint
- 11 transmit and 11 receive unidirectional endpoints
- 16 KB dynamically configured FIFO RAM

- Eight DMA master channels
- Two top-level maskable general-purpose interrupts
- Low-power wake-up on activity
- VBUS control interrupts for external analog VBUS control
- Session request protocol (SRP) and host negotiation protocol (HNP) capability
- Host transaction scheduling in hardware
- Soft connect or disconnect feature
- Full- and high-speed physical layer UTMI+ level 3 interface for on-chip PHY
- Backwards compatible with existing USB 1.1 hosts
- Support for Battery Charging Specification Revision 1.1

Only device requirements or system bandwidth limit the number of active endpoints at one time because each endpoint operates independently from the next. Software determines the type of transfer for each endpoint individually and also the manner in which it is transferred between the USB controller and memory (DMA or interrupt-based). The USB uses endpoint zero solely for receive and transmit control transfer. These transfers are used for device configuration and information gathering.

USB Functional Description

The following sections describe the function of the USB OTG interface.

USB Architectural Concepts

The USB controller operates in either of two USB operation modes (peripheral or host mode) at a given time.

In peripheral mode, the USB controller encodes, decodes, checks, and directs all USB packets sent and received, responding appropriately to host requests. Data is transferred from the processor core memory into the Tx FIFOs of the device onto USB as IN packets. In the other direction, USB OUT packets are received into the Rx FIFOs (having been sent from the host) and transferred to system memory for processing or storage. In peripheral mode, the USB controller acts as a slave device to another USB host; either a personal computer or another OTG host controller.

When operating in host mode, the USB controller uses simple hosting capabilities to master point-to-point connections with another USB peripheral, initiating transfers on the bus for the peripheral to respond. USB IN packets are received into the Rx FIFOs for transfer into the processor core memory. Data written into Tx FIFOs is transmitted onto the bus as USB OUT packets. In this mode, the USB controller encodes, decodes, and checks USB packets sent and received. The controller automatically schedules isochronous and interrupt transfers from the endpoint buffers. It performs one transaction every *n* frames, where n represents the polling interval programmed for the endpoint.

Any of the endpoints can be programmed to be written to or read from using the DMA master channels. This configuration provides the most efficient means of transferring data between the controller and on-chip memory.

USB endpoints 0 through 11 have DMA interrupt lines (`USB_DMA_IRQ`) providing a total of eight DMA request lines.

The USB provides two top-level maskable interrupts, each of which can be sourced from any or all of transmit endpoint status, receive endpoint status or global USB status. See Interrupt Signals for details.

The RAM interface of the USB controller supports a single block of synchronous single-port RAM used to buffer the USB packets.

16K bytes of SRAM are available.

The UTMI+ level 3 PHY interface provides a means of connecting a selection of high- or full-speed PHYs to the controller, from device-only PHYs through full OTG-compliant PHYs.

For details of the PHY interface, See UTMI Interface.

ATTENTION: Check the processor data sheet for requirements regarding minimum system clock frequency needed for proper USB operation.

The USB controller is configured as either a USB OTG A device or B device depending on the type of plug inserted into its USB receptacle. The state of the `USB_ID` (connector ID) pin determines this configuration.

The USB controller uses an asynchronous wake-up circuit to detect when another B device is asserting its D+ pull-up. This activity initiates the SRP (session request protocol) when all other clocks are off.

This slow clock is derived from SCLK0 and enabled using the `USB_PHY_CTL.EN` bit.

Use of the controller for OTG functionality requires the capability to:

- Drive VBUS (as a default A device powering the bus)
- Discharge VBUS (speeding up the time for VBUS to fall below the *SessionEnd* threshold as a B device checking initial conditions)
- Charge VBUS to 2.1 V (when initiating SRP as a B device).

The UTMI interface drives these controls, but the controller also provides a separate interrupt register, `USB_VBUS_CTL`, which represents the drive VBUS, discharge VBUS, and charge VBUS signaling. See the register section for more information on these controls.

Multi-Point Support

The USB controller has the facility, when operating in host mode, to act as the host to a range of USB peripheral devices.

High-speed, full-speed, or low-speed devices connect to the USB controller through a USB hub.

The USB controller, as part of its support for multiple devices, permits individual allocation of the functions of the target to the different Rx and Tx endpoints implemented. Furthermore, the USB controller can make this allocation dynamically, allowing the devices from the targeted peripheral list to be used in different combinations. The numbers of Tx and Rx endpoints implemented in the controller limit the combinations of peripheral devices that can be used together. Devices can only be added where the required endpoints remain available.

On-Chip Bus Interfaces

The USB controller uses two 32-bit wide independent bus interfaces, a master and a slave, to communicate with a processor-based subsystem. The slave interface allows the processor core to access the control and status registers (including DMA master registers) and the endpoint FIFOs. The integrated DMA uses the master interface to drive data into or out of the endpoint FIFOs with minimal processor core interaction. For more information, see USB Block Diagram.

FIFO Configuration

Each bidirectional endpoint (provided as two unidirectional endpoints) has its own endpoint number (0 for control, 1 on up for data transfer). Although two endpoints could use the same number, the endpoints can support different transfer types. Each of these bidirectional endpoints has a fixed region of the SRAM in the USB controller to which it has access. This feature dictates to some extent the types of transfers that can be used for that particular endpoint. This restriction follows from the maximum size of USB packets, which varies with each transfer type. The ***FIFO Sizes and Transfer Types*** table lists the endpoint FIFO configuration, with an indication of the transfer types possible for that particular buffer size.

Table 28-1: FIFO Sizes and Transfer Types

Bidirectional Endpoint (Rx and Tx)	**FIFO Size (each direction)**	**USB Transfer Types**
0	64 bytes	Size fixed for control transfers
1–4	Dynamically configured in powers of 2 from 8 to 8192 bytes	Bulk, Interrupt, Isochronous
1–11	Dynamically configured in powers of 2 from 8 to 8192 bytes	Bulk, Interrupt, Isochronous

Each endpoint FIFO can buffer one or two packets (in double-buffered mode). Double-buffering is recommended for most applications to improve efficiency by reducing the frequency of servicing for each endpoint.

Double-buffering bulk transactions means that data transfers over the USB are not slowed when packets are loaded or unloaded from the FIFO in the time it takes to transfer a packet. Double-buffering isochronous transactions allows more time to load or unload the FIFO. It also allows the usage of the SOF interrupt to service the endpoint rather than the endpoint interrupt. This functionality has the following advantages:

- Easy detection of lost packets
- Regular interrupt timing (making it easier to source or sink the data)

- If the USB controller uses more than one isochronous endpoint, one interrupt can service all the endpoints.

The USB controller uses the transmit or receive FIFO address registers to specify the address of each endpoint FIFO.

Clocking

The USB controller uses the system clock SCLK0 to generate an internal clock (CLK) used to clock the USB registers.

For proper operation, refer to device datasheet for minimum system clock SCLK0 value.

NOTE: For best performance (best signal integrity), follow the guidelines in the data sheet for selecting an input clock frequency.

When the controller is in the SUSPEND state and when no session is active, the clock and much of the USB controller is stopped to reduce power consumption. The clock becomes operational again when RESUME signaling is detected on the USB lines.

UTMI Interface

The interface to the on-chip PHY uses the industry-standard UTMI+ (universal transceiver macro interface) level 3.

This interface provides full- and high-speed device and OTG functionality and supports communication to a hub.

The PHY is a mixed-signal block and includes the following:

- Full-speed and high-speed drivers and receivers (single-ended and differential)
- Full-speed and high-speed CDR
- Full-speed or high-speed shift registers, NRZI encode or decode and bit-stuff encode or decode
- Data line pull-up and pull-down resistors
- VBUS and `USB_ID` level detection
- Host disconnect detection

Although the UTMI specification indicates that VBUS charging, driving and discharging happen inside the PHY, for process-restricting and power reasons, implement these functions off-chip in a separate USB charge-pump chip.

ADSP-BF70x USB Register List

The Universal Serial Bus controller (USB) is a multi-point high-speed dual-role USB 2.0-compliant controller. The USB controller can operate in a traditional USB peripheral-only mode as well as the host mode presented in the On-The-Go (OTG) supplement to the USB 2.0 Specification, Rev 1.0a; June 24, 2003; USB-IF. A set of registers governs USB controller operations. For more information on USB controller functionality, see the USB controller register descriptions.

Table 28-2: ADSP-BF70x USB Register List

Name	Description
USB_BAT_CHG	Battery Charging Control Register
USB_CT_HHSRTN	Host High-Speed Return to Normal Register
USB_CT_HSBT	High-Speed Timeout Register
USB_CT_UCH	Chirp Timeout Register
USB_DEV_CTL	Device Control Register
USB_DMA[n]_ADDR	DMA Channel n Address Register
USB_DMA[n]_CNT	DMA Channel n Count Register
USB_DMA[n]_CTL	DMA Channel n Control Register
USB_DMA_IRQ	DMA Interrupt Register
USB_EP0I_CFGDATA[N]	EP0 Configuration Information Register
USB_EP0I_CNT[N]	EP0 Number of Received Bytes Register
USB_EP0I_CSR[N]_H	EP0 Configuration and Status (Host) Register
USB_EP0I_CSR[N]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0I_NAKLIMIT[N]	EP0 NAK Limit Register
USB_EP0I_TYPE[N]	EP0 Connection Type Register
USB_EP0_CFGDATA[n]	EP0 Configuration Information Register
USB_EP0_CNT[n]	EP0 Number of Received Bytes Register
USB_EP0_CSR[n]_H	EP0 Configuration and Status (Host) Register
USB_EP0_CSR[n]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0_NAKLIMIT[n]	EP0 NAK Limit Register
USB_EP0_TYPE[n]	EP0 Connection Type Register
USB_EPINFO	Endpoint Information Register
USB_EPI[N]_RXCNT	EPn Number of Bytes Received Register
USB_EPI[N]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EPI[N]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EPI[N]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EPI[N]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EPI[N]_RXTYPE	EPn Receive Type Register
USB_EPI[N]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EPI[N]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EPI[N]_TXINTERVAL	EPn Transmit Polling Interval Register

Table 28-2: ADSP-BF70x USB Register List (Continued)

Name	Description
USB_EPI[N]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EPI[N]_TXTYPE	EPn Transmit Type Register
USB_EP[n]_RXCNT	EPn Number of Bytes Received Register
USB_EP[n]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EP[n]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EP[n]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EP[n]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EP[n]_RXTYPE	EPn Receive Type Register
USB_EP[n]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EP[n]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EP[n]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EP[n]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EP[n]_TXTYPE	EPn Transmit Type Register
USB_FADDR	Function Address Register
USB_FIFOB[n]	FIFO Byte (8-Bit) Register
USB_FIFOH[n]	FIFO Half-Word (16-Bit) Register
USB_FIFO[n]	FIFO Word (32-Bit) Register
USB_FRAME	Frame Number Register
USB_FS_EOF1	Full-Speed EOF 1 Register
USB_HS_EOF1	High-Speed EOF 1 Register
USB_IEN	Common Interrupts Enable Register
USB_INDEX	Index Register
USB_INTRRX	Receive Interrupt Register
USB_INTRRXE	Receive Interrupt Enable Register
USB_INTRTX	Transmit Interrupt Register
USB_INTRTXE	Transmit Interrupt Enable Register
USB_IRQ	Common Interrupts Register
USB_LINKINFO	Link Information Register
USB_LPM_ATTR	LPM Attribute Register
USB_LPM_CTL	LPM Control Register
USB_LPM_FADDR	LPM Function Address Register

Table 28-2: ADSP-BF70x USB Register List (Continued)

Name	Description
USB_LPM_IEN	LPM Interrupt Enable Register
USB_LPM_IRQ	LPM Interrupt Status Register
USB_LS_EOF1	Low-Speed EOF 1 Register
USB_MP[n]_RXFUNCADDR	MPn Receive Function Address Register
USB_MP[n]_RXHUBADDR	MPn Receive Hub Address Register
USB_MP[n]_RXHUBPORT	MPn Receive Hub Port Register
USB_MP[n]_TXFUNCADDR	MPn Transmit Function Address Register
USB_MP[n]_TXHUBADDR	MPn Transmit Hub Address Register
USB_MP[n]_TXHUBPORT	MPn Transmit Hub Port Register
USB_PHY_CTL	PHY Control Register
USB_PLL_OSC	PLL and Oscillator Control Register
USB_POWER	Power and Device Control Register
USB_RAMINFO	RAM Information Register
USB_RQPKTCNT[n]	EPn Request Packet Count Register
USB_RXFIFOADDR	Receive FIFO Address Register
USB_RXFIFOSZ	Receive FIFO Size Register
USB_SOFT_RST	Software Reset Register
USB_TESTMODE	Testmode Register
USB_TXFIFOADDR	Transmit FIFO Address Register
USB_TXFIFOSZ	Transmit FIFO Size Register
USB_VBUS_CTL	VBUS Control Register
USB_VPLEN	VBUS Pulse Length Register

ADSP-BF70x USB Interrupt List

Table 28-3: ADSP-BF70x USB Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
77	USB0_STAT	USB0 Status/FIFO Data Ready	Level	
78	USB0_DATA	USB0 DMA Status/Transfer Complete	Level	

ADSP-BF70x USB Trigger List

Table 28-4: ADSP-BF70x USB Trigger List Masters

Trigger ID	Name	Description	Sensitivity
36	USB0_DATA	USB0 DMA Status/Transfer Complete	Level

Table 28-5: ADSP-BF70x USB Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

USB Block Diagram

The ***USB OTG Controller Block Diagram*** shows the functional blocks within the USB. For more information about the blocks, see the USB Functional Description.

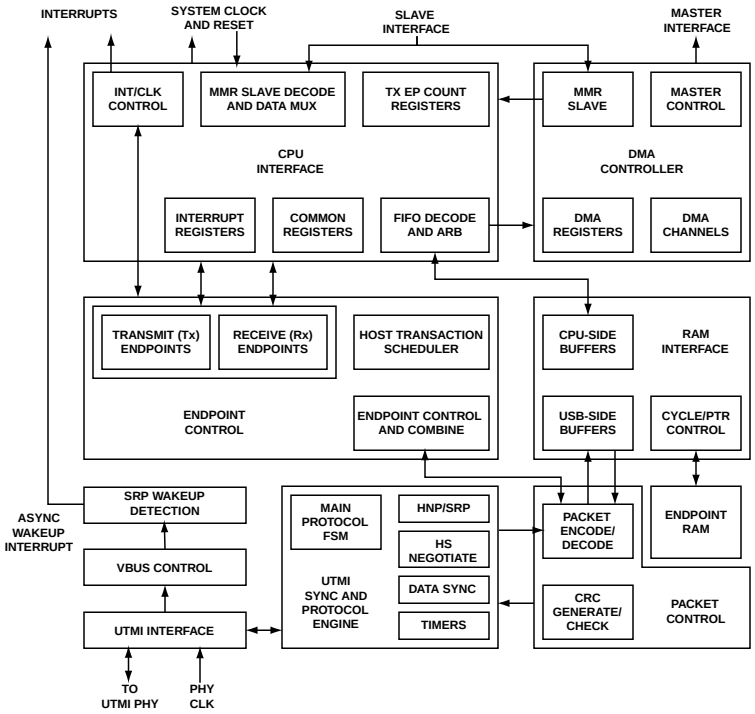

Figure 28-1: USB OTG Controller Block Diagram

USB Definitions

A list of common USB terms and their definitions as used in this specification and based on the USB controller follows:

'A' Device

The USB device with a mini-A plug inserted into its receptacle. The A device always supplies power to VBUS.

'B' Device

The USB device with a standard-B or mini-B plug inserted into its receptacle. The B device starts a session as the peripheral.

Bidirectional endpoint

An endpoint that can concurrently support both receive and transfer packets.

Control endpoint

An endpoint used only for transfer of USB control packets for setup and configuration. In all USB devices, the control endpoint refers to the bidirectional endpoint 0.

Dual role device

A USB device that can operate either as the USB host in an OTG session or as a traditional USB peripheral.

Endpoint

A single physical communication channel for USB, implemented as a FIFO and control logic for that endpoint. Each endpoint has an associated USB transfer type, maximum packet size, bandwidth requirement, endpoint number, and (often) a fixed transfer direction.

Frame

A regular, fixed 1 ms timeslot that can contain several transactions. The transfer type determines the permissible transactions for a given endpoint.

HNP

Host negotiation protocol. Part of the USB OTG supplement that allows the host function to be transferred between two connected dual role devices.

Packet

The lowest level of data exchange on USB. The transfer type and buffer size of the USB peripheral determine the size.

PHY

The PHY is a transceiver circuit that implements the physical layer of USB. For full speed USB OTG, this circuit includes line drivers and receivers, pull-up, or pull-down resistors as well as device ID and VBUS level detection.

Session

A period during which USB transfers take place within an OTG connection. The A device (drives VBUS) or B device (initiates SRP) can initiate this period. VBUS is powered during a session.

SRP

Session request protocol. Part of the USB OTG supplement that allows a B device to turn on VBUS and initiate a USB session.

Transaction

Collection of one or more packets in sequence

Transfer

Collection of one or more transfers in sequence

Unidirectional endpoint

Endpoint with its direction fixed in a single direction (for example, it can only receive packets from the USB) in both host and peripheral modes.

USB References

The following references provide further information regarding the USB.

- *On-The-Go Supplement to the USB 2.0 Specification*, Rev 1.0a, June 24, 2003, USB-IF
- *Universal Serial Bus Specification 2.0*

USB Operating Modes

The USB OTG interface can operate in peripheral mode or host mode.

When the USB controller operates in peripheral mode, the controller can be attached to a conventional host (such as a personal computer) or another OTG device operating in host mode. The second device can be high speed or full speed. When linked to another peripheral device, the USB controller can also act as the host. If the other device is also a dual role controller, the two devices can switch roles, as needed.

The role the USB controller takes depends on the way the devices are cabled together. Each USB cable has an A and a B device end. If the A end of the cable is plugged into the device containing the USB controller, the USB controller takes the role of the host device. It goes into host mode (in this case, the USB_DEV_CTL.HOSTMODE bit is set to 1). If the B of the cable is plugged in, the USB controller goes instead into peripheral mode (and the USB_DEV_CTL.HOSTMODE bit remains at 0).

When both devices contain dual role controllers, signaling can be used to switch the roles of the two devices, without switching the cable connecting the two devices. See Host Negotiation Protocol for details on the conditions under which the USB controller can switch between peripheral and host mode.

NOTE: The multi-point capability of the USB controller is associated with a range of registers recording the allocation of device functions to individual endpoints and device function characteristics. These characteristics include endpoint number, operating speed, and transaction type on an endpoint-by-endpoint basis. These registers are principally associated with the use of the USB controller as the host to a number of devices. However, set the registers when the core is used as the host for a single target device.

To enable the USB:

1. Configure the USB PLL multiplier settings in the USB PLL control register. Check the processor data sheet for the requirements for input clock frequency.
2. Enable the USB PHY by setting the USB_PHY_CTL.EN bit.
3. Poll the bit in the USB PLL control register to ensure that the USB PLL has locked to the new frequency.

Peripheral Mode

USB OTG interface operations for the peripheral mode differ from host mode in a number of ways. The following sections describe peripheral mode operations.

Endpoint Setup

In peripheral mode, the USB uses a few endpoint-specific configuration bits when setting up an endpoint for transfer for all types of peripheral transfer. The configuration determines how the processor core interacts with the endpoint FIFO.

One key parameter required before a transfer can occur through an endpoint is the maximum USB packet size that the endpoint can support. The software sets this value. It depends on various system constraints. These constraints include the size of hardware FIFO available and system latencies as well as the USB transfer type and class used. The USB_EP[n]_TXMAXP or USB_EP[n]_RXMAXP registers define the maximum amount of data that can be transferred to the selected endpoint in a single frame. The value must match the programmed maximum individual packet size (*MaxPktSize*) of the standard endpoint descriptor for the endpoint.

For transmit endpoints, program the maximum packet size using the USB_EP[n]_TXMAXP. For receive endpoints, the USB uses the USB_EP[n]_RXMAXP register. The maximum packet size must not exceed the actual hardware endpoint FIFO size.

The settings in the USB_RXFIFOSZ or USB_TXFIFOSZ register determine the corresponding sizes of the transmit or receive FIFOs, as well as, single or double buffered mode for endpoints 1 to 11.

Because the USB controller uses a 32-bit interface, choose an even number for the value of *MaxPktSize*. This selection simplifies transferring data between FIFOs and the processor core.

Configure more setup parameters using the USB_EP[n]_TXCSR_H or USB_EP[n]_RXCSR_H register (depending on whether the endpoint in question is for receive or transmit). The USB uses the USB_EP[n]_RXCSR_H.DMAREQEN bit in this register to enable the assertion of the appropriate DMA request whenever the endpoint is able to receive or transmit another packet. The USB uses the USB_EP[n]_RXCSR_H.AUTOCLR and USB_EP[n]_RXCSR_H.AUTOREQ bits to set the FIFO ready triggers (USB_EP[n]_RXCSR_H.RXPKTRDY and USB_EP[n]_TXCSR_H.TXPKTRDY) automatically whenever a packet is transferred to streamline DMA operation for transfers that span multiple packets. Note, however, that the USB cannot use USB_EP[n]_RXCSR_H.AUTOCLR and USB_EP[n]_RXCSR_H.AUTOREQ bits with high-bandwidth endpoints. Refer to the following register sections for more information: USB_EP[n]_TXCSR_H,USB_EPI[N]_RXCSR_P,USB_EPI[N]_RXCSR_H,USB_EP[n]_TXCSR_P.

IN Transactions as a Peripheral

When the USB controller operates in peripheral mode, the transmit FIFOs handle data for IN transactions. The maximum size of data packet that can be placed in a FIFO for a transmit endpoint is programmable. When applicable, the value written to the USB_EP[n]_TXMAXP register for that endpoint determines the size (maximum payload multiplied by the number of transactions per micro-frame).

The maximum packet size set for any endpoint must not exceed the FIFO size. (See FIFO Configuration.)

ATTENTION: Do not write to the USB_EP[n]_TXMAXP register while there is data in the FIFO, as unexpected results can occur.

The following sections describe the two types of packet buffering used for IN transactions.

Single packet buffering.

Set the USB_EP[n]_TXCSR_P.TXPKTRDY bit as each packet for transmission is loaded into the transmit FIFO. If the USB_EP[n]_TXCSR_P.AUTOSET bit is set, the USB_EP[n]_TXCSR_P.TXPKTRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, and where auto-set cannot be used (high-bandwidth isochronous or interrupt transactions), always set the USB_EP[n]_TXCSR_P.TXPKTRDY bit manually (for example by the processor core).

When the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set, either manually or automatically, the USB_EP[n]_TXCSR_P.NEFIFO bit is also set and the packet is ready to be sent. When the packet is successfully sent, both the USB_EP[n]_TXCSR_P.TXPKTRDY and USB_EP[n]_TXCSR_P.NEFIFO bits are cleared. The USB controller generates the appropriate transmit endpoint interrupt (if enabled). The next packet can then be loaded into the FIFO.

Double packet buffering.

Set the USB_EP[n]_TXCSR_P.TXPKTRDY bit as each packet for transmission is loaded into the transmit FIFO. If the USB_EP[n]_TXCSR_P.AUTOSET bit is set, the USB_EP[n]_TXCSR_P.TXPKTRDY bit is automatically set when a maximum-sized packet is loaded into the FIFO. For packet sizes less than the maximum, and where auto-set cannot be used (high-bandwidth isochronous or interrupt transactions), always set the USB_EP[n]_TXCSR_P.TXPKTRDY bit manually (for example by the processor core).

When the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set, either manually or automatically, the USB_EP[n]_TXCSR_P.NEFIFO bit also is set. The USB_EP[n]_TXCSR_P.TXPKTRDY bit is then immediately cleared (and an interrupt generated, if enabled). A second packet can now be loaded into the transmit FIFO and the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set again (either manually or automatically if the packet is the maximum size). Both packets are now ready for transmission.

When the first packet is successfully sent, the USB_EP[n]_TXCSR_P.TXPKTRDY bit is cleared. The USB controller generates the appropriate transmit endpoint interrupt (if enabled) to signal that another packet can now be loaded into the transmit FIFO. The state of the USB_EP[n]_TXCSR_P.NEFIFO bit indicates how many packets can be loaded. If the USB_EP[n]_TXCSR_P.NEFIFO bit is set, then there is another packet in the FIFO and only one more packet can be loaded. If the USB_EP[n]_TXCSR_P.NEFIFO bit is cleared, then there are no packets in the FIFO and two more packets can be loaded.

OUT Transactions as a Peripheral

When the USB controller operates in peripheral mode, the receive FIFOs handle data for OUT transactions.

The value written to the USB_EP[n]_RXMAXP register for an endpoint determines the maximum amount of data received by a receive endpoint in any frame. The value is programmable. The maximum packet size must not exceed the FIFO size.

The value written to the USB_EP[n]_RXMAXP register for an endpoint determines maximum amount of data received by a receive endpoint in any micro-frame (in high-speed mode). The value is programmable. It is the maximum payload multiplied by the number of transactions per micro-frame. The maximum packet size must not exceed the FIFO size.

If the size of the receive endpoint FIFO is less than twice the maximum packet size for this endpoint, only one data packet can be buffered in the FIFO. Single buffering is selected. (The size is set in the USB_EP[n]_RXMAXP register). When a packet is received and placed in the receive FIFO, the USB_EP[n]_RXCSR_P.RXPKTRDY bit and the USB_EP[n]_RXCSR_P.FIFOFULL bit are set. The USB controller generates the appropriate receive endpoint interrupt (if enabled) to signal that a packet can now be unloaded from the FIFO. After the packet is unloaded, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit to allow reception of more packets. If the USB_EP[n]_RXCSR_P.AUTOCLR bit is set and a maximum-sized packet is unloaded from the FIFO, the USB_EP[n]_RXCSR_P.RXPKTRDY bit is cleared automatically. The USB_EP[n]_RXCSR_P.FIFOFULL bit is also cleared. For packet sizes less than the maximum, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit manually (for example by the processor core).

If double packet buffering is enabled, then two data packets can be buffered. When the first packet for reception is loaded into the receive FIFO, the USB_EP[n]_RXCSR_P.RXPKTRDY bit is set. The USB controller generates the appropriate receive endpoint interrupt (if enabled) to signal that a packet can now be unloaded from the FIFO. The

USB_EP[n]_RXCSR_P.FIFOFULL bit is not set. This bit is only set if a second packet is received and loaded into the receive FIFO.

After the first packet is unloaded, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit to allow reception of further packets. If the USB_EP[n]_RXCSR_P.AUTOCLR bit is set and a maximum-sized packet is unloaded from the FIFO, the USB_EP[n]_RXCSR_P.RXPKTRDY bit is cleared automatically. For packet sizes less than the maximum, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit manually (for example by the processor core).

If the USB_EP[n]_RXCSR_P.FIFOFULL bit is set to 1 when USB_EP[n]_RXCSR_P.RXPKTRDY is cleared, the USB controller first clears the USB_EP[n]_RXCSR_P.FIFOFULL bit. The controller then sets the USB_EP[n]_RXCSR_P.RXPKTRDY bit again, indicating that there is another packet waiting in the FIFO for unloading.

High-Bandwidth Isochronous or Interrupt Transactions

High-bandwidth isochronous or interrupt transactions use much the same protocol as other isochronous or interrupt transactions. There are, however, some special features to conducting high-bandwidth transactions.

- When setting the maximum packet size handled by the endpoint in the USB_EP[n]_TXMAXP/ USB_EP[n]_RXMAXP registers, set the maximum number of transactions per micro-frame using the USB_EP[n]_TXMAXP.MULTM1 and USB_EP[n]_RXMAXP.MULTM1 bits.

  The maximum number of transactions (2 or 3) also represents the maximum number of sections in which any single high-bandwidth packet can be transferred. The configuration sets the maximum size of the packet to 2 or 3 times the maximum payload specified for the endpoint in the same register.

  NOTE: The maximum payload that can be sent in any transaction is 1K byte.

- When sending packets, set the USB_EP[n]_TXCSR_P.TXPKTRDY bit using the application software. Similarly, when unloading packets from the receive endpoint FIFO, clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit using the application software.

  CAUTION: The AutoSet and AutoClear functions cannot be used to set and clear these bits in high-bandwidth transactions.

- The transmission of packets as a number of sections introduces a further type of error – the transmission of incomplete packets.

  For transmit endpoints, transmitting incomplete packets principally applies when the interface is in peripheral mode. It occurs when the transmission fails to receive enough IN tokens from the host to send all the parts of the data packet. It can also apply to high-bandwidth interrupt transactions in host mode where the core does not receive any response from the device to which the packet is transmitted. In both cases, the USB_EP[n]_TXCSR_P.INCOMPTX bit is set.

  For receive endpoints, an incomplete packet issue can occur. The PIDs of the received parts of the data packet show that one or more parts of the data packet have not been received. When this event happens, the USB_EP[n]_RXCSR_P.INCOMPRX bit is set. Usually this bit is set in peripheral mode. However, it can also be

set in host mode (using the USB_EP[n]_RXCSR_H.INCOMPRX bit). This event occurs when the USB communicates with a device that fails to respond in accordance with the USB protocol.

High Bandwidth Isochronous or Interrupt IN Endpoints

In high-speed mode, transmit endpoints configured for high-bandwidth isochronous or interrupt transactions can transmit up to three USB packets in any micro-frame. The transmission occurs with a payload of up to 1024 bytes in each packet, corresponding to a data transfer rate of up to 3072 bytes per micro-frame.

The ***High Bandwidth IN Endpoints*** figure provides an overview of high-bandwidth IN endpoints in USB.

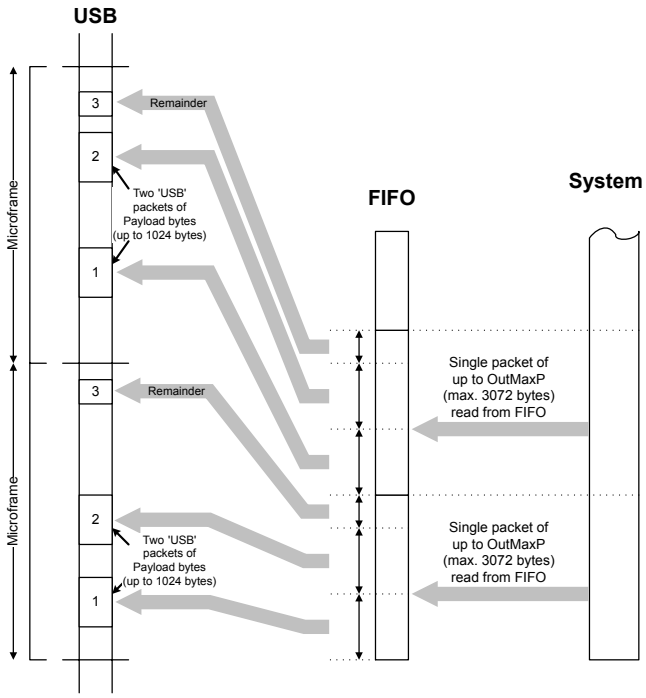

Figure 28-2: High Bandwidth IN Endpoints

The USB controller supports these transfers by permitting the loading of data packets with up to three times the normal packet size into the associated FIFO in a single transaction. From the software viewpoint in the processor core, the ***High Bandwidth IN Endpoints*** figure describes the operation for single or double packet buffering (as appropriate). One exception is that the USB_EP[n]_TXCSR_P.TXPKTRDY bit must always be set manually (for example by the processor core) as the auto set feature does not operate with high-bandwidth isochronous transfers.

The USB controller automatically splits any data packet loaded into the FIFO that is larger than the maximum into USB packets of the maximum payload, or smaller, for transmission. The following settings define the number of USB packets transmitted per micro-frame and the maximum payload in each packet:

- Use the USB_EP[n]_TXMAXP.MAXPAY bits to set the maximum payload in any USB packet
- Use the USB_EP[n]_TXMAXP.MULTM1 bits to set the maximum number of such packets for transmission in one micro-frame (2 or 3)

Together, these settings define the maximum size of packet that can be loaded into the FIFO.

At least one USB packet always is sent. The number of further USB packets sent in the same micro-frame depends on the amount of data loaded into the FIFO. The USB_EP[n]_TXCSR_P.TXPKTRDY bit is cleared and an interrupt is generated only when all the packets have been sent. Each USB packet is sent in response to an IN token. If, at the end of a micro-frame, the USB controller has not received enough IN tokens to send all the USB packets, the remaining data is flushed from the FIFO. (For example, one of the IN tokens received is corrupt). The USB_EP[n]_TXCSR_P.TXPKTRDY bit is cleared and the USB_EP[n]_TXCSR_P.INCOMPTX bit is set to indicate that not all of the data loaded into the FIFO transmitted.

High-Bandwidth Isochronous or Interrupt OUT Endpoints

In high-speed mode, isochronous receive endpoints can receive up to three USB packets in any micro-frame. The reception occurs with a payload of up to 1024 bytes in each packet, corresponding to a data transfer rate of up to 3072 bytes per micro-frame. Similarly, the USB controller can receive high-bandwidth interrupt transactions in host mode, but there is no support for high-bandwidth interrupt transactions in peripheral mode.

The ***High-Bandwidth OUT Endpoints*** figure shows an overview of high-bandwidth OUT endpoints.

The USB controller supports this rate by automatically combining all the USB packets received during a micro-frame into a single packet of up to 3 normal packets within the receive FIFO. From the software viewpoint in the processor core, the ***High Bandwidth IN Endpoints*** figure describes the operation for single or double packet buffering (as appropriate). One exception is that the USB_EP[n]_RXCSR_P.RXPKTRDY bit always must be cleared manually (for example by the processor core) because the auto-clear function does not operate with high-bandwidth isochronous transfers.

The maximum number of USB packets that can be received in any micro-frame and the maximum payload of these packets are configured as follows:

- Use the USB_EP[n]_RXMAXP.MAXPAY bits to set the maximum payload in any USB packet
- Use the USB_EP[n]_RXMAXP.MULTM1 bits to set the maximum number of these packets that can be received in a micro-frame (2 or 3)

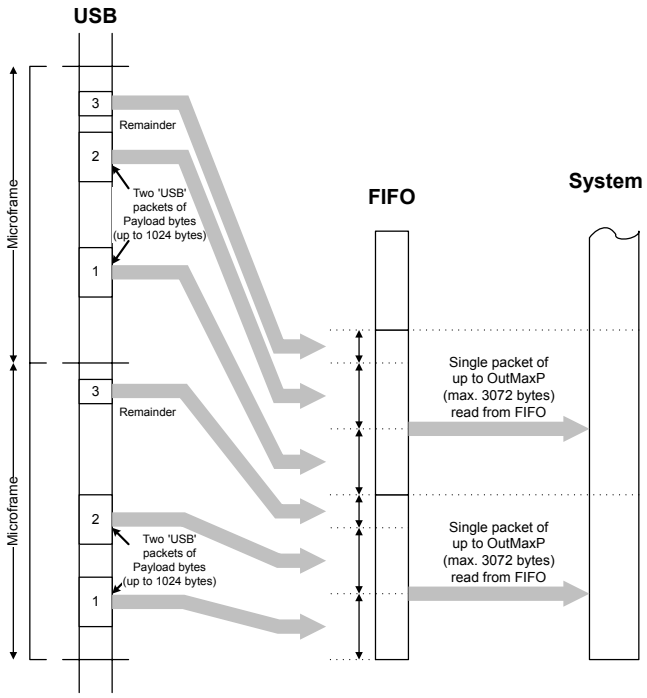

Figure 28-3: High-Bandwidth OUT Endpoints

The number of USB packets sent in any micro-frame depends on the amount of data for transfer, and is indicated through the PIDs used for the individual packets. If the indicated number of packets have not been received by the end of a micro-frame, the `USB_EP[n]_RXCSR_P.INCOMPRX` bit is set to indicate that the data in the FIFO is incomplete. An interrupt is still generated to allow the data that has been received to be read from the FIFO.

Peripheral Transfer Work Flows

The USB transfer types (control, bulk, isochronous, and interrupt transfers) have different system requirements as well as individual USB transfer-specific features. Software handles each type differently. There is no uniform way of doing transfers across all transfer types using the USB controller.

The following sections provide some guidelines for peripheral mode transfer flows for each of the transfer types, in both IN (transmit) and OUT (receive) directions. For bulk endpoints, the optimal transfer flow depends on whether the final size of the transfer is known or unknown. The USB driver class in use determines whether the transfer size is known or not. Some drivers define the complete transfer size, and others operate on a packet-by-packet basis using a short packet to denote the end of a transfer. (A short packet is less than the value configured in the `USB_EP[n]_TXMAXP` register or less than the value configured in the `USB_EP[n]_RXMAXP` register).

Each of the work flows uses the following common steps.

1. Configure the endpoint control and status registers and the `USB_EP[n]_TXMAXP` or `USB_EP[n]_RXMAXP` value.
2. Configure the appropriate data transfer mechanism (DMA or interrupt setup).
3. Data transfer occurs.

The work flows do not describe the actions of the USB controller immediately preceding the endpoint setup. (For example, the reception of an IN/OUT token from the host, token validity checking, or NAK generation, among others). Note also that there is no error-handling contained in the work flows (for example, checking the `USB_EP[n]_RXCSR_P.FIFOFULL` bit before writing data).

The proceeding sections use terms packets, frames, and transfers with their strict USB definitions (see USB Definitions).

Control Transactions as a Peripheral

Endpoint 0 is the main control endpoint of the USB controller. As such, the routines required to service endpoint 0 are more complicated than the routines required to service other endpoints.

The software is required to handle all the standard device requests that the USB controller sends or receives through endpoint 0. The *Universal Serial Bus Specification*, Revision 2.0, Chapter 9 describes the requirements. The protocol for these device requests involves different numbers and types of transactions per transfer. To accommodate this functionality, the processor must take a state machine approach to command decoding and handling.

The standard device requests a USB peripheral receives fits into three categories:

- Zero data requests (in which the command includes all the information)
- Write requests (in which extra data follows the command)
- Read requests (in which the device sends data back to the host)

The following sections describe the sequence of actions that the software must perform to process these different types of device request.

Write Requests

The host sends an 8-byte command followed by a write request that contains an extra packet (or packets) of data. An example of a write standard device request is `SET_DESCRIPTOR`.

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The `USB_EP[n]_RXCSR_P.RXPKTRDY` bit is also set. The host then reads and decodes the 8-byte command from the endpoint 0 FIFO.

As with a zero data request, write to the `USB_EP0_CSR[n]_P` register to set the `USB_EP0_CSR[n]_P.SPKTRDY` bit. (The event indicates that the host read the command from the FIFO). But, in this case, do not set the `USB_EP0_CSR[n]_P.DATAEND` bit (indicating that more data is expected).

When a second endpoint 0 interrupt is received, the `USB_EP0_CSR[n]_P` register is read to check the endpoint status. The `USB_EP0_CSR[n]_P.RXPKTRDY` bit is set to indicate that a data packet is received. Read the `USB_EP0_CNT[n]` register to determine the size of this data packet. The data packet can then be read from the endpoint 0 FIFO.

If the length of the data associated with the request is greater than the maximum packet size for endpoint 0, the host sends more data packets. (The WLENGTH field in the command indicates the length of the data). In this case, the USB_EP0_CSR[n]_P.SPKTRDY bit is set, but do not set the USB_EP0_CSR[n]_P.DATAEND bit.

When all the expected data packets have been received, software writes to the USB_EP0_CSR[n]_P register to set the USB_EP0_CSR[n]_P.SPKTRDY bit and to set the USB_EP0_CSR[n]_P.DATAEND bit (indicating that no more data is expected).

When the host moves to the status stage of the request, software generates another endpoint 0 interrupt to indicate that the request has completed. No further action is required from the software; the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or cannot be executed, then when the host decodes it, software must write to the USB_EP0_CSR[n]_P register. This operation sets the USB_EP0_CSR[n]_P.SPKTRDY bit and the USB_EP0_CSR[n]_P.SENDSTALL bit. When the host sends more data, the USB controller sends a stall to tell the host that the request was not executed. Software generates an endpoint 0 interrupt and the USB_EP0_CSR[n]_P.SENTSTALL bit is set.

If the host sends more data after the USB_EP0_CSR[n]_P.DATAEND has been set, then the USB controller sends a stall. Software generates an endpoint 0 interrupt and the USB_EP0_CSR[n]_P.SENTSTALL bit is set.

Read Requests

The function sends the 8-byte command followed by read requests containing a packet (or packets) of data to the host. Examples of standard device requests for read are:

- GET_CONFIGURATION
- GET_INTERFACE
- GET_DESCRIPTOR
- GET_STATUS
- SYNCH_FRAME

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The USB_EP[n]_RXCSR_P.RXPKTRDY bit is also set. The host then reads and decodes the 8-byte command from the endpoint 0 FIFO. Write the USB_EP0_CSR[n]_P.SPKTRDY bit (indicating that the command has been read from the FIFO).

The data to transmit to the host is written to the endpoint 0 FIFO. If the size of the transmit data is greater than the maximum packet size for endpoint 0, only the maximum packet size is written to the FIFO. The USB_EP0_CSR[n]_P.TXPKTRDY bit is then set (indicating that there is a packet in the FIFO to be sent). When the packet has been sent to the host, software generates another endpoint 0 interrupt. The next data packet can be written to the FIFO.

When the last data packet has been written to the FIFO, software writes to the USB_EP0_CSR[n]_P register to set the USB_EP0_CSR[n]_P.TXPKTRDY bit and to set the USB_EP0_CSR[n]_P.DATAEND bit. (This activity indicates that there is no more data after this packet).

When the host moves to the status stage of the request, software generates another endpoint 0 interrupt to indicate that the request has completed. No further action is required from the software; the interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when the host decodes it, software must write to the USB_EP0_CSR[n]_P register. This operation sets the USB_EP0_CSR[n]_P.SPKTRDY bit and the USB_EP0_CSR[n]_P.SENDSTALL bit. When the host requests data, the USB controller sends a stall to tell the host that the request was not executed. Software generates an endpoint 0 interrupt and the USB_EP0_CSR[n]_P.SENTSTALL bit is set.

If the host requests more data after USB_EP0_CSR[n]_P.DATAEND is set, then the USB controller sends a stall. Software generates an endpoint 0 interrupt and the USB_EP0_CSR[n]_P.SENTSTALL bit is set.

Zero Data Requests

Zero data requests have all their information included in the 8-byte command and do not require transfer of extra data.

Examples of standard device requests for zero data are:

- SET_FEATURE
- CLEAR_FEATURE
- SET_ADDRESS
- SET_CONFIGURATION
- SET_INTERFACE

As with all requests, the sequence of events begins when the software receives an endpoint 0 interrupt. The USB_EP0_CSR[n]_P.RXPKTRDY bit is also set. The host must then read and decode the 8-byte command from the endpoint 0 FIFO, and take appropriate action. For example, if the command is SET_FEATURE, the host writes 7-bit address value contained in the command to the USB_FADDR register.

Software must set the USB_EP0_CSR[n]_P.SPKTRDY bit (indicating that the command has been read from the FIFO) and the USB_EP0_CSR[n]_P.DATAEND bit (indicating that no further data is expected for this request).

When the host moves to the status stage of the request, the USB controller generates a second endpoint 0 interrupt, indicating that the request has completed. No further action is required from the software; the second interrupt is just a confirmation that the request completed successfully.

If the command is an unrecognized command, or for some other reason cannot be executed, then when the host decodes it, the USB_EP0_CSR[n]_P.SPKTRDY bit is set which sets the USB_EP0_CSR[n]_P.SENDSTALL bit. When the host moves to the status stage of the request, the USB controller sends a stall to tell the host that the

request was not executed. The USB controller generates a second endpoint 0 interrupt and sets the `USB_EP0_CSR[n]_P.SENTSTALL` bit.

If the host sends more data after the `USB_EP0_CSR[n]_P.DATAEND` bit is set, then the USB controller sends a stall. It generates an endpoint 0 interrupt and sets the `USB_EP0_CSR[n]_P.SENTSTALL` bit.

Endpoint 0 States

When the USB operates as a peripheral, the endpoint 0 control has three modes (IDLE, Tx, and Rx). The modes correspond to the phases of the control transfer and the state that endpoint 0 enters during phases of the transfer. (See Endpoint 0 Service Routine as Peripheral.)

IDLE is the default mode on power-up or reset. The processor sets `RxPktRdy` bit when endpoint 0 is in IDLE state, indicating a new device request. Once the processor unloads the device request from the FIFO, the USB decodes the descriptor. It determines whether there is a data phase and, if so, the direction of the data phase of the control transfer (to set the FIFO direction).

Depending on the direction of the data phase, endpoint 0 goes into either Tx state or Rx state. If there is no data phase, endpoint 0 remains in IDLE state to accept the next device request.

The processor must take different actions at the different phases of the possible transfers (for example, loading the FIFO, setting `TxPktRdy`). The ***Endpoint 0 Control States*** figure shows the actions for the phase. The USB changes the FIFO direction depending on the direction of the data phase, independently of the processor.

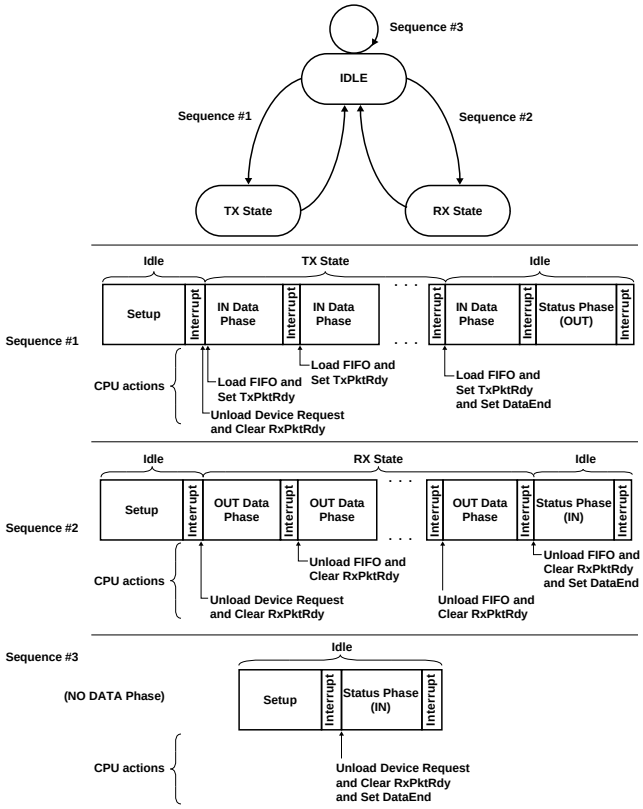

Figure 28-4: Endpoint 0 Control States

Endpoint 0 Service Routine as Peripheral

The USB controller generates an endpoint 0 interrupt when:

- The USB controller sets the USB_EP0_CSR[n]_P.RXPKTRDY bit after a valid token has been received and data has been written to the FIFO.
- The USB controller clears the USB_EP0_CSR[n]_P.TXPKTRDY bit after the data packet in the FIFO has been successfully transmitted to the host.
- The USB controller sets the USB_EP0_CSR[n]_P.SENTSTALL bit after a control transaction is ended due to a protocol violation.
- The USB controller sets the USB_EP0_CSR[n]_P.SETUPEND bit because a control transfer has ended before USB_EP0_CSR[n]_P.DATAEND is set.

Whenever the endpoint 0 service routine is entered, the firmware must first check whether the current control transfer has been ended. The transfer can end due to either a stall condition or a premature end-of-control transfer. If the control transfer ends due to a stall condition, the USB controller sets the USB_EP0_CSR[n]_P.SENTSTALL bit. If the control transfer ends due to a premature end-of-control transfer, the USB controller sets USB_EP0_CSR[n]_P.SETUPEND. In either case, the firmware must abort processing the current control transfer and set the state to IDLE.

Once the firmware has determined that an illegal bus state did not generate the interrupt, the next action depends on the endpoint state.

If endpoint 0 is in IDLE state, the only valid reason the USB controller can generate an interrupt is due to the core receiving data from the USB bus. The service routine must check for this state by testing the USB_EP0_CSR[n]_P.RXPKTRDY bit. If the USB controller sets the bit, then the core has received a SETUP packet. The processor unloads this packet from the FIFO and decodes it to determine the next action. Depending on the command contained within the SETUP packet, endpoint 0 enters one of the following three states.

- If the command is a single packet transaction (SET_ADDRESS, SET_INTERFACE and the others) without a data phase, the endpoint remains in the IDLE state.
- If the command has an OUT data phase (SET_DESCRIPTOR and others), the endpoint enters the Rx state.
- If the command has an IN data phase (GET_DESCRIPTOR and others), the endpoint enters the Tx state.

NOTE: Command transactions all include a field that indicates the amount of data the host expects to receive or send.

If the endpoint is in Tx state, the interrupt indicates that the core has received an IN token and data from the FIFO has been sent. The firmware must respond to this event by:

- Placing more data in the FIFO when the host still expects more data
- Setting the USB_EP0_CSR[n]_P.DATAEND bit to indicate that the data phase is complete

Once the data phase of the transaction completes, endpoint 0 returns to the IDLE state to await the next control transaction.

If the endpoint is in the Rx state, the interrupt indicates that a data packet has been received. The firmware must respond by unloading the received data from the FIFO. The firmware must then determine whether it has received all of the expected data. If it has, the firmware must set the `USB_EP0_CSR[n]_P.DATAEND` bit and return endpoint 0 to IDLE state. If more data is expected, the firmware must set the `USB_EP0_CSR[n]_P.SPKTRDY` bit to indicate that it has read the data in the FIFO and leave the endpoint in the Rx state.

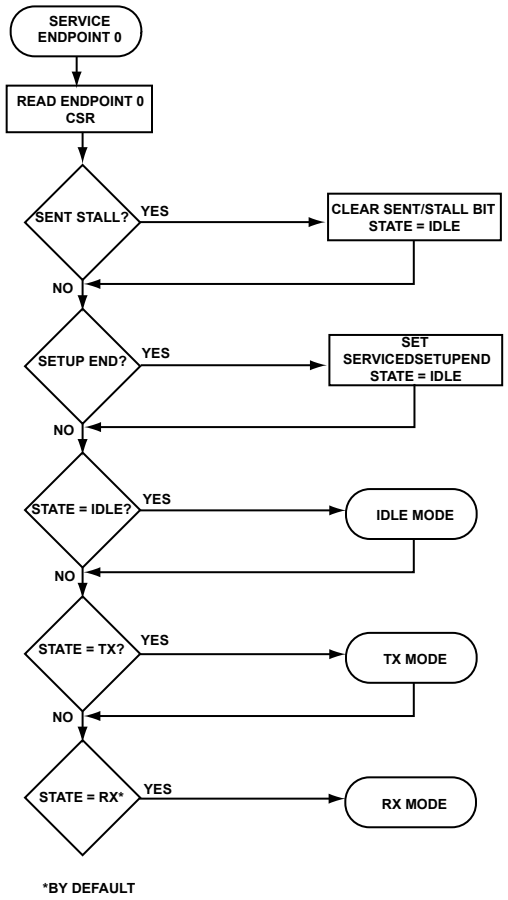

Figure 28-5: Endpoint 0 Service Routine

Idle Mode

The endpoint 0 control must select IDLE mode at power-on or reset. The endpoint 0 control returns to this mode when the Rx and Tx modes terminate.

As shown in the ***Endpoint 0 Idle Mode (Setup Phase)*** figure, the SETUP phase of control transfer is handled in IDLE mode.

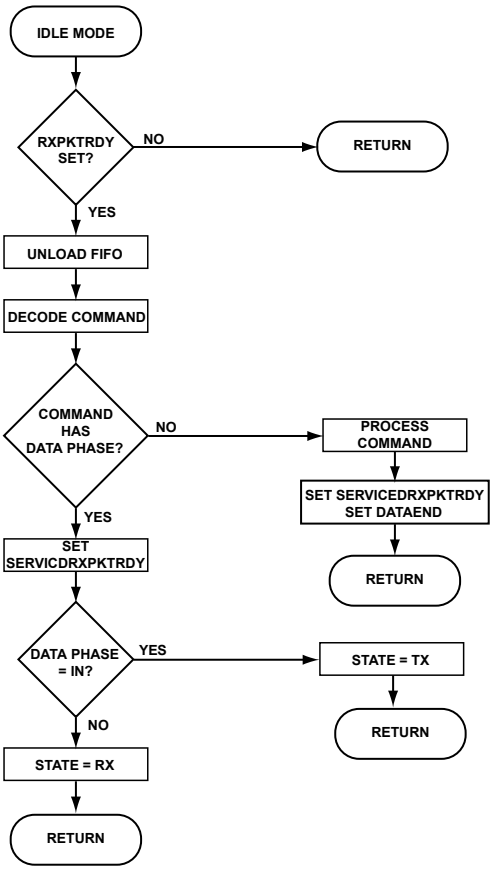

Figure 28-6: Endpoint 0 Idle Mode (Setup Phase)

Tx Mode

Refer to the ***Endpoint 0 Tx Mode*** figure. When the endpoint is in Tx state, all arriving IN tokens must be treated as part of a data phase until the required amount of data has been sent to the host. If either a SETUP or an OUT token is received while the endpoint is in the Tx state, a `USB_EP0_CSR[n]_P.SETUPEND` condition occurs. The core expects only IN tokens.

Three events can cause the Tx mode to terminate before the expected amount of data has been sent:

- The host sends an invalid token which sets the `USB_EP0_CSR[n]_P.SETUPEND` bit.
- The firmware sends a packet containing less than the maximum packet size for endpoint 0.
- The firmware sends an empty data packet.

Until the transaction is terminated, when the firmware receives an interrupt which indicates that a packet has been sent from the FIFO, it simply loads the FIFO. An interrupt is generated when the USB controller clears `USB_EP0_CSR[n]_P.TXPKTRDY`.

When the firmware forces the termination of a transfer (by sending a short or empty data packet), it must set the `USB_EP0_CSR[n]_P.DATAEND` bit. This event indicates to the core that the data phase is complete and that the core will receive an acknowledge packet next.

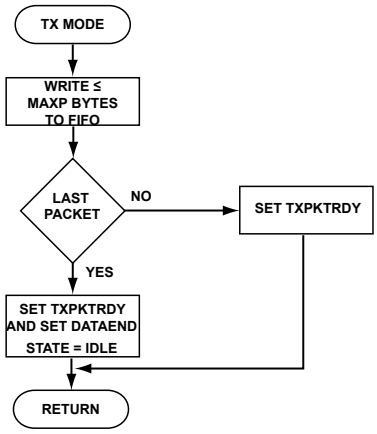

Figure 28-7: Endpoint 0 Tx Mode

Rx Mode

Refer to the ***Endpoint 0 Rx Mode*** figure. In Rx mode, all arriving data must be treated as part of a data phase until the expected amount of data is received. If either a SETUP or an IN token is received while the endpoint is in Rx state, a `USB_EP0_CSR[n]_P.SETUPEND` condition occurs since the core expects only OUT tokens.

Three events can cause the Rx mode to terminate before the expected amount of data is received:

- The host sends an invalid token which sets the `USB_EP0_CSR[n]_P.SETUPEND` bit.
- The host sends a packet which contains less than the maximum packet size for endpoint 0.
- The host sends an empty data packet.

The transaction terminates when the firmware receives an interrupt which indicates that new data has arrived (`USB_EP0_CSR[n]_P.RXPKTRDY` bit is set). Until the transaction terminates, firmware must unload the FIFO and clear `USB_EP0_CSR[n]_P.RXPKTRDY` by setting the `USB_EP0_CSR[n]_P.SPKTRDY` bit.

When the firmware detects the termination of a transfer (by receiving either the expected amount of data or an empty data packet), it must set the `USB_EP0_CSR[n]_P.DATAEND` bit. This event indicates to the core that the data phase is complete and that the core will receive an acknowledge packet next.

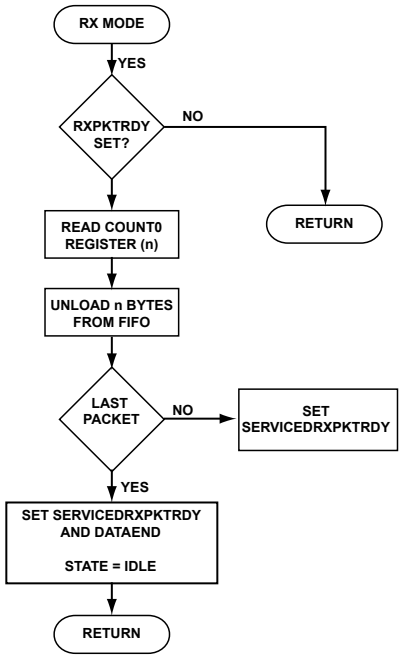

Figure 28-8: Endpoint 0 Rx Mode

Peripheral Mode, Bulk IN, Transfer Size Known

For this process, the maximum size of an individual packet (*MaxPktSize*) in bytes and the complete transfer size (*TxferSize*) in bytes, must be known.

1. Load *MaxPktSize* into the `USB_EP[n]_TXMAXP` register.
2. Set the following bits: `USB_EP[n]_TXCSR_P.DMAREQEN` = 1, `USB_EP[n]_TXCSR_P.AUTOSET` = 1, `USB_EP[n]_TXCSR_P.ISO` = 0, `USB_EP[n]_TXCSR_P.FRCDATATGL`= 0.
3. Load the *TxferSize* value into the `USB_DMA[n]_CNT` register.
4. Configure the DMA controller to write the data into the corresponding Tx FIFO address.
5. On each `USB_DMA[n]_CNT` transition, the DMA controller writes a new packet into the FIFO. The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is automatically set when each new packet is written.

   ADDITIONAL INFORMATION: Repeat Step 5 for each full packet of the transfer. Even if the final packet is a short packet, the USB controller automatically detects the packet because the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set.

Peripheral Mode, Bulk IN, Transfer Size Unknown

For this process, assume the maximum individual packet size (*MaxPktSize*) in bytes is an even number of bytes.

1. Load *MaxPktSize* into the `USB_EP[n]_TXMAXP` register.

2. Set the following bits: USB_EP[n]_TXCSR_P.DMAREQEN = 1, USB_EP[n]_TXCSR_P.AUTOSET = 1, USB_EP[n]_TXCSR_P.ISO = 0, USB_EP[n]_TXCSR_P.FRCDATATGL= 0.
3. Configure the DMA controller to write *MaxPktSize*/2 half words into the corresponding Tx FIFO address on each USB_DMA[n]_CNT.
4. Set up an ISR, sensitive to the DMA work-block-complete interrupt, that writes a remaining short packet into the Tx FIFO using processor core DMA. Then, set the USB_EP[n]_TXCSR_P.TXPKTRDY bit or toggle the USB_EP[n]_TXCSR_P.TXPKTRDY bit to send a zero-length packet.
5. On each USB_DMA[n]_CNT transition, the DMA controller writes a new packet into the FIFO. The USB_EP[n]_TXCSR_P.TXPKTRDY bit is set automatically when each new packet is written.

   ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer. The ISR from step 4 manages the final short or zero-length packet.

Peripheral Mode, ISO IN, Small MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is less than 128 bytes and is an even number of bytes. Assume that double buffering is enabled, and the auto-set feature is not used (because packets are often less than *MaxPktSize*).

1. Load *MaxPktSize* into the USB_EP[n]_TXMAXP register.
2. Set the following bits: USB_EP[n]_TXCSR_P.ISO = 1.
3. Preload the first two packets into the endpoint Tx FIFO and set the USB_EP[n]_TXCSR_P.TXPKTRDY bit.
4. Set up an ISR, sensitive to the USB_IRQ.SOF interrupt, which writes a new packet into the Tx FIFO and sets the USB_EP[n]_TXCSR_P.TXPKTRDY bit.
5. Set the USB_IEN.SOF bit = 1 to generate an interrupt on each start of frame.

   ADDITIONAL INFORMATION: Repeat step 5 for each ISO packet.

Peripheral Mode, ISO IN, Large MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is greater than 128 bytes and is an even number of bytes. Assume that double buffering is enabled, and the auto-set feature is not used (because packets are often less than *MaxPktSize*).

1. Load *MaxPktSize* into the USB_EP[n]_TXMAXP register.
2. Set the USB_EP[n]_TXCSR_P.ISO bit = 1.
3. Set the USB_POWER.ISOUPDT bit = 1 to prevent the initial packet loaded into the FIFO from transmitting on the USB until the next 1ms frame.
4. Load the total number of bytes for the first two packets into the USB_DMA[n]_CNT register.

5. Configure the DMA controller to pre-load the two packets into the corresponding Tx FIFO address and set the USB_EP[n]_TXCSR_P.TXPKTRDY bit.
6. Set up an ISR, sensitive to the USB_IRQ.SOF interrupt, which writes a new packet into the Tx FIFO by configuring the DMA controller to load the packet.
7. Set the USB_IEN.SOF bit = 1 to generate an interrupt on each start of frame.

   ADDITIONAL INFORMATION: Repeat step 7 for each ISO packet.

Peripheral Mode, Bulk OUT, Transfer Size Known

For this process, the maximum individual packet size (*MaxPktSize*) in bytes and the complete transfer size (*TxferSize*) in bytes must be known.

1. Load *MaxPktSize* into USB_EP[n]_RXMAXP.
2. Set the following bits: USB_EP[n]_RXCSR_P.DMAREQEN = 1, USB_EP[n]_RXCSR_P.AUTOCLR= 1, USB_EP[n]_RXCSR_P.ISO = 0, USB_EP[n]_RXCSR_P.CLRDATATGL = 0, USB_EP[n]_RXCSR_P.DMAREQMODE = 0.
3. Configure the DMA controller to read the full *TxferSize*/2 half words from the corresponding Rx FIFO address.
4. On each USB_DMA[n]_CNT transition, the DMA controller reads another packet from the FIFO. The USB controller automatically clears the USB_EP[n]_RXCSR_P.RXPKTRDY bit when each new packet is read.

   ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer. If *TxferSize* is not an exact multiple of *MaxPktSize*, the final USB_DMA[n]_CNT transition causes the DMA controller to read out only the short packet that remains.

Peripheral Mode, Bulk OUT, Transfer Size Unknown

For this process, the maximum individual packet size (*MaxPktSize*) in bytes must be known.

1. Load *MaxPktSize* into USB_EP[n]_RXMAXP.
2. Set the following bits: USB_EP[n]_RXCSR_P.DMAREQEN= 1, USB_EP[n]_RXCSR_P.AUTOCLR = 1, USB_EP[n]_RXCSR_P.ISO = 0, USB_EP[n]_RXCSR_P.CLRDATATGL= 0, USB_EP[n]_RXCSR_P.DMAREQMODE = 1.
3. Set the appropriate bit in the USB_INTRRXE register.
4. Configure the DMA controller to read *MaxPktSize*/2 half words from the corresponding Rx FIFO address on each USB_DMA[n]_CNT transition.
5. Set up an ISR, sensitive to the Rx interrupt, which reads the USB_EP[n]_RXCNT register and then transfers USB_EP[n]_RXCNT bytes (in half words) from the Rx FIFO to the processor core.

ADDITIONAL INFORMATION: Depending on the number of bytes in the FIFO, configure the DMA to read the data, or read it with the processor core.

ADDITIONAL INFORMATION: On each USB_DMA[n]_CNT transition, the DMA controller reads a packet from the FIFO. The USB controller automatically clears the USB_EP[n]_RXCSR_P.RXPKTRDY bit when each new packet is read.

ADDITIONAL INFORMATION: Repeat step 5 for each full packet of the transfer.

6. If a packet is received that is less than *MaxPktSize*, the Rx interrupt goes high, and the ISR from step 5 reads out the remaining short packet.

Peripheral Mode, ISO OUT, Small MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is less than 128 bytes. Assume that double buffering is enabled.

1. Load the *MaxPktSize* value into the USB_EP[n]_RXMAXP register.
2. Set the USB_EP[n]_RXCSR_P.ISO bit = 1.
3. Set up an ISR, sensitive to the USB_IRQ.SOF interrupt, that reads the USB_EP[n]_RXCSR_P.FIFOFULL bit and then reads the USB_EP0_CNT[n].RXCNT status register. The ISR removes one or two packets (equal to the USB_EP0_CNT[n].RXCNT number of bytes) from the FIFO. It then clears the USB_EP[n]_RXCSR_P.RXPKTRDY bit.
4. Set the USB_IEN.SOF bit = 1 to generate an interrupt on each start of frame.

   ADDITIONAL INFORMATION: Repeat step 4 for each ISO packet.

Peripheral Mode, ISO OUT, Large MaxPktSize

For this process, the maximum individual packet size (*MaxPktSize*) in bytes is greater than 128 bytes. Assume that double buffering is enabled.

1. Load *MaxPktSize* into the USB_EP[n]_RXMAXP register.
2. Set the USB_EP[n]_RXCSR_P.ISO bit = 1.
3. Set up an ISR (sensitive to the USB_IRQ.SOF interrupt), that reads the USB_EP[n]_RXCSR_P.FIFOFULL bit, and then reads the USB_EP[n]_RXCNT status register. The ISR configures the DMA controller to remove one or two packets (equal to the USB_EP[n]_RXCNT number of bytes) from the FIFO.
4. Set up an ISR, sensitive to the DMA work-block-complete interrupt to clear the USB_EP[n]_RXCSR_P.RXPKTRDY.
5. Set the USB_IEN.SOF bit = 1 to generate an interrupt on each start of frame.

   ADDITIONAL INFORMATION: Repeat step 5 for each ISO packet.

Peripheral Mode Suspend

When no activity has occurred on the USB for 3 ms, the USB controller enters suspend mode. If the suspend interrupt (USB_IRQ.SUSPEND) is enabled, an interrupt is generated now.

When resume signaling is detected, the USB controller exits suspend mode. If the USB_IRQ.RESUME interrupt is enabled, an interrupt is generated. The processor core can also force the USB controller to exit suspend mode by setting the USB_POWER.RESUME bit. This event initiates a remote wake-up. When this bit is set, the USB controller exits suspend mode and drives resume signaling onto the bus. The processor core must clear this bit after 10 ms (a maximum of 15 ms) to end resume-signaling.

NOTE: The USB_IRQ.RESUME interrupt is not generated when the processor core exits suspend mode. This interrupt is not generated when the software initiates remote wake-up.

Start of Frame (SOF) Packets

When the USB controller operates in peripheral mode, it receives a start of frame packet from the host every millisecond when in full-speed mode, or every 125 microseconds when in high-speed mode.

When the USB controller receives a SOF packet, it writes the 11-bit frame number contained in the packet into the USB_FRAME register. An output pulse, lasting one USB clock bit period, is generated. A start of frame interrupt is also generated (if enabled by the USB_IRQ.SOF bit).

After the USB controller has started to receive SOF packets, the controller expects one every millisecond (faster in high-speed mode). If the USB controller does not receive an SOF packet after 1.00358 ms (faster in high-speed mode), it is assumed that the packet is lost. A start of frame pulse (together with a USB_IRQ.SOF interrupt) is still generated even though the USB_FRAME register is not updated. The USB controller continues to generate an SOF pulse every millisecond (faster in high-speed mode). It resynchronizes these pulses to the received SOF packets when it successfully receives these packets again.

Soft Connect/Soft Disconnect

In peripheral mode, the USB controller sets or clears the USB_POWER.SOFTCONN bit to switch between normal mode and non-driving mode. When USB_POWER.SOFTCONN=1, the USB controller is in normal mode and the D+/D– lines of the USB bus are enabled. When the USB_POWER.SOFTCONN=0, the PHY is put into non-driving mode and D+ and D– are three-stated. The USB controller appears to have been disconnected from the USB bus.

After system reset, USB_POWER.SOFTCONN=0. From that point, the USB controller appears disconnected until the software has set USB_POWER.SOFTCONN =1. The application software can then choose when to set the PHY to its normal mode. Systems with a lengthy initialization procedure can use this functionality to ensure that initialization is complete and the system is ready to perform enumeration before connecting to the USB. Once the USB_POWER.SOFTCONN bit is set to 1, the software can also clear this bit to 0 to simulate a disconnect.

Error Handling As a Peripheral

The host can abort a control transfer due to a protocol error on the USB. The function controller software can also abort the transfer (for example, because it cannot process the command).

The USB controller automatically detects protocol errors and sends a stall packet to the host under the following conditions.

1. The host sends more data during the OUT data phase of a write request than specified in the command. This condition is detected when the host sends an OUT token after the USB_EP0_CSR[n]_P.DATAEND bit is set.
2. The host requests more data during the IN data phase of a read request than specified in the command. This condition is detected when the host sends an IN token after the USB_EP0_CSR[n]_P.DATAEND bit is set.
3. The host sends more than *MaxPktSize* data bytes in an OUT data packet.
4. The host sends a non-zero length DATA1 packet during the status phase of a read request.

When the USB controller has sent the stall packet, it sets the USB_EP0_CSR[n]_P.SENTSTALL bit and generates an interrupt. When the software receives an endpoint 0 interrupt with the USB_EP0_CSR[n]_P.SENTSTALL bit set, it aborts the current transfer, clears the USB_EP0_CSR[n]_P.SENTSTALL bit, and returns to the IDLE state.

If the host enters the status phase before all the data for the request transfers, or sends a new SETUP packet before completing the current transfer, then it prematurely ends the transfer. The USB_EP0_CSR[n]_P.SETUPEND bit is set and an endpoint 0 interrupt is generated. When the software receives an endpoint 0 interrupt with the USB_EP0_CSR[n]_P.SETUPEND bit set, it aborts the current transfer, sets the USB_EP0_CSR[n]_P.SSETUPEND bit, and returns to the IDLE state. If the USB_EP0_CSR[n]_P.RXPKTRDY bit is set, it indicates that the host has sent another SETUP packet and the software must then process this command.

If the software wants to abort the current transfer, because it cannot process the command or has some other internal error, then it must set the USB_EP0_CSR[n]_P.SENTSTALL bit. The USB controller then sends a stall packet to the host, sets the USB_EP0_CSR[n]_P.SENTSTALL bit, and generates an endpoint 0 interrupt.

Stalls Issued to Control Transfers

In peripheral mode, the USB controller automatically issues a stall handshake to a control transfer under the following conditions:

1. The host sends more data during an OUT data phase of a control transfer than specified in the device request during the SETUP phase. The USB controller detects this condition when the host sends an OUT token (instead of an IN token) after the processor core unloads the last OUT packet and sets the USB_EP0_CSR[n]_P.DATAEND bit.
2. The host requests more data during an IN data phase of a control transfer than specified in the device request during the SETUP phase. The USB controller detects this condition when the host sends an IN token (instead of an OUT token) after the processor core clears USB_EP[n]_TXCSR_P.TXPKTRDY and sets USB_EP0_CSR[n]_P.DATAEND. The processor sets USB_EP0_CSR[n]_P.DATAEND in response to the host-issued ACK for the last packet.
3. The host sends more than *MaxPktSize* data with an OUT data token.
4. The host sends the wrong PID (packet identifier) for the OUT status phase of a control transfer.
5. The host sends more than a zero length data packet for the OUT status phase.

Zero Length OUT Data Packets in Control Transfers

The USB controller uses a zero-length OUT data packet to indicate the end of a control transfer. In normal operation, such packets must only be received after the entire length of the device request transfers (for example, after the processor core has set the USB_EP0_CSR[n]_P.DATAEND bit). If the host sends a zero-length OUT data packet before the entire length of device request transfers, this packet signals the premature end of the transfer. In this case, the USB controller automatically flushes any IN token the processor core has loaded for the data phase from the FIFO and sets the USB_EP0_CSR[n]_P.SETUPEND bit.

Host Mode

USB OTG interface operations in host mode differ from peripheral mode in a number of ways. The following sections describe host mode operations.

Transaction Scheduling

When operating as a host, the USB controller maintains a frame counter.

If the target function is a full-speed device, the USB controller automatically sends an SOF packet at the start of each frame or micro-frame.

If the target function is a low-speed device, a K state is transmitted on the bus to act as a *keep-alive*. It stops the low-speed device from going into suspend mode.

After the SOF packet is transmitted, the USB controller cycles through all the endpoints looking for active transactions. An active transaction is defined as an Rx endpoint for which the USB_EP[n]_RXCSR_H.REQPKT bit is set or a Tx endpoint for which the USB_EP[n]_TXCSR_H.TXPKTRDY bit is set.

An active isochronous or interrupt transaction only starts if:

- It is found on the first transaction scheduler cycle of a frame.
- The interval counter for that endpoint has counted down to zero.

This functionality ensures that only one interrupt or isochronous transaction occurs per endpoint per *n* frames or micro frames (or, up to three, if high-bandwidth support is selected). *n* is the interval set in the USB_EP[n]_TXINTERVAL or USB_EP[n]_RXINTERVAL register for that endpoint.

An active bulk transaction starts immediately, provided there is sufficient time left in the frame to complete the transaction before the next SOF packet is due. If the transaction must be retried, then it is not retried until the transaction scheduler has checked all the other endpoints for active transactions first. (For example, the transaction is retried because a NAK was received or the target function did not respond). This check ensures that an endpoint that is sending many NAKs does not block other transactions on the bus. The USB controller permits specifying a limit (USB_EP[n]_TXINTERVAL or USB_EP[n]_RXINTERVAL registers) to the length of time in which NAKs can be received from a particular target before the endpoint is timed out.

Endpoint Setup and Data Transfer

When the HOST_MODE bit is set to 1, the USB controller operates as a host for point-to-point communications with another USB device. Or, when attached to a hub, the USB controller operates as a host for communication with a range of devices in a multi-point set-up.

The USB controller supports high-speed, full-speed, and low-speed USB functions, both for point-to-point communication and for operation through a hub.

Where necessary, the core automatically carries out the necessary transaction translation to allow usage of a low-speed or full-speed device with a USB 2.0 hub.

The USB controller supports control, bulk, isochronous, or interrupt transactions.

Transfers between the subsystem and endpoint FIFOs in host mode are similar to peripheral mode. See the descriptions of processor core-to-FIFO data transfer in Peripheral Mode.

Control Transaction as a Host

Host control transactions are conducted through endpoint 0. The software handles all the standard device requests that are sent or received through endpoint 0 (as described in *Universal Serial Bus Specification*, Revision 2.0, Chapter 9).

For a USB peripheral, there are three categories of standard device requests:

- ***Zero data requests.*** Comprised of a SETUP command followed by an IN status phase. The command includes all the information.
- ***Write requests.*** Comprised of a SETUP command, followed by an OUT data phase followed by an IN status phase. Extra data follows the command.
- ***Read requests*** Comprised of a SETUP command, followed by an IN data phase followed by an OUT status phase. The device is required to send data back to the host.

A timeout can be set to limit the length of time during which the USB controller retries a transaction that the target continually NAKs. This limit can be between 2 and 2^{15} frames or micro frames and is set through the USB_EP0_NAKLIMIT[n] register.

The following sections describe the steps taken in different phases of a control transaction and the actions of the core when issuing standard device requests.

Set up Phase as a Host

The processor core driving the host device performs the following actions for the SETUP phase of a control transaction.

1. Load the 8 bytes of the required device request command into the endpoint 0 FIFO.
2. Set the USB_EP0_CSR[n]_H.SETUPPKT bit and USB_EP0_CSR[n]_H.TXPKTRDY bit. These bits must be set together.

The USB controller then sends a SETUP token followed by the 8-byte command to endpoint 0 of the addressed device, retrying as necessary.

3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt (for example, set USB_INTRTXE.EP0). The processor core then reads the USB_EP0_CSR[n]_H register to establish whether the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or the USB_EP0_CSR[n]_H.NAKTO bits are set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target did not accept the command (for example, because the target device does not support it) and issues a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller tried to send the SETUP packet and the following data packet three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller received a NAK response to each attempt to send the SETUP packet, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the USB_EP0_CSR[n]_H.NAKTO bit.

4. If none of USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR or USB_EP0_CSR[n]_H.NAKTO bits are set, the SETUP phase is correctly acknowledged. The processor core can proceed to the following IN data phase, OUT data phase or IN status phase specified for the particular standard device request.

IN Data Phase as a Host

The processor core driving the host device performs the following actions for the IN data phase of a control transaction.

1. Set the USB_EP0_CSR[n]_H.REQPKT bit.
2. Wait while the USB controller sends the IN token and then receives the required data back.
3. When the USB controller generates the endpoint 0 interrupt (for example, by setting the USB_INTRTXE.EP0 bit), read the USB_EP0_CSR[n]_H register. Determine whether the USB_EP0_CSR[n]_H.RXSTALL bit, the USB_EP0_CSR[n]_H.TOERR bit, the USB_EP0_CSR[n]_H.NAKTO bit, or the USB_EP0_CSR[n]_H.RXPKTRDY bit is set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller has tried to send the required IN token three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or clear USB_EP0_CSR[n]_H.REQPKT before clearing the USB_EP0_CSR[n]_H.NAKTO bit to abort the transaction.

4. If the USB_EP0_CSR[n]_H.RXPKTRDY bit is set, the processor core reads the data from the endpoint 0 FIFO, then clears USB_EP0_CSR[n]_H.RXPKTRDY.
5. If further data is expected, the processor core must repeat the previous steps.

   When all the data is successfully received, the processor core can proceed to the OUT status phase of the control transaction.

OUT Data as a Host (Control)

The processor core driving the host device performs the following actions for the OUT data phase of a control transaction.

1. Load the data to be sent into the endpoint 0 FIFO.
2. Set the USB_EP0_CSR[n]_H.TXPKTRDY bit.

   The USB controller sends an OUT token followed by the data from the FIFO to endpoint 0 of the addressed device, retrying as necessary.
3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt (for example by setting the USB_INTRTX.EP0 bit). The processor core can then read the USB_EP0_CSR[n]_H to establish whether the USB_EP0_CSR[n]_H.RXSTALL bit, the USB_EP0_CSR[n]_H.TOERR bit, or the USB_EP0_CSR[n]_H.NAKTO bit is set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1 the USB controller has tried to send the OUT token and the following data packet three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the OUT token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the USB_EP0_CSR[n]_H.NAKTO bit.

   If none of the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or USB_EP0_CSR[n]_H.NAKTO bits are set, the OUT data is correctly acknowledged.
4. If further data must be sent, the processor core must repeat the previous steps.

   When all the data is successfully sent, the processor core proceeds to the IN status phase of the control transaction.

IN Status Phase as a Host (Following SETUP Phase or OUT Data Phase)

The processor core driving the host device performs the following actions for the IN status phase of a control transaction.

1. Set the USB_EP0_CSR[n]_H.STATUSPKT and USB_EP0_CSR[n]_H.REQPKT bits. These bits must be set together.

2. Wait while the USB controller both sends an IN token and receives a response from the USB peripheral.
3. When the USB controller generates the endpoint 0 interrupt (for example, it sets the USB_INTRTX.EP0 bit), read the USB_EP0_CSR[n]_H register to establish whether the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, USB_EP0_CSR[n]_H.NAKTO, or the USB_EP0_CSR[n]_H.RXPKTRDY bits are set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target could not complete the command and so has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller has tried to send the required IN token three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or clear USB_EP0_CSR[n]_H.REQPKT before clearing the USB_EP0_CSR[n]_H.NAKTO bit to abort the transaction.
4. If the USB_EP0_CSR[n]_H.RXPKTRDY bit is set, the processor core typically clears it.

OUT Status Phase as a Host (Following IN Data Phase)

The processor core driving the host device performs the following actions for the OUT status phase of a control transaction.

1. Set USB_EP0_CSR[n]_H.STATUSPKT and USB_EP0_CSR[n]_H.TXPKTRDY bits. These bits must be set together.
2. Wait while the USB controller both sends the OUT token and a zero-length DATA1 packet.
3. At the end of the attempt to send the data, the USB controller generates an endpoint 0 interrupt. The processor core then reads the USB_EP0_CSR[n]_H register to discover when the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or USB_EP0_CSR[n]_H.NAKTO bits are set.

   If USB_EP0_CSR[n]_H.RXSTALL=1, the target could not complete the command and so has issued a stall response.

   If USB_EP0_CSR[n]_H.TOERR=1, the USB controller has tried to send the STATUS packet and the following data packet three times without getting a response.

   If USB_EP0_CSR[n]_H.NAKTO=1, the USB controller has received a NAK response to each attempt to send the IN token, for longer than the time set in the USB_EP0_NAKLIMIT[n] register. Direct the USB controller to either clear the USB_EP0_CSR[n]_H.NAKTO bit to continue trying this transaction (until it times out again) or to flush the FIFO to abort the transaction before clearing the USB_EP0_CSR[n]_H.NAKTO bit.
4. If none of the USB_EP0_CSR[n]_H.RXSTALL, USB_EP0_CSR[n]_H.TOERR, or USB_EP0_CSR[n]_H.NAKTO bits are set, the status phase is correctly acknowledged.

Host IN Transactions

When the USB controller operates as a host, IN transactions are handled like OUT transactions are handled when the USB controller is operating as a peripheral. First, the USB controller sets USB_EP[n]_RXCSR_H.REQPKT bit to initiate the transaction. This bit indicates to the transaction scheduler that there is an active transaction on this endpoint. The transaction scheduler then sends an IN token to the target function.

When the packet is received and placed in the Rx FIFO, the USB_EP[n]_RXCSR_H.RXPKTRDY bit is set. The appropriate Rx endpoint interrupt is generated (if enabled) to signal that the processor can now unload a packet can now from the FIFO. When the processor unloads the packet, USB_EP[n]_RXCSR_H.RXPKTRDY is cleared. The USB controller uses the USB_EP[n]_RXCSR_H.AUTOCLR bit to clear the USB_EP[n]_RXCSR_H.RXPKTRDY bit automatically when the processor unloads a maximum sized packet from the FIFO. There is also an USB_EP[n]_RXCSR_H.AUTOREQ bit that automatically sets the USB_EP[n]_RXCSR_H.REQPKT bit when the USB_EP[n]_RXCSR_H.RXPKTRDY bit is cleared. The USB controller can use the USB_EP[n]_RXCSR_H.AUTOCLR and USB_EP[n]_RXCSR_H.AUTOREQ bits with an external DMA controller to perform complete bulk transfers without processor core intervention.

If the target function responds to a bulk or interrupt IN token with a NAK, the USB controller retries the transaction until the NAK limit set in the USB_EP0_NAKLIMIT[n] register is reached. If the target function responds with a stall, the USB controller does not retry the transaction, but sets the USB_EP[n]_RXCSR_H.RXSTALL bit to interrupt the processor core. If the target function does not respond to the IN token within the required time, the USB controller retries the transaction.(USB controller also retries the transaction if there was a CRC or bit-stuff error in the packet). If, after three attempts, the target function still has not responded, the USB controller clears the USB_EP[n]_RXCSR_H.REQPKT bit and interrupts the processor core with the DATAERROR_R bit in USB_RXCSR set.

Host OUT Transactions

When the USB controller operates as a host, OUT transactions are handled like IN transactions are handled when the USB controller operates as a peripheral.

The USB_EP[n]_TXCSR_H.TXPKTRDY bit must be set as the processor loads each packet into the Tx FIFO. The USB controller uses the USB_EP[n]_TXCSR_H.AUTOSET bit to cause the USB_EP[n]_TXCSR_H.TXPKTRDY bit to be automatically set when the processor loads a maximum sized packet into the FIFO. The USB controller can use the USB_EP[n]_TXCSR_H.AUTOSET bit with an external DMA controller to perform complete bulk transfers without processor core intervention.

If the target function responds to the OUT token with a NAK, the USB controller retries the transaction until the NAK limit set in the USB_EP0_NAKLIMIT[n] register is reached. If the target function responds with a stall, the USB controller does not retry the transaction, but sets the USB_EP[n]_TXCSR_H.RXSTALL bit to interrupt the processor core. If the target function does not respond to the OUT token within the required time, the USB controller retries the transaction.(USB controller also retries the transaction if there was a CRC or bit-stuff error in the packet). If, after three attempts, the target function still has not responded, the USB controller flushes the FIFO and sets the USB_EP[n]_TXCSR_H.TXTOERR bit to interrupt the processor core.

Multi-Point Support

The following sections describe the multi-point support of the USB controller.

- Allocating Devices to Endpoints
- Multi-Point Operation
- Multi-Point Bandwidth Considerations

Allocating Devices to Endpoints

The separate functions of the connected devices are allocated to the endpoints within the USB controller through a group of three registers. The registers are associated with each implemented Rx or Tx endpoint (including endpoint 0).

The registers are:

- USB_MP[n]_TXFUNCADDR/ USB_MP[n]_RXFUNCADDR
- USB_MP[n]_TXHUBADDR/ USB_MP[n]_RXHUBADDR
- USB_MP[n]_TXHUBPORT/ USB_MP[n]_RXHUBPORT

The location of these registers depends on which of the endpoints is being addressed.

Record the address of the target function that is accessed through the selected endpoint in the transmit and receive function address registers. Record this information separately for each Tx and Rx endpoint used. In particular, set both USB_MP[n]_TXFUNCADDR and USB_MP[n]_RXFUNCADDR for endpoint 0.

The USB controller uses the transmit and receive hub address and hub port registers when a full-speed or low-speed device is connected to it through a high-speed USB 2.0 hub. The hub carries out the required transaction translation between high-speed transmission and low-speed or full-speed transmission. In this situation, the USB_MP[n]_TXHUBADDR/ USB_MP[n]_RXHUBADDR and USB_MP[n]_TXHUBPORT/ USB_MP[n]_RXHUBPORT registers must record the address of the hub that carries out the transaction translation. It must also record the address of the port of that hub through which the associated Tx or Rx endpoint must access the device.

If endpoint 0 is connected to a hub, then set both the Tx and the Rx versions of these registers for this endpoint. The USB controller also uses hub address registers to record whether the hub offers multiple transaction translators or just a single transaction translator. This configuration has a significant effect on the overall bandwidth that can be achieved.

In addition to recording the address of the target function, record the endpoint number and operating speed of the target device and the type of transaction that is executed. For a Tx endpoint, set this information in the USB_EP[n]_TXTYPE register when the index register is set to select the required endpoint. For an Rx endpoint, set this information in the USB_EP[n]_RXTYPE register when the index register is set to select the required endpoint. In both cases, record the endpoint number in bits 3–0, select the transaction type through bits 5–4, and select the operating speed through bits 7–6.

Set only the speed for endpoint 0 because endpoint 0 only has the facilities to handle control transactions and therefore is always associated with a device endpoint 0. Use bits 7–6 of the Type 0 register to set the speed. The register is located at address 0x1A when the index register is set to 0.

Multi-Point Operation

After allocating functions to endpoints and recording the operating speed of the target device, multi-point operations can be configured. Most operations in a multi-point set-up are the same as for the equivalent actions where the core is attached to a single other device.

However, more steps are required when:

- The option of dynamically switching the allocation of functions to endpoints is taken (for example, to allow the support of a wider range of devices).
- The control packets normally associated with endpoint 0 are handled through a different endpoint.

If dynamic allocation is used, the program must monitor the current data toggle state associated with the endpoint and with each of the devices that are allocated to that endpoint. This knowledge allows the program to select the correct data toggle state when switching occurs between one device and the other. (This action is the programs responsibility. The core cannot determine what data toggle state is expected when a function switches in and out of use.)

The data toggle state can be switched from its current state by writing to the appropriate `USB_EP[n]_TXCSR_H` or `USB_EP[n]_RXCSR_H` register. This activity sets the data toggle write enable and data toggle bits that are included in the registers when the core is in host mode.

Data toggle write enable and data toggle bits are also included in the `USB_EP0_CSR[n]_H` register. However, control operations carried out through endpoint 0 of the core normally leave the data toggle in the expected state.

Where control packets are handled through an endpoint other than endpoint 0, programs must prompt for each setup token to be sent. Programs must set the `USB_EP[n]_TXCSR_H.SETUPPKT` bit when the core operates in host mode, along with the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit. If the `USB_EP[n]_TXCSR_H.SETUPPKT` bit is not set, an OUT token is sent.

Use endpoint 0 of the USB controller to handle control packets for all of the devices attached to the controller, and to switch the allocation of this endpoint, as appropriate. Sending the correct token is ensured, as is ensuring that the data toggle is correctly set for this endpoint.

Using a different endpoint for this function is possible, as described, but note the following:

- The control function must be allocated to an Rx/Tx endpoint pair (with the same endpoint number).
- The chosen endpoints must each be associated with FIFOs that can accommodate the packet size associated with EP0 transactions at the chosen operating speed. The size is a minimum of 8 bytes for low-speed or full-speed transactions but 64 bytes for high-speed transactions.

Multi-Point Bandwidth Considerations

The available bandwidth determines the ability of a multi-point system to cope with isochronous transactions.

Once an endpoint is set up, hardware handles all scheduling. However, as with PC-based EHCI/OHCI/UHCI hosts, before opening a periodic pipe (for use by isochronous or interrupt traffic), software must determine that there is sufficient bandwidth available.

The bandwidth required for different transactions can be determined using algorithms similar to the ones used with PC-based hosts (detailed in Section 5.11.3 of the USB 2.0 Specification).

The available bandwidth is greater where the hub used supports multiple transaction translators.

Babble Interrupt

If the bus is still active at the end of a frame, the USB controller assumes that the function it is connected to has malfunctioned. It suspends all transactions, and generates a babble interrupt (`USB_IRQ.RSTBABBLE`). The USB controller does not start a transaction until the bus is inactive for at least the minimum inter-packet delay. The controller also does not start a transaction unless it can be finished before the end of the frame.

To recover from a babble error condition, the processor must take the following actions inside the interrupt service routine.

1. Turn off VBUS. Wait until the VBUS level indicator reads b#01.
2. Turn on VBUS. Wait until the VBUS level indicator reads b#11.
3. Set the `USB_IRQ.SESSREQ` bit.

The VBUS level indicator is the `USB_DEV_CTL.VBUS` bit field.

NOTE: Because VBUS is sourced external to the processor, make sure that the hardware design connects a GPIO or the dedicated `USB_VBUS` signal to the external source. This connection enables software to turn VBUS on and off.

VBUS Events

The USB On-The-Go specification defines a series of thresholds to which the devices involved in point-to-point communications must respond.

- VBUS Valid (between 4.4 V and 4.75 V)
- Session Valid for A device (between 0.8 V and 2.1 V)
- Session End (between 0.2 V and 0.8 V)

The critical thresholds and the processor response depend on whether the device is an A device or a B device and the circumstances of the event. The following sections describe these actions.

Actions as an A Device

VBUS >VBUS Valid with session initiated by USB controller. VBUS level indicator = b#11 and the session bit is set. When VBUS is greater than VBUS valid, the USB controller selects host mode and waits for a device to connect. It then generates a connect interrupt. The processor resets and enumerates the connected B device.

VBUS > Session valid with session initiated by B device. VBUS level indicator = b#10 and the session bit is clear. When VBUS is greater than session valid, the USB controller generates a session request interrupt. The processor sets the session bit. The USB controller either stays in host mode or changes to peripheral mode, depending upon the state of the pull-up resistor on the B device. For more information, refer to the host negotiation protocol of the OTG specification. The state of the host mode bit indicates the selected mode.

VBUS below VBUS Valid while the Session bit remains set. VBUS level indicator b#11 and the session bit is set. This event indicates a problem with the VBUS power level. For example, the battery power could have dropped too low to sustain VBUS valid. Or, the B device could be drawing more current than the A device can provide. In either case, the USB controller automatically terminates the session and generates a VBUS error interrupt.

To recover from this VBUS error condition, the processor must take the following actions inside the VBUS error interrupt handler.

- Turn off VBUS and wait until the `USB_DEV_CTL.VBUS` reads b#01.
- Turn on VBUS and wait until the `USB_DEV_CTL.VBUS` reads b#11.
- Set the `USB_DEV_CTL.SESSION` bit

The `USB_DEV_CTL.VBUS` bit field indicates the VBUS level.

NOTE: Because VBUS is sourced external to the processor, make sure that the hardware design connects a GPIO or the dedicated `DrvVBUS` signal to the external source. Then, the software can be used to turn VBUS on and off.

Actions as a B Device

VBUS > Session Valid. VBUS level indicator = b#10 and session bit is clear. This event indicates activity from the A device. The USB controller sets the session bit and disconnects the pull down resistor on the D+ line.

VBUS < Session Valid. While the session bit remains set, VBUS level indicator = b#01 and session bit is set. This event indicates that the A device has lost power (or become disconnected). The USB controller clears the session bit and generates a disconnect interrupt. The processor ends the session.

VBUS < Session End. VBUS level indicator = b#00. This event is the condition under which a B device can initiate a session request. If the session bit is set, then after 2 ms of SE0 on the bus, the USB controller starts SRP by first pulsing the data line, then pulsing the `USB_VBUS` signal.

Host Mode Reset

If the USB_POWER.RESET is set while the USB controller is in host mode, the USB controller generates reset signaling on the bus. The processor core must keep this bit set for 20 ms to ensure correct resetting of the target device. After the processor core clears the bit, the USB controller starts its frame counter and transaction scheduler.

Host Mode Suspend

The controller has a suspend mode that allows power savings for the processor. The mode operates as follows.

Entry into Suspend mode. When operating as a host, the USB controller can be prompted to go into suspend mode by setting the USB_POWER.SUSPEND bit. When this bit is set, the USB controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions start. No SOF packets are generated. If the USB_POWER.SUSPEND bit is set, the UTMI+ PHY goes into low-power mode when the USB controller goes into suspend mode and stops the clock.

Sending Resume Signaling. When the application requires the USB controller to leave suspend mode, it must clear and then set the USB_POWER.RESUME bit, and leave it set for 20 ms. While the USB_POWER.RESUME bit is high, the USB controller generates resume signaling on the bus. After 20 ms, the processor core must clear the USB_POWER.RESUME bit, at which point the frame counter and transaction scheduler start.

Responding to Remote Wake-up. If resume signaling is detected from the target while the USB controller is in suspend mode, the UTMI+ PHY is brought out of low-power mode and the clock restarts. The USB controller then exits suspend mode and automatically sets the USB_POWER.RESUME bit to take over generating the resume signaling from the target. If the USB_IRQ.RESUME bit=1, software generates an interrupt.

Suspending and Resuming the Controller

With the introduction of link power management, there are two basic methods to suspend and resume the USB controller. The ***Basic LPM transaction*** diagram demonstrates these two methods.

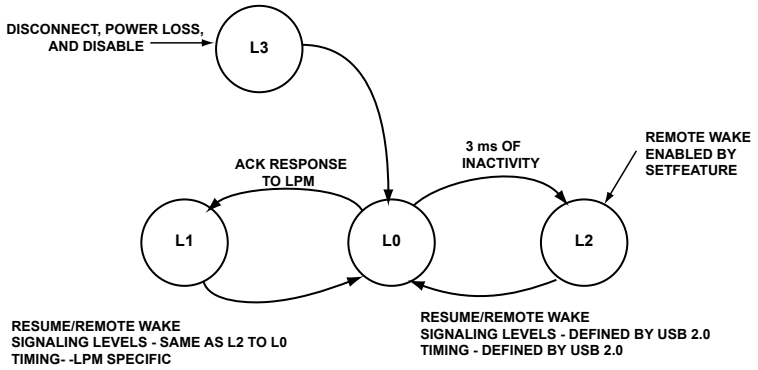

Figure 28-9: Basic LPM Transaction

The procedure that suspends and resumes the USB controller depends on whether the core operates as a device or a host, and the method of suspend desired. The following sections describe these options.

Suspend or Resume by Inactivity on the USB Bus (L0 to L2 State) in Peripheral Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a peripheral, the USB controller monitors activity on the USB and when no activity has occurred for 3 ms, the controller goes into suspend mode. If the USB_IRQ.SUSPEND interrupt has been enabled, the USB controller now generates an interrupt. The USB_IRQ.SUSPEND output also goes low (if enabled).

   The *POWERDWN* signal is also asserted to indicate that the application can stop USB_CLKIN to save power. *POWERDWN* then remains asserted until either power is removed from the bus (indicating that the device has been disconnected) or resume signaling or reset signaling is detected on the bus.

2. When resume signaling occurs on the bus, the USB_CLKIN must be restarted, if necessary. The USB controller then automatically exits suspend mode. If the USB_IRQ.RESUME interrupt is enabled, the USB controller generates an interrupt.

3. Initiating a remote wake-up. To initiate a remote wake-up while the controller is in suspend mode, set the USB_POWER.RESUME bit=1. (If USB_CLKIN has been stopped, it must be restarted before this write can occur.) The software must leave then this bit set for approximately 10 ms (minimum of 2 ms, a maximum of 15 ms) before resetting it to 0. By this time the hub is driving resume signaling on the USB.

NOTE: The USB_IRQ.RESUME interrupt is not generated when the software initiates a remote wake-up.

Suspend or Resume by Inactivity on the USB Bus (L0 To L2 State) in Host Mode

The following steps occur in this mode.

1. Entry into suspend mode. When operating as a host, the USB controller can be prompted to go into suspend mode by setting the USB_POWER.SUSPEND bit. When this bit is set, the USB controller completes the current transaction then stops the transaction scheduler and frame counter. No further transactions start and no SOF packets are generated. If the USB_POWER.SUSEN bit is set, the UTMI+ PHY goes into low-power mode when the controller goes into suspend mode and stops USB_CLKIN.

2. Sending resume signaling. When the application requires the controller to leave suspend mode, it clears the USB_POWER.SUSPEND bit, sets the USB_POWER.RESUME bit, and leaves it set for 20 ms. While the USB_POWER.RESUME bit is high, the controller generates resume signaling on the bus. After 20 ms, the processor core must clear the USB_POWER.RESUME bit, at which point the frame counter and transaction scheduler start.

3. Responding to remote wake-up. If resume signaling is detected from the target while the USB controller is in suspend mode, the UTMI+ PHY is brought out of low-power mode and restarts USB_CLKIN. The controller then exits suspend mode and automatically sets the USB_POWER.RESUME bit to 1 to take over generating the resume signaling from the target. If the USB_IRQ.RESUME interrupt is enabled, the USB controller generates an interrupt.

Suspend or Resume by an LPM Transaction (L0 To L1 State) in Peripheral Mode

The following steps occur in this mode.

1. Enter into suspend mode. When operating as a peripheral, the controller never initiates an LPM suspend (transition from the L0 state to the L1 state). Rather, the controller only suspends at the request of the host. Configure the USB_LPM_CTL register appropriately to enable the LPM feature. The USB controller uses the register field USB_LPM_CTL.EN bit to enable and support extended and LPM transactions. The USB controller uses the USB_LPM_CTL.TX field to instruct the hardware that it is ready to suspend and to respond to the next LPM transaction with an ACK. In this case, the controller responds to the next LPM transaction with an ACK if all other conditions are met. The ***Response to LPM Transaction*** table summarizes the response of the USB controller to an LPM transaction.

Table 28-6: Response to LPM Transaction

LPMXMT	LPMCNTRL	Data Pending (Resides in Tx FIFOs)	Response to Next LPM Transaction
1'b0, 1'b0 1'b1 1'b1	2'b00, 2'b10 2'b00 2'b10	Do-not-care	Timeout
1'b0, 1'b1	2'b01	Do-not-care	STALL
1'b0	2'b11	Do-not-care	NYET
1'b1	2'b11	Yes	NYET
1'b1	2'b11	No	ACK

For all cases in the table in which the controller responds (no timeout occurs), an LPM interrupt is generated in the USB_LPM_IRQ register. The controller responds with an ACK only if there is no data pending in any of the Tx endpoint FIFOs. If there is data pending, the USB controller responds with a NYET.

Once an LPM transaction is successfully received, three events occur:

a. The USB_LPM_ATTR register is updated with values received in the LPM transaction. See the "Register Descriptions" section of this chapter for complete information on this register.

b. The controller suspends 9 µs after transmitting the ACK. The host or the controller can drive resume signaling 50 µs after this event. During the 9 µs interval, the host can continue to transmit the LPM transaction. The controller responds with an ACK in this case regardless of the USB_LPM_CTL.TX bit value.

c. An interrupt is generated informing software of the response (an ACK in this case). An ACK response is the indication to software that the controller has suspended.

   Since the primary purpose of LPM is to save power, software reads the USB_LPM_ATTR register to determine the attributes of the suspend. Software must make a determination based on these attributes whether there are more potential power savings in the system. In making this determination, note that if the host

initiates the resume signaling, the controller must respond to packet transmissions within the time specified by USB_LPM_ATTR.HIRD + 10 µs.

2. When resume signaling occurs on the bus. When the host resumes the bus, it drives resume signaling for a minimum time specified by the host initiated resume duration bit field (USB_LPM_ATTR.HIRD). The controller must be able to respond to traffic within the time HIRD + 10 µs. The controller transitions to a normal operating state automatically and a resume interrupt is generated in the USB_LPM_IRQ register.

   However for this event to occur, the inputs *CLK* and *XCLK* must be available. To facilitate the resume timing requirement, a negative ACK (NAK) is provided using the USB_LPM_CTL.NAK bit. If this bit is set to 1'b1, all endpoints respond to any transaction (other then an LPM) with a NAK. This bit only takes effect after the controller has suspended LPM. Typically, this bit is asserted when the USB_LPM_CTL.TX field is also asserted. Using this feature can simplify the resume timing requirement because the controller only needs *XCLK* to respond (with a NAK) to traffic. Software can continue to restore the system to normal operation while the controller responds to all transactions with a NAK. After software completely restores the system, it can then clear the USB_LPM_CTL.NAK bit.

3. Initiating remote wake-up. To initiate a remote wake-up while in suspend mode, the controller writes a 1'b1 to theUSB_LPM_CTL.RESUME bit. This bit is self clearing. Writing a 1'b1 drives resume signaling on the bus for 50 µs. The host responds by driving resume for 60 µs to 990 µs. 10 µs after the host stops driving resume, the controller transitions to its normal operational state and is ready for packet transmission. A resume interrupt is generated in the USB_LPM_IRQ register.

Suspend or Resume by an LPM Transaction (L0 to L1 State) in Host Mode

The following steps occur in this mode.

1. Enter into suspend mode. When operating as a host, the controller initiates an LPM suspend (transition from the L0 state to the L1 state) by initiating an LPM transaction as follows.

   a. Software sets up the desired attributes of the suspend in the USB_LPM_ATTR register. Enabling remote wake-up and a large HIRD gives the peripheral more opportunity to conserve power.

   b. Enable all LPM interrupts in the USB_LPM_IEN register.

   c. Software writes 0x01 to the USB_LPM_CTL register to initiate the transaction.

   d. An interrupt is generated to inform software of the response to the LPM transaction. If an ACK was received, then the controller suspends automatically within 8 µs. This event indicates that the controller has suspended.

   If the response from the device has a bit stuff error or a PID error, then an USB_LPM_IRQ.LPMERR interrupt is generated. The hardware immediately attempts the LPM transaction two more times. The device does not suspend for 8 µs after the initial LPM so it can respond to either of these subsequent LPM transactions. If an LPM timeout has occurred three times, the USB_LPM_IRQ.LPMNC and the USB_LPM_IRQ.LPMERR interrupts are set. Now, software is unaware of the device state and must deduce it by other means.

2. Send resume signaling. Software generates resume signaling as follows.

a. Enable all LPM interrupts in the USB_LPM_IEN register.

b. Software writes to the USB_LPM_CTL.RESUME bit which is self-clearing. This operation causes resume signaling on the bus for the time specified in the USB_LPM_ATTR.HIRD bit field. Hardware assumes that the last LPM transaction that caused the suspend used this value.

c. After HIRD + 10 µs, the controller transitions to its normal operational state and is ready for packet transmission and a USB_LPM_IRQ.LPMRES interrupt is generated.

NOTE: Prior to resuming, software must ensure that the system is restored from a low-power state and that the inputs *CLK* and *XCLK* are available.

3. Responding to remote wake-up. If the remote wake-up feature is enabled in the LPM transaction that caused the suspend, then the device can drive resume signaling on the bus. When this event occurs, the device drives resume signaling BUS for 50 µs. The controller immediately begins driving resume signaling on the BUS and continues for 60 µs. 10 µs after completion of the resume signaling, the controller transitions to its normal operating state and is ready for packet transmission. Then, the USB_LPM_IRQ.LPMRES interrupt is generated.

USB Event Control

The following sections provide information on the use of interrupts, reset, and the reporting of errors and interface status.

Interrupt Signals

The "Interrupt Table" section at the beginning of this chapter shows the two interrupts generated from the USB controller.

The software generates interrupts from control endpoint zero under the following conditions

- When a control transaction ends before the end of the data is transferred
- When a data packet is sent or received from the endpoint 0 FIFOs

The USB controller generates interrupts from transmit endpoints (USB_INTRTX) under the following conditions:

- A packet is sent from the TX FIFO (host and peripheral mode)
- After three attempts at transmitting a packet, no valid handshake packet is received (host mode)

The software generates interrupts from receive endpoints (USB_INTRRX) under the following conditions:

- A packet is received into the RX FIFO (host and peripheral mode)
- A stall handshake is received (host mode)
- After three attempts at receiving a packet, no data packet is received (host mode)

The software generates interrupts from the USB status (USB_IRQ) under the following conditions:

- When VBUS drops below the VBUS valid threshold during a session (A device only)

- When SRP signaling is detected (A device only)
- When device disconnect is detected (host mode)
- When a session ends (peripheral mode)
- When a device connection is detected (host mode)
- At start of frame (SOF)
- When reset signaling is detected on USB (peripheral mode)
- When babble is detected (host mode)
- In suspend mode, when resume signaling is detected on USB
- When suspend signaling is detected (peripheral mode)

The software generates interrupts for the following VBUS control requests:

- Drive VBUS greater than 4.4 V (default A device)
- Stop driving VBUS
- Start charging VBUS (peripheral mode)
- Stop charging VBUS
- Start discharging VBUS (peripheral mode)
- Stop discharging VBUS

Interrupt Handling

When interrupted with a USB interrupt, the processor core must read the interrupt status register to determine which endpoints have caused the interrupt and jump to the appropriate routine. If multiple endpoints have caused the interrupt, endpoint 0 must be serviced first, followed by the other endpoints. The ***USB Interrupt Service Routine*** figure shows a flowchart for the USB interrupt service routine.

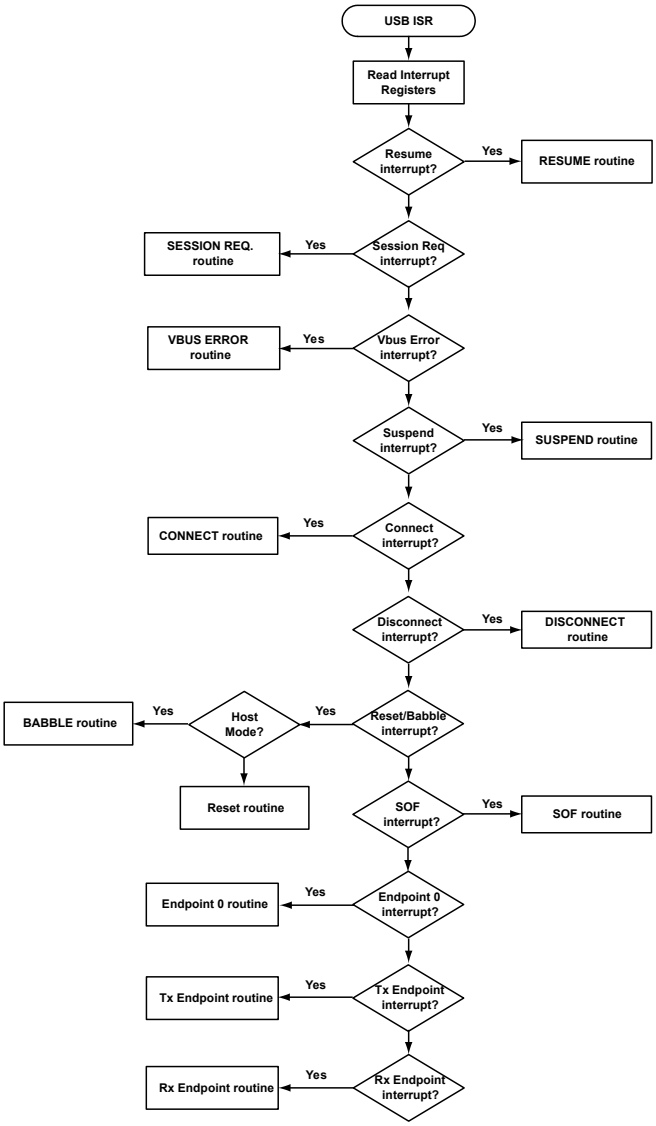

Figure 28-10: USB Interrupt Service Routine

Reset Signals

The USB controller includes an active-high synchronous hardware reset sourced from the processor core. Another source of peripheral reset is through the USB, when USB reset signaling is detected on the I/O lines. Per the USB 2.0 Specification, this state is entered when both the D+ and D- inputs are driven low for 2.5 ms or more. (The USB host typically holds the reset for greater than 10 ms).

Reset in Peripheral Mode

When the USB controller detects a reset, it performs the following actions:

- Sets the `USB_FADDR` register to zero
- Sets the `USB_INDEX` register to zero

- Flushes all endpoint FIFOs
- Clears all control and status registers
- Enables all interrupts
- Generates a reset interrupt

The USB controller reset does not affect the USB_IRQ and USB_VBUS_CTL registers. These registers are only reset (along with the ones listed) during a system reset.

If the USB_POWER.HSEN bit was set, the USB controller also tries to negotiate for high-speed operation. The USB_POWER.HSMODE bit indicates whether high-speed operation is selected.

When the application software receives a reset interrupt, it closes any open pipes and waits for bus enumeration to begin.

USB Reset in Host Mode

If the USB_POWER.RESET bit =1 while the USB controller is in host mode, the controller generates reset signaling on the bus.

If the USB_POWER.HSEN bit =1, the controller also tries to negotiate for high-speed operation.

The processor core must keep the USB_POWER.RESET bit set for at least 20 ms to ensure correct resetting of the target device. After the processor core clears the bit, the USB controller starts its frame counter and transaction scheduler.

The USB controller uses the USB_POWER.HSMODE bit to select high-speed operation.

USB Programming Model

The following sections describe the USB OTG programming model.

Peripheral Mode Flow Charts

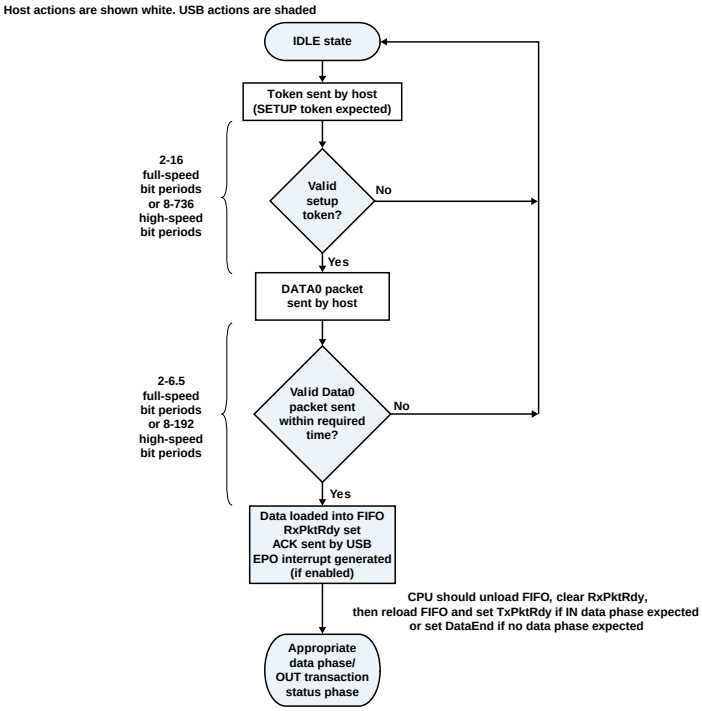

Figure 28-11: USB Control Setup Phase

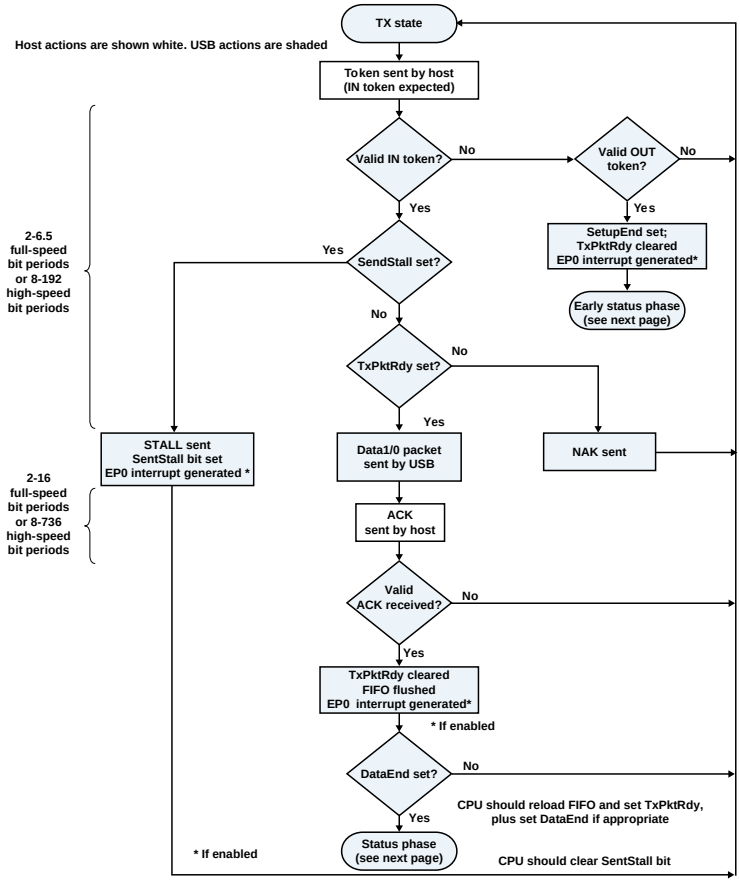

Figure 28-12: Control In Data Phase

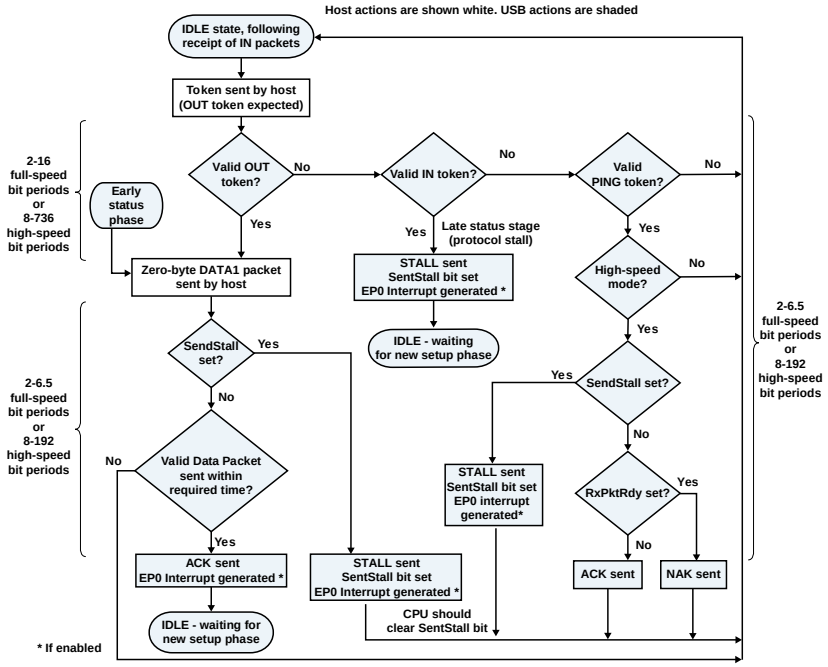

Figure 28-13: Control In Status Phase

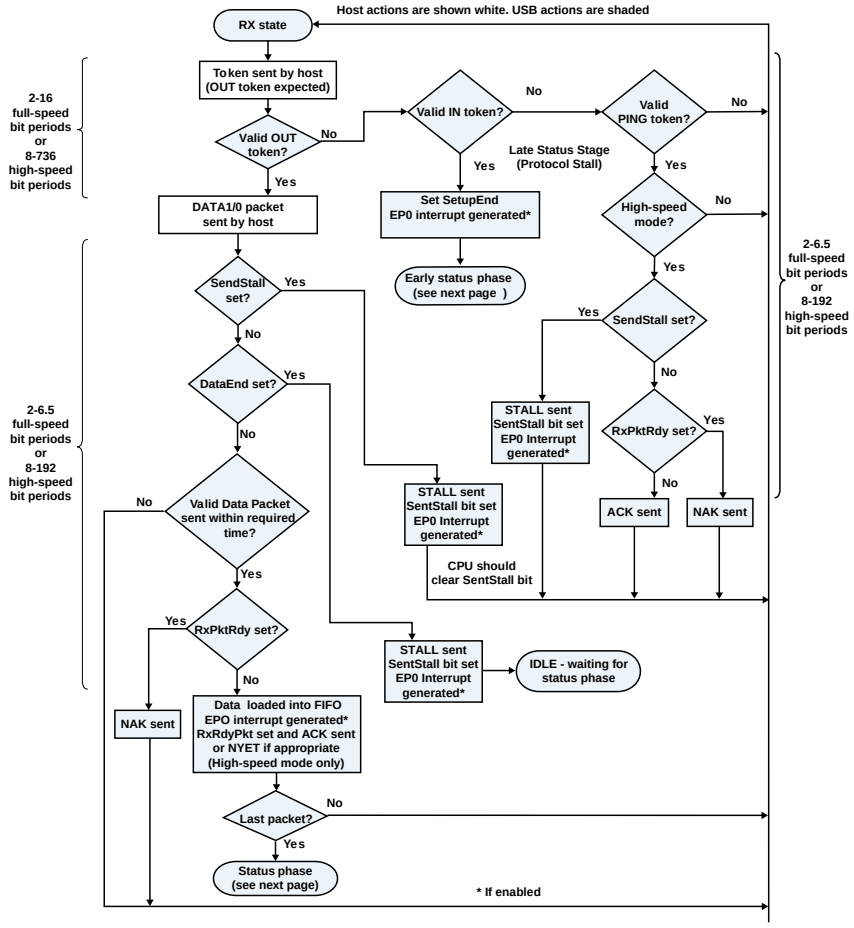

Figure 28-14: Control Out Data Phase

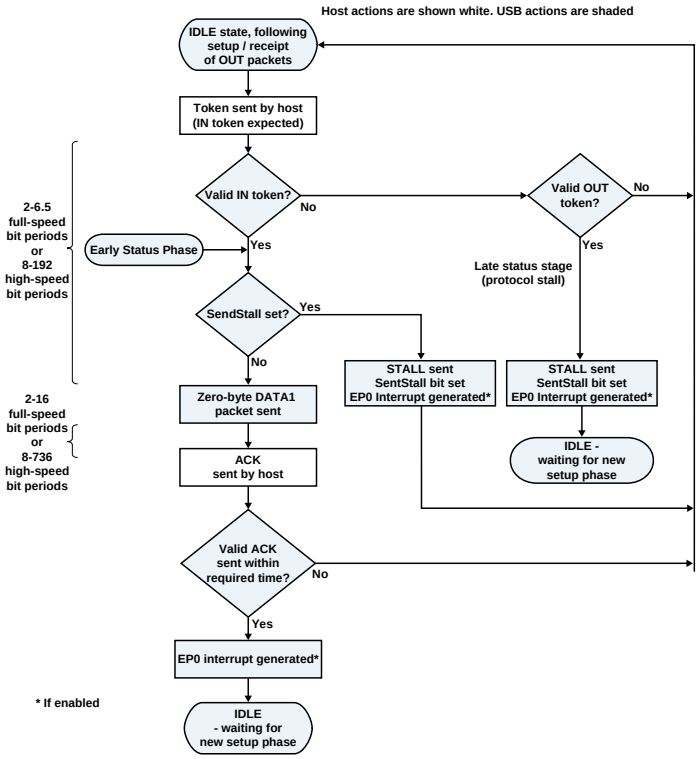

Figure 28-15: Control Out Status Phase

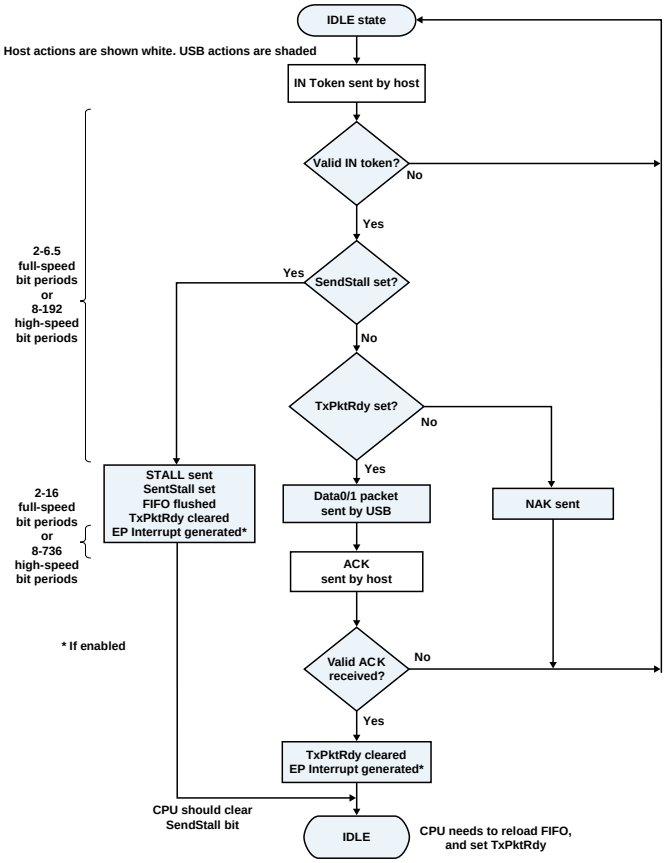

Figure 28-16: Bulk/Low Bandwidth Interrupt In Transaction

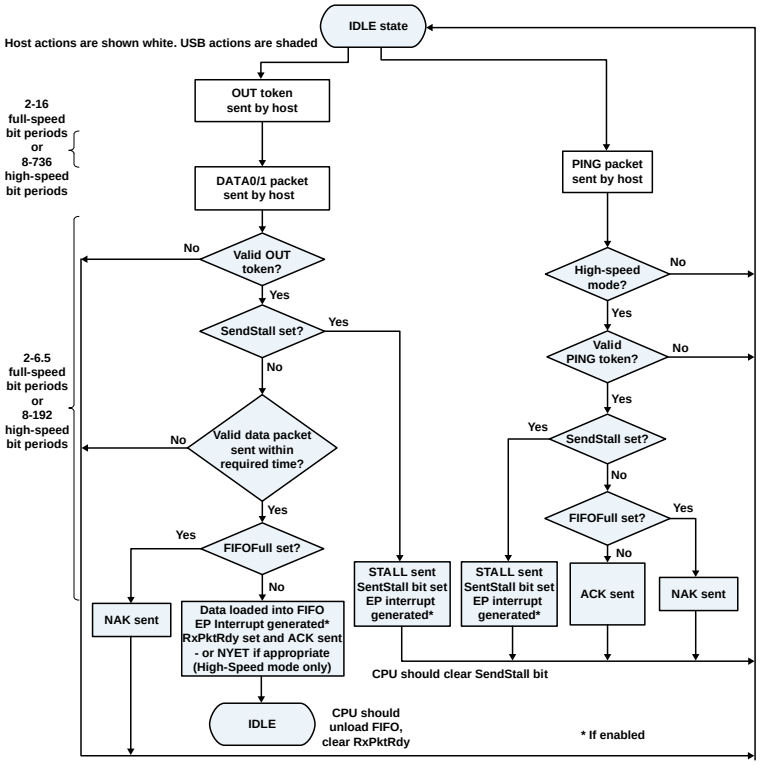

Figure 28-17: Bulk/Low Bandwidth Interrupt Out Transaction

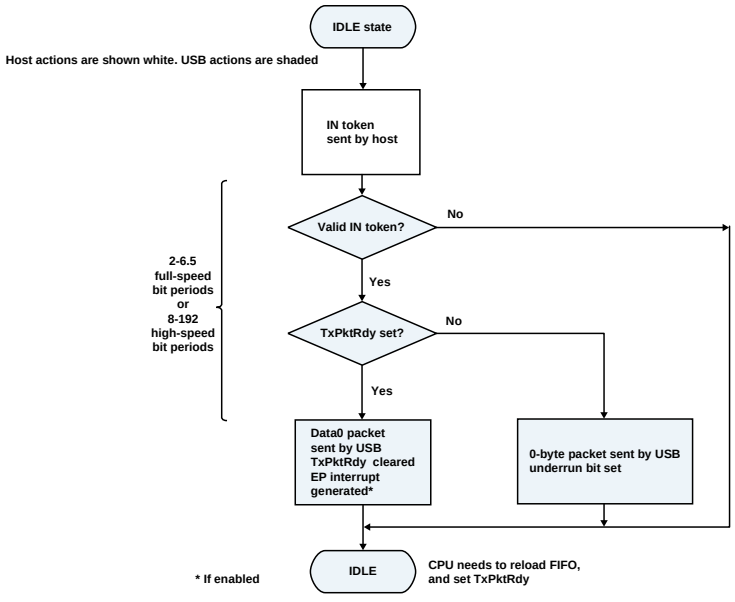

Figure 28-18: Full-speed/Low Bandwidth Isochronous In Transaction

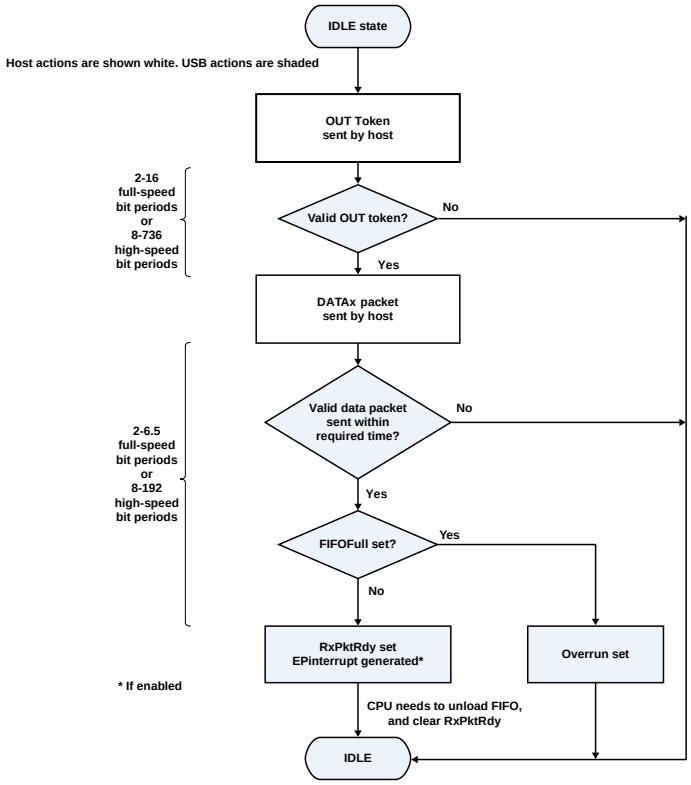

Figure 28-19: Full-speed/Low Bandwidth Isochronous Out Transaction

Host Mode Flow Charts

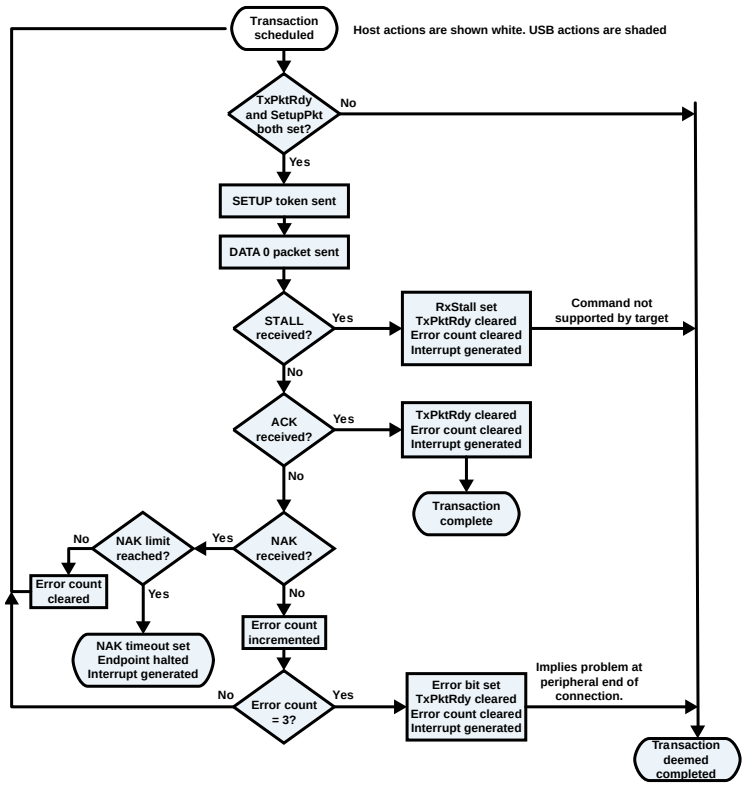

Figure 28-20: USB Control Setup Phase

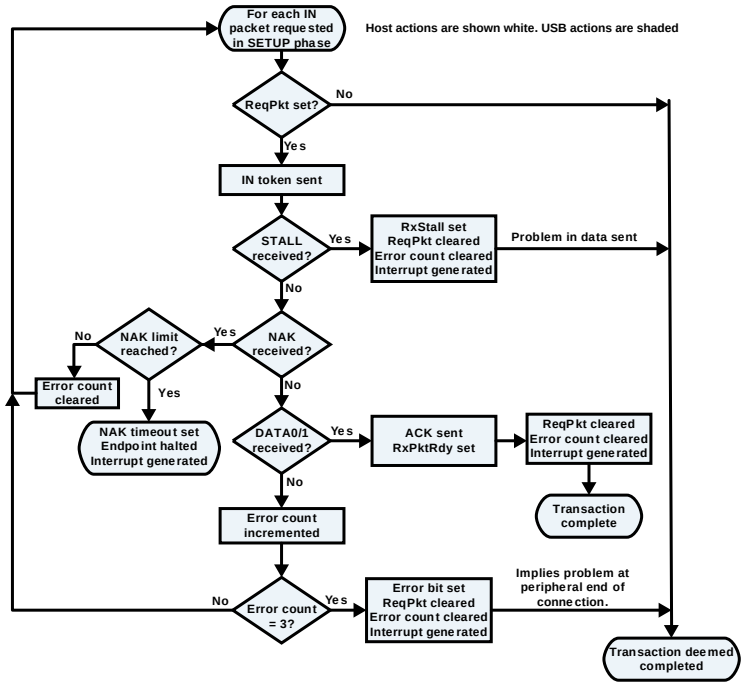

Figure 28-21: Control In Data Phase

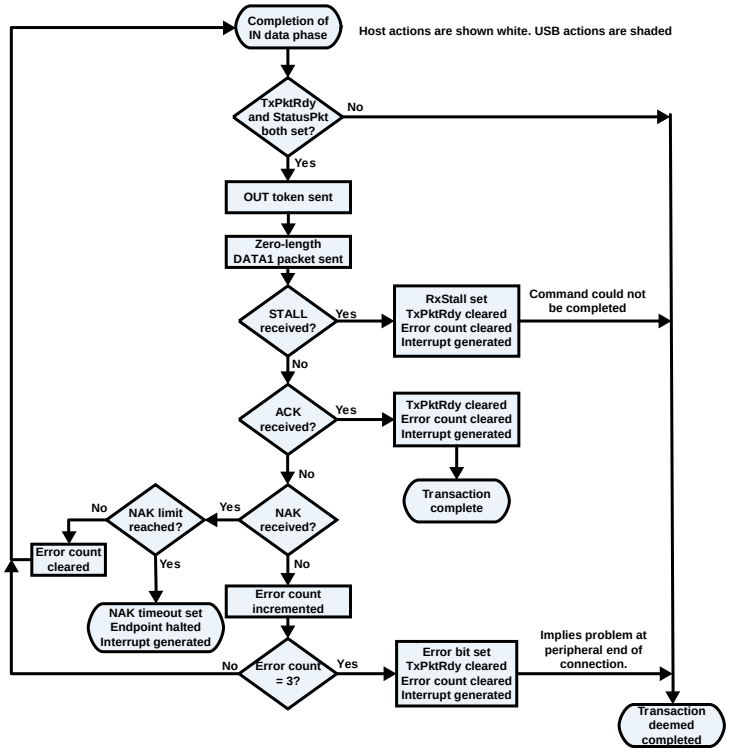

Figure 28-22: Control In Data Status Phase

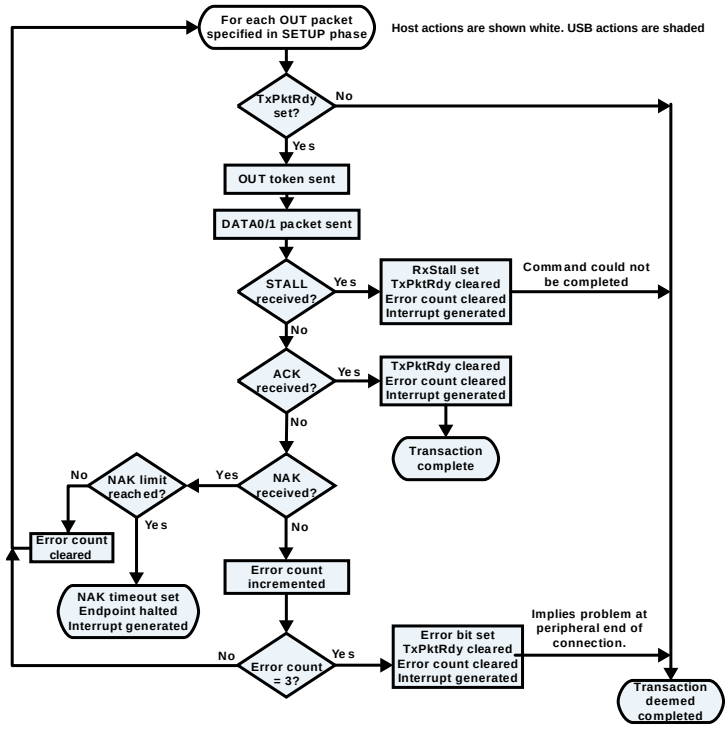

Figure 28-23: Control Out Data Phase

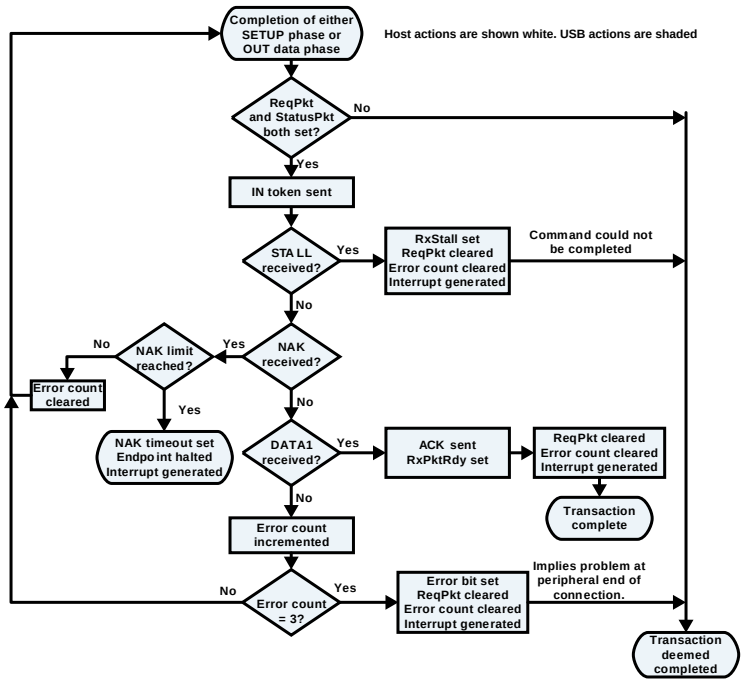

Figure 28-24: Control Out Data Status Phase

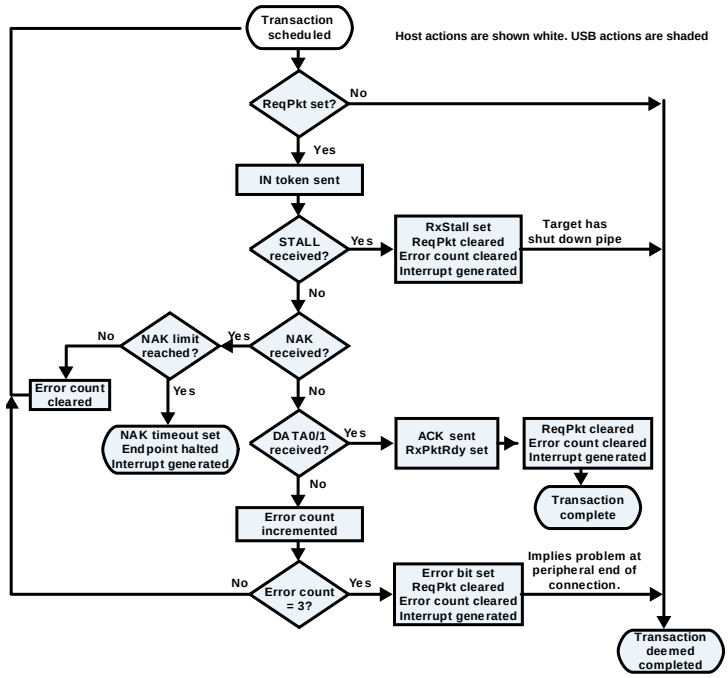

Figure 28-25: Bulk/Low Bandwidth Interrupt In Transaction

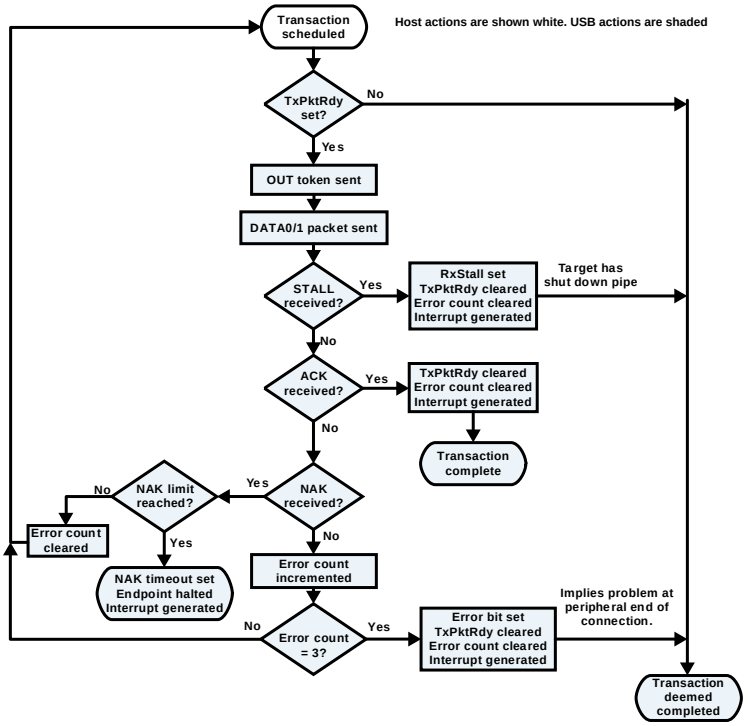

Figure 28-26: Bulk/Low Bandwidth Interrupt Out Transaction

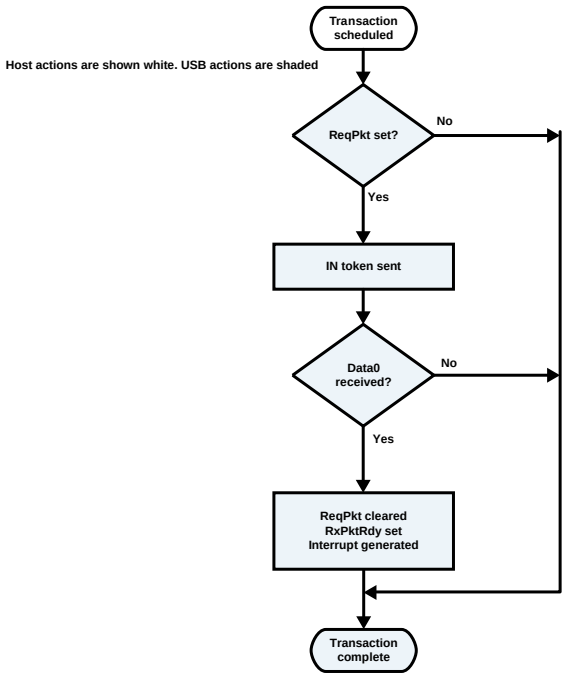

Figure 28-27: Full-Speed/Low Bandwidth Isochronous In Transaction

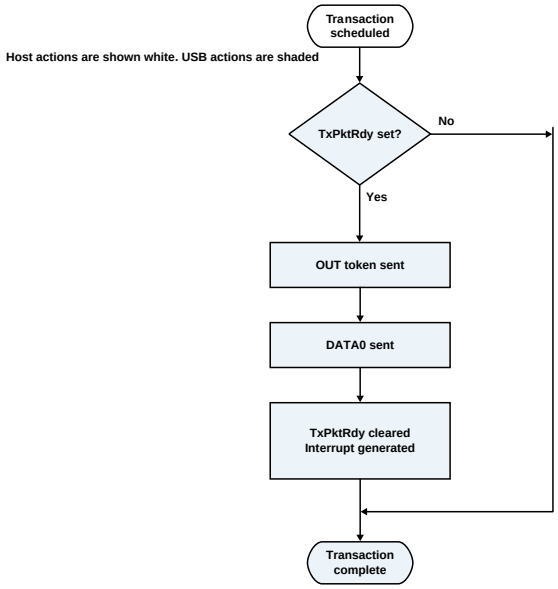

Figure 28-28: Full-Speed/Low Bandwidth Isochronous Out Transaction

DMA Mode Flow Charts

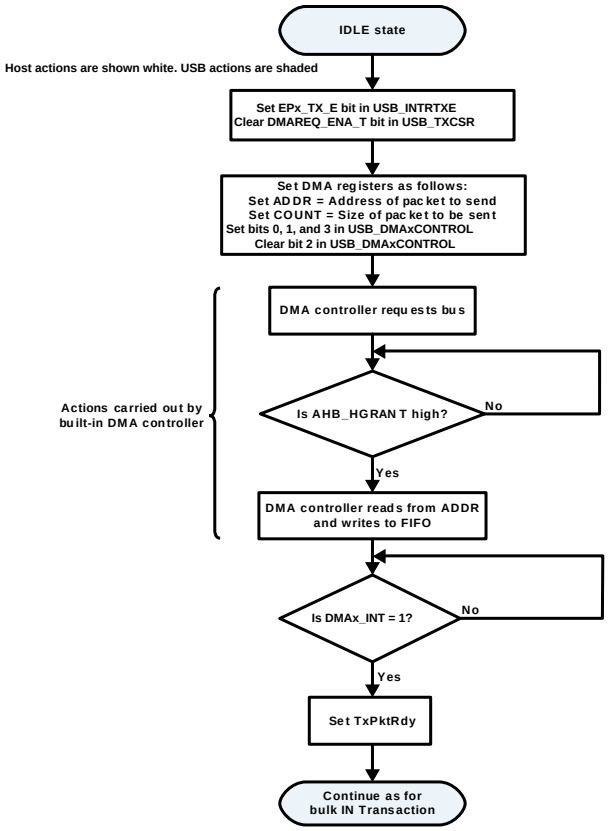

Figure 28-29: Single Packet Transmit During DMA Operation

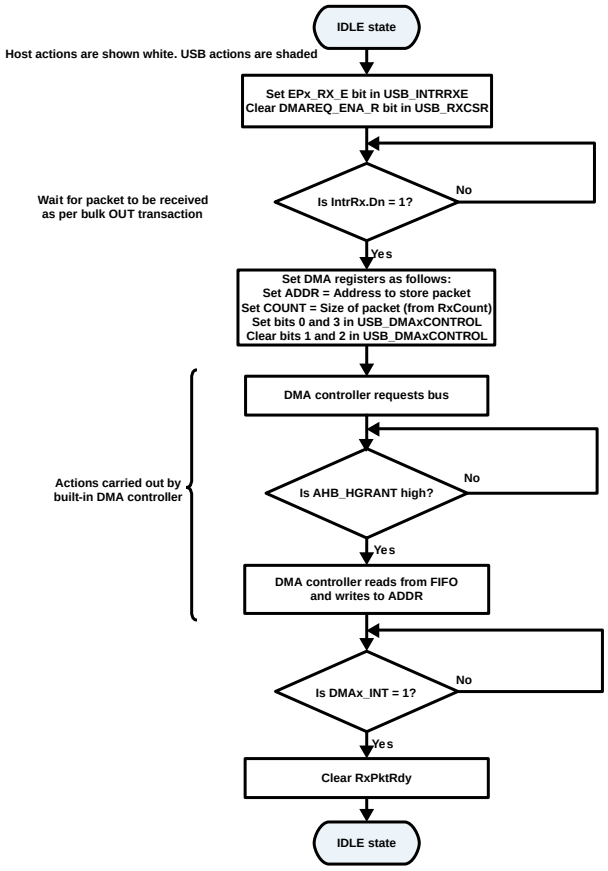

Figure 28-30: Single Packet Receive During DMA Operation

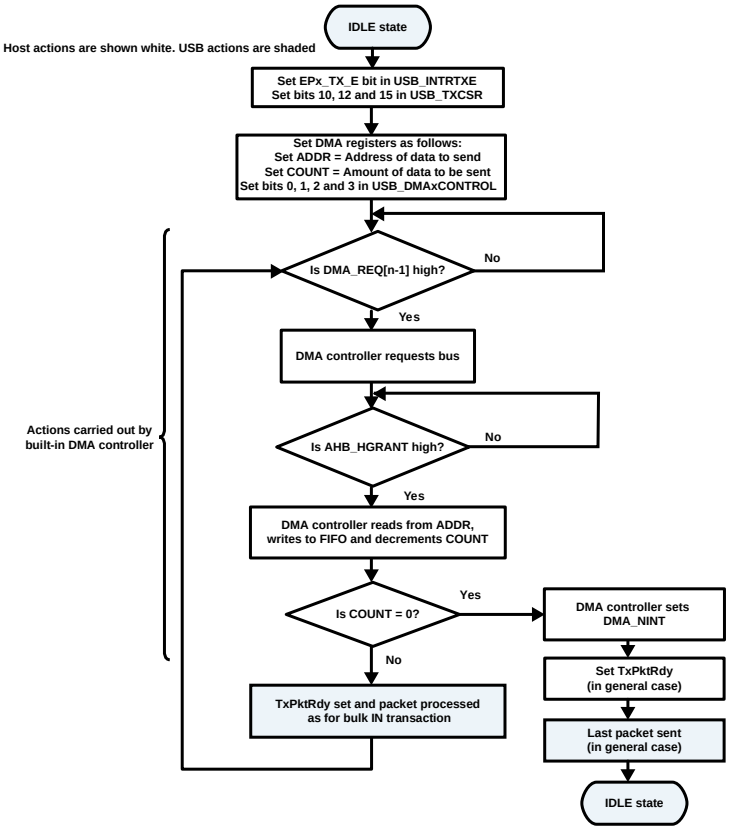

Figure 28-31: Multiple Packet Transmit During DMA Operation

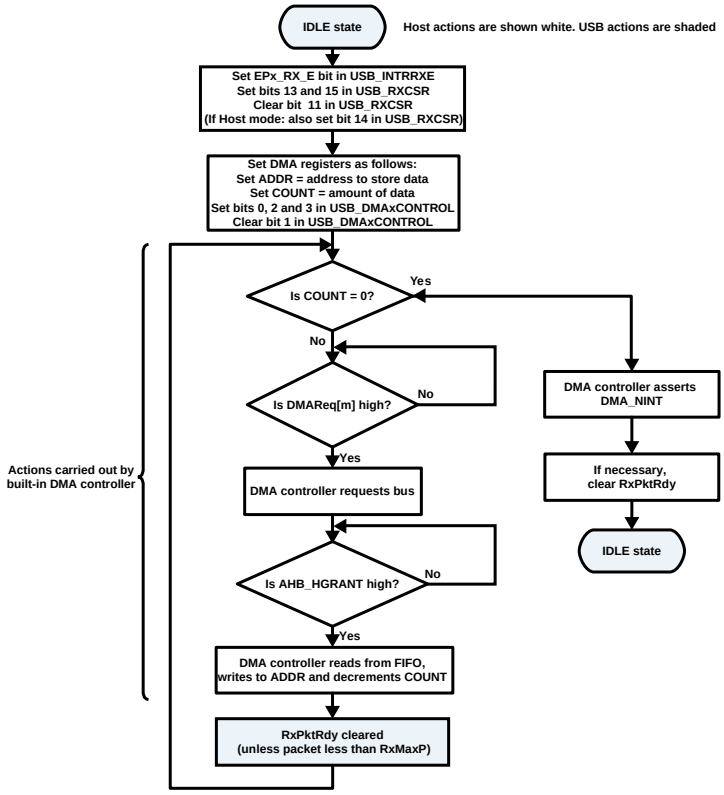

Figure 28-32: Multiple Packet Receive During DMA Operation (Data Size Known)

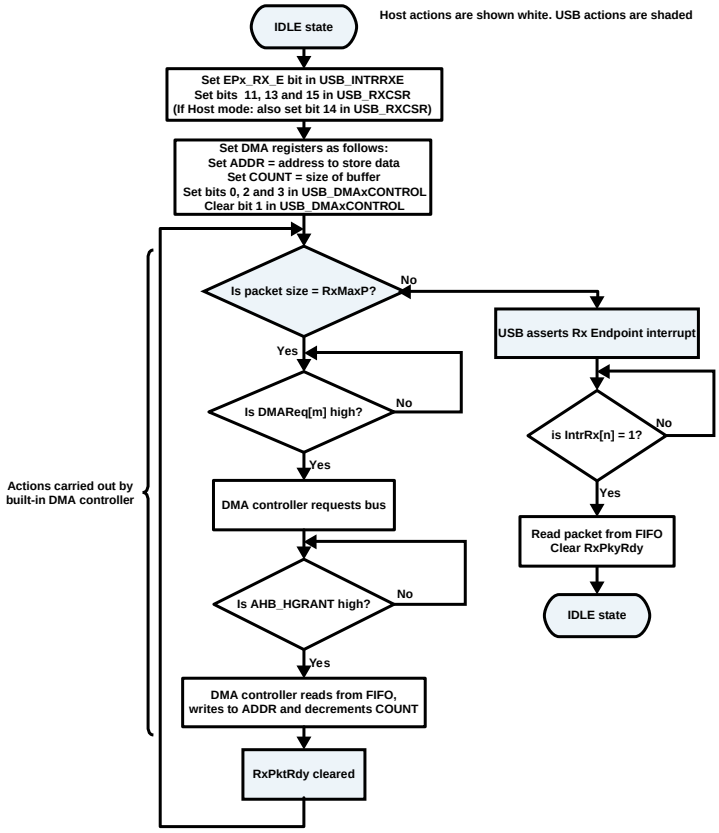

Figure 28-33: Multiple Packet Receive During DMA Operation (Data Size Not-known)

OTG Session Request

To conserve power, the USB on-the-go supplement allows VBUS to only power-up when required and to turn off when the bus is not in use.

The A device on the bus always supplies VBUS. The USB controller samples the `USB_ID` input from the PHY to determine whether it is the A device or the B device. The signal is pulled low when an A-type plug is sensed (signifying that the USB controller is the A device). The input is taken high when a B-type plug is sensed (signifying that the USB controller is the B device).

Starting a Session

When the device containing the USB controller wants to start a session, the processor core must set the `USB_DEV_CTL.SESSION` bit. The USB controller then enables ID pin sensing. This activity results in the `USB_ID` input either being taken low if an A-type connection is detected or high if a B-type connection is detected. The `USB_DEV_CTL.BDEVICE` bit is also set to indicate whether the USB controller has adopted the role of the A device or the B device.

The USB controller is the A device. The USB controller then enters host mode (the A device is always the default host). It waits for VBUS to go above the VBUS valid threshold, as indicated when the `USB_DEV_CTL.VBUS` bits go to 11.

The USB controller then waits for a peripheral to be connected. When the USB controller detects a peripheral, it generates a connect interrupt (USB_IRQ.CON bit) (if enabled). It sets either the USB_DEV_CTL.FSDEV or USB_DEV_CTL.LSDEV bits, depending on whether a full-speed peripheral or a low-speed peripheral was detected. The processor core then resets this peripheral. To end the session, the processor core must clear the USB_DEV_CTL.SESSION bit.

The USB controller is the B device. The USB controller requests a session using the session request protocol defined in the USB on-the-go supplement. This functionality is accomplished by setting the USB_DEV_CTL.SESSION bit.

At the end of the session, typically the USB controller clears the USB_DEV_CTL.SESSION bit. But, the processor core can also clear it when the application software must perform a software disconnect. For more information, see the description of the USB_DEV_CTL register. The USB controller switches on the pull-up resistor on D+. This activity signals to the A device to end the session.

Detecting Activity

When the other device of the OTG set-up wants to start a session, USB controller either:

- Raises VBUS above the session valid threshold (if A device) or
- First pulses the data line, then pulses VBUS (if B device)

Depending on which of these actions happens, the USB controller can determine whether it is the A device or the B device in the current set-up and act accordingly. (The USB_DEV_CTL.VBUS bits=10 indicates if it is the A device.)

If VBUS is raised above the session valid threshold, the USB controller is the B device. The USB controller sets the USB_DEV_CTL.SESSION bit. When reset signaling is detected on the bus, a reset interrupt (USB_IRQ.RSTBABBLE=1) is generated (if enabled) that the processor core interprets as the start of a session. The USB controller is in peripheral mode as the B device is the default peripheral.

At the end of the session, the A device turns off the power to VBUS. When VBUS drops below the session valid threshold, the USB controller detects this state and clears the USB_DEV_CTL.SESSION bit to indicate that the session has ended. (The USB_DEV_CTL.VBUS bits=01 indicates that VBUS has dropped below the session valid threshold). A disconnect interrupt (USB_IRQ.DISCON bit) is also generated (if enabled).

If data line or VBUS pulsing is detected, the USB controller is the A device. The controller generates a USB_IRQ.SESSREQ interrupt to indicate that the B device is requesting a session. The processor core must then start a session by setting the USB_DEV_CTL.SESSION bit.

Host Negotiation Protocol

When the USB controller is the A device (USB_ID low, USB_DEV_CTL.BDEVICE=0), the controller automatically enters host mode when a session starts.

When the USB controller is the B device (USB_ID high, USB_DEV_CTL.BDEVICE=1), the controller automatically enters peripheral mode when a session starts. The processor core can request that the USB controller become the host by setting the USB_DEV_CTL.HOSTREQ bit. This bit can be set either when requesting a session start by setting the USB_DEV_CTL.SESSION bit or at any time after a session has started.

When the USB controller enters suspend mode (no activity on the bus for 3 ms), and assuming the USB_DEV_CTL.HOSTREQ bit remains set, it then enters host mode. It begins host negotiation (as specified in the USB OTG supplement), causing the PHY to disconnect the pull-up resistor on the D+ line. This event causes the A device to switch to peripheral mode and to connect its own pull-up resistor. When the USB controller detects this activity, it generates a connect interrupt (USB_IRQ.CON bit). The controller also sets the USB_POWER.RESET bit to begin resetting the A device. (The USB controller begins this reset sequence automatically to ensure that reset is started as required within 1 ms of the A device connecting its pull-up resistor). The processor core must wait at least 20 ms, then clear the USB_POWER.RESET bit and enumerate the A device.

When the USB controller-based B device has finished using the bus, the processor core must put it into suspend mode by setting the USB_POWER.SUSPEND. The A device detects this state and either terminates the session or reverts to host mode. If the A device is USB controller-based, it generates a disconnect interrupt (USB_IRQ.DISCON bit) if enabled.

Wake-up from Hibernate State

To conserve power when the chip is idle, systems often use power-down modes to shut down power and clocks to various parts of the chip. Hibernate state saves the most power (core clock, peripherals clocks, and internal power are off; only external power is on).

During normal operation, the software can decide that the chip has been idle for a long enough period. The software determines that there is no immediate need for the clocks to be active and the chip can be put into a power-down mode, such as hibernate. This period of inactivity occurs when there is a USB suspend state (idle on the bus for greater than 3 ms) or if no OTG session is valid. The USB controller uses the USB_POWER.SUSPEND bit and USB_DEV_CTL.VBUS status bits to indicate these states.

Before the system software (driver) pushes processor into the hibernate state, the software has to make sure that the USB_PHY_CTL.HIBER bit is set. Setting this bit activates the non-idle activity detection logic in the PHY. The non-idle activity detection logic in the analog PHY detects any non-idle activity on the USB bus. This logic wakes up the processor and generates a low to high transition on the SYS_EXTWAKE pin.

To use non-idle activity detection logic as a wake-up source for the processor, enable the USB wake-up source by programming the appropriate bits in the DPM wake-up enable register (DPM_WAKE_EN). After the processor wakes up, USB is listed as the wake-up source in the DPM wake-up status (DPM_WAKE_STAT) register. The external power-up sequence chip uses the SYS_EXTWAKE pin to power up SDRAM or another external peripheral. The processor typically goes through these steps when it comes out of hibernate state.

After the chip comes out of hibernate state, the software has to make sure that the USB_PHY_CTL.RESTORE bit is set. This setting deactivates the non-idle activity detection logic and ensures proper USB functionality.

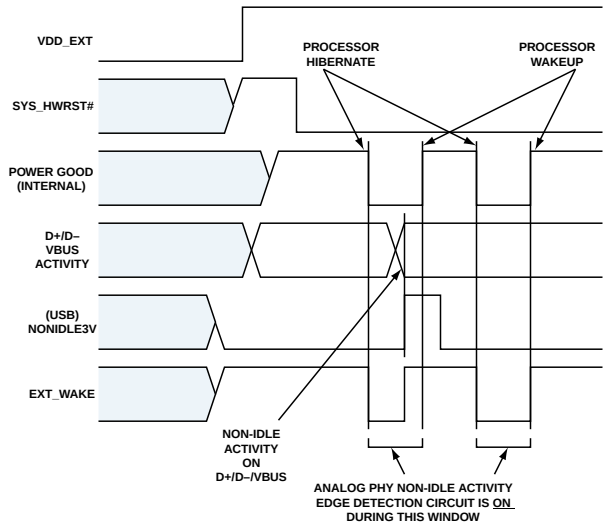

Figure 28-34: Timing Diagram of EXT_WAKE Pin

The interrupt is asserted when either of the following events occur:

- Non-idle signaling occurs during the USB suspend state (including USB reset signaling)
- VBUS falls below the session valid threshold

Wake up from Deep Sleep State

To conserve power when the chip is idle, systems often use power down modes to either shut down the clocks to various parts of the chip or power. In the deep sleep state, the core is idle. The power dissipation on the `VDD_INT` power domain is reduced by gating all the core and system clocks and disabling the PLL.

A USB event (active high polarity only) can take the processor out of deep sleep. To retain the USB communication after exiting deep sleep, the PHY must be configured appropriately to retain the state of D+/D-, thus avoiding the need to re-enumerate. The following condition must be satisfied from the USB PHY.

Before entering deep sleep, the `USB_PHY_CTL.HIBER` bit is set (= 1) and the `USB_PHY_CTL.RESTORE` bit is cleared (= 0) in the PHY control register. This operation is usually done in the USB suspend handler.

After exiting deep sleep, the `USB_PHY_CTL.HIBER` bit is cleared (= 0) and the `USB_PHY_CTL.RESTORE` bit is set (= 1) in the PHY control register. This operation is usually done in the DPM event handler.

```
main()
{
/* Set up USB as a wakeup event. Active high polarity only */
*pREG_DPM0_WAKE_POL = BITM_DPM_WAKE_POL_WS5;
    *pREG_DPM0_WAKE_EN =  BITM_DPM_WAKE_EN_WS5;
}

USB_SUSPEND_HANDLER()
{
```

```
          /* Set HIBER bit and CLEAR RESTORE bit */

          *pREG_USB0_PHY_CTL |= BITM_USB_PHY_CTL_HIBBER;
          *pREG_USB0_PHY_CTL &=  ~(BITM_USB_PHY_CTL_RESTORE);
          /* Enter DeepSleep mode */
      }

      DPM0_EVT_Int_Handler()  /* this is invoked when the processor comes out of
Deep Sleep */
      {
          /* Clear the HIBER bit and Set the RESTORE bit */
         *pREG_USB0_PHY_CTL &=  ~(BITM_USB_PHY_CTL_HIBER);
           *pREG_USB0_PHY_CTL |=  BITM_USB_PHY_CTL_RESTORE;
      }
```

Data Transfer

Whether the USB controller is operating in host or peripheral mode, data channels through the endpoint FIFOs to construct packets that are sent or received over the USB. The USB controller uses the Rx FIFOs to receive OUT packets when in peripheral mode and IN packets when operating in host mode. Similarly, The USB controller uses the Tx FIFOs to transmit IN packets when in peripheral mode and OUT packets as a host.

Data can be moved between the FIFOs and memory using either DMA or core accesses. Each endpoint FIFO has its own individually programmable options so that each can be set up separately. The system must treat each transfer type differently. Data transfers of significant size almost certainly require DMA to move the data around; but the processor can handle smaller packet sizes completely.

Each data endpoint supports both double and single-buffering modes. In single-buffered operation, the processor loads and unloads FIFOs on a packet-by-packet basis. Double-buffering imposes less burden on the system by allowing two packets to be buffered in a FIFO before it is necessary to use DMA or interrupts to service the FIFO. Double-buffering mode is automatically enabled when a *MaxPktSize* is set for an endpoint that is equal to or less than half the size in bytes of that FIFO.

Loading or Unloading Packets from Endpoints

Transfers to and from the FIFOs can be 32-bit, 16-bit, or 8-bit. When using core accesses, use the same width for transfers associated with one data packet, so that data is consistently byte, half-word, or word aligned. The last transfer can, however, contain fewer bytes than the previous transfers in order to complete an 8-bit or 16-bit transfer.

When using the DMA to access the FIFOs, the starting DMA address must be word aligned, or aligned on a 32-bit boundary. The packet transfer starts with a word transfer, but half-word or byte transfers can be added at the end to handle any leftovers.

DMA Master Channels

The USB controller provides eight DMA master channels.

These channels provide a more efficient transfer of larger amounts of data between the FIFOs and the processor core, and the channels free up the processor core for other tasks. The processor uses the DMA control registers to configure and control each of these channels.

Each DMA controller can operate in one of two DMA modes: 0 or 1. When operating in mode 0, the DMA controller can only be programmed to load or unload one packet, so processor intervention is required for each packet transferred over the USB. The DMA controller can use this mode with any endpoint, whether it uses control, bulk, isochronous, or interrupt transactions.

When operating in DMA mode 1, the DMA controller can only be programmed to load or unload a complete bulk transfer, which can be many packets. After set up, the DMA controller loads or unloads the packets, interrupting the processor only when the transfer has completed. DMA mode 1 can only be used with endpoints that use bulk transactions. It is most valuable where large blocks of data are transferred to a bulk endpoint. The USB protocol requires splitting such packets into a series of packets of *MaxPktSize* for the endpoint.

The DMA controller can use mode 1 to avoid the overhead of having to interrupt the processor after each individual packet. It interrupts the processor only after the transfer completes. In some cases, the block of data transferred consists of a predefined number of these packets that the controlling software counts through the transfer process. In other cases, the last packet in the series can be less than the maximum packet size. The receiver can use this short packet to signal the end of the transfer. If the total size of the transfer is an exact multiple of the maximum packet size, the transmitting software must send a null packet for the receiver to detect.

NOTE: Each channel can be independently programmed for the selected operating mode.

For bulk OUT transfers using DMA mode 1, the DMA request line is asserted only when:

- There is an edge transition of the state of the `USB_EP[n]_RXCSR_H.RXPKTRDY`, and
- A payload of *MaxPktSize* has been received

If a data packet is in the FIFO prior to setting the DMA request mode bits, the DMA request line is not asserted when the DMA is enabled. DMA is enabled using the `USB_DMA[n]_CTL.EN` bit. (DMA request mode bits are `USB_EP[n]_RXCSR_H.DMAREQMODE` or `USB_EP[n]_RXCSR_P.DMAREQMODE`). The data is not read from the Rx FIFO in this situation, resulting in a DMA hang. However, since the packet arrived before DMA request mode and DMA request enable bits (`USB_EP[n]_RXCSR_H.DMAREQEN` or `USB_EP[n]_RXCSR_P.DMAREQEN`) were enabled, an Rx interrupt is generated for the corresponding endpoint. Therefore, the software must set the DMA request mode to request mode 0 to unload the pre-received packet. The Rx interrupt service routine can be similar to the following:

If `USB_EP[n]_RXCNT` = MaxPktSize

Switch to DMA mode 0 and unload the packet (in mode 0, the DMA request enable is always asserted, whenever there is data in the FIFO)

Set the `USB_EP[n]_RXCNT` to MaxPktSize so as to unload only one packet

If `USB_EP[n]_RXCSR_H.AUTOCLR` is set, it is not necessary to clear `USB_EP[n]_RXCSR_H.RXPKTRDY` manually

Switch back to DMA Mode 1 and set the count to

(Total_Count – MaxPktSize)

Else

Handle as normal for case of short packet

DMA transfers can be 8-bit, 16-bit, or 32-bit. All transfers associated with one packet (except for the last) must be of the same width, so that the data is consistently byte-aligned or word-aligned. The last transfer can contain fewer bytes than the previous transfers to complete an odd-byte or odd-word transfer.

DMA Bus Cycles

The DMA controller uses incrementing bursts of an unspecified length on the peripheral DMA bus. The controller starts a new burst when it is first granted bus mastership and when the peripheral address starts a new 1 KB block. (The controller is granted bus mastership at the start of a USB packet or when regaining the bus after losing it following a partial packet).

When unloading packets from the FIFOs, the DMA controller requests ahead to the USB controller. The DMA controller starts the transfer with two BUSY cycles while it is gets the first word from the FIFO. All subsequent words of the packet are immediately available and no further BUSY cycles are required. The DMA controller is associated with a two-word buffer, so it does not lose data when it loses bus mastership in the middle of unloading a packet. When the controller regains bus mastership, it can continue unloading the packet without adding any BUSY cycles.

The DMA start address (written to the `USB_DMA[n]_ADDR` register) must be word aligned. The DMA controller supports split transactions and retries.

The DMA request lines are individually enabled using the appropriate DMA request enable bit (there are four options: Tx peripheral and host and Rx peripheral and host) and operate in two modes, referred to as DMA request mode 0 and DMA request mode 1. The operating mode is configured using the appropriate DMA request mode bit (there are four options: Tx peripheral and host and Rx peripheral and host).

NOTE: When operating in host mode, if the `USB_EP[n]_TXCSR_H.RXSTALL` bit or the `USB_EP[n]_TXCSR_H.TXTOERR` is set following three failed transmit attempts, the DMA request line is disabled until the bits are cleared.

The mode selected also affects the generation of endpoint interrupts (if enabled). In DMA request mode 0, no interrupt is generated when packets are received but the appropriate endpoint interrupt is generated to prompt the loading of all packets. In DMA request mode 1, the endpoint interrupt is suppressed except following the receipt of a short packet (one less than `USB_EP[n]_RXMAXP` bytes).

Table 28-7: Endpoint Interrupt Associated with the Receive Packet Ready Bit=1

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
0	X	YES
1	0	NO

Table 28-7: Endpoint Interrupt Associated with the Receive Packet Ready Bit=1 (Continued)

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
1	1	Only is short packet

Table 28-8: Endpoint Interrupt Associated with the Receive Packet Ready Bit=0

DMAReqEnab	DMAReqMode	EP Interrupt Generated?
0	X	YES
1	0	YES
1	1	NO

NOTE: Set the `USB_EP[n]_TXMAXP`/`USB_EP[n]_RXMAXP` registers to an even number of bytes for proper interrupt generation in DMA mode 1.

DMA transfers can be 8-bit, 16-bit, or 32-bit as required. However, all transfers associated with one packet (except for the last) must be of the same width so that the data is consistently byte-aligned, word-aligned, or double-word-aligned. The last transfer can contain fewer bytes than the previous transfers to complete an odd-byte or odd-word transfer.

NOTE: Disable DMA requests before changing the DMA request mode bit. In particular, do not set the `USB_EP[n]_TXCSR_H.DMAREQMODE` bit to zero either before or in the same cycle as the corresponding `USB_EP[n]_TXCSR_H.DMAREQEN` bit is cleared to zero.

Transferring Packets Using DMA

Both the channel and the endpoint must be programmed appropriately to use the the DMA master channels to access the USB controller FIFOs. Many variations are possible. The following sections detail the standard set ups used for the basic actions of transferring individual and multiple packets.

Individual Rx Endpoint Packet

The transfer of individual packets normally uses DMA mode 0. Program the USB controller Rx endpoint as follows.

1. Set to 1 the relevant bit in the `USB_INTRRXE` register.
2. Set to 0 the DMA enable bit of the appropriate `USB_EP[n]_RXCSR_H.DMAREQEN`/ `USB_EP[n]_RXCSR_P.DMAREQEN` register. (There is no need to set the USB controller to support DMA for this operation.)
3. When the USB controller receives a packet, it generates the appropriate endpoint interrupt (using the `USB_INTRRXE` register). The processor must then program the appropriate DMA master channel as follows.
   - Configure the `USB_DMA[n]_ADDR` register with the memory address to store the packet
   - Configure the `USB_DMA[n]_CNT` register with the size of packet (determined by reading the USB controller `USB_RQPKTCNT[n]` register)

- Configure the USB_DMA[n]_CTL register using the following bit settings:USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.DIR=0, USB_DMA[n]_CTL.MODE=0

The DMA controller then requests bus mastership and transfers the packet to memory. It interrupts the processor when it has completed the transfer. The processor then clears the USB_EP[n]_RXCSR_H.RXPKTRDY bit.

Individual Tx Endpoint Packet

Using DMA mode 0, program a USB controller Tx endpoint as follows.

1. Set to 1 the relevant bit in the USB_INTRTXE register.
2. Set to 0 the DMA enable bit of the appropriate USB_EP[n]_TXCSR_H.DMAREQEN/ USB_EP[n]_TXCSR_P.DMAREQEN register. (There is no need to set the USB controller to support DMA for this operation.)
3. When the FIFO can accommodate data, the USB controller interrupts the processor with the appropriate Tx endpoint interrupt. The processor must then program the DMA channel as follows:
   - Configure the USB_DMA[n]_ADDR register with the memory address to store the packet.
   - Configure the USB_DMA[n]_CNT register with the size of packet.
   - Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.DIR=1, USB_DMA[n]_CTL.MODE=0.

The DMA controller then requests bus mastership and transfers the packet to the USB controller FIFO. When it has completed the transfer, it generates a DMA interrupt. The processor then sets the USB_EP[n]_TXCSR_H.TXPKTRDY bit.

Multiple Rx Endpoint Packets

The transfer of multiple packets normally uses DMA mode 1. Program the DMA controller using the DMA registers:

- Configure the USB_DMA[n]_ADDR register with the memory address of data block to send.
- Configure the USB_DMA[n]_CNT register with the maximum size of data buffer.
- Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.IE=1, USB_DMA[n]_CTL.DIR=0, USB_DMA[n]_CTL.MODE=1.

Program the USB controller Rx endpoint as follows:

1. Set to 1 the relevant bit in the USB_INTRRX register.
2. Set to 1 the USB_EP[n]_RXCSR_H.AUTOCLR, USB_EP[n]_RXCSR_H.DMAREQEN, and USB_EP[n]_RXCSR_H.DMAREQMODE bits of the appropriate receive control and status register (host or peripheral). In host mode, set to 1 the USB_EP[n]_RXCSR_H.AUTOREQ and USB_EP[n]_RXCSR_H.DMAREQMODE bits.

As the USB controller receives each packet, the DMA master channel requests bus mastership and transfers the packet to memory. With USB_EP[n]_RXCSR_H.AUTOCLR set, the USB controller automatically clears its USB_EP[n]_RXCSR_H.RXPKTRDY bit. This process continues automatically until the USB controller receives a short packet (one of less than the maximum packet size for the endpoint) signifying the end of the transfer. The DMA controller does not transfer this short packet: instead the USB controller interrupts the processor by generating the appropriate endpoint interrupt. The processor can then read the USB_EP[n]_RXCNT register to see the size of the short packet. It either unloads the packet manually or reprograms the DMA controller in mode 0 to unload the packet.

The USB_DMA[n]_ADDR register is incremented as the DMA controller unloads the packets. The processor determines the size of the transfer by comparing the current value of USB_DMA[n]_ADDR with the start address of the memory buffer.

If the size of the transfer exceeds the data buffer size, the DMA controller stops unloading the FIFO and interrupts the processor.

Multiple Tx Endpoint Packets

Using DMA mode 1 for a Tx endpoint, program the DMA controller as follows:

- Configure the USB_DMA[n]_ADDR register with the memory address of data block to send.
- Configure the USB_DMA[n]_CNT register with the size of the data block.
- Configure the USB_DMA[n]_CTL register using the following bit settings: USB_DMA[n]_CTL.EN=1, USB_DMA[n]_CTL.IE=1 USB_DMA[n]_CTL.DIR=1, USB_DMA[n]_CTL.MODE=1.

Program the USB controller Tx endpoint as follows:

1. Set to 1 the relevant bit in the USB_INTRTXE register.
2. Set to 1 the USB_EP[n]_TXCSR_H.AUTOSET and USB_EP[n]_TXCSR_H.DMAREQEN bits of the appropriate transmit control and status register (host or peripheral).

When the FIFO in the USB controller becomes available, the DMA controller requests bus mastership and transfers a packet to the FIFO. With USB_EP[n]_TXCSR_H.AUTOSET set, the USB controller automatically sets the USB_EP[n]_TXCSR_H.TXPKTRDY bit. This process continues until the entire data block is transferred to the USB controller.

The DMA controller then interrupts the processor by taking the appropriate USB_DMA_IRQ register bit low.

- If the last packet loaded was less than the maximum packet size for the endpoint, the USB_EP[n]_TXCSR_H.TXPKTRDY bit is not set for this packet. The processor must respond to the DMA interrupt by setting the USB_EP[n]_TXCSR_H.TXPKTRDY bit to allow the last short packet to be sent.
- If the last packet loaded is the maximum packet size, then the appropriate action depends on whether the transfer is under the control of an application. One example is the mass storage software on Windows system that keeps count of the individual packets sent.

- If the transfer is not under such control, the processor must respond to the DMA interrupt by setting the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit. This operation sends a null packet for the receiving software to interpret as indicating the end of the transfer.

ADSP-BF70x USB Register Descriptions

Universal Serial Bus Controller (USB) contains the following registers.

Table 28-9: ADSP-BF70x USB Register List

Name	Description
USB_BAT_CHG	Battery Charging Control Register
USB_CT_HHSRTN	Host High-Speed Return to Normal Register
USB_CT_HSBT	High-Speed Timeout Register
USB_CT_UCH	Chirp Timeout Register
USB_DEV_CTL	Device Control Register
USB_DMA[n]_ADDR	DMA Channel n Address Register
USB_DMA[n]_CNT	DMA Channel n Count Register
USB_DMA[n]_CTL	DMA Channel n Control Register
USB_DMA_IRQ	DMA Interrupt Register
USB_EP0I_CFGDATA[N]	EP0 Configuration Information Register
USB_EP0I_CNT[N]	EP0 Number of Received Bytes Register
USB_EP0I_CSR[N]_H	EP0 Configuration and Status (Host) Register
USB_EP0I_CSR[N]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0I_NAKLIMIT[N]	EP0 NAK Limit Register
USB_EP0I_TYPE[N]	EP0 Connection Type Register
USB_EP0_CFGDATA[n]	EP0 Configuration Information Register
USB_EP0_CNT[n]	EP0 Number of Received Bytes Register
USB_EP0_CSR[n]_H	EP0 Configuration and Status (Host) Register
USB_EP0_CSR[n]_P	EP0 Configuration and Status (Peripheral) Register
USB_EP0_NAKLIMIT[n]	EP0 NAK Limit Register
USB_EP0_TYPE[n]	EP0 Connection Type Register
USB_EPINFO	Endpoint Information Register
USB_EPI[N]_RXCNT	EPn Number of Bytes Received Register
USB_EPI[N]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EPI[N]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register

Table 28-9: ADSP-BF70x USB Register List (Continued)

Name	Description
USB_EPI[N]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EPI[N]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EPI[N]_RXTYPE	EPn Receive Type Register
USB_EPI[N]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EPI[N]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EPI[N]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EPI[N]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EPI[N]_TXTYPE	EPn Transmit Type Register
USB_EP[n]_RXCNT	EPn Number of Bytes Received Register
USB_EP[n]_RXCSR_H	EPn Receive Configuration and Status (Host) Register
USB_EP[n]_RXCSR_P	EPn Receive Configuration and Status (Peripheral) Register
USB_EP[n]_RXINTERVAL	EPn Receive Polling Interval Register
USB_EP[n]_RXMAXP	EPn Receive Maximum Packet Length Register
USB_EP[n]_RXTYPE	EPn Receive Type Register
USB_EP[n]_TXCSR_H	EPn Transmit Configuration and Status (Host) Register
USB_EP[n]_TXCSR_P	EPn Transmit Configuration and Status (Peripheral) Register
USB_EP[n]_TXINTERVAL	EPn Transmit Polling Interval Register
USB_EP[n]_TXMAXP	EPn Transmit Maximum Packet Length Register
USB_EP[n]_TXTYPE	EPn Transmit Type Register
USB_FADDR	Function Address Register
USB_FIFOB[n]	FIFO Byte (8-Bit) Register
USB_FIFOH[n]	FIFO Half-Word (16-Bit) Register
USB_FIFO[n]	FIFO Word (32-Bit) Register
USB_FRAME	Frame Number Register
USB_FS_EOF1	Full-Speed EOF 1 Register
USB_HS_EOF1	High-Speed EOF 1 Register
USB_IEN	Common Interrupts Enable Register
USB_INDEX	Index Register
USB_INTRRX	Receive Interrupt Register
USB_INTRRXE	Receive Interrupt Enable Register
USB_INTRTX	Transmit Interrupt Register

Table 28-9: ADSP-BF70x USB Register List (Continued)

Name	Description
USB_INTRTXE	Transmit Interrupt Enable Register
USB_IRQ	Common Interrupts Register
USB_LINKINFO	Link Information Register
USB_LPM_ATTR	LPM Attribute Register
USB_LPM_CTL	LPM Control Register
USB_LPM_FADDR	LPM Function Address Register
USB_LPM_IEN	LPM Interrupt Enable Register
USB_LPM_IRQ	LPM Interrupt Status Register
USB_LS_EOF1	Low-Speed EOF 1 Register
USB_MP[n]_RXFUNCADDR	MPn Receive Function Address Register
USB_MP[n]_RXHUBADDR	MPn Receive Hub Address Register
USB_MP[n]_RXHUBPORT	MPn Receive Hub Port Register
USB_MP[n]_TXFUNCADDR	MPn Transmit Function Address Register
USB_MP[n]_TXHUBADDR	MPn Transmit Hub Address Register
USB_MP[n]_TXHUBPORT	MPn Transmit Hub Port Register
USB_PHY_CTL	PHY Control Register
USB_PLL_OSC	PLL and Oscillator Control Register
USB_POWER	Power and Device Control Register
USB_RAMINFO	RAM Information Register
USB_RQPKTCNT[n]	EPn Request Packet Count Register
USB_RXFIFOADDR	Receive FIFO Address Register
USB_RXFIFOSZ	Receive FIFO Size Register
USB_SOFT_RST	Software Reset Register
USB_TESTMODE	Testmode Register
USB_TXFIFOADDR	Transmit FIFO Address Register
USB_TXFIFOSZ	Transmit FIFO Size Register
USB_VBUS_CTL	VBUS Control Register
USB_VPLEN	VBUS Pulse Length Register

Battery Charging Control Register

The USB_BAT_CHG register controls USB controller battery change-related features.

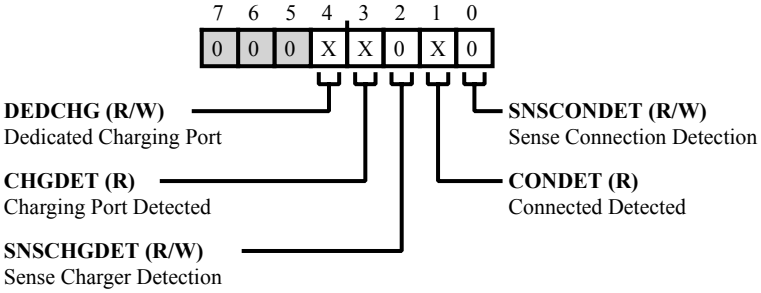

Figure 28-35: USB_BAT_CHG Register Diagram

Table 28-10: USB_BAT_CHG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/W)	DEDCHG	Dedicated Charging Port. The USB_BAT_CHG.DEDCHG bit is asserted if both D+ and D- are high. This can be used to determine if the attached device is a dedicated charging port. This bit is the decode of LineState[1] and LineState[0]. This bit is only valid when a session is initiated, which enables a pullup on D+ when acting as a B-device.
3 (R/NW)	CHGDET	Charging Port Detected. The USB_BAT_CHG.CHGDET bit indicates when a charging port is detected. This bit indicates that D+/- is above V_{DAT_REF} and below V_{LGC}.
2 (R/W)	SNSCHGDET	Sense Charger Detection. The USB_BAT_CHG.SNSCHGDET bit enables charger detection. Setting this bit enables VD_SRC and ID_SINK.
1 (R/NW)	CONDET	Connected Detected. The USB_BAT_CHG.CONDET bit is valid when USB_BAT_CHG.SNSCONDET is enabled. This bit reflects the inverse of D+ (!LineState[0]). If nothing is connected, D+ is pulled high. If a charger or USB port is connected, D+ is pulled low.
0 (R/W)	SNSCONDET	Sense Connection Detection. The USB_BAT_CHG.SNSCONDET bit enables connection detection. Enabling this bit enables I_{DP_SRC} and R_{DM_DWN}.

Host High-Speed Return to Normal Register

The USB_CT_HHSRTN register selects the delay from the end of the high-speed resume signaling (acting as a host) to the return to normal mode operation. This value is multiplied by 4 times the XCLK period (or 16.7 ns). The default setting corresponds to a delay of 100us.

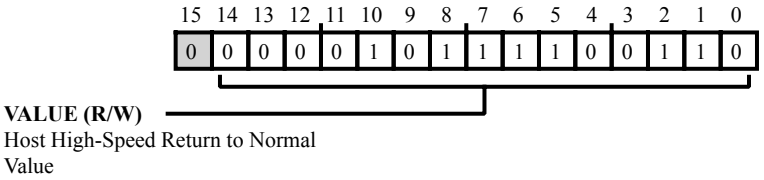

Figure 28-36: USB_CT_HHSRTN Register Diagram

Table 28-11: USB_CT_HHSRTN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Host High-Speed Return to Normal Value.

High-Speed Timeout Register

The USB_CT_HSBT register selects an amount of time to add to the minimum high-speed timeout in units of 64 bit times. The USB 2.0 specification section 7.1.19.2 states that the controller must not timeout less than 736 bit times and must timeout after 816 bit times. The value in USB_CT_HSBT is multiplied by 64-bit times and added to the minimum 736 bit times. Settings less than 1 violate the USB 2.0 specification, making the controller non-compliant.

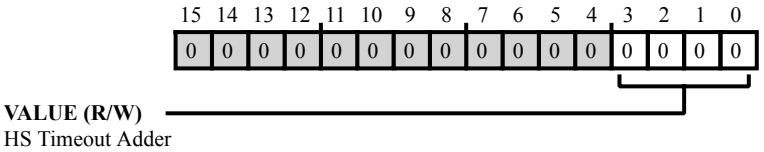

Figure 28-37: USB_CT_HSBT Register Diagram

Table 28-12: USB_CT_HSBT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
3:0 (R/W)	VALUE	HS Timeout Adder. The USB_CT_HSBT.VALUE bits selects an amount of time to add to the minimum high-speed timeout in units of 64 bit times.	
		0	HS Timeout = 736 (bit time)
		1	HS Timeout = 800 (bit time)
		2	HS Timeout = 864 (bit time)

Chirp Timeout Register

The USB_CT_UCH register selects chirp timeout value. The value is multiplied by 4 times the XCLK period (or 67ns). The default setting is 1.1ms.

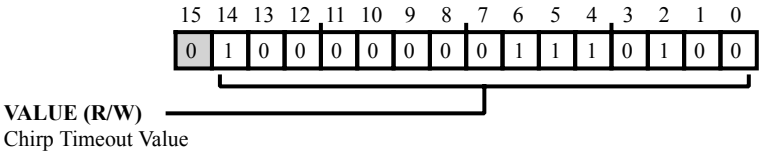

Figure 28-38: USB_CT_UCH Register Diagram

Table 28-13: USB_CT_UCH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14:0 (R/W)	VALUE	Chirp Timeout Value. The USB_CT_UCH.VALUE bits select the chirp timeout value.

Device Control Register

The USB_DEV_CTL register selects whether the USB controller is operating in peripheral mode or in host mode. It is used for controlling and monitoring the VBUS line.

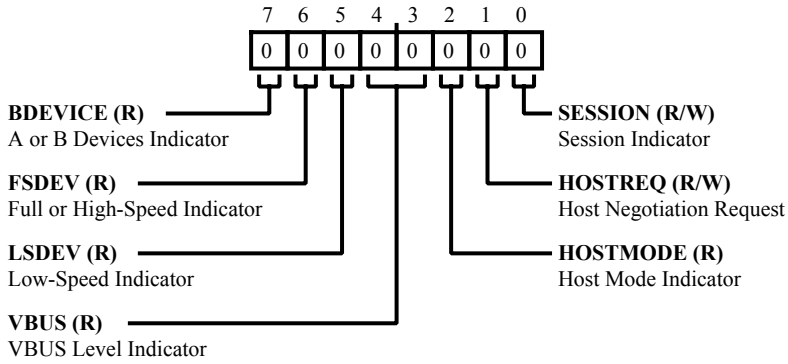

Figure 28-39: USB_DEV_CTL Register Diagram

Table 28-14: USB_DEV_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	BDEVICE	A or B Devices Indicator. The USB_DEV_CTL.BDEVICE bit indicates whether the USB controller is operating as the A device or the B device. This bit is only valid while a session is in progress.	
		0	A Device Detected
		1	B Device Detected
6 (R/NW)	FSDEV	Full or High-Speed Indicator. The USB_DEV_CTL.FSDEV bit is set when a full-speed or high-speed device is detected being connected to the port. High-speed devices are distinguished from full-speed by checking for high-speed chirps when the device detects a USB controller reset. This bit is only valid in host mode.	
		0	Not Detected
		1	Full or High-Speed Detected
5 (R/NW)	LSDEV	Low-Speed Indicator. The USB_DEV_CTL.LSDEV bit is set when a low-speed device is detected being connected to the port. This bit is only valid in host mode.	
		0	Not Detected
		1	Low-Speed Detected

Table 28-14: USB_DEV_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4:3 (R/NW)	VBUS	VBUS Level Indicator. The USB_DEV_CTL.VBUS bits indicated the current VBUS level.	
		0	Below SessionEnd
		1	Above SessionEnd, below AValid
		2	Above AValid, below VBUSValid
		3	Above VBUSValid
2 (R/NW)	HOSTMODE	Host Mode Indicator. The USB_DEV_CTL.HOSTMODE bit is set when the USB controller is acting as a host.	
		0	Peripheral Mode
		1	Host Mode
1 (R/W)	HOSTREQ	Host Negotiation Request. When the USB_DEV_CTL.HOSTREQ bit is set, the USB controller initiates the host negotiation when suspend mode is entered. This bit is cleared when host negotiation is completed. The USB_DEV_CTL.HOSTREQ bit applies when the USB controller is operating as a B device only.	
		0	No Request
		1	Place Request
0 (R/W)	SESSION	Session Indicator. When operating as an A device, the USB_DEV_CTL.SESSION bit is set or cleared by the processor core to start or end a session. When operating as a B device, the USB_DEV_CTL.SESSION bit is set or cleared by the USB controller when a session starts or ends. This bit is also set by the processor core to initiate the session request protocol. When the USB controller is in suspend mode, the bit may be cleared by the processor core to perform a software disconnect.	
		0	Not Detected
		1	Detected Session

DMA Channel n Address Register

The USB_DMA[n]_ADDR register indicates the location in on-chip memory where DMA data is written or read. The address must be aligned to 32-bit words (The lower two address bits are always zero.) This register increments as the DMA transfer progresses.

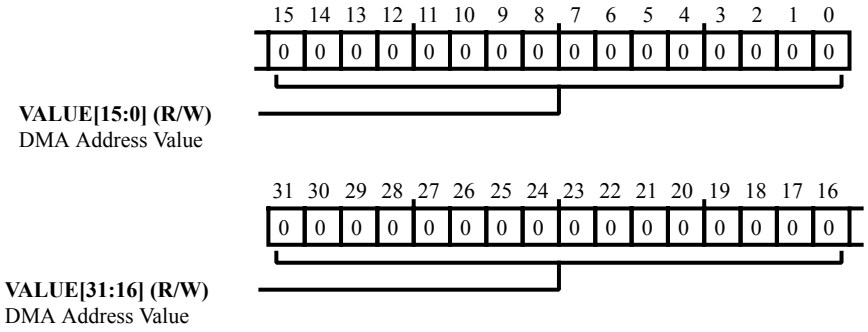

Figure 28-40: USB_DMA[n]_ADDR Register Diagram

Table 28-15: USB_DMA[n]_ADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Address Value. The USB_DMA[n]_ADDR.VALUE bits hold the address value for the location in on-chip memory where DMA data is written or read.

DMA Channel n Count Register

The USB_DMA[n]_CNT register holds the DMA count, indicating the number of bytes to be transferred for a given DMA work block. If this field is set to zero, no data is transferred, and an interrupt is generated.

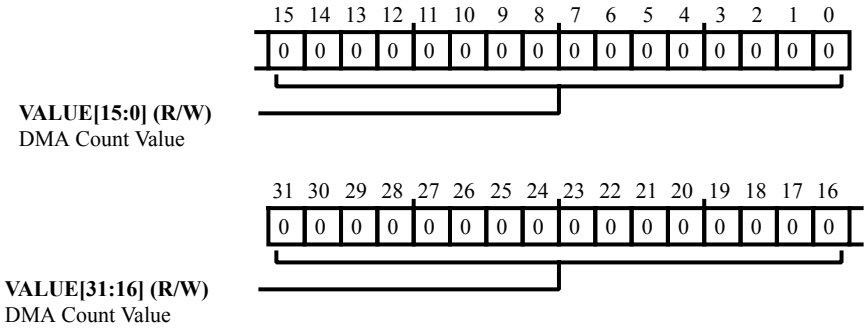

Figure 28-41: USB_DMA[n]_CNT Register Diagram

Table 28-16: USB_DMA[n]_CNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	DMA Count Value. The USB_DMA[n]_CNT.VALUE bits indicate the number of bytes to be transferred for a given DMA work block.

DMA Channel n Control Register

There is a USB_DMA[n]_CTL register for each DMA master channel. This register assigns, configures, and controls each endpoint with a corresponding DMA master channel.

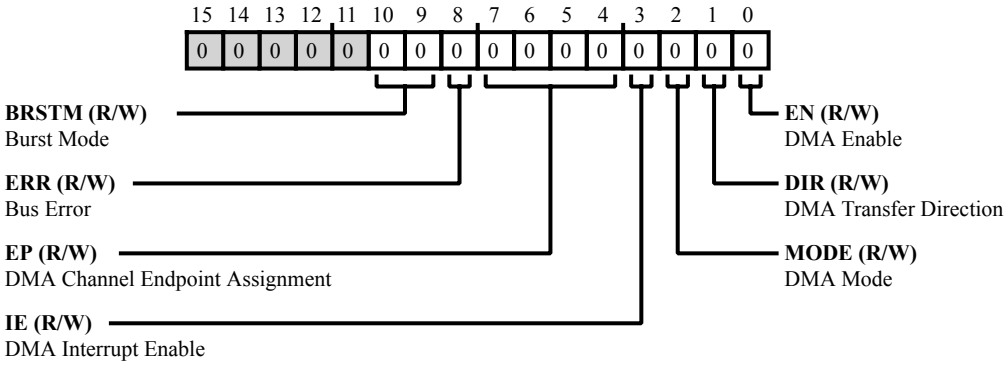

Figure 28-42: USB_DMA[n]_CTL Register Diagram

Table 28-17: USB_DMA[n]_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10:9 (R/W)	BRSTM	Burst Mode. The USB_DMA[n]_CTL.BRSTM bits select the type or length of burst transfer used by the corresponding DMA channel to transfer data.	
		0	Unspecified Length
		1	INCR4 or Unspecified Length
		2	INCR8, INCR4, or Unspecified Length
		3	INCR16, INCR8, INCR4, or Unspecified Length
8 (R/W)	ERR	Bus Error. The USB_DMA[n]_CTL.ERR bit indicates when a peripheral bus error has been encountered by the master channel. This bit is cleared by software.	
		0	No Status
		1	Bus Error
7:4 (R/W)	EP	DMA Channel Endpoint Assignment. The USB_DMA[n]_CTL.EP bits select the endpoint assignments for the DMA channel. (Enumeration values not shown are reserved.)	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3

Table 28-17: USB_DMA[n]_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15
3 (R/W)	IE	DMA Interrupt Enable. The USB_DMA[n]_CTL.IE bit enables DMA interrupts for the DMA channel, enabling operation of the channels corresponding bit in the USB_DMA_IRQ register.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	MODE	DMA Mode. The USB_DMA[n]_CTL.MODE bit selects whether the DMA channel operates in DMA mode 0 or operates in DMA mode 1. Note that DMA mode 1 may only be used with bulk endpoints.	
		0	DMA Mode 0
		1	DMA Mode 1
1 (R/W)	DIR	DMA Transfer Direction. The USB_DMA[n]_CTL.DIR bit selects the DMA channel transfer direction, which must be selected for use with receive endpoints (DMA write=0) or transmit endpoints (DMA read=1).	
		0	DMA Write (for Rx Endpoint)
		1	DMA Read (for Tx Endpoint)

Table 28-17: USB_DMA[n]_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	EN	DMA Enable. The `USB_DMA[n]_CTL.EN` bit enables the DMA channel to start the DMA transfer.	
		0	Disable DMA
		1	Enable DMA (Start Transfer)

DMA Interrupt Register

The USB_DMA_IRQ register indicates which of the DMA master channels have a pending interrupt. The USB controller generates the interrupt when the corresponding DMA count register (USB_DMA[n]_CNT) reaches zero. The USB controller clears this register when it is read.

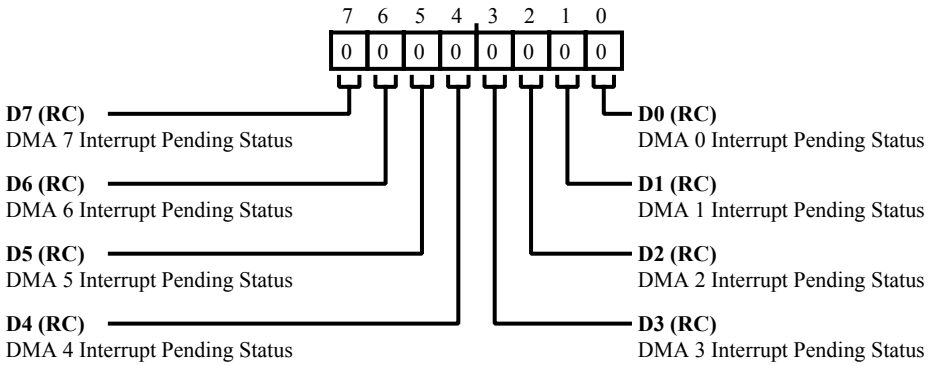

Figure 28-43: USB_DMA_IRQ Register Diagram

Table 28-18: USB_DMA_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	D7	DMA 7 Interrupt Pending Status. The USB_DMA_IRQ.D7 indicates whether there is a DMA 7 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
6 (RC/NW)	D6	DMA 6 Interrupt Pending Status. The USB_DMA_IRQ.D6 indicates whether there is a DMA 6 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
5 (RC/NW)	D5	DMA 5 Interrupt Pending Status. The USB_DMA_IRQ.D5 indicates whether there is a DMA 5 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
4 (RC/NW)	D4	DMA 4 Interrupt Pending Status. The USB_DMA_IRQ.D4 indicates whether there is a DMA 4 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt

Table 28-18: USB_DMA_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RC/NW)	D3	DMA 3 Interrupt Pending Status. The `USB_DMA_IRQ.D3` indicates whether there is a DMA 3 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
2 (RC/NW)	D2	DMA 2 Interrupt Pending Status. The `USB_DMA_IRQ.D2` indicates whether there is a DMA 2 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
1 (RC/NW)	D1	DMA 1 Interrupt Pending Status. The `USB_DMA_IRQ.D1` indicates whether there is a DMA 1 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt
0 (RC/NW)	D0	DMA 0 Interrupt Pending Status. The `USB_DMA_IRQ.D0` indicates whether there is a DMA 0 interrupt pending.	
		0	No Pending Interrupt
		1	Pending DMA Interrupt

EP0 Configuration Information Register

The USB_EP0I_CFGDATA[N] register describes the USB controller hardware configuration. This register only exists for endpoint 0.

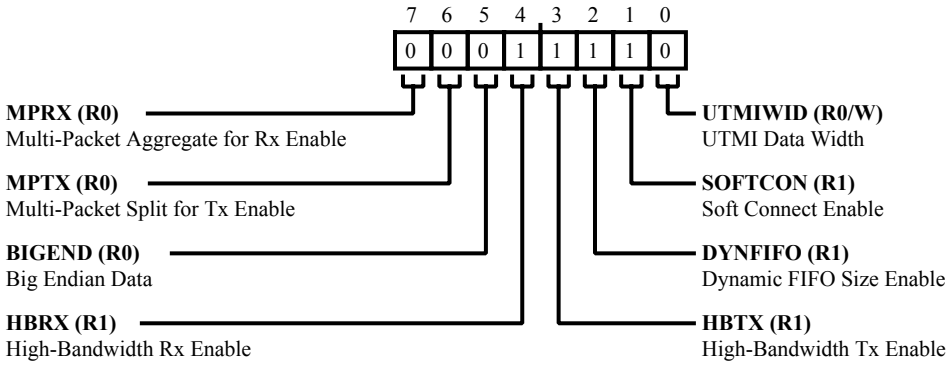

Figure 28-44: USB_EP0I_CFGDATA[N] Register Diagram

Table 28-19: USB_EP0I_CFGDATA[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R0/NW)	MPRX	Multi-Packet Aggregate for Rx Enable. The USB_EP0I_CFGDATA[N].MPRX bit indicates whether the USB controller aggregates receive packets into bulk packets before the processor core reads the data.	
		0	No Aggregate Rx Bulk Packets
		1	Aggregate Rx Bulk Packets
6 (R0/NW)	MPTX	Multi-Packet Split for Tx Enable. The USB_EP0I_CFGDATA[N].MPTX bit indicates whether the USB controller permits transmit of large packets through writing to bulk endpoints. The USB controller splits the transmit data into packets, which are appropriately sized for transmit.	
		0	No Split Tx Bulk Packets
		1	Split Tx Bulk Packets
5 (R0/NW)	BIGEND	Big Endian Data. The USB_EP0I_CFGDATA[N].BIGEND bit indicates whether the USB controller uses big endian configuration or little endian configuration.	
		0	Little Endian Configuration
		1	Big Endian Configuration

Table 28-19: USB_EP0I_CFGDATA[N] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R1/NW)	HBRX	High-Bandwidth Rx Enable. The `USB_EP0I_CFGDATA[N].HBRX` bit indicates whether the USB controller supports high-bandwidth receive ISO endpoint.	
		0	No High-Bandwidth Rx
		1	High-Bandwidth Rx
3 (R1/NW)	HBTX	High-Bandwidth Tx Enable. The `USB_EP0I_CFGDATA[N].HBTX` bit indicates whether the USB controller supports high-bandwidth transmit ISO endpoint.	
		0	No High-Bandwidth Tx
		1	High-Bandwidth Tx
2 (R1/NW)	DYNFIFO	Dynamic FIFO Size Enable. The `USB_EP0I_CFGDATA[N].DYNFIFO` bit indicates whether the USB controller uses dynamic FIFO size support (on products supporting this feature), enabling the dynamic FIFO registers. These registers are accessed using the configuration set in the endpoints indexed FIFO size and FIFO address registers, except for endpoint 0.	
		0	No Dynamic FIFO Size
		1	Dynamic FIFO Size
1 (R1/NW)	SOFTCON	Soft Connect Enable. The `USB_EP0I_CFGDATA[N].SOFTCON` bit indicates whether the USB controller uses soft connect.	
		0	No Soft Connect
		1	Soft Connect
0 (R0/W)	UTMIWID	UTMI Data Width. The `USB_EP0I_CFGDATA[N].UTMIWID` bit indicates whether the USB controller uses an 8-bit or 16-bit UTMI data width.	
		0	8-bit UTMI Data Width
		1	16-bit UTMI Data Width

EP0 Number of Received Bytes Register

The USB_EP0I_CNT[N] register indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change. It is only valid while the USB_EP0_CSR[n]_H.RXPKTRDY bit or USB_EP0_CSR[n]_P.RXPKTRDY bit is set.

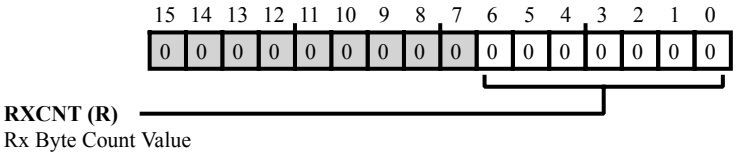

Figure 28-45: USB_EP0I_CNT[N] Register Diagram

Table 28-20: USB_EP0I_CNT[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/NW)	RXCNT	Rx Byte Count Value. The USB_EP0I_CNT[N].RXCNT bits holds the number of data bytes currently in line ready to be read from the Rx FIFO. The value returned changes as the FIFO is unloaded. It is only valid while the USB_EP0_CSR[n]_H.RXPKTRDY bit or USB_EP0_CSR[n]_P.RXPKTRDY bit is set.

EP0 Configuration and Status (Host) Register

The USB_EP0I_CSR[N]_H register provides control and status bits for endpoint 0 in host mode. Note that some bits may be set to clear automatically.

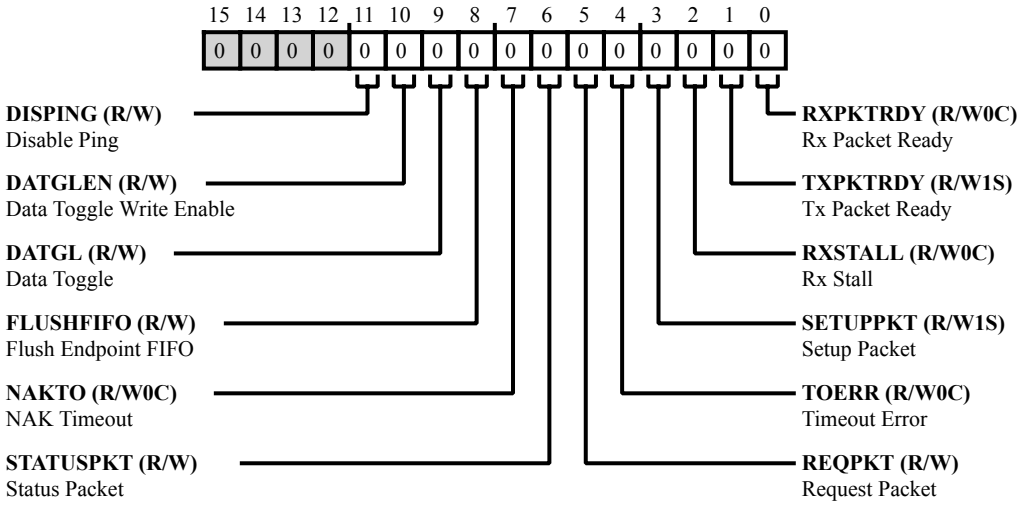

Figure 28-46: USB_EP0I_CSR[N]_H Register Diagram

Table 28-21: USB_EP0I_CSR[N]_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	DISPING	Disable Ping. The USB_EP0I_CSR[N]_H.DISPING bit disables (in host mode) high-speed PING tokens for the data and status phases of a control transfer.	
		0	Issue PING tokens
		1	Do not issue PING
10 (R/W)	DATGLEN	Data Toggle Write Enable. The USB_EP0I_CSR[N]_H.DATGLEN bit enables (in host mode) the USB controller to write the current state of the endpoint 0 USB_EP0I_CSR[N]_H.DATGL bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL

Table 28-21: USB_EP0I_CSR[N]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	DATGL	Data Toggle. The `USB_EP0I_CSR[N]_H.DATGL` bit indicates (in host mode) the current state of the endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP0I_CSR[N]_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint 0 FIFO and clear the `USB_EP0I_CSR[N]_H.TXPKTRDY` and `USB_EP0I_CSR[N]_H.RXPKTRDY` bits. The `USB_EP0I_CSR[N]_H.FLUSHFIFO` bit should only be set if the `USB_EP0I_CSR[N]_H.TXPKTRDY` and `USB_EP0I_CSR[N]_H.RXPKTRDY` bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W0C)	NAKTO	NAK Timeout. The `USB_EP0I_CSR[N]_H.NAKTO` bit indicates (in host mode) when endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the `USB_EP0_NAKLIMIT[n]` register. The processor core should clear this bit to allow the endpoint to continue.	
		0	No Status
		1	Endpoint Halted (NAK Timeout)
6 (R/W)	STATUSPKT	Status Packet. The `USB_EP0I_CSR[N]_H.STATUSPKT` bit directs (in host mode) the USB controller to perform a status stage transaction. This bit is set at the same time as the `USB_EP0I_CSR[N]_H.TXPKTRDY` and `USB_EP0I_CSR[N]_H.RXPKTRDY` bits. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the status stage transaction.	
		0	No Request
		1	Request Status Transaction

Table 28-21: USB_EP0I_CSR[N]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	REQPKT	Request Packet. The `USB_EP0I_CSR[N]_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when the `USB_EP0I_CSR[N]_H.RXPKTRDY` bit is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W0C)	TOERR	Timeout Error. The `USB_EP0I_CSR[N]_H.TOERR` bit indicates (in host mode) when three attempts have been made to perform a transaction with no response from the peripheral. The processor core should clear this bit. An interrupt is generated when this bit is set.	
		0	No Status
		1	Timeout Error
3 (R/W1S)	SETUPPKT	Setup Packet. The `USB_EP0I_CSR[N]_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP0I_CSR[N]_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP token
2 (R/W0C)	RXSTALL	Rx Stall. The `USB_EP0I_CSR[N]_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0I_CSR[N]_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

Table 28-21: USB_EP0I_CSR[N]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0I_CSR[N]_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 Configuration and Status (Peripheral) Register

The USB_EP0I_CSR[N]_P register provides control and status bits for endpoint 0 in peripheral mode. Note that some bits may be set to clear automatically.

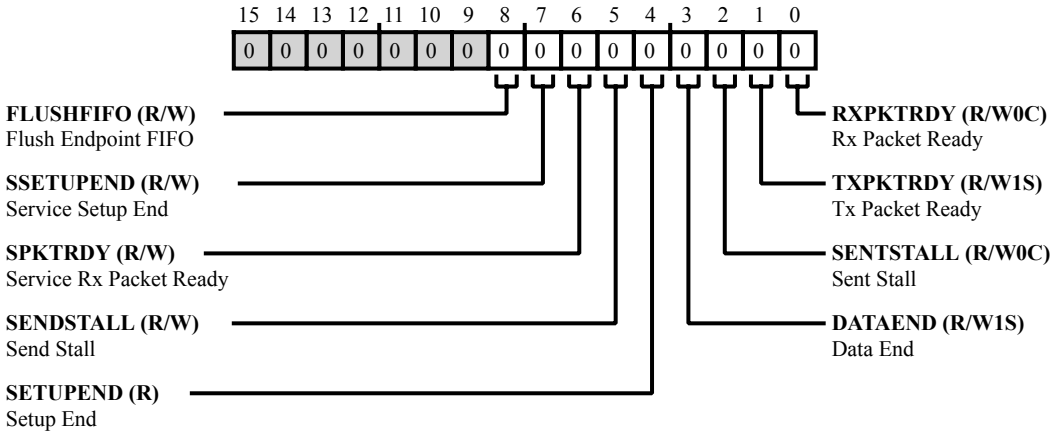

Figure 28-47: USB_EP0I_CSR[N]_P Register Diagram

Table 28-22: USB_EP0I_CSR[N]_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP0I_CSR[N]_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint 0 FIFO and clear the USB_EP0I_CSR[N]_P.TXPKTRDY and USB_EP0I_CSR[N]_P.RXPKTRDY bits. The USB_EP0I_CSR[N]_P.FLUSHFIFO bit should only be set if the USB_EP0I_CSR[N]_P.TXPKTRDY and USB_EP0I_CSR[N]_P.RXPKTRDY bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W)	SSETUPEND	Service Setup End. The USB_EP0I_CSR[N]_P.SSETUPEND bit is set (in peripheral mode) by the processor core to clear the USB_EP0I_CSR[N]_P.SETUPEND. This bit is cleared automatically.	
		0	No Action
		1	Clear SETUPEND Bit

Table 28-22: USB_EP0I_CSR[N]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SPKTRDY	Service Rx Packet Ready. The `USB_EP0I_CSR[N]_P.SPKTRDY` bit is set (in peripheral mode) by the processor core to clear the `USB_EP0I_CSR[N]_P.RXPKTRDY` bit. This bit is cleared automatically.	
		0	No Action
		1	Clear RXPKTRDY Bit
5 (R/W)	SENDSTALL	Send Stall. The `USB_EP0I_CSR[N]_P.SENDSTALL` bit is set (in peripheral mode) by the processor core to terminate the current transaction. The STALL handshake is transmitted, then this bit automatically is cleared.	
		0	No Action
		1	Terminate Current Transaction
4 (R/NW)	SETUPEND	Setup End. The `USB_EP0I_CSR[N]_P.SETUPEND` bit indicates (in peripheral mode) when a control transaction ends before the `USB_EP0I_CSR[N]_P.DATAEND` bit is set. An interrupt is generated and the FIFO is flushed at this time. This bit is cleared when the processor core sets the `USB_EP0I_CSR[N]_P.SSETUPEND` bit.	
		0	No Status
		1	Setup Ended before DATAEND
3 (R/W1S)	DATAEND	Data End. The `USB_EP0I_CSR[N]_P.DATAEND` bit is set (in peripheral mode) by the processor core sets when the core: • Sets the `USB_EP0I_CSR[N]_P.TXPKTRDY` bit for the last data packet. • Clears the `USB_EP0I_CSR[N]_P.RXPKTRDY` bit after unloading the last data packet. • Sets the `USB_EP0I_CSR[N]_P.TXPKTRDY` bit for a zero-length data packet. The `USB_EP0I_CSR[N]_P.DATAEND` bit is cleared automatically.	
		0	No Status
		1	Data End Condition
2 (R/W0C)	SENTSTALL	Sent Stall. The `USB_EP0I_CSR[N]_P.SENTSTALL` bit is set (in peripheral mode) when a STALL handshake is transmitted. The processor core should clear this bit.	
		0	No Status
		1	Transmitted STALL Handshake

Table 28-22: USB_EP0I_CSR[N]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0I_CSR[N]_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	
		1	Set this bit after loading a data packet into the FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0I_CSR[N]_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core clears this bit by setting the `USB_EP0I_CSR[N]_P.SPKTRDY` bit.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 NAK Limit Register

The USB_EP0I_NAKLIMIT[N] register determines the number of frames/micro-frames after which endpoint 0 should timeout on receiving a stream of NAK responses for bulk endpoints.

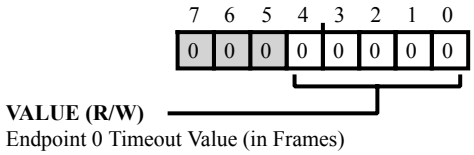

Figure 28-48: USB_EP0I_NAKLIMIT[N] Register Diagram

Table 28-23: USB_EP0I_NAKLIMIT[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Endpoint 0 Timeout Value (in Frames). The USB_EP0I_NAKLIMIT[N].VALUE bits hold the endpoint 0 timeout value (number of frames).

EP0 Connection Type Register

The `USB_EP0I_TYPE[N]` register selects the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub.

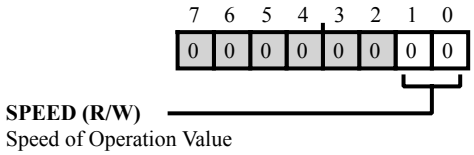

Figure 28-49: USB_EP0I_TYPE[N] Register Diagram

Table 28-24: USB_EP0I_TYPE[N] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	SPEED	Speed of Operation Value. The `USB_EP0I_TYPE[N].SPEED` bits select the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as Processor Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed

EP0 Configuration Information Register

The USB_EP0_CFGDATA[n] register describes the USB controller hardware configuration. This register only exists for endpoint 0.

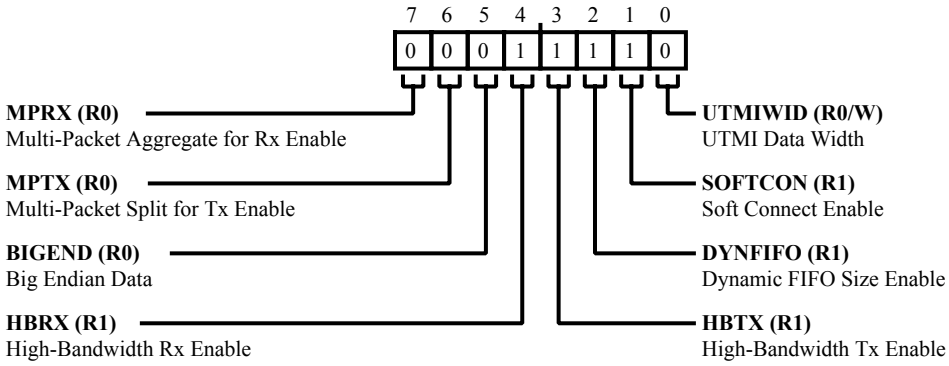

Figure 28-50: USB_EP0_CFGDATA[n] Register Diagram

Table 28-25: USB_EP0_CFGDATA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R0/NW)	MPRX	Multi-Packet Aggregate for Rx Enable. The USB_EP0_CFGDATA[n].MPRX bit indicates whether the USB controller aggregates receive packets into bulk packets before the processor core reads the data.	
		0	No Aggregate Rx Bulk Packets
		1	Aggregate Rx Bulk Packets
6 (R0/NW)	MPTX	Multi-Packet Split for Tx Enable. The USB_EP0_CFGDATA[n].MPTX bit indicates whether the USB controller permits transmit of large packets through writing to bulk endpoints. The USB controller splits the transmit data into packets, which are appropriately sized for transmit.	
		0	No Split Tx Bulk Packets
		1	Split Tx Bulk Packets
5 (R0/NW)	BIGEND	Big Endian Data. The USB_EP0_CFGDATA[n].BIGEND bit indicates whether the USB controller uses big endian configuration or little endian configuration.	
		0	Little Endian Configuration
		1	Big Endian Configuration

Table 28-25: USB_EP0_CFGDATA[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R1/NW)	HBRX	High-Bandwidth Rx Enable. The `USB_EP0_CFGDATA[n].HBRX` bit indicates whether the USB controller supports high-bandwidth receive ISO endpoint.	
		0	No High-Bandwidth Rx
		1	High-Bandwidth Rx
3 (R1/NW)	HBTX	High-Bandwidth Tx Enable. The `USB_EP0_CFGDATA[n].HBTX` bit indicates whether the USB controller supports high-bandwidth transmit ISO endpoint.	
		0	No High-Bandwidth Tx
		1	High-Bandwidth Tx
2 (R1/NW)	DYNFIFO	Dynamic FIFO Size Enable. The `USB_EP0_CFGDATA[n].DYNFIFO` bit indicates whether the USB controller uses dynamic FIFO size support (on products supporting this feature), enabling the dynamic FIFO registers. These registers are accessed using the configuration set in the endpoints indexed FIFO size and FIFO address registers, except for endpoint 0.	
		0	No Dynamic FIFO Size
		1	Dynamic FIFO Size
1 (R1/NW)	SOFTCON	Soft Connect Enable. The `USB_EP0_CFGDATA[n].SOFTCON` bit indicates whether the USB controller uses soft connect.	
		0	No Soft Connect
		1	Soft Connect
0 (R0/W)	UTMIWID	UTMI Data Width. The `USB_EP0_CFGDATA[n].UTMIWID` bit indicates whether the USB controller uses an 8-bit or 16-bit UTMI data width.	
		0	8-bit UTMI Data Width
		1	16-bit UTMI Data Width

EP0 Number of Received Bytes Register

The USB_EP0_CNT[n] register indicates the number of received data bytes in the endpoint 0 FIFO. The value returned changes as the contents of the FIFO change. It is only valid while the USB_EP0_CSR[n]_H.RXPKTRDY bit or USB_EP0_CSR[n]_P.RXPKTRDY bit is set.

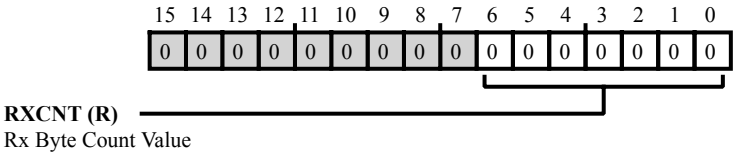

Figure 28-51: USB_EP0_CNT[n] Register Diagram

Table 28-26: USB_EP0_CNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/NW)	RXCNT	Rx Byte Count Value. The USB_EP0_CNT[n].RXCNT bits holds the number of data bytes currently in line ready to be read from the Rx FIFO. The value returned changes as the FIFO is unloaded. It is only valid while the USB_EP0_CSR[n]_H.RXPKTRDY bit or USB_EP0_CSR[n]_P.RXPKTRDY bit is set.

EP0 Configuration and Status (Host) Register

The USB_EP0_CSR[n]_H register provides control and status bits for endpoint 0 in host mode. Note that some bits may be set to clear automatically.

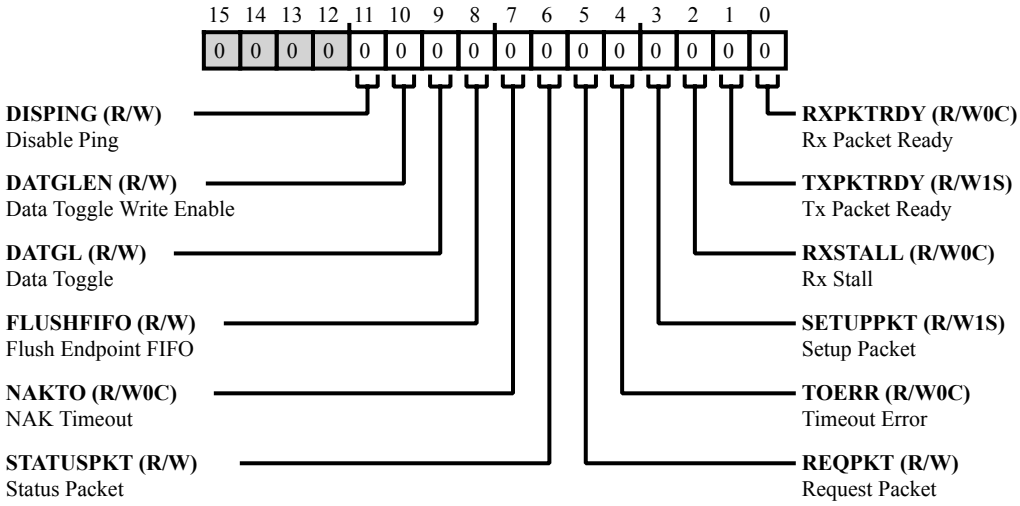

Figure 28-52: USB_EP0_CSR[n]_H Register Diagram

Table 28-27: USB_EP0_CSR[n]_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	DISPING	Disable Ping. The USB_EP0_CSR[n]_H.DISPING bit disables (in host mode) high-speed PING tokens for the data and status phases of a control transfer.	
		0	Issue PING tokens
		1	Do not issue PING
10 (R/W)	DATGLEN	Data Toggle Write Enable. The USB_EP0_CSR[n]_H.DATGLEN bit enables (in host mode) the USB controller to write the current state of the endpoint 0 USB_EP0_CSR[n]_H.DATGL bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL

Table 28-27: USB_EP0_CSR[n]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	DATGL	Data Toggle. The USB_EP0_CSR[n]_H.DATGL bit indicates (in host mode) the current state of the endpoint 0 data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP0_CSR[n]_H.FLUSHFIFO bit directs (in host mode) the USB controller to flush data from the endpoint 0 FIFO and clear the USB_EP0_CSR[n]_H.TXPKTRDY and USB_EP0_CSR[n]_H.RXPKTRDY bits. The USB_EP0_CSR[n]_H.FLUSHFIFO bit should only be set if the USB_EP0_CSR[n]_H.TXPKTRDY and USB_EP0_CSR[n]_H.RXPKTRDY bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W0C)	NAKTO	NAK Timeout. The USB_EP0_CSR[n]_H.NAKTO bit indicates (in host mode) when endpoint 0 is halted following the receipt of NAK responses for longer than the time set by the USB_EP0_NAKLIMIT[n] register. The processor core should clear this bit to allow the endpoint to continue.	
		0	No Status
		1	Endpoint Halted (NAK Timeout)
6 (R/W)	STATUSPKT	Status Packet. The USB_EP0_CSR[n]_H.STATUSPKT bit directs (in host mode) the USB controller to perform a status stage transaction. This bit is set at the same time as the USB_EP0_CSR[n]_H.TXPKTRDY and USB_EP0_CSR[n]_H.RXPKTRDY bits. Setting this bit ensures that the data toggle is set to 1 so that a DATA1 packet is used for the status stage transaction.	
		0	No Request
		1	Request Status Transaction

Table 28-27: USB_EP0_CSR[n]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	REQPKT	Request Packet. The `USB_EP0_CSR[n]_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when the `USB_EP0_CSR[n]_H.RXPKTRDY` bit is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W0C)	TOERR	Timeout Error. The `USB_EP0_CSR[n]_H.TOERR` bit indicates (in host mode) when three attempts have been made to perform a transaction with no response from the peripheral. The processor core should clear this bit. An interrupt is generated when this bit is set.	
		0	No Status
		1	Timeout Error
3 (R/W1S)	SETUPPKT	Setup Packet. The `USB_EP0_CSR[n]_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP0_CSR[n]_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP token
2 (R/W0C)	RXSTALL	Rx Stall. The `USB_EP0_CSR[n]_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0_CSR[n]_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

Table 28-27: USB_EP0_CSR[n]_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0_CSR[n]_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 Configuration and Status (Peripheral) Register

The USB_EP0_CSR[n]_P register provides control and status bits for endpoint 0 in peripheral mode. Note that some bits may be set to clear automatically.

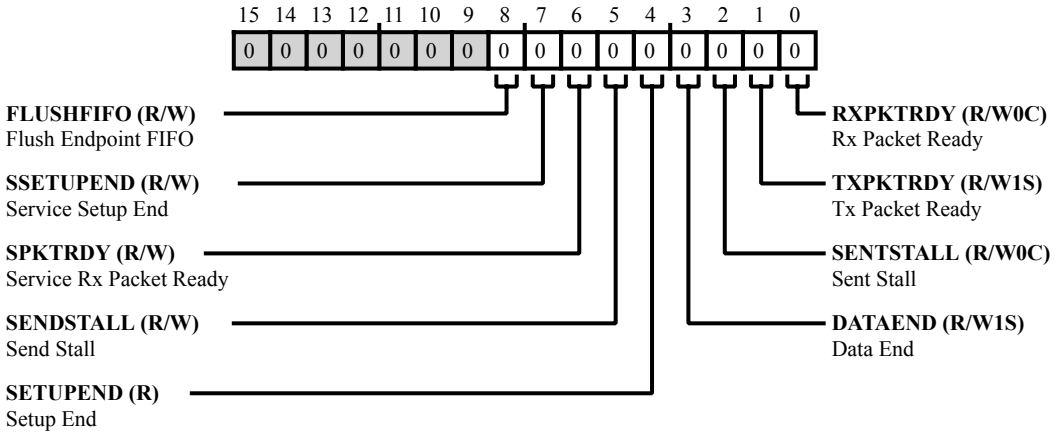

Figure 28-53: USB_EP0_CSR[n]_P Register Diagram

Table 28-28: USB_EP0_CSR[n]_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP0_CSR[n]_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint 0 FIFO and clear the USB_EP0_CSR[n]_P.TXPKTRDY and USB_EP0_CSR[n]_P.RXPKTRDY bits. The USB_EP0_CSR[n]_P.FLUSHFIFO bit should only be set if the USB_EP0_CSR[n]_P.TXPKTRDY and USB_EP0_CSR[n]_P.RXPKTRDY bits are set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
7 (R/W)	SSETUPEND	Service Setup End. The USB_EP0_CSR[n]_P.SSETUPEND bit is set (in peripheral mode) by the processor core to clear the USB_EP0_CSR[n]_P.SETUPEND. This bit is cleared automatically.	
		0	No Action
		1	Clear SETUPEND Bit

Table 28-28: USB_EP0_CSR[n]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	SPKTRDY	Service Rx Packet Ready. The USB_EP0_CSR[n]_P.SPKTRDY bit is set (in peripheral mode) by the processor core to clear the USB_EP0_CSR[n]_P.RXPKTRDY bit. This bit is cleared automatically.	
		0	No Action
		1	Clear RXPKTRDY Bit
5 (R/W)	SENDSTALL	Send Stall. The USB_EP0_CSR[n]_P.SENDSTALL bit is set (in peripheral mode) by the processor core to terminate the current transaction. The STALL handshake is transmitted, then this bit automatically is cleared.	
		0	No Action
		1	Terminate Current Transaction
4 (R/NW)	SETUPEND	Setup End. The USB_EP0_CSR[n]_P.SETUPEND bit indicates (in peripheral mode) when a control transaction ends before the USB_EP0_CSR[n]_P.DATAEND bit is set. An interrupt is generated and the FIFO is flushed at this time. This bit is cleared when the processor core sets the USB_EP0_CSR[n]_P.SSETUPEND bit.	
		0	No Status
		1	Setup Ended before DATAEND
3 (R/W1S)	DATAEND	Data End. The USB_EP0_CSR[n]_P.DATAEND bit is set (in peripheral mode) by the processor core sets when the core: • Sets the USB_EP0_CSR[n]_P.TXPKTRDY bit for the last data packet. • Clears the USB_EP0_CSR[n]_P.RXPKTRDY bit after unloading the last data packet. • Sets the USB_EP0_CSR[n]_P.TXPKTRDY bit for a zero-length data packet. The USB_EP0_CSR[n]_P.DATAEND bit is cleared automatically.	
		0	No Status
		1	Data End Condition
2 (R/W0C)	SENTSTALL	Sent Stall. The USB_EP0_CSR[n]_P.SENTSTALL bit is set (in peripheral mode) when a STALL handshake is transmitted. The processor core should clear this bit.	
		0	No Status
		1	Transmitted STALL Handshake

Table 28-28: USB_EP0_CSR[n]_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP0_CSR[n]_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	
		1	Set this bit after loading a data packet into the FIFO
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP0_CSR[n]_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core clears this bit by setting the `USB_EP0_CSR[n]_P.SPKTRDY` bit.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EP0 NAK Limit Register

The USB_EP0_NAKLIMIT[n] register determines the number of frames/micro-frames after which endpoint 0 should timeout on receiving a stream of NAK responses for bulk endpoints.

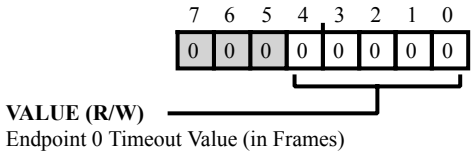

Figure 28-54: USB_EP0_NAKLIMIT[n] Register Diagram

Table 28-29: USB_EP0_NAKLIMIT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4:0 (R/W)	VALUE	Endpoint 0 Timeout Value (in Frames). The USB_EP0_NAKLIMIT[n].VALUE bits hold the endpoint 0 timeout value (number of frames).

EP0 Connection Type Register

The USB_EP0_TYPE[n] register selects the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub.

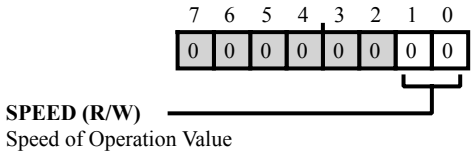

Figure 28-55: USB_EP0_TYPE[n] Register Diagram

Table 28-30: USB_EP0_TYPE[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (R/W)	SPEED	Speed of Operation Value. The USB_EP0_TYPE[n].SPEED bits select the USB controller operating speed for endpoint 0 when acting as a host connected to devices through a hub. In these instances. the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as Processor Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed

Endpoint Information Register

The USB_EPINFO register allows read-back of the number of Tx and Rx endpoints available

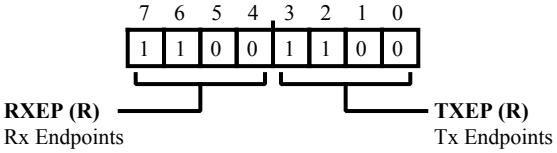

Figure 28-56: USB_EPINFO Register Diagram

Table 28-31: USB_EPINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	RXEP	Rx Endpoints. The USB_EPINFO.RXEP bits indicate the number of receive endpoints. excluding EP0.
3:0 (R/NW)	TXEP	Tx Endpoints. The USB_EPINFO.TXEP bits indicate the number of transmit endpoints, excluding EP0.

EPn Number of Bytes Received Register

The USB_EPI[N]_RXCNT register indicates the number of received data bytes in the endpoint receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while the USB_EP[n]_RXCSR_H.RXPKTRDY bit or USB_EP[n]_RXCSR_P.RXPKTRDY bit is set.

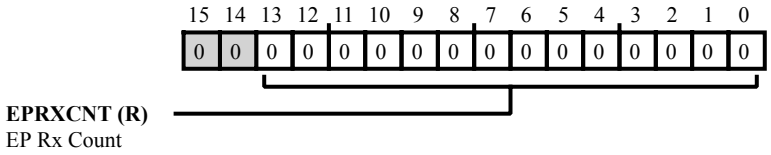

Figure 28-57: USB_EPI[N]_RXCNT Register Diagram

Table 28-32: USB_EPI[N]_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/NW)	EPRXCNT	EP Rx Count. The USB_EPI[N]_RXCNT.EPRXCNT bits hold the number of data bytes ready to be read from the receive FIFO.

EPn Receive Configuration and Status (Host) Register

The USB_EPI[N]_RXCSR_H register provides (in host mode) control and status bits for transfers through the currently selected receive endpoint.

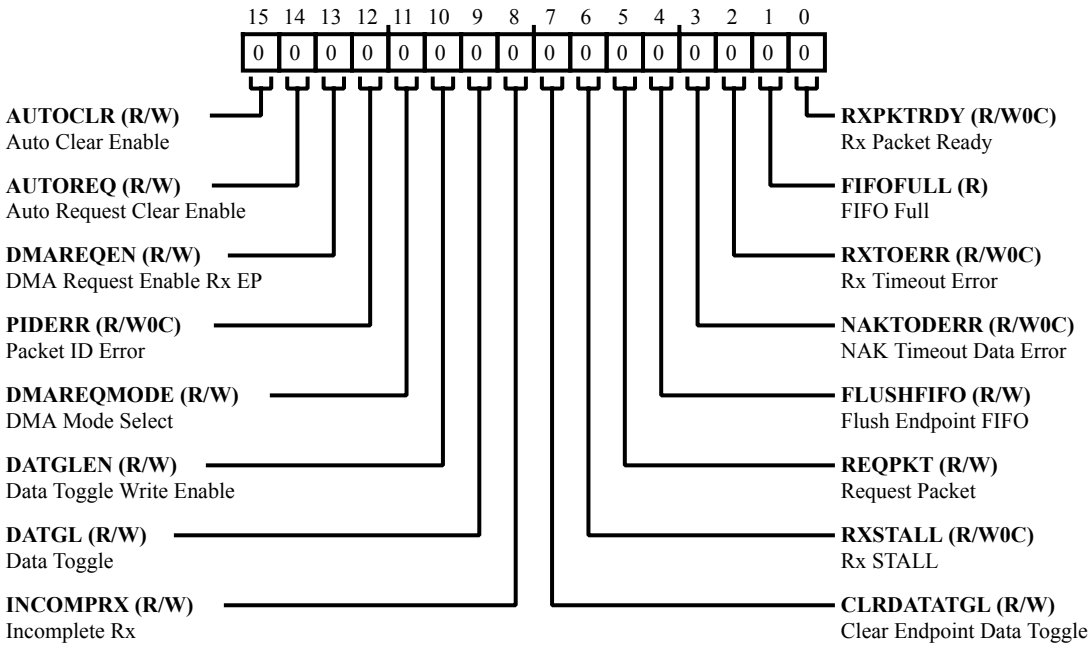

Figure 28-58: USB_EPI[N]_RXCSR_H Register Diagram

Table 28-33: USB_EPI[N]_RXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EPI[N]_RXCSR_H.AUTOCLR bit directs (in host mode) the USB controller to automatically clear the USB_EPI[N]_RXCSR_H.RXPKTRDY bit when a packet of size USB_EP[n]_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EPI[N]_RXCSR_H.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EP[n]_RXMAXP value. The USB controller auto clears the USB_EPI[N]_RXCSR_H.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the USB_EPI[N]_RXCSR_H.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	AUTOREQ	Auto Request Clear Enable. The USB_EPI[N]_RXCSR_H.AUTOREQ bit directs (in host mode) the USB controller to automatically clear the USB_EPI[N]_RXCSR_H.REQPKT bit when USB_EPI[N]_RXCSR_H.RXPKTRDY bit is cleared. This bit is automatically cleared when a short packet is received.	
		0	Disable Auto Request Clear
		1	Enable Auto Request Clear
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EPI[N]_RXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 28-33: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W0C)	PIDERR	Packet ID Error. The `USB_EPI[N]_RXCSR_H.PIDERR` bit indicates (in host mode) when a PID error occurs for isochronous transactions. This bit is ignored in host mode for bulk or interrupt transactions.	
		0	No Status
		1	PID Error
11 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_RXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EPI[N]_RXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
10 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EPI[N]_RXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EPI[N]_RXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The `USB_EPI[N]_RXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	INCOMPRX	Incomplete Rx. The `USB_EPI[N]_RXCSR_H.INCOMPRX` bit indicates (in host mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when `USB_EPI[N]_RXCSR_H.RXPKTRDY` is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx

Table 28-33: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_RXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	RXSTALL	Rx STALL. The `USB_EPI[N]_RXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
5 (R/W)	REQPKT	Request Packet. The `USB_EPI[N]_RXCSR_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when `USB_EPI[N]_RXCSR_H.RXPKTRDY` is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EPI[N]_RXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit. The `USB_EPI[N]_RXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EPI[N]_RXCSR_H.RXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 28-33: USB_EPI[N]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W0C)	NAKTODERR	NAK Timeout Data Error. The USB_EPI[N]_RXCSR_H.NAKTODERR bit indicates (in host mode for isochronous transfers) a NAK timeout data error when the USB_EPI[N]_RXCSR_H.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when the USB_EPI[N]_RXCSR_H.RXPKTRDY bit is cleared. The USB_EPI[N]_RXCSR_H.NAKTODERR bit indicates (in host mod for bulk transfers) when a receive endpoint is halted following the receipt of NAK responses greater than the limit set in the USB_EP[n]_RXINTERVAL register. The processor should clear this bit to allow the endpoint to continue. If double packet buffering is enabled, the USB_EPI[N]_RXCSR_H.REQPKT bit should also be set in the same cycle as this bit is cleared.	
		0	No Status
		1	NAK Timeout Data Error
2 (R/W0C)	RXTOERR	Rx Timeout Error. The USB_EPI[N]_RXCSR_H.RXTOERR bit indicates (in host mode) when three attempts have been made to receive a packet and no data packet has been received. The USB controller generates an interrupt for this condition. The processor should clear this bit. Note that USB_EPI[N]_RXCSR_H.RXTOERR is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Rx Timeout Error
1 (R/NW)	FIFOFULL	FIFO Full. The USB_EPI[N]_RXCSR_H.FIFOFULL bit indicates (in host mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The USB_EPI[N]_RXCSR_H.RXPKTRDY is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Configuration and Status (Peripheral) Register

The USB_EPI[N]_RXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected receive endpoint.

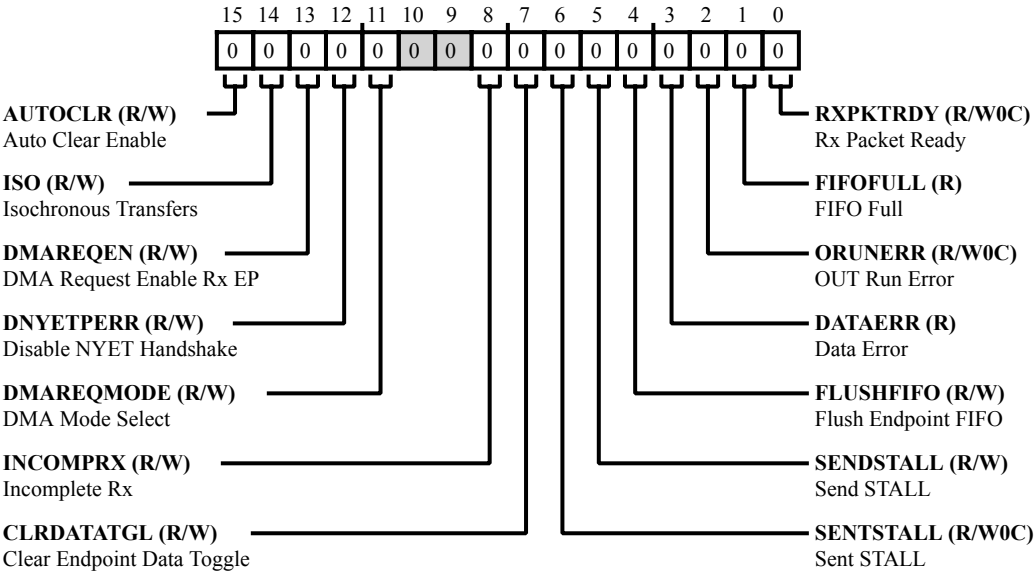

Figure 28-59: USB_EPI[N]_RXCSR_P Register Diagram

Table 28-34: USB_EPI[N]_RXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EPI[N]_RXCSR_P.AUTOCLR bit directs (in peripheral mode) the USB controller to automatically clear the USB_EPI[N]_RXCSR_P.RXPKTRDY bit when a packet of size USB_EP[n]_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EPI[N]_RXCSR_P.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EP[n]_RXMAXP value. The USB controller auto clears the USB_EPI[N]_RXCSR_P.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the USB_EPI[N]_RXCSR_P.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	ISO	Isochronous Transfers. The USB_EPI[N]_RXCSR_P.ISO bit selects (in peripheral mode) between isochronous transfers and bulk/interrupt transfers.	
		0	This bit should be cleared for bulk or interrupt transfers.
		1	This bit should be set for isochronous transfers.
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EPI[N]_RXCSR_P.DMAREQEN bit enables (in peripheral mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 28-34: USB_EPI[N]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	DNYETPERR	Disable NYET Handshake. The `USB_EPI[N]_RXCSR_P.DNYETPERR` bit disables (in peripheral mode for high speed bulk/interrupt transactions) NYET handshakes. When this bit is set, all successful receive packets are ACK'd, including the point at which the FIFO becomes full. The `USB_EPI[N]_RXCSR_P.DNYETPERR` bit must be set for all interrupt endpoints in high speed mode.	
		0	Enable NYET Handshake
		1	Disable NYET Handshake
11 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_RXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the `USB_EPI[N]_RXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
8 (R/W)	INCOMPRX	Incomplete Rx. The `USB_EPI[N]_RXCSR_P.INCOMPRX` bit indicates (in peripheral mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when `USB_EPI[N]_RXCSR_P.RXPKTRDY` is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_RXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EPI[N]_RXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when a STALL handshake is transmitted. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted

Table 28-34: USB_EPI[N]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SENDSTALL	Send STALL. The USB_EPI[N]_RXCSR_P.SENDSTALL bit is set (in peripheral mode) by the processor to send a STALL handshake. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Action
		1	Request STALL Handshake
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPI[N]_RXCSR_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPI[N]_RXCSR_P.RXPKTRDY bit. The USB_EPI[N]_RXCSR_P.FLUSHFIFO bit should only be set if the USB_EPI[N]_RXCSR_P.RXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
3 (R/NW)	DATAERR	Data Error. The USB_EPI[N]_RXCSR_P.DATAERR bit indicates (in peripheral mode for isochronous transfers) when the USB_EPI[N]_RXCSR_P.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when USB_EPI[N]_RXCSR_P.RXPKTRDY is cleared. The USB_EPI[N]_RXCSR_P.DATAERR bit is always zero for bulk endpoints in peripheral mode.	
		0	No Status
		1	Data Error
2 (R/W0C)	ORUNERR	OUT Run Error. The USB_EPI[N]_RXCSR_P.ORUNERR bit indicates (in peripheral mode for isochronous transfers) when an OUT packet cannot be loaded into the receive FIFO. The processor should clear this bit. The USB_EPI[N]_RXCSR_P.ORUNERR bit always returns zero in bulk mode.	
		0	No Status
		1	OUT Run Error
1 (R/NW)	FIFOFULL	FIFO Full. The USB_EPI[N]_RXCSR_P.FIFOFULL bit indicates (in peripheral mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full

Table 28-34: USB_EPI[N]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EPI[N]_RXCSR_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Polling Interval Register

The USB_EPI[N]_RXINTERVAL register defines the polling interval for the currently-selected receive endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EPI[N]_RXINTERVAL register for each configured receive endpoint, except endpoint 0. The transfer types related to speed, interval value, and interval operation are as follows:

- Interrupt: Speed = low-speed or full -speed, USB_EPI[N]_RXINTERVAL = 1-255, and Operation = polling interval is m frames.
- Interrupt: Speed = high-speed, USB_EPI[N]_RXINTERVAL = 1-16, and Operation = polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed = full-speed or high-speed, USB_EPI[N]_RXINTERVAL = 1-16, and Operation = polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed = full-speed or high-speed, USB_EPI[N]_RXINTERVAL = 2-16, and Operation = NAK limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EPI[N]_RXINTERVAL value of 0 or 1 disables the NAK timeout function.

Not all products support high-speed operation or micro-frames. These features do not apply for products that only support low/full-speed operation.

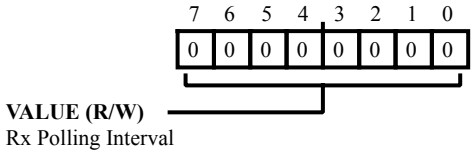

Figure 28-60: USB_EPI[N]_RXINTERVAL Register Diagram

Table 28-35: USB_EPI[N]_RXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Rx Polling Interval. The USB_EPI[N]_RXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers and select the number of frames (or microframes, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Receive Maximum Packet Length Register

The USB_EPI[N]_RXMAXP register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame.

Note that a value greater than the maximum allowed of 1023 for full-speed USB operation produces unpredictable results. Also, note that the total amount of data represented by the value written to this register must not exceed the receive FIFO size, and should not exceed half the FIFO size if double-buffering is required.

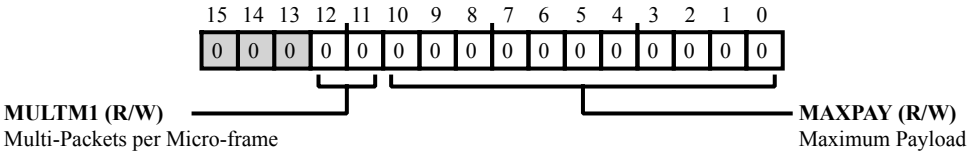

Figure 28-61: USB_EPI[N]_RXMAXP Register Diagram

Table 28-36: USB_EPI[N]_RXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-frame. The USB_EPI[N]_RXMAXP.MULTM1 bits select the number of high-speed, high-bandwidth isochronous or interrupt packets that may be transferred in a micro-frame. The valid number of packets per micro-frame is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller combines multiple packets received within a micro-frame into a single packet in the FIFO.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EPI[N]_RXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024, but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EPI[N]_RXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Receive Type Register

The USB_EPI[N]_RXTYPE register selects the endpoint number and transaction protocol to use for the currently selected receive endpoint. There is a USB_EPI[N]_RXTYPE register for each receive endpoint. Note that this register is only used in host mode.

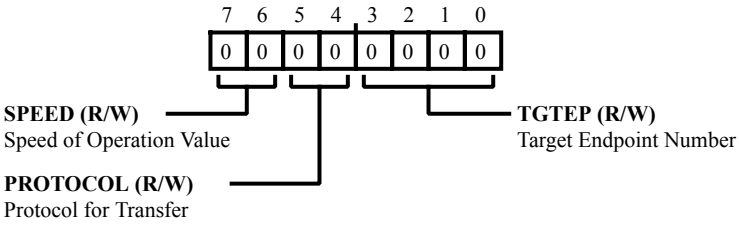

Figure 28-62: USB_EPI[N]_RXTYPE Register Diagram

Table 28-37: USB_EPI[N]_RXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EPI[N]_RXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances, the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When it is not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EPI[N]_RXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EPI[N]_RXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the receive endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 28-37: USB_EPI[N]_RXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Transmit Configuration and Status (Host) Register

The USB_EPI[N]_TXCSR_H register provides (in host mode) control and status bits for transfers through the currently-selected transmit endpoint.

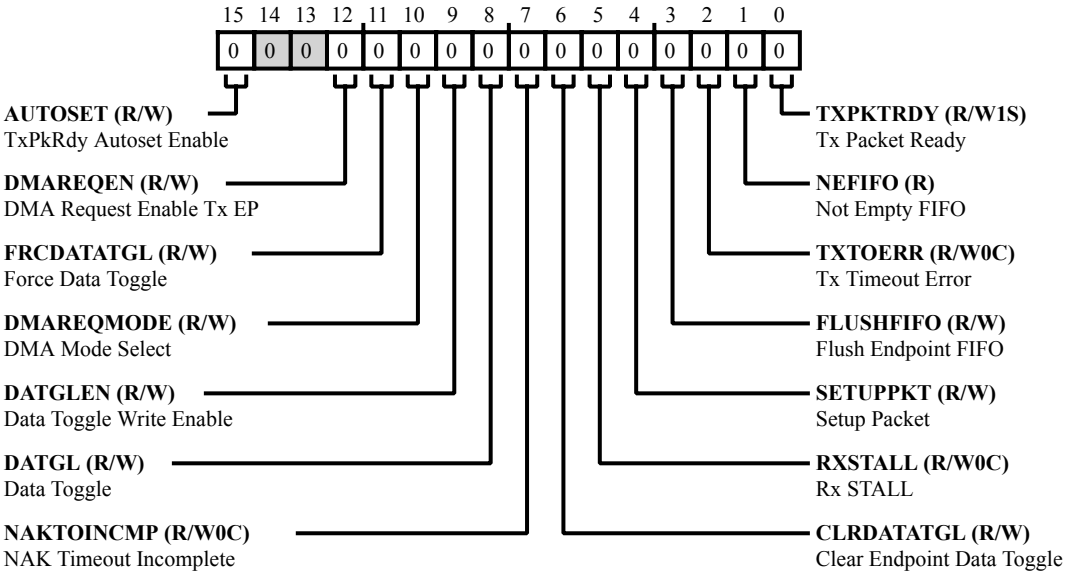

Figure 28-63: USB_EPI[N]_TXCSR_H Register Diagram

Table 28-38: USB_EPI[N]_TXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EPI[N]_TXCSR_H.AUTOSET bit enables (in host mode) the automatic setting of the USB_EPI[N]_TXCSR_H.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EPI[N]_TXCSR_H.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EPI[N]_TXCSR_H.AUTOSET bit should not be set for high-bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The USB_EPI[N]_TXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 28-38: USB_EPI[N]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EPI[N]_TXCSR_H.FRCDATATGL` bit forces (in host mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_TXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared during the cycle before or the same cycle that the `USB_EPI[N]_TXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
9 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EPI[N]_TXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EPI[N]_TXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
8 (R/W)	DATGL	Data Toggle. The `USB_EPI[N]_TXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is set
		1	DATA1 is set

Table 28-38: USB_EPI[N]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W0C)	NAKTOINCMP	NAK Timeout Incomplete. The USB_EPI[N]_TXCSR_H.NAKTOINCMP bit indicates (for bulk endpoints in host mode) when the transmit endpoint is halted following the receipt of NAK responses for longer than the time set in the USB_EP[n]_TXINTERVAL register. The processor should clear this bit, allowing the endpoint to continue. For products supporting high-speed operation, for high-bandwidth isochronous endpoints in host mode, this bit indicates when no response is received from the device to which the packet is being sent.	
		0	No Status
		1	NAK Timeout Over Maximum
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The USB_EPI[N]_TXCSR_H.CLRDATATGL bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	RXSTALL	Rx STALL. The USB_EPI[N]_TXCSR_H.RXSTALL bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
4 (R/W)	SETUPPKT	Setup Packet. The USB_EPI[N]_TXCSR_H.SETUPPKT bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the USB_EPI[N]_TXCSR_H.TXPKTRDY bit is set.	
		0	No Request
		1	Send SETUP Token
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EPI[N]_TXCSR_H.FLUSHFIFO bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EPI[N]_TXCSR_H.TXPKTRDY bit. The USB_EPI[N]_TXCSR_H.FLUSHFIFO bit should only be set if the USB_EPI[N]_TXCSR_H.TXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 28-38: USB_EPI[N]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	TXTOERR	Tx Timeout Error. The `USB_EPI[N]_TXCSR_H.TXTOERR` bit indicates (in host mode) when three attempts have been made to send a packet and no handshake packet has been received. The USB controller generates an interrupt for this condition, clears the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit, and flushes the FIFO. The processor should clear this bit. Note that `USB_EPI[N]_TXCSR_H.TXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Tx Timeout Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EPI[N]_TXCSR_H.NEFIFO` bit indicates (in host mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in the `USB_INTRTXE` register), the USB controller generates an interrupt for this condition. Note that the `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EPI[N]_TXCSR_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Configuration and Status (Peripheral) Register

The USB_EPI[N]_TXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected transmit endpoint.

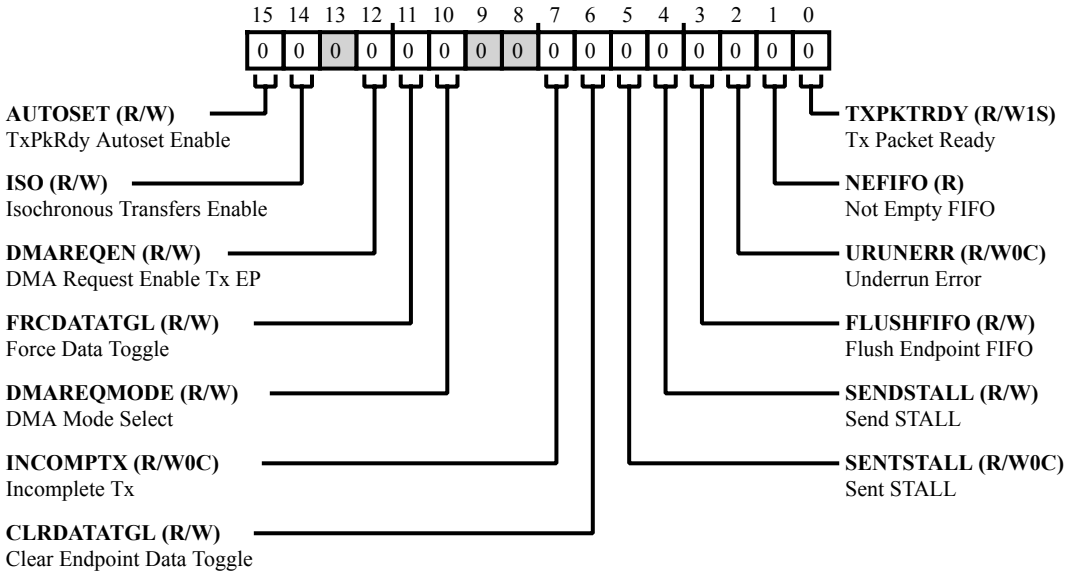

Figure 28-64: USB_EPI[N]_TXCSR_P Register Diagram

Table 28-39: USB_EPI[N]_TXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EPI[N]_TXCSR_P.AUTOSET bit enables (in peripheral mode) automatic setting of the USB_EPI[N]_TXCSR_P.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EPI[N]_TXCSR_P.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EPI[N]_TXCSR_P.AUTOSET bit should not be set for high-bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset

Table 28-39: USB_EPI[N]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	ISO	Isochronous Transfers Enable. The `USB_EPI[N]_TXCSR_P.ISO` bit enables (in peripheral mode) the transmit endpoint for isochronous transfers. This bit should be disabled for bulk or interrupt endpoints.	
		0	Disable Tx EP Isochronous Transfers
		1	Enable Tx EP Isochronous Transfers
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The `USB_EPI[N]_TXCSR_P.DMAREQEN` bit enables (in peripheral mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EPI[N]_TXCSR_P.FRCDATATGL` bit forces (in peripheral mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EPI[N]_TXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared during the cycle before or in the same cycle that the `USB_EPI[N]_TXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
7 (R/W0C)	INCOMPTX	Incomplete Tx. The `USB_EPI[N]_TXCSR_P.INCOMPTX` bit indicates (for high-bandwidth isochronous endpoints in peripheral mode) when a large packet has been split into two or three packets for transmission, but insufficient IN tokens have been received to send all parts.	
		0	No Status
		1	Incomplete Tx (Insufficient IN Tokens)

Table 28-39: USB_EPI[N]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EPI[N]_TXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EPI[N]_TXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when the USB controller transmits a STALL handshake. When this condition occurs, the USB controller flushes the FIFO and clears the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted
4 (R/W)	SENDSTALL	Send STALL. The `USB_EPI[N]_TXCSR_P.SENDSTALL` bit (in peripheral mode) is set by the processor to issue a STALL handshake to an IN token. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Request
		1	Request STALL Handshake Transmission
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EPI[N]_TXCSR_P.FLUSHFIFO` bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit. The `USB_EPI[N]_TXCSR_P.FLUSHFIFO` bit should only be set if the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO
2 (R/W0C)	URUNERR	Underrun Error. The `USB_EPI[N]_TXCSR_P.URUNERR` bit indicates (in peripheral mode) when an IN token is received while the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit is not set. The processor should clear this bit.	
		0	No Status
		1	Underrun Error

Table 28-39: USB_EPI[N]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EPI[N]_TXCSR_P.NEFIFO` bit indicates (in peripheral mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in the `USB_INTRTXE` register), the USB controller generates an interrupt for this condition. Note that the `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EPI[N]_TXCSR_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Polling Interval Register

The USB_EPI[N]_TXINTERVAL register defines the polling interval for the currently-selected transmit endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EPI[N]_TXINTERVAL register for each configured transmit endpoint, except endpoint 0. The transfer types related to the speed, interval value, and interval operation are as follows:

- Interrupt: Speed = low-speed or full-speed, USB_EPI[N]_TXINTERVAL = 1-255, and Operation = polling interval is m frames.
- Interrupt: Speed = high-speed, USB_EPI[N]_TXINTERVAL = 1-16, and Operation = polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed = full-speed or high-speed, USB_EPI[N]_TXINTERVAL = 1-16, and Operation = polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed = full-speed or high-speed, USB_EPI[N]_TXINTERVAL = 2-16, and Operation = NAK limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EPI[N]_TXINTERVAL value of 0 or 1 disables the NAK timeout function.

Not all products support high-speed operation or micro-frames. These features do not apply for products that only support low/full-speed operation.

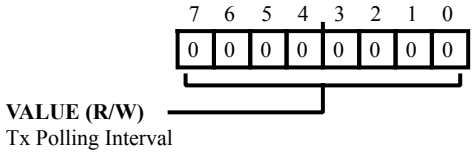

Figure 28-65: USB_EPI[N]_TXINTERVAL Register Diagram

Table 28-40: USB_EPI[N]_TXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Tx Polling Interval. The USB_EPI[N]_TXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers. The USB_EPI[N]_TXINTERVAL.VALUE bits select the number of frames (or micro-frames, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Transmit Maximum Packet Length Register

The USB_EPI[N]_TXMAXP register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. The USB_EPI[N]_TXMAXP register provides indexed access to the maximum packet length register for each Tx endpoint, except endpoint 0.

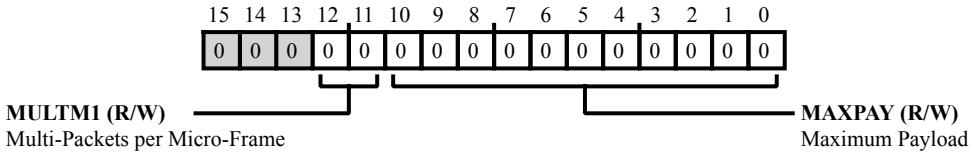

Figure 28-66: USB_EPI[N]_TXMAXP Register Diagram

Table 28-41: USB_EPI[N]_TXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-Frame. The USB_EPI[N]_TXMAXP.MULTM1 bits select the number of high-speed, high-bandwidth isochronous or interrupt packets that may be transferred in a micro-frame. The valid number of packets per micro-frame is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller splits the FIFO data into multiple packets less than or equal to the maximum payload size.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EPI[N]_TXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EPI[N]_TXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Transmit Type Register

The USB_EPI[N]_TXTYPE register selects the endpoint number and transaction protocol to use for the currently selected transmit endpoint. There is a USB_EPI[N]_TXTYPE register for each transmit endpoint. Note that this register is only used in host mode.

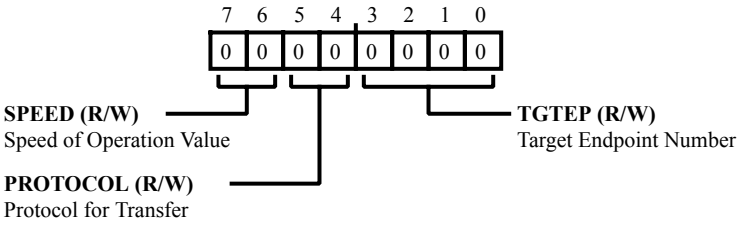

Figure 28-67: USB_EPI[N]_TXTYPE Register Diagram

Table 28-42: USB_EPI[N]_TXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EPI[N]_TXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances, the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EPI[N]_TXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EPI[N]_TXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the transmit endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 28-42: USB_EPI[N]_TXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Number of Bytes Received Register

The USB_EP[n]_RXCNT register indicates the number of received data bytes in the endpoint receive FIFO. The value returned changes as the contents of the FIFO change and is only valid while the USB_EP[n]_RXCSR_H.RXPKTRDY bit or USB_EP[n]_RXCSR_P.RXPKTRDY bit is set.

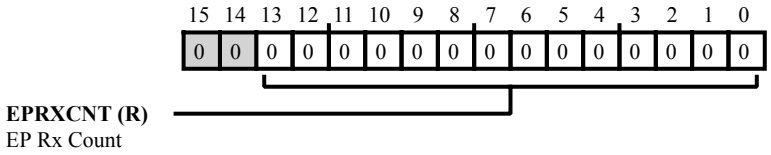

Figure 28-68: USB_EP[n]_RXCNT Register Diagram

Table 28-43: USB_EP[n]_RXCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
13:0 (R/NW)	EPRXCNT	EP Rx Count. The USB_EP[n]_RXCNT.EPRXCNT bits hold the number of data bytes ready to be read from the receive FIFO.

EPn Receive Configuration and Status (Host) Register

The USB_EP[n]_RXCSR_H register provides (in host mode) control and status bits for transfers through the currently selected receive endpoint.

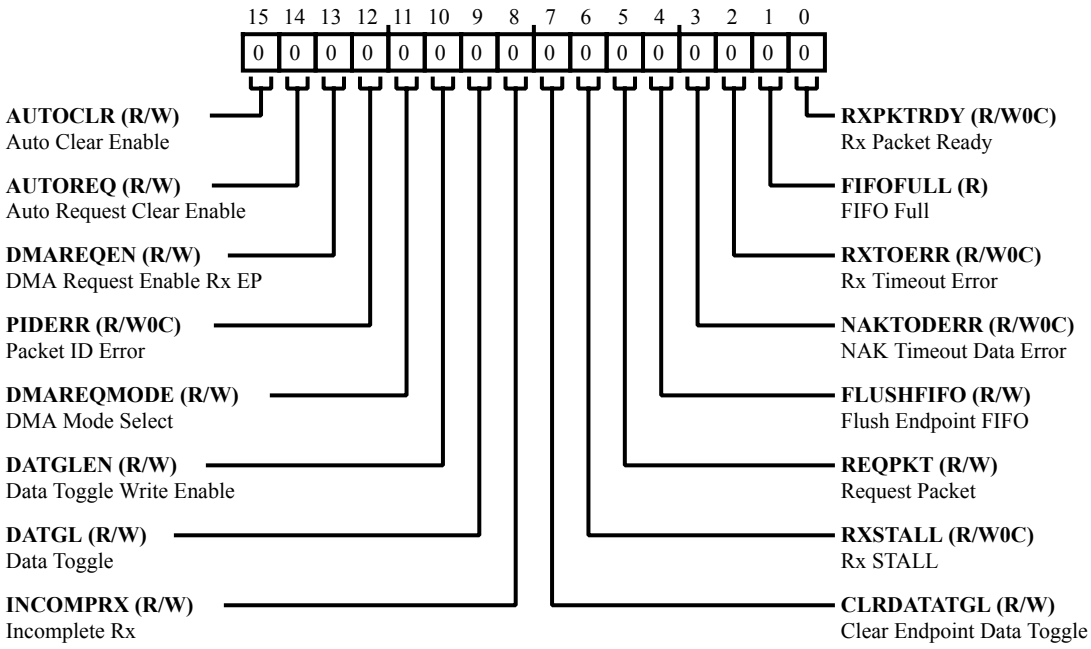

Figure 28-69: USB_EP[n]_RXCSR_H Register Diagram

Table 28-44: USB_EP[n]_RXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EP[n]_RXCSR_H.AUTOCLR bit directs (in host mode) the USB controller to automatically clear the USB_EP[n]_RXCSR_H.RXPKTRDY bit when a packet of size USB_EP[n]_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EP[n]_RXCSR_H.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EP[n]_RXMAXP value. The USB controller auto clears the USB_EP[n]_RXCSR_H.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the USB_EP[n]_RXCSR_H.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	AUTOREQ	Auto Request Clear Enable. The USB_EP[n]_RXCSR_H.AUTOREQ bit directs (in host mode) the USB controller to automatically clear the USB_EP[n]_RXCSR_H.REQPKT bit when USB_EP[n]_RXCSR_H.RXPKTRDY bit is cleared. This bit is automatically cleared when a short packet is received.	
		0	Disable Auto Request Clear
		1	Enable Auto Request Clear
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EP[n]_RXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 28-44: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W0C)	PIDERR	Packet ID Error. The USB_EP[n]_RXCSR_H.PIDERR bit indicates (in host mode) when a PID error occurs for isochronous transactions. This bit is ignored in host mode for bulk or interrupt transactions.	
		0	No Status
		1	PID Error
11 (R/W)	DMAREQMODE	DMA Mode Select. The USB_EP[n]_RXCSR_H.DMAREQMODE bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the USB_EP[n]_RXCSR_H.DMAREQEN bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
10 (R/W)	DATGLEN	Data Toggle Write Enable. The USB_EP[n]_RXCSR_H.DATGLEN bit enables (in host mode) the USB controller to write the current state of the endpoint USB_EP[n]_RXCSR_H.DATGL bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
9 (R/W)	DATGL	Data Toggle. The USB_EP[n]_RXCSR_H.DATGL bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is Set
		1	DATA1 is Set
8 (R/W)	INCOMPRX	Incomplete Rx. The USB_EP[n]_RXCSR_H.INCOMPRX bit indicates (in host mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when USB_EP[n]_RXCSR_H.RXPKTRDY is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx

Table 28-44: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_RXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	RXSTALL	Rx STALL. The `USB_EP[n]_RXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
5 (R/W)	REQPKT	Request Packet. The `USB_EP[n]_RXCSR_H.REQPKT` bit directs (in host mode) the USB controller to request an IN transaction. This bit is cleared when `USB_EP[n]_RXCSR_H.RXPKTRDY` is set.	
		0	No Request
		1	Send IN Tokens to Device
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP[n]_RXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit. The `USB_EP[n]_RXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 28-44: USB_EP[n]_RXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W0C)	NAKTODERR	NAK Timeout Data Error. The `USB_EP[n]_RXCSR_H.NAKTODERR` bit indicates (in host mode for isochronous transfers) a NAK timeout data error when the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when the `USB_EP[n]_RXCSR_H.RXPKTRDY` bit is cleared. The `USB_EP[n]_RXCSR_H.NAKTODERR` bit indicates (in host mod for bulk transfers) when a receive endpoint is halted following the receipt of NAK responses greater than the limit set in the `USB_EP[n]_RXINTERVAL` register. The processor should clear this bit to allow the endpoint to continue. If double packet buffering is enabled, the `USB_EP[n]_RXCSR_H.REQPKT` bit should also be set in the same cycle as this bit is cleared.	
		0	No Status
		1	NAK Timeout Data Error
2 (R/W0C)	RXTOERR	Rx Timeout Error. The `USB_EP[n]_RXCSR_H.RXTOERR` bit indicates (in host mode) when three attempts have been made to receive a packet and no data packet has been received. The USB controller generates an interrupt for this condition. The processor should clear this bit. Note that `USB_EP[n]_RXCSR_H.RXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Rx Timeout Error
1 (R/NW)	FIFOFULL	FIFO Full. The `USB_EP[n]_RXCSR_H.FIFOFULL` bit indicates (in host mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP[n]_RXCSR_H.RXPKTRDY` is set (in host mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Configuration and Status (Peripheral) Register

The `USB_EP[n]_RXCSR_P` register provides (in peripheral mode) control and status bits for transfers through the currently selected receive endpoint.

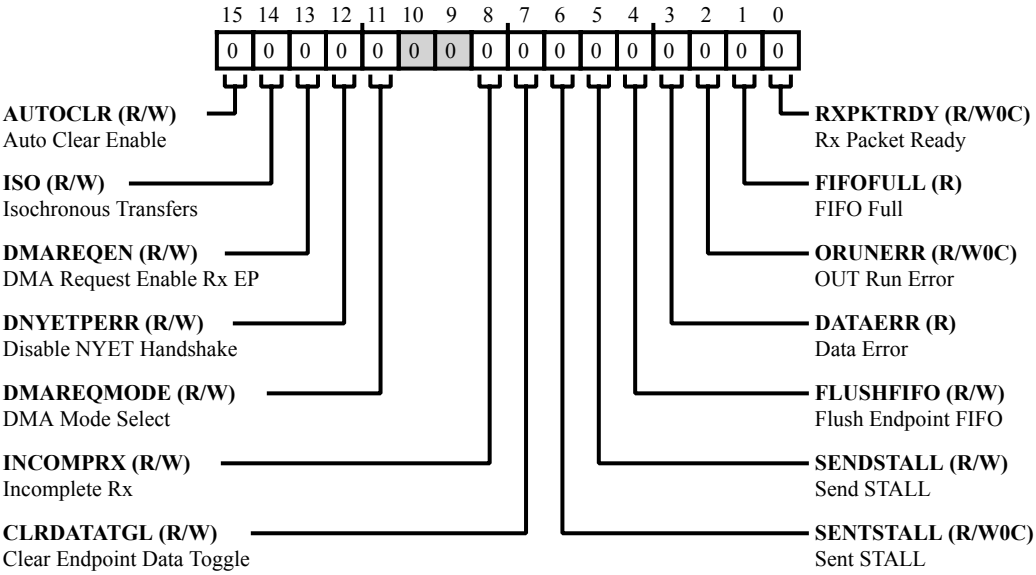

Figure 28-70: USB_EP[n]_RXCSR_P Register Diagram

Table 28-45: USB_EP[n]_RXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOCLR	Auto Clear Enable. The USB_EP[n]_RXCSR_P.AUTOCLR bit directs (in peripheral mode) the USB controller to automatically clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit when a packet of size USB_EP[n]_RXMAXP bytes has been unloaded from the receive FIFO. When packets of less than the maximum packet size are unloaded, the processor must clear USB_EP[n]_RXCSR_P.RXPKTRDY manually. When using the DMA to unload the receive FIFO, data is read from the receive FIFO in four byte chunks, regardless of the USB_EP[n]_RXMAXP value. The USB controller auto clears the USB_EP[n]_RXCSR_P.RXPKTRDY bit as follows. (In the following: Remainder=(RxMaxP/4), and PktSz-Clearing-RxPktRdy=Actual-Bytes-Read-Packet-Sizes-That-Clear-RxPktRdy.) • Remainder=0, Bytes-Read=RxMaxP, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2, RxMaxP-3 • Remainder=3, Bytes Read=RxMaxP+1, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1, RxMaxP-2 • Remainder=2, Bytes Read=RxMaxP+2, PktSz-Clearing-RxPktRdy=RxMaxP, RxMaxP-1 • Remainder=1, Bytes Read=RxMaxP+3, PktSz-Clearing-RxPktRdy=RxMaxP For products supporting high-speed operation, the USB_EP[n]_RXCSR_P.AUTOCLR bit should not be set for high-bandwidth isochronous endpoints.	
		0	Disable Auto Clear
		1	Enable Auto Clear
14 (R/W)	ISO	Isochronous Transfers. The USB_EP[n]_RXCSR_P.ISO bit selects (in peripheral mode) between isochronous transfers and bulk/interrupt transfers.	
		0	This bit should be cleared for bulk or interrupt transfers.
		1	This bit should be set for isochronous transfers.
13 (R/W)	DMAREQEN	DMA Request Enable Rx EP. The USB_EP[n]_RXCSR_P.DMAREQEN bit enables (in peripheral mode) DMA requests for this receive endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 28-45: USB_EP[n]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12 (R/W)	DNYETPERR	Disable NYET Handshake. The USB_EP[n]_RXCSR_P.DNYETPERR bit disables (in peripheral mode for high speed bulk/interrupt transactions) NYET handshakes. When this bit is set, all successful receive packets are ACK'd, including the point at which the FIFO becomes full. The USB_EP[n]_RXCSR_P.DNYETPERR bit must be set for all interrupt endpoints in high speed mode.	
		0	Enable NYET Handshake
		1	Disable NYET Handshake
11 (R/W)	DMAREQMODE	DMA Mode Select. The USB_EP[n]_RXCSR_P.DMAREQMODE bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared the cycle before or the same cycle that the USB_EP[n]_RXCSR_P.DMAREQEN bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
8 (R/W)	INCOMPRX	Incomplete Rx. The USB_EP[n]_RXCSR_P.INCOMPRX bit indicates (in peripheral mode for high-bandwidth isochronous or interrupt transfers) when the received packet is incomplete because parts of the packet were not received. This bit is cleared when USB_EP[n]_RXCSR_P.RXPKTRDY is cleared. For all other modes, this bit is zero.	
		0	No Status
		1	Incomplete Rx
7 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The USB_EP[n]_RXCSR_P.CLRDATATGL bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
6 (R/W0C)	SENTSTALL	Sent STALL. The USB_EP[n]_RXCSR_P.SENTSTALL bit indicates (in peripheral mode) when a STALL handshake is transmitted. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted

Table 28-45: USB_EP[n]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SENDSTALL	Send STALL. The USB_EP[n]_RXCSR_P.SENDSTALL bit is set (in peripheral mode) by the processor to send a STALL handshake. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Action
		1	Request STALL Handshake
4 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The USB_EP[n]_RXCSR_P.FLUSHFIFO bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the USB_EP[n]_RXCSR_P.RXPKTRDY bit. The USB_EP[n]_RXCSR_P.FLUSHFIFO bit should only be set if the USB_EP[n]_RXCSR_P.RXPKTRDY bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO
3 (R/NW)	DATAERR	Data Error. The USB_EP[n]_RXCSR_P.DATAERR bit indicates (in peripheral mode for isochronous transfers) when the USB_EP[n]_RXCSR_P.RXPKTRDY bit is set and the data packet has a CRC or bit-stuff error. This bit is cleared when USB_EP[n]_RXCSR_P.RXPKTRDY is cleared. The USB_EP[n]_RXCSR_P.DATAERR bit is always zero for bulk endpoints in peripheral mode.	
		0	No Status
		1	Data Error
2 (R/W0C)	ORUNERR	OUT Run Error. The USB_EP[n]_RXCSR_P.ORUNERR bit indicates (in peripheral mode for isochronous transfers) when an OUT packet cannot be loaded into the receive FIFO. The processor should clear this bit. The USB_EP[n]_RXCSR_P.ORUNERR bit always returns zero in bulk mode.	
		0	No Status
		1	OUT Run Error
1 (R/NW)	FIFOFULL	FIFO Full. The USB_EP[n]_RXCSR_P.FIFOFULL bit indicates (in peripheral mode) when no more packets can be loaded into the receive FIFO.	
		0	No Status
		1	FIFO Full

Table 28-45: USB_EP[n]_RXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W0C)	RXPKTRDY	Rx Packet Ready. The `USB_EP[n]_RXCSR_P.RXPKTRDY` is set (in peripheral mode) when a data packet is received. An interrupt is generated (if enabled) when this bit is set. The processor core should clear this bit when the packet is read from the FIFO.	
		0	No Rx Packet
		1	Rx Packet in Endpoint FIFO

EPn Receive Polling Interval Register

The USB_EP[n]_RXINTERVAL register defines the polling interval for the currently-selected receive endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EP[n]_RXINTERVAL register for each configured receive endpoint, except endpoint 0. The transfer types related to speed, interval value, and interval operation are as follows:

- Interrupt: Speed = low-speed or full -speed, USB_EP[n]_RXINTERVAL = 1-255, and Operation = polling interval is m frames.
- Interrupt: Speed = high-speed, USB_EP[n]_RXINTERVAL = 1-16, and Operation = polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed = full-speed or high-speed, USB_EP[n]_RXINTERVAL = 1-16, and Operation = polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed = full-speed or high-speed, USB_EP[n]_RXINTERVAL = 2-16, and Operation = NAK limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EP[n]_RXINTERVAL value of 0 or 1 disables the NAK timeout function.

Not all products support high-speed operation or micro-frames. These features do not apply for products that only support low/full-speed operation.

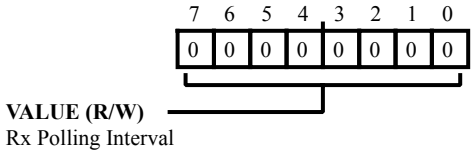

Figure 28-71: USB_EP[n]_RXINTERVAL Register Diagram

Table 28-46: USB_EP[n]_RXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Rx Polling Interval. The USB_EP[n]_RXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers and select the number of frames (or microframes, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Receive Maximum Packet Length Register

The USB_EP[n]_RXMAXP register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame.

Note that a value greater than the maximum allowed of 1023 for full-speed USB operation produces unpredictable results. Also, note that the total amount of data represented by the value written to this register must not exceed the receive FIFO size, and should not exceed half the FIFO size if double-buffering is required.

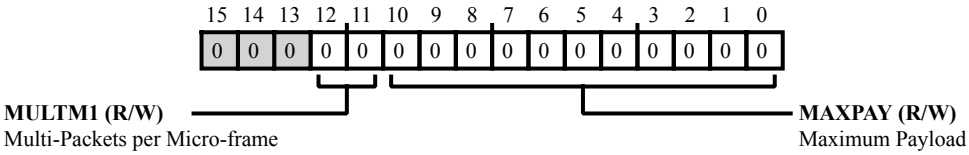

Figure 28-72: USB_EP[n]_RXMAXP Register Diagram

Table 28-47: USB_EP[n]_RXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-frame. The USB_EP[n]_RXMAXP.MULTM1 bits select the number of high-speed, high-bandwidth isochronous or interrupt packets that may be transferred in a micro-frame. The valid number of packets per micro-frame is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller combines multiple packets received within a micro-frame into a single packet in the FIFO.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EP[n]_RXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024, but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EP[n]_RXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Receive Type Register

The USB_EP[n]_RXTYPE register selects the endpoint number and transaction protocol to use for the currently selected receive endpoint. There is a USB_EP[n]_RXTYPE register for each receive endpoint. Note that this register is only used in host mode.

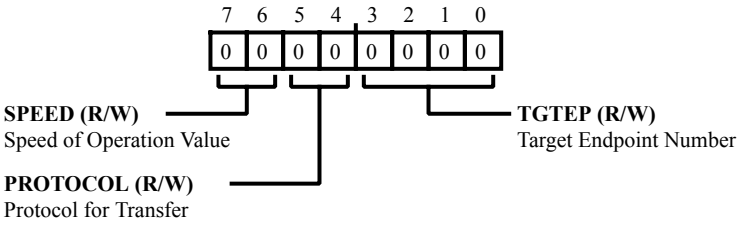

Figure 28-73: USB_EP[n]_RXTYPE Register Diagram

Table 28-48: USB_EP[n]_RXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EP[n]_RXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances, the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When it is not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EP[n]_RXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EP[n]_RXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the receive endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 28-48: USB_EP[n]_RXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

EPn Transmit Configuration and Status (Host) Register

The USB_EP[n]_TXCSR_H register provides (in host mode) control and status bits for transfers through the currently-selected transmit endpoint.

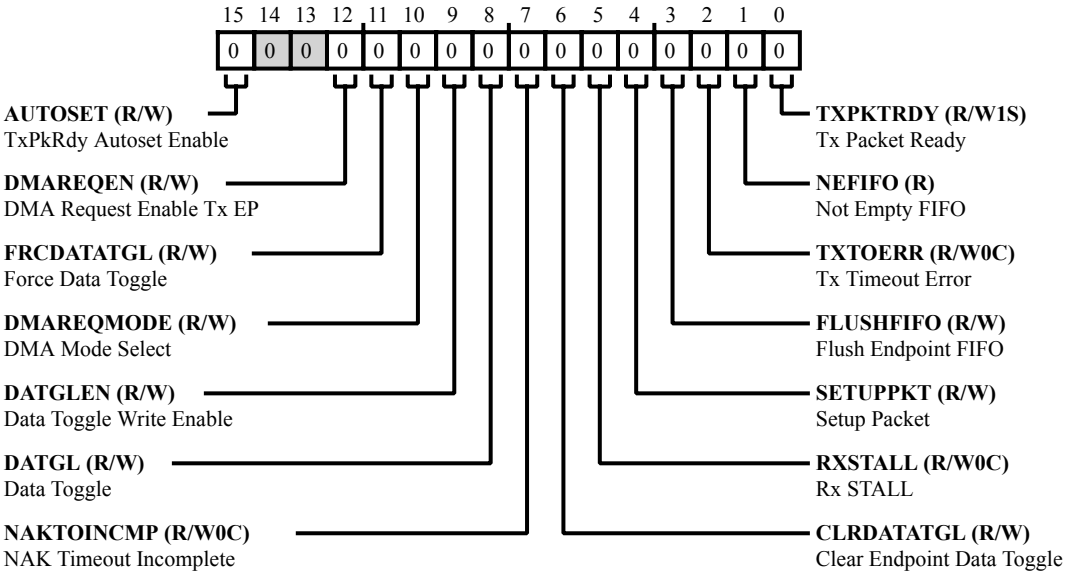

Figure 28-74: USB_EP[n]_TXCSR_H Register Diagram

Table 28-49: USB_EP[n]_TXCSR_H Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EP[n]_TXCSR_H.AUTOSET bit enables (in host mode) the automatic setting of the USB_EP[n]_TXCSR_H.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EP[n]_TXCSR_H.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EP[n]_TXCSR_H.AUTOSET bit should not be set for high-bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The USB_EP[n]_TXCSR_H.DMAREQEN bit enables (in host mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request

Table 28-49: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EP[n]_TXCSR_H.FRCDATATGL` bit forces (in host mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_TXCSR_H.DMAREQMODE` bit selects (in host mode) between DMA request mode 1 or 0. This bit must not be cleared during the cycle before or the same cycle that the `USB_EP[n]_TXCSR_H.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
9 (R/W)	DATGLEN	Data Toggle Write Enable. The `USB_EP[n]_TXCSR_H.DATGLEN` bit enables (in host mode) the USB controller to write the current state of the endpoint `USB_EP[n]_TXCSR_H.DATGL` bit. This bit is automatically cleared once the new value is written.	
		0	Disable Write to DATGL
		1	Enable Write to DATGL
8 (R/W)	DATGL	Data Toggle. The `USB_EP[n]_TXCSR_H.DATGL` bit indicates (in host mode) the current state of the endpoint data toggle. If D10 is high, this bit may be written with the required setting of the data toggle. If D10 is low, any value written to this bit is ignored. This bit is only used in host mode.	
		0	DATA0 is set
		1	DATA1 is set

Table 28-49: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W0C)	NAKTOINCMP	NAK Timeout Incomplete. The `USB_EP[n]_TXCSR_H.NAKTOINCMP` bit indicates (for bulk endpoints in host mode) when the transmit endpoint is halted following the receipt of NAK responses for longer than the time set in the `USB_EP[n]_TXINTERVAL` register. The processor should clear this bit, allowing the endpoint to continue. For products supporting high-speed operation, for high-bandwidth isochronous endpoints in host mode, this bit indicates when no response is received from the device to which the packet is being sent.	
		0	No Status
		1	NAK Timeout Over Maximum
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_TXCSR_H.CLRDATATGL` bit is set (in host mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	RXSTALL	Rx STALL. The `USB_EP[n]_TXCSR_H.RXSTALL` bit indicates (in host mode) when a STALL handshake is received. The processor core should clear this bit.	
		0	No Status
		1	Stall Received from Device
4 (R/W)	SETUPPKT	Setup Packet. The `USB_EP[n]_TXCSR_H.SETUPPKT` bit directs (in host mode) the USB controller to send a SETUP token instead of an OUT token for the transaction. This bit is set at the same time as the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is set.	
		0	No Request
		1	Send SETUP Token
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP[n]_TXCSR_H.FLUSHFIFO` bit directs (in host mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit. The `USB_EP[n]_TXCSR_H.FLUSHFIFO` bit should only be set if the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush Endpoint FIFO

Table 28-49: USB_EP[n]_TXCSR_H Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W0C)	TXTOERR	Tx Timeout Error. The `USB_EP[n]_TXCSR_H.TXTOERR` bit indicates (in host mode) when three attempts have been made to send a packet and no handshake packet has been received. The USB controller generates an interrupt for this condition, clears the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit, and flushes the FIFO. The processor should clear this bit. Note that `USB_EP[n]_TXCSR_H.TXTOERR` is valid only when the endpoint is operating in bulk or interrupt mode.	
		0	No Status
		1	Tx Timeout Error
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EP[n]_TXCSR_H.NEFIFO` bit indicates (in host mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in the `USB_INTRTXE` register), the USB controller generates an interrupt for this condition. Note that the `USB_EP[n]_TXCSR_H.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP[n]_TXCSR_H.TXPKTRDY` bit should be set (in host mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Configuration and Status (Peripheral) Register

The USB_EP[n]_TXCSR_P register provides (in peripheral mode) control and status bits for transfers through the currently selected transmit endpoint.

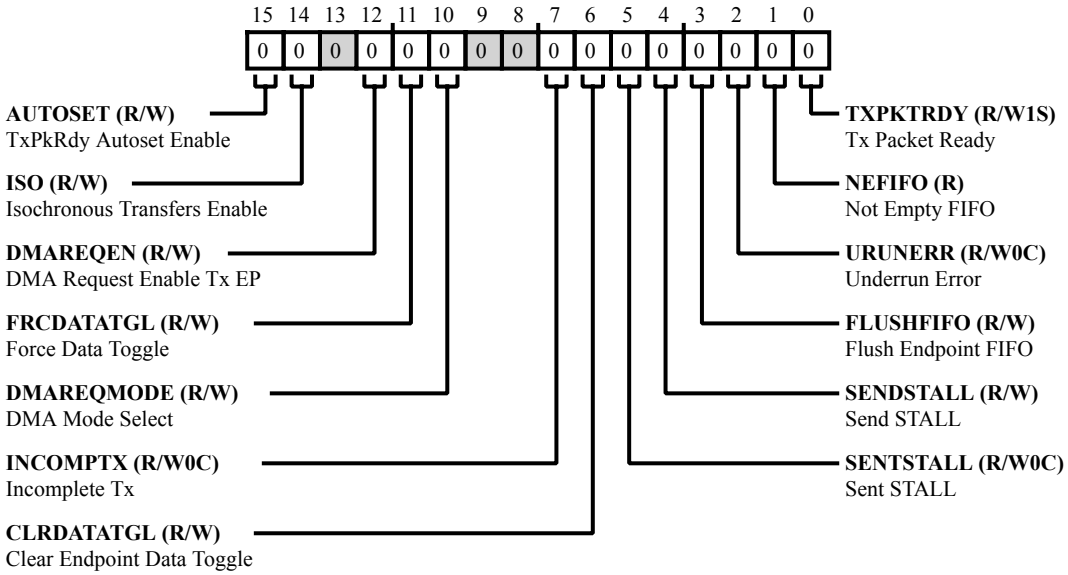

Figure 28-75: USB_EP[n]_TXCSR_P Register Diagram

Table 28-50: USB_EP[n]_TXCSR_P Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/W)	AUTOSET	TxPkRdy Autoset Enable. The USB_EP[n]_TXCSR_P.AUTOSET bit enables (in peripheral mode) automatic setting of the USB_EP[n]_TXCSR_P.TXPKTRDY bit when the maximum data packet size (USB_EP[n]_TXMAXP) is loaded into the transmit FIFO. The USB_EP[n]_TXMAXP value must be a word (4-byte) multiple. If a packet less than the maximum packet size is loaded, the USB_EP[n]_TXCSR_P.TXPKTRDY bit needs to be set manually. For products supporting high-speed operation, this USB_EP[n]_TXCSR_P.AUTOSET bit should not be set for high-bandwidth endpoints (endpoints with USB_EP[n]_TXMAXP value greater than 1).	
		0	Disable Autoset
		1	Enable Autoset

Table 28-50: USB_EP[n]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	ISO	Isochronous Transfers Enable. The `USB_EP[n]_TXCSR_P.ISO` bit enables (in peripheral mode) the transmit endpoint for isochronous transfers. This bit should be disabled for bulk or interrupt endpoints.	
		0	Disable Tx EP Isochronous Transfers
		1	Enable Tx EP Isochronous Transfers
12 (R/W)	DMAREQEN	DMA Request Enable Tx EP. The `USB_EP[n]_TXCSR_P.DMAREQEN` bit enables (in peripheral mode) DMA requests for this transmit endpoint.	
		0	Disable DMA Request
		1	Enable DMA Request
11 (R/W)	FRCDATATGL	Force Data Toggle. The `USB_EP[n]_TXCSR_P.FRCDATATGL` bit forces (in peripheral mode) the endpoint data toggle to switch and clears the data packet from the FIFO, regardless of whether an ACK was received. This feature can be used by interrupt transmit endpoints that are used to communicate rate feedback for isochronous endpoints.	
		0	No Action
		1	Toggle Endpoint Data
10 (R/W)	DMAREQMODE	DMA Mode Select. The `USB_EP[n]_TXCSR_P.DMAREQMODE` bit selects (in peripheral mode) between DMA request mode 1 or 0. This bit must not be cleared during the cycle before or in the same cycle that the `USB_EP[n]_TXCSR_P.DMAREQEN` bit is cleared. In DMA request mode 0, the DMA is programmed to load one packet at a time. Processor intervention is required for each packet. DMA mode 1 can be used with bulk endpoints to transmit multiple packets without processor intervention.	
		0	DMA Request Mode 0
		1	DMA Request Mode 1
7 (R/W0C)	INCOMPTX	Incomplete Tx. The `USB_EP[n]_TXCSR_P.INCOMPTX` bit indicates (for high-bandwidth isochronous endpoints in peripheral mode) when a large packet has been split into two or three packets for transmission, but insufficient IN tokens have been received to send all parts.	
		0	No Status
		1	Incomplete Tx (Insufficient IN Tokens)

Table 28-50: USB_EP[n]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	CLRDATATGL	Clear Endpoint Data Toggle. The `USB_EP[n]_TXCSR_P.CLRDATATGL` bit is set (in peripheral mode) by the processor to reset the endpoint data toggle to 0.	
		0	No Action
		1	Reset EP Data Toggle to 0
5 (R/W0C)	SENTSTALL	Sent STALL. The `USB_EP[n]_TXCSR_P.SENTSTALL` bit indicates (in peripheral mode) when the USB controller transmits a STALL handshake. When this condition occurs, the USB controller flushes the FIFO and clears the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit. The processor should clear this bit.	
		0	No Status
		1	STALL Handshake Transmitted
4 (R/W)	SENDSTALL	Send STALL. The `USB_EP[n]_TXCSR_P.SENDSTALL` bit (in peripheral mode) is set by the processor to issue a STALL handshake to an IN token. The processor clears this bit to terminate the stall condition. This bit has no effect for isochronous transfers.	
		0	No Request
		1	Request STALL Handshake Transmission
3 (R/W)	FLUSHFIFO	Flush Endpoint FIFO. The `USB_EP[n]_TXCSR_P.FLUSHFIFO` bit directs (in peripheral mode) the USB controller to flush data from the endpoint FIFO and clear the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit. The `USB_EP[n]_TXCSR_P.FLUSHFIFO` bit should only be set if the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is set. Note that setting this bit at other times may cause data corruption.	
		0	No Flush
		1	Flush endpoint FIFO
2 (R/W0C)	URUNERR	Underrun Error. The `USB_EP[n]_TXCSR_P.URUNERR` bit indicates (in peripheral mode) when an IN token is received while the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is not set. The processor should clear this bit.	
		0	No Status
		1	Underrun Error

Table 28-50: USB_EP[n]_TXCSR_P Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	NEFIFO	Not Empty FIFO. The `USB_EP[n]_TXCSR_P.NEFIFO` bit indicates (in peripheral mode) when there is at least one packet in the transmit FIFO. This bit is cleared automatically when a data packet has been transmitted. If the endpoints transmit interrupt is enabled (in the `USB_INTRTXE` register), the USB controller generates an interrupt for this condition. Note that the `USB_EP[n]_TXCSR_P.TXPKTRDY` bit is also automatically cleared prior to loading a second packet into a double-buffered FIFO.	
		0	FIFO Empty
		1	FIFO Not Empty
0 (R/W1S)	TXPKTRDY	Tx Packet Ready. The `USB_EP[n]_TXCSR_P.TXPKTRDY` bit should be set (in peripheral mode) by the processor core after loading a data packet into the FIFO. This bit is cleared automatically when the data packet is transmitted. An interrupt is generated (if enabled) when the bit is cleared.	
		0	No Tx Packet
		1	Tx Packet in Endpoint FIFO

EPn Transmit Polling Interval Register

The USB_EP[n]_TXINTERVAL register defines the polling interval for the currently-selected transmit endpoint for interrupt, isochronous, and bulk transfers. There is a USB_EP[n]_TXINTERVAL register for each configured transmit endpoint, except endpoint 0. The transfer types related to the speed, interval value, and interval operation are as follows:

- Interrupt: Speed = low-speed or full-speed, USB_EP[n]_TXINTERVAL = 1-255, and Operation = polling interval is m frames.
- Interrupt: Speed = high-speed, USB_EP[n]_TXINTERVAL = 1-16, and Operation = polling interval is $2^{(m-1)}$ micro-frames.
- Isochronous: Speed = full-speed or high-speed, USB_EP[n]_TXINTERVAL = 1-16, and Operation = polling interval is $2^{(m-1)}$ frames or micro-frames.
- Bulk: Speed = full-speed or high-speed, USB_EP[n]_TXINTERVAL = 2-16, and Operation = NAK limit is $2^{(m-1)}$ frames or micro-frames.

Note that a USB_EP[n]_TXINTERVAL value of 0 or 1 disables the NAK timeout function.

Not all products support high-speed operation or micro-frames. These features do not apply for products that only support low/full-speed operation.

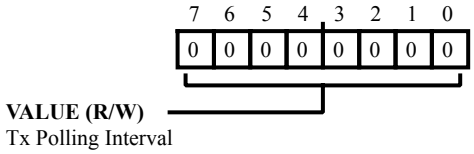

Figure 28-76: USB_EP[n]_TXINTERVAL Register Diagram

Table 28-51: USB_EP[n]_TXINTERVAL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Tx Polling Interval. The USB_EP[n]_TXINTERVAL.VALUE bits define the polling interval value for interrupt and isochronous transfers. The USB_EP[n]_TXINTERVAL.VALUE bits select the number of frames (or micro-frames, if the processor supports high-speed operation) after which the endpoint should timeout on receiving a stream of NAK responses for bulk and control endpoints. Note that the USB controller halts transfers to control endpoints if the host receives NAK responses for more frames than the limit set by this register.

EPn Transmit Maximum Packet Length Register

The USB_EP[n]_TXMAXP register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. The USB_EP[n]_TXMAXP register provides indexed access to the maximum packet length register for each Tx endpoint, except endpoint 0.

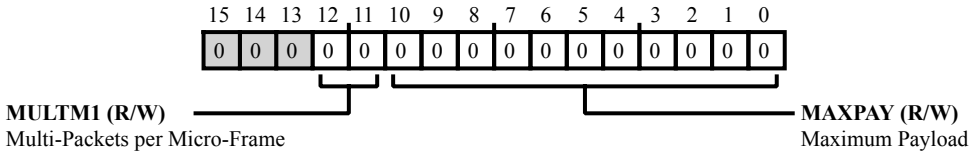

Figure 28-77: USB_EP[n]_TXMAXP Register Diagram

Table 28-52: USB_EP[n]_TXMAXP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:11 (R/W)	MULTM1	Multi-Packets per Micro-Frame. The USB_EP[n]_TXMAXP.MULTM1 bits select the number of high-speed, high-bandwidth isochronous or interrupt packets that may be transferred in a micro-frame. The valid number of packets per micro-frame is 1-3 which corresponds to settings 0-2. If this field is not zero, the USB controller splits the FIFO data into multiple packets less than or equal to the maximum payload size.
10:0 (R/W)	MAXPAY	Maximum Payload. The USB_EP[n]_TXMAXP.MAXPAY bits select the maximum number of bytes that may be transferred per transaction. This field can be up to 1024 but is subject to constraints by the USB specification based on endpoint mode and speed. This field should not exceed the FIFO size for the endpoint, or half the FIFO size if double buffering is used. This value should match the wMaxPacketSize field of the standard endpoint descriptor (USB 2.0 spec, section 9). The USB_EP[n]_TXMAXP.MAXPAY bits must be set to an even number of bytes for proper interrupt generation in DMA mode 1.

EPn Transmit Type Register

The USB_EP[n]_TXTYPE register selects the endpoint number and transaction protocol to use for the currently selected transmit endpoint. There is a USB_EP[n]_TXTYPE register for each transmit endpoint. Note that this register is only used in host mode.

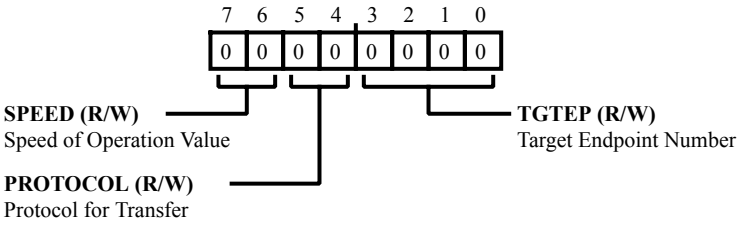

Figure 28-78: USB_EP[n]_TXTYPE Register Diagram

Table 28-53: USB_EP[n]_TXTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7:6 (R/W)	SPEED	Speed of Operation Value. The USB_EP[n]_TXTYPE.SPEED bits select the USB controller operating speed for the endpoint when acting as a host connected to devices through a hub. In these instances, the USB controller must issue split transactions under certain conditions. If a device is directly connected (not through a hub), all endpoints use the same speed as which the controller is connected. When not connected to devices through a hub, program this field with 00.	
		0	Same Speed as the Core
		1	High-Speed
		2	Full-Speed
		3	Low-Speed
5:4 (R/W)	PROTOCOL	Protocol for Transfer. The USB_EP[n]_TXTYPE.PROTOCOL bits select the transfer protocol for the endpoint.	
		0	Control
		1	Isochronous
		2	Bulk
		3	Interrupt
3:0 (R/W)	TGTEP	Target Endpoint Number. The USB_EP[n]_TXTYPE.TGTEP bits select (for endpoints 1-11) the target endpoint. This value should be set to the endpoint number contained in the transmit endpoint descriptor returned during device enumeration. Endpoint 0 always uses target endpoint number 0. (Enumeration values not shown are reserved.)	

Table 28-53: USB_EP[n]_TXTYPE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	Endpoint 0
		1	Endpoint 1
		2	Endpoint 2
		3	Endpoint 3
		4	Endpoint 4
		5	Endpoint 5
		6	Endpoint 6
		7	Endpoint 7
		8	Endpoint 8
		9	Endpoint 9
		10	Endpoint 10
		11	Endpoint 11
		12	Endpoint 12
		13	Endpoint 13
		14	Endpoint 14
		15	Endpoint 15

Function Address Register

The USB_FADDR register contains the device address used in peripheral mode. The processor writes this register with the address received through a SET_ADDRESS command from the host.

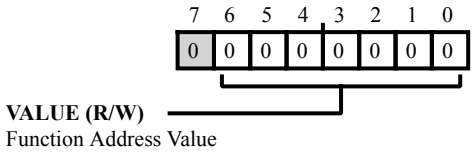

Figure 28-79: USB_FADDR Register Diagram

Table 28-54: USB_FADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Function Address Value. The USB_FADDR.VALUE bits contain the address of the peripheral part of the transaction.

FIFO Byte (8-Bit) Register

Writes to the USB_FIFOB[n] register go to the endpoint Tx FIFO and reads from the USB_FIFOB[n] register come from the endpoint Rx FIFO. The USB_FIFOB[n], USB_FIFOH[n], and USB_FIFO[n] registers are the same register. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (USB_FIFO[n] register) writes and reads, which are more efficient. If the USB packet is a non-word (4-byte) size, the program should use a half-word (USB_FIFOH[n] register) or byte (USB_FIFOB[n] register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

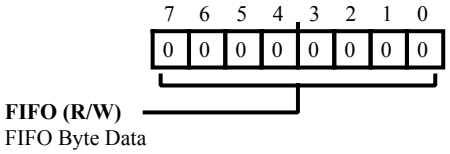

Figure 28-80: USB_FIFOB[n] Register Diagram

Table 28-55: USB_FIFOB[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	FIFO	FIFO Byte Data. The USB_FIFOB[n].FIFO bits provide byte access to the USB Tx and Rx endpoint FIFOs.

FIFO Half-Word (16-Bit) Register

Writes to the USB_FIFOH[n] register go to the endpoint Tx FIFO and reads from the USB_FIFOH[n] register come from the endpoint Rx FIFO. The USB_FIFOB[n], USB_FIFOH[n], and USB_FIFO[n] registers are the same register. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (USB_FIFO[n] register) writes and reads, which are more efficient. If the USB packet is a non-word (4-byte) size, programs should use a half-word (USB_FIFOH[n] register) or byte (USB_FIFOB[n] register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

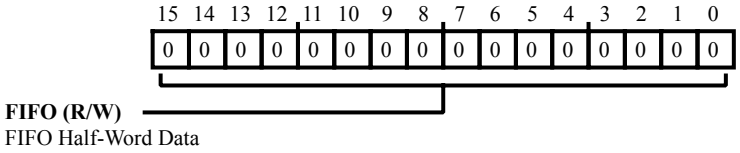

Figure 28-81: USB_FIFOH[n] Register Diagram

Table 28-56: USB_FIFOH[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	FIFO	FIFO Half-Word Data. The USB_FIFOH[n].FIFO bits provide half-word access to the USB Tx and Rx endpoint FIFOs.

FIFO Word (32-Bit) Register

Writes to the USB_FIFO[n] register go to the endpoint Tx FIFO and reads from the USB_FIFO[n] register come from the endpoint Rx FIFO. The USB_FIFOB[n], USB_FIFOH[n], and USB_FIFO[n] registers are the same registers. These registers exist at the same address. Typically, programs should load and unload the FIFO using word (USB_FIFO[n] register) writes and reads, which are more efficient. If the USB packet is a non-word (4-byte) size, programs should use a half-word (USB_FIFOH[n] register) or byte (USB_FIFOB[n] register) read or write at the end when loading or unloading the FIFO.

Note that (for correct USB controller operation) programs should not mix byte, half-word, or word accesses, except for the last few bytes if the size of the packet is odd (not a multiple of the size they were using).

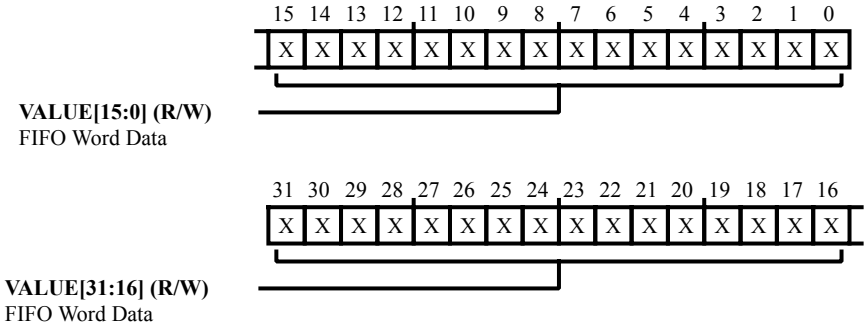

Figure 28-82: USB_FIFO[n] Register Diagram

Table 28-57: USB_FIFO[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	FIFO Word Data. The USB_FIFO[n].VALUE bits provide word access to the USB Tx and Rx endpoint FIFOs.

Frame Number Register

The USB_FRAME register contains the frame number of the last received frame. The data in this register has bit 10 as the MSB and bit 0 as the LSB.

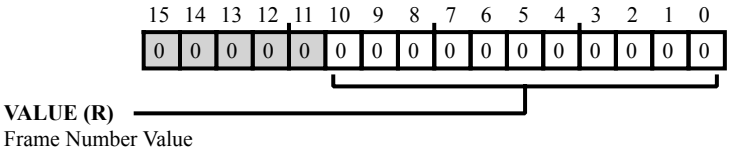

Figure 28-83: USB_FRAME Register Diagram

Table 28-58: USB_FRAME Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
10:0 (R/NW)	VALUE	Frame Number Value. The USB_FRAME.VALUE bits contains the frame number of the last received frame. The data in this field has bit 10 as the MSB and bit 0 as the LSB.

Full-Speed EOF 1 Register

The `USB_FS_EOF1` register defines the minimum time gap allowed between the start of the last transaction and the end of frame for full-speed transactions.

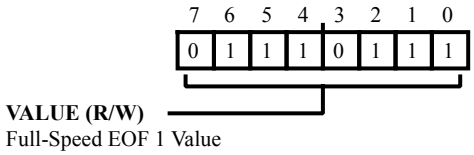

Figure 28-84: USB_FS_EOF1 Register Diagram

Table 28-59: USB_FS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Full-Speed EOF 1 Value. The `USB_FS_EOF1.VALUE` bits set the time before the end of the frame to stop beginning new transactions (in units of 533.3ns) for full-speed transactions. The default setting corresponds to 63.46us.

High-Speed EOF 1 Register

The USB_HS_EOF1 register defines the minimum time gap allowed between the start of the last transaction and the end of frame for high-speed transactions.

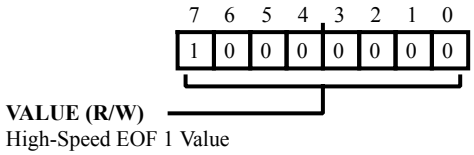

Figure 28-85: USB_HS_EOF1 Register Diagram

Table 28-60: USB_HS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	High-Speed EOF 1 Value. The USB_HS_EOF1.VALUE sets the time before the end of the frame to stop beginning new transactions (in units of 133.3ns) for high-speed transactions. The default setting corresponds to 17.07us.

Common Interrupts Enable Register

The USB_IEN register enables interrupts for USB controller system sources. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_IRQ register is set.

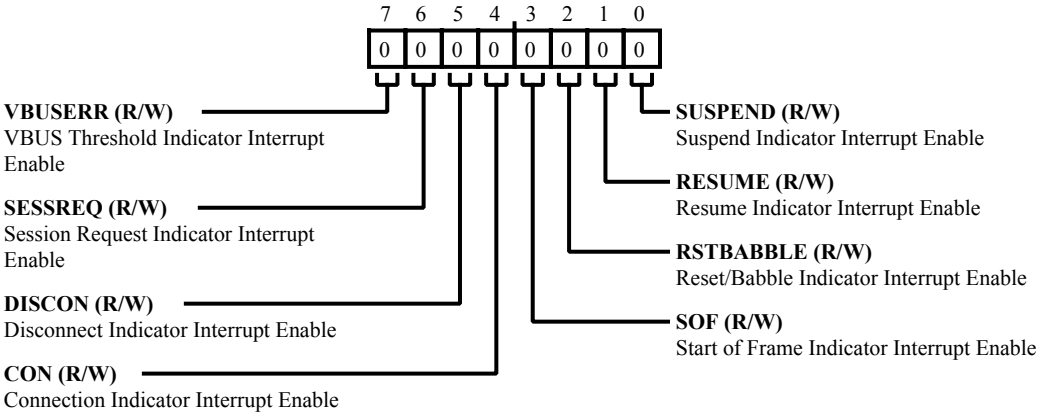

Figure 28-86: USB_IEN Register Diagram

Table 28-61: USB_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	VBUSERR	VBUS Threshold Indicator Interrupt Enable. The USB_IEN.VBUSERR bit enables the USB_IRQ.VBUSERR interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	SESSREQ	Session Request Indicator Interrupt Enable. The USB_IEN.SESSREQ bit enables the USB_IRQ.SESSREQ interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	DISCON	Disconnect Indicator Interrupt Enable. The USB_IEN.DISCON bit enables the USB_IRQ.DISCON interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	CON	Connection Indicator Interrupt Enable. The USB_IEN.CON bit enables the USB_IRQ.CON interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 28-61: USB_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	SOF	Start of Frame Indicator Interrupt Enable. The `USB_IEN.SOF` bit enables the `USB_IRQ.SOF` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	RSTBABBLE	Reset/Babble Indicator Interrupt Enable. The `USB_IEN.RSTBABBLE` bit enables the `USB_IRQ.RSTBABBLE` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	RESUME	Resume Indicator Interrupt Enable. The `USB_IEN.RESUME` bit enables the `USB_IRQ.RESUME` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	SUSPEND	Suspend Indicator Interrupt Enable. The `USB_IEN.SUSPEND` bit enables the `USB_IRQ.SUSPEND` interrupt.	
		0	Disable Interrupt
		1	Enable Interrupt

Index Register

The USB_INDEX register contains an index value for mirrored addressing of USB controller endpoint control and status registers.

There is one set of registers, but they are mirrored at two address locations if the endpoint is selected by the USB_INDEX register. An endpoint's register set only appears in the indexed location if the USB_INDEX register is written with that endpoint number. You can read/write an endpoint's register in either the directly mapped location which is always visible, or in the indexed location which is only visible if the USB_INDEX register is written with the endpoint number. The USB_INDEX register and indexed address locations only affect address decoding. For example, loading a 0 into the USB_INDEX register selects endpoint 0 access.

The USB_INDEX register can be used for indexed access of the directly mapped control/status registers from USB controller address offset 0x100-0x1FF. For products supporting the dynamic FIFO size feature, the endpoint Tx/Rx size and address registers always use the USB_INDEX register, there is no direct mapping for these endpoint-specific registers. The multi-point USB_MP[n]_TXFUNCADDR, USB_MP[n]_TXHUBADDR, USB_MP[n]_TXHUBPORT, USB_MP[n]_RXFUNCADDR, USB_MP[n]_RXHUBADDR, and USB_MP[n]_RXHUBPORT registers only have direct mapping, no indexed mapping.

Before accessing an endpoint's control/status registers using the indexed range, write the endpoint number to the USB_INDEX register to ensure that the correct control/status registers appear in the indexed range of the memory map.

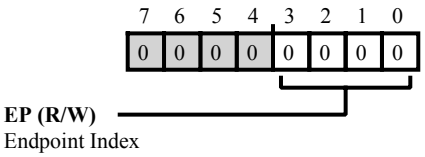

Figure 28-87: USB_INDEX Register Diagram

Table 28-62: USB_INDEX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W)	EP	Endpoint Index. The USB_INDEX.EP bits select mirrored access for an endpoints indexed control and status registers. Valid values for this bit field are 0-11.

Receive Interrupt Register

The USB_INTRRX register indicates which interrupts are currently active for the receive (Rx) endpoints. Note that the USB controller automatically clears this register when it is read.

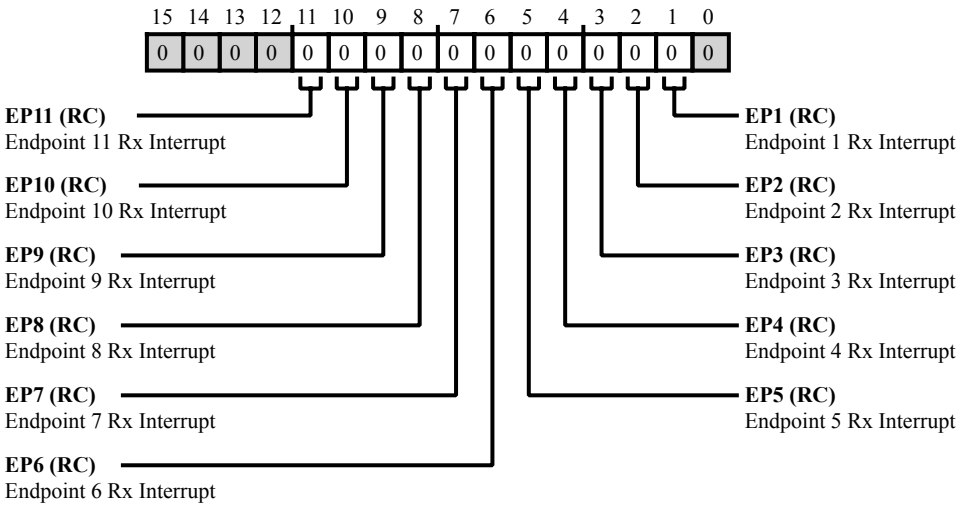

Figure 28-88: USB_INTRRX Register Diagram

Table 28-63: USB_INTRRX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (RC/NW)	EP11	Endpoint 11 Rx Interrupt. The USB_INTRRX.EP11 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
10 (RC/NW)	EP10	Endpoint 10 Rx Interrupt. The USB_INTRRX.EP10 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
9 (RC/NW)	EP9	Endpoint 9 Rx Interrupt. The USB_INTRRX.EP9 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 28-63: USB_INTRRX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (RC/NW)	EP8	Endpoint 8 Rx Interrupt. The USB_INTRRX.EP8 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
7 (RC/NW)	EP7	Endpoint 7 Rx Interrupt. The USB_INTRRX.EP7 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	EP6	Endpoint 6 Rx Interrupt. The USB_INTRRX.EP6 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	EP5	Endpoint 5 Rx Interrupt. The USB_INTRRX.EP5 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
4 (RC/NW)	EP4	Endpoint 4 Rx Interrupt. The USB_INTRRX.EP4 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	EP3	Endpoint 3 Rx Interrupt. The USB_INTRRX.EP3 bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 28-63: USB_INTRRX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (RC/NW)	EP2	Endpoint 2 Rx Interrupt. The `USB_INTRRX.EP2` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	EP1	Endpoint 1 Rx Interrupt. The `USB_INTRRX.EP1` bit indicates whether or not a receive interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Receive Interrupt Enable Register

The USB_INTRRXE register enables interrupts for the receive (Rx) endpoints. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_INTRRX register is set.

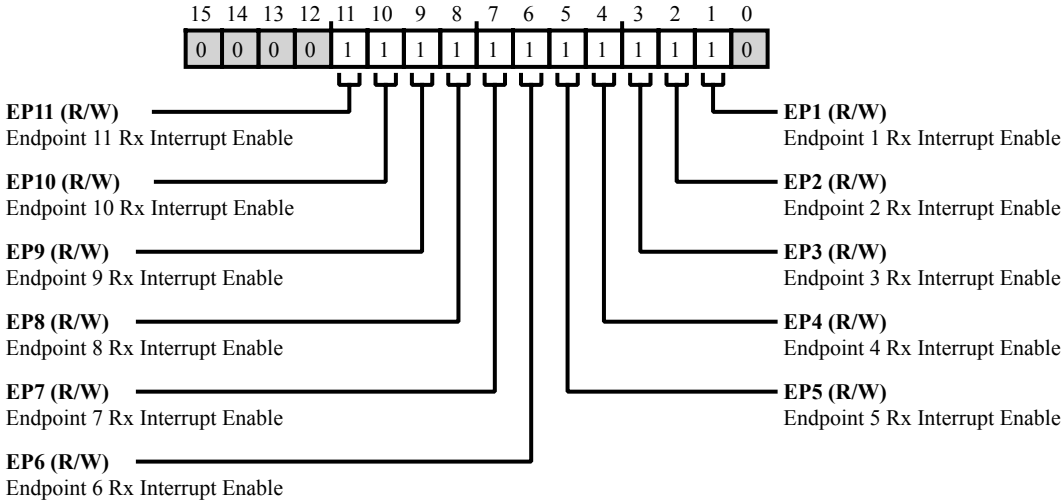

Figure 28-89: USB_INTRRXE Register Diagram

Table 28-64: USB_INTRRXE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EP11	Endpoint 11 Rx Interrupt Enable. The USB_INTRRXE.EP11 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
10 (R/W)	EP10	Endpoint 10 Rx Interrupt Enable. The USB_INTRRXE.EP10 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
9 (R/W)	EP9	Endpoint 9 Rx Interrupt Enable. The USB_INTRRXE.EP9 bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 28-64: USB_INTRRXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	EP8	Endpoint 8 Rx Interrupt Enable. The `USB_INTRRXE.EP8` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
7 (R/W)	EP7	Endpoint 7 Rx Interrupt Enable. The `USB_INTRRXE.EP7` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	EP6	Endpoint 6 Rx Interrupt Enable. The `USB_INTRRXE.EP6` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	EP5	Endpoint 5 Rx Interrupt Enable. The `USB_INTRRXE.EP5` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	EP4	Endpoint 4 Rx Interrupt Enable. The `USB_INTRRXE.EP4` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	EP3	Endpoint 3 Rx Interrupt Enable. The `USB_INTRRXE.EP3` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	EP2	Endpoint 2 Rx Interrupt Enable. The `USB_INTRRXE.EP2` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 28-64: USB_INTRRXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	EP1	Endpoint 1 Rx Interrupt Enable. The `USB_INTRRXE.EP1` bit enables the receive interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Transmit Interrupt Register

The USB_INTRTX register indicates which interrupts are currently active for endpoint 0 and the transmit (Tx) endpoints. Note that the USB controller automatically clears this register when it is read.

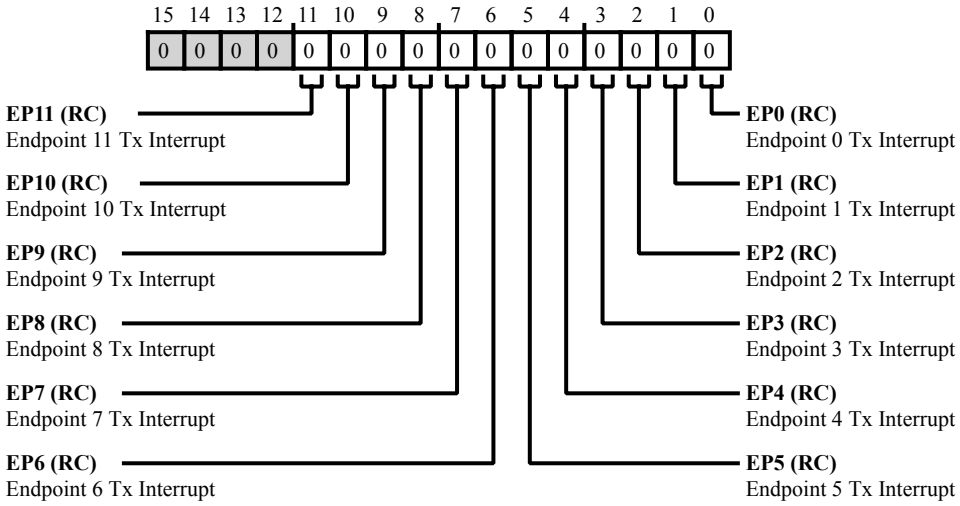

Figure 28-90: USB_INTRTX Register Diagram

Table 28-65: USB_INTRTX Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (RC/NW)	EP11	Endpoint 11 Tx Interrupt. The USB_INTRTX.EP11 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
10 (RC/NW)	EP10	Endpoint 10 Tx Interrupt. The USB_INTRTX.EP10 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
9 (RC/NW)	EP9	Endpoint 9 Tx Interrupt. The USB_INTRTX.EP9 bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 28-65: USB_INTRTX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (RC/NW)	EP8	Endpoint 8 Tx Interrupt. The `USB_INTRTX.EP8` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
7 (RC/NW)	EP7	Endpoint 7 Tx Interrupt. The `USB_INTRTX.EP7` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	EP6	Endpoint 6 Tx Interrupt. The `USB_INTRTX.EP6` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	EP5	Endpoint 5 Tx Interrupt. The `USB_INTRTX.EP5` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
4 (RC/NW)	EP4	Endpoint 4 Tx Interrupt. The `USB_INTRTX.EP4` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	EP3	Endpoint 3 Tx Interrupt. The `USB_INTRTX.EP3` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Table 28-65: USB_INTRTX Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (RC/NW)	EP2	Endpoint 2 Tx Interrupt. The `USB_INTRTX.EP2` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	EP1	Endpoint 1 Tx Interrupt. The `USB_INTRTX.EP1` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending
0 (RC/NW)	EP0	Endpoint 0 Tx Interrupt. The `USB_INTRTX.EP0` bit indicates whether or not a transmit interrupt is pending for this endpoint.	
		0	No Interrupt
		1	Interrupt Pending

Transmit Interrupt Enable Register

The USB_INTRTXE register enables interrupts for endpoint 0 and the transmit (Tx) endpoints. Enabling an interrupt in this register directs the USB controller to generate an interrupt if the corresponding interrupt pending bit in the USB_INTRTX register is set.

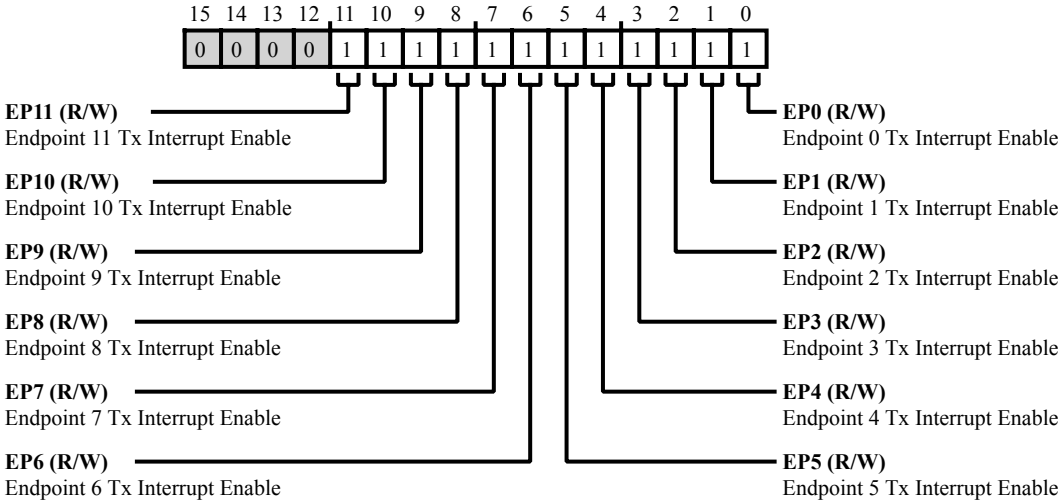

Figure 28-91: USB_INTRTXE Register Diagram

Table 28-66: USB_INTRTXE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	EP11	Endpoint 11 Tx Interrupt Enable. The USB_INTRTXE.EP11 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
10 (R/W)	EP10	Endpoint 10 Tx Interrupt Enable. The USB_INTRTXE.EP10 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
9 (R/W)	EP9	Endpoint 9 Tx Interrupt Enable. The USB_INTRTXE.EP9 bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 28-66: USB_INTRTXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	EP8	Endpoint 8 Tx Interrupt Enable. The `USB_INTRTXE.EP8` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
7 (R/W)	EP7	Endpoint 7 Tx Interrupt Enable. The `USB_INTRTXE.EP7` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
6 (R/W)	EP6	Endpoint 6 Tx Interrupt Enable. The `USB_INTRTXE.EP6` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
5 (R/W)	EP5	Endpoint 5 Tx Interrupt Enable. The `USB_INTRTXE.EP5` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	EP4	Endpoint 4 Tx Interrupt Enable. The `USB_INTRTXE.EP4` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	EP3	Endpoint 3 Tx Interrupt Enable. The `USB_INTRTXE.EP3` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	EP2	Endpoint 2 Tx Interrupt Enable. The `USB_INTRTXE.EP2` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 28-66: USB_INTRTXE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	EP1	Endpoint 1 Tx Interrupt Enable. The `USB_INTRTXE.EP1` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt
0 (R/W)	EP0	Endpoint 0 Tx Interrupt Enable. The `USB_INTRTXE.EP0` bit enables the transmit interrupt for this endpoint.	
		0	Disable Interrupt
		1	Enable Interrupt

Common Interrupts Register

The USB_IRQ register indicates which interrupts are currently active for USB controller system sources. Note that the USB controller automatically clears this register when it is read.

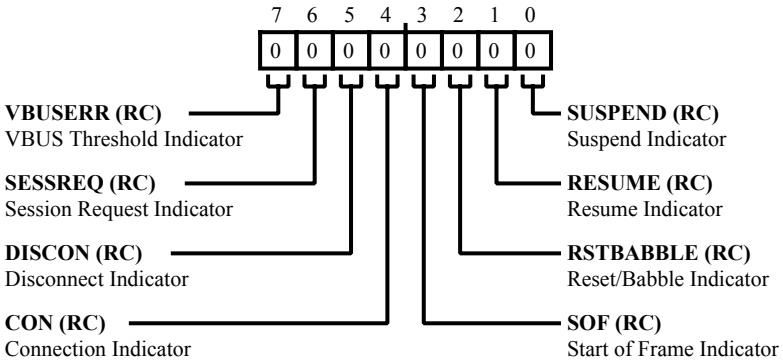

Figure 28-92: USB_IRQ Register Diagram

Table 28-67: USB_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (RC/NW)	VBUSERR	VBUS Threshold Indicator. The USB_IRQ.VBUSERR bit indicates whether the USB controller has detected that the VBUS is below the VBUS valid threshold. This bit is valid only when the USB controller is an A device. Note that the USB_IRQ.VBUSERR bit and the USB_VBUS_CTL.DRVINT bit share an interrupt source line.	
		0	No Interrupt
		1	Interrupt Pending
6 (RC/NW)	SESSREQ	Session Request Indicator. The USB_IRQ.SESSREQ bit indicates whether the USB controller has detected a session request signal. This bit is valid only when the USB controller is an A device.	
		0	No Interrupt
		1	Interrupt Pending
5 (RC/NW)	DISCON	Disconnect Indicator. The USB_IRQ.DISCON bit indicates whether the USB controller has detected a device disconnect (host mode) or has detected a session end (peripheral mode).	
		0	No Interrupt
		1	Interrupt Pending

Table 28-67: USB_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (RC/NW)	CON	Connection Indicator. The USB_IRQ.CON bit indicates whether the USB controller has detected a device connection. This bit is valid only in host mode.	
		0	No Interrupt
		1	Interrupt Pending
3 (RC/NW)	SOF	Start of Frame Indicator. The USB_IRQ.SOF bit indicates whether the USB controller has detected a start of a frame.	
		0	No Interrupt
		1	Interrupt Pending
2 (RC/NW)	RSTBABBLE	Reset/Babble Indicator. The USB_IRQ.RSTBABBLE bit indicates whether the USB controller has detected reset signalling on the bus. In host mode, the USB controller also indicates when the USB controller detects babble. Note that the USB_IRQ.RSTBABBLE bit is only active after the first SOF has been sent.	
		0	No Interrupt
		1	Interrupt Pending
1 (RC/NW)	RESUME	Resume Indicator. The USB_IRQ.RESUME bit indicates whether the USB controller has detected resume signaling on the bus while the USB controller is in suspend mode.	
		0	No Interrupt
		1	Interrupt Pending
0 (RC/NW)	SUSPEND	Suspend Indicator. The USB_IRQ.SUSPEND bit indicates whether the USB controller has detected suspend signalling on the bus. This bit is valid only in peripheral mode.	
		0	No Interrupt
		1	Interrupt Pending

Link Information Register

The USB_LINKINFO register specifies the PHY-related delays.

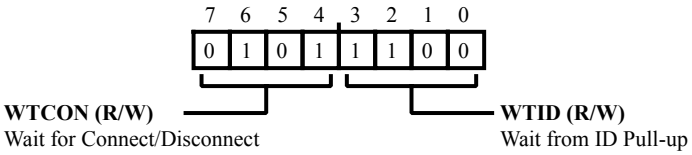

Figure 28-93: USB_LINKINFO Register Diagram

Table 28-68: USB_LINKINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/W)	WTCON	Wait for Connect/Disconnect. The USB_LINKINFO.WTCON bits set the wait time to be applied to allow the users to connect or disconnect filter.
3:0 (R/W)	WTID	Wait from ID Pull-up. The USB_LINKINFO.WTID bits set the delay to be applied from IDPULLUP being asserted to IDDIG being considered valid.

LPM Attribute Register

The USB_LPM_ATTR register defines the link power management (LPM) attributes for LPM transactions and sleep/wake operation. In peripheral mode, the USB_LPM_ATTR register contains values received in the most recent, accepted (ACK'd) LPM transaction. In host mode, the USB_LPM_ATTR register contains values (loaded by software) that set up the next LPM transaction. The USB controller inserts the LPM values within the next LPM transaction.

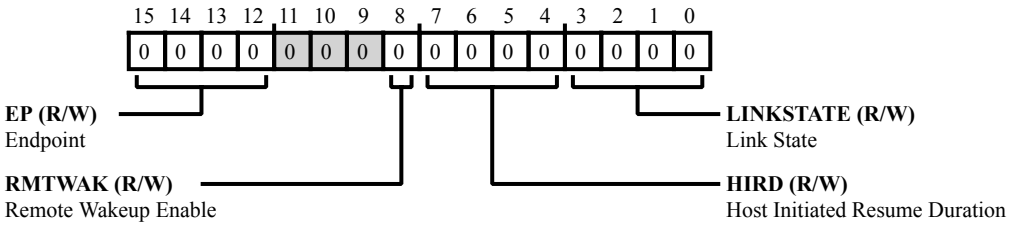

Figure 28-94: USB_LPM_ATTR Register Diagram

Table 28-69: USB_LPM_ATTR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15:12 (R/W)	EP	Endpoint. The USB_LPM_ATTR.EP bits select the endpoint in the token packet of the LPM transaction.	
8 (R/W)	RMTWAK	Remote Wakeup Enable. The USB_LPM_ATTR.RMTWAK bit enables remote wakeup. This bit is applied on a temporary basis only and is only applied to the current suspend state. After the current suspend cycle, the remote wakeup capability that was negotiated during enumeration applies.	
		0	Disable Remote Wakeup
		1	Enable Remote Wakeup
7:4 (R/W)	HIRD	Host Initiated Resume Duration. The USB_LPM_ATTR.HIRD bits select the host-initiated resume duration. This value is the minimum time that the host drives resume on the bus. The value in this register corresponds to an actual resume time of: Resume Time = 50us + HIRD*75us. This equation produces results in a range of 50us to 1200us.	
3:0 (R/W)	LINKSTATE	Link State. The USB_LPM_ATTR.LINKSTATE bits is value is provided by the host to the peripheral to indicate what state the peripheral must transition to after the receipt and acceptance of a LPM transaction. (Enumerations not shown are reserved.)	
		1	Sleep State (L1)

LPM Control Register

The USB_LPM_CTL register controls link power management (LPM) operations, including LPM enable, NAK, resume, and mode transition.

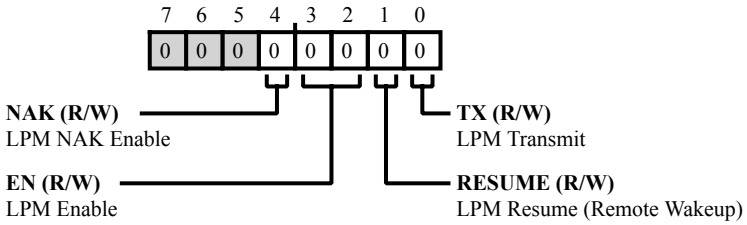

Figure 28-95: USB_LPM_CTL Register Diagram

Table 28-70: USB_LPM_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	NAK	LPM NAK Enable. The USB_LPM_CTL.NAK bit enables (in peripheral mode) a NAK-all-non-LPM transactions mode for all end points, forcing a NAK response to all transactions other than an LPM transaction. This bit only takes effect after the controller has been LPM suspended. In this case, the USB controller continues to NAK, until this bit has been cleared by software.	
		0	Disable LPM NAK
		1	Enable LPM NAK
3:2 (R/W)	EN	LPM Enable. The USB_LPM_CTL.EN bits enable (In peripheral mode) LPM operations. The LPM operation may be enabled at different levels, which determine the response of the USB controller to LPM transactions.	
		0	Disable LPM. LPM and extended transactions are not supported. The USB controller does not respond to LPM transactions, and these transaction timeout.
		1	Disable LPM. LPM and extended transactions are not supported. The USB controller does not respond to LPM transactions, and these transaction timeout.
		2	Enable Extended Transactions. LPM is not supported, but extended transactions are supported. The USB controller responds to an LPM transaction with a STALL.
		3	Enable LPM and Extended Transactions. Both LPM and extended transactions are supported. The USB controller responds with a NYET or an ACK as determined by the value of LPMXMT and other conditions.

Table 28-70: USB_LPM_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	RESUME	LPM Resume (Remote Wakeup). The USB_LPM_CTL.RESUME bit initiates resume (remote wakeup). The operation of this bit differs from the USB_POWER.RESUME bit in that the LPM resume signal timing is controlled by hardware. When set, the USB controller asserts resume signaling for 50us in host mode or asserts resume signaling for the time specified by the USB_LPM_ATTR.HIRD field in device mode. The USB_LPM_CTL.RESUME bit is self-clearing.	
		0	No Action
		1	LPM Resume
0 (R/W)	TX	LPM Transmit. The USB_LPM_CTL.TX bit puts the USB controller in LPM transmit mode. But, this mode operates differently in host mode versus peripheral mode. In peripheral mode, this bit is set by software to instruct the controller to transition to the L1 state upon receipt of the next LPM transaction. This bit is only effective if LPM enable (USB_LPM_CTL.EN) is set to 0x3. The LPM transmit enable bit can be set in the same cycle as LPM enable. If the USB_LPM_CTL.TX and USB_LPM_CTL.EN bits are enabled, the USB controller can respond in the following ways: • If no data is pending (all transmit FIFOs are empty), the USB controller responds with an ACK, clears the USB_LPM_CTL.TX bit, and generates a software interrupt. • If data is pending (data resides in at least one transmit FIFO), the USB controller responds with a NYET, does not clear the USB_LPM_CTL.TX bit, and generates a software interrupt. In host mode, this bit is set by software to transmit an LPM transaction. This bit is self-clearing. The USB controller clears this bit immediately on receipt of any token or after three timeouts have occurred.	
		0	Disable LPM Tx
		1	Enable LPM Tx

LPM Function Address Register

The USB_LPM_FADDR register selects the link power management (LPM) function address.

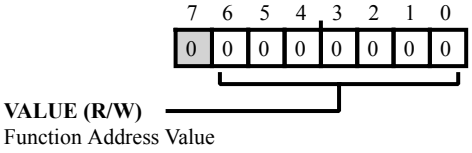

Figure 28-96: USB_LPM_FADDR Register Diagram

Table 28-71: USB_LPM_FADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Function Address Value. The USB_LPM_FADDR.VALUE bits hold the LPM function address value that the USB controller places in the LPM payload.

LPM Interrupt Enable Register

The USB_LPM_IEN register enables the link power management (LPM) related interrupts. When an interrupt is enabled in this register and the corresponding interrupt is pending in USB_LPM_IRQ, the USB controller generates the interrupt. When an interrupt is disabled in this register, the corresponding interrupt may be pending in USB_LPM_IRQ, but the USB controller does not generate an interrupt.

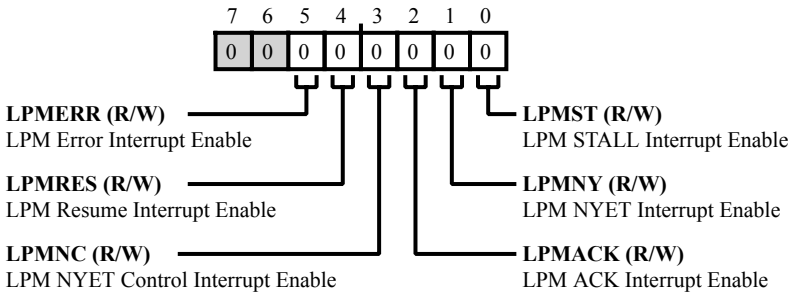

Figure 28-97: USB_LPM_IEN Register Diagram

Table 28-72: USB_LPM_IEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	LPMERR	LPM Error Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
4 (R/W)	LPMRES	LPM Resume Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
3 (R/W)	LPMNC	LPM NYET Control Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
2 (R/W)	LPMACK	LPM ACK Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt
1 (R/W)	LPMNY	LPM NYET Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt

Table 28-72: USB_LPM_IEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/W)	LPMST	LPM STALL Interrupt Enable.	
		0	Disable Interrupt
		1	Enable Interrupt

LPM Interrupt Status Register

The USB_LPM_IRQ register indicates link power management (LPM) related interrupt status. The USB controller clears this register when it is read.

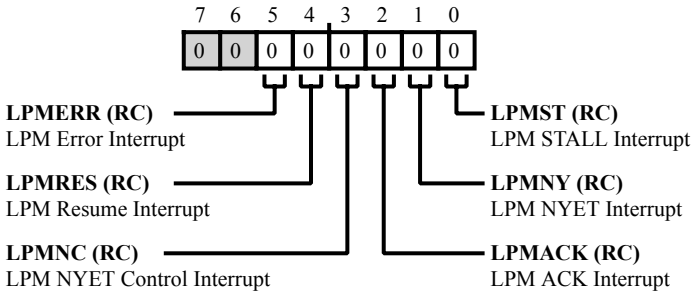

Figure 28-98: USB_LPM_IRQ Register Diagram

Table 28-73: USB_LPM_IRQ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (RC/NW)	LPMERR	LPM Error Interrupt. The USB_LPM_IRQ.LPMERR bit indicates an LPM error interrupt condition. This interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set if an LPM transaction is received that has a USB_LPM_ATTR.LINKSTATE field that is not supported. The USB controller responds to the transaction with a STALL. Note that the USB controller updates the USB_LPM_ATTR register, so software can observe the non-compliant LPM packet payload. In host mode, this bit is set if the response to a LPM transaction is received with a bit stuff or PID error. No suspend occurs and the state of the device is now unknown.	
		0	No Interrupt Pending
		1	Interrupt Pending
4 (RC/NW)	LPMRES	LPM Resume Interrupt. The USB_LPM_IRQ.LPMRES bit indicates that the USB controller has been resumed for any reason. This bit is mutually exclusive from the USB_POWER.RESUME bit.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 28-73: USB_LPM_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (RC/NW)	LPMNC	LPM NYET Control Interrupt. The USB_LPM_IRQ.LPMNC bit indicates an LPM NYET control interrupt condition. This interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with a NYET due to data pending in the transmit FIFOs. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, the USB_LPM_CTL.TX field is set to 1, and there is data pending in the transmit FIFOs. In host mode, this bit is set when an LPM transaction has been transmitted, but has failed to complete. The transaction failure is due to a timeout or bit errors in the response for three attempts.	
		0	No Interrupt Pending
		1	Interrupt Pending
2 (RC/NW)	LPMACK	LPM ACK Interrupt. The USB_LPM_IRQ.LPMACK bit indicates an LPM ACK interrupt condition. This interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with an ACK. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, the USB_LPM_CTL.TX field is set to 1, and there is no data pending in the controller transmit FIFOs. In host mode, this bit is set when an LPM transaction is transmitted, and the device responds with an ACK.	
		0	No Interrupt Pending
		1	Interrupt Pending
1 (RC/NW)	LPMNY	LPM NYET Interrupt. The USB_LPM_IRQ.LPMNY bit indicates an LPM NYET interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. In peripheral mode, this bit is set when an LPM transaction is received, and the USB controller responds with a NYET. This interrupt may only occur when the USB_LPM_CTL.EN field is set to 11, and the USB_LPM_CTL.TX field is set to 0. In host mode, this bit is set when an LPM transaction is transmitted and the device responds with a NYET.	
		0	No Interrupt Pending
		1	Interrupt Pending

Table 28-73: USB_LPM_IRQ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (RC/NW)	LPMST	LPM STALL Interrupt. The `USB_LPM_IRQ.LPMST` bit indicates an LPM STALL interrupt condition, but this interrupt has differing conditions for host mode versus peripheral mode. This bit is set when an LPM transaction is received, and the USB controller responds with a STALL. This interrupt may only occur when the `USB_LPM_CTL.EN` field is set to 01. In host mode, this bit is set when an LPM transaction is transmitted, and the device responds with a STALL.	
		0	No Interrupt Pending
		1	Interrupt Pending

Low-Speed EOF 1 Register

The USB_LS_EOF1 register defines the minimum time gap allowed between the start of the last transaction and the end of frame for low-speed transactions.

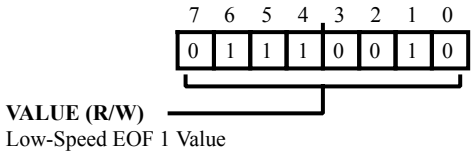

Figure 28-99: USB_LS_EOF1 Register Diagram

Table 28-74: USB_LS_EOF1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	Low-Speed EOF 1 Value. The USB_LS_EOF1.VALUE bits set the time before end of frame to stop beginning new transactions (in units of 1.067us) for low-speed transactions. The default setting corresponds to 121.6us.

MPn Receive Function Address Register

The USB_MP[n]_RXFUNCADDR register specifies the receive endpoint's target address in host mode. This register is not used in device mode. Note that the USB_MP[n]_RXFUNCADDR register does not exist for EP0.

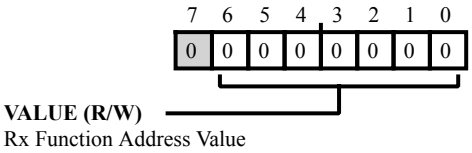

Figure 28-100: USB_MP[n]_RXFUNCADDR Register Diagram

Table 28-75: USB_MP[n]_RXFUNCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Rx Function Address Value. The USB_MP[n]_RXFUNCADDR.VALUE bits hold the address of the target device for this endpoint.

MPn Receive Hub Address Register

The USB_MP[n]_RXHUBADDR register specifies the hub address of the endpoint in host mode. This register is not used in device mode. Note that this register only needs to be programmed when a full-speed or low-speed device is connected to a high-speed hub, which carries out the necessary transaction translation. Note that the USB_MP[n]_RXHUBADDR register does not exist for EP0.

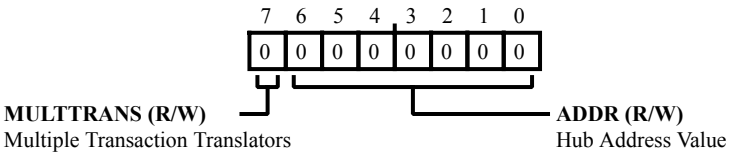

Figure 28-101: USB_MP[n]_RXHUBADDR Register Diagram

Table 28-76: USB_MP[n]_RXHUBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	MULTTRANS	Multiple Transaction Translators. The USB_MP[n]_RXHUBADDR.MULTTRANS bit should be set if the hub has multiple transaction translators.	
		0	Single Transaction Translator
		1	Multiple Transaction Translators
6:0 (R/W)	ADDR	Hub Address Value. The USB_MP[n]_RXHUBADDR.ADDR bits hold the address of the hub to which this device is connected.	

MPn Receive Hub Port Register

The USB_MP[n]_RXHUBPORT register specifies the hub port for full-speed and low-speed endpoints in host mode. This register is not used in device mode. The USB_MP[n]_RXHUBPORT register lets the USB controller support SPLIT transactions. Note that the USB_MP[n]_RXHUBPORT register does not exist for EP0.

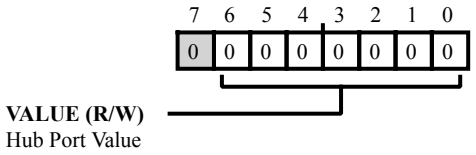

Figure 28-102: USB_MP[n]_RXHUBPORT Register Diagram

Table 28-77: USB_MP[n]_RXHUBPORT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Hub Port Value. The USB_MP[n]_RXHUBPORT.VALUE bits hold the hub port value of the target device for this endpoint.

MPn Transmit Function Address Register

The USB_MP[n]_TXFUNCADDR register specifies the transmit endpoint's target address in host mode. This register is not used in device mode. Note that the USB_MP[n]_TXFUNCADDR register must be setup for EP0. (The USB_MP[n]_RXFUNCADDR register does not exist for EP0.)

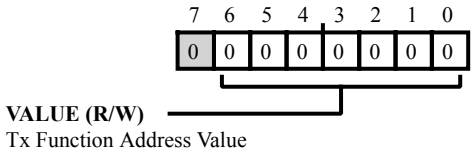

Figure 28-103: USB_MP[n]_TXFUNCADDR Register Diagram

Table 28-78: USB_MP[n]_TXFUNCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Tx Function Address Value. The USB_MP[n]_TXFUNCADDR.VALUE bits hold the address of the target device for this endpoint.

MPn Transmit Hub Address Register

The USB_MP[n]_TXHUBADDR register specifies the hub address of the endpoint in host mode. This register is not used in device mode. Note that this register only needs to be programmed when a full-speed or low-speed device is connected to a high-speed hub, which carries out the necessary transaction translation. Also, note that EP0 only uses the USB_MP[n]_TXHUBADDR register. (The USB_MP[n]_RXHUBADDR register does not exist for EP0.)

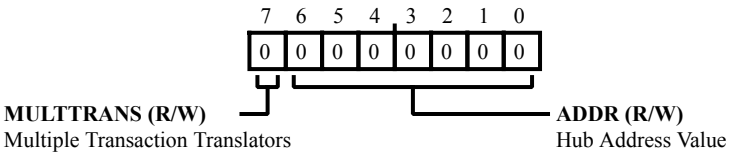

Figure 28-104: USB_MP[n]_TXHUBADDR Register Diagram

Table 28-79: USB_MP[n]_TXHUBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	MULTTRANS	Multiple Transaction Translators. The USB_MP[n]_TXHUBADDR.MULTTRANS bit should be set if the hub has multiple transaction translators.	
		0	Single Transaction Translator
		1	Multiple Transaction Translators
6:0 (R/W)	ADDR	Hub Address Value. The USB_MP[n]_TXHUBADDR.ADDR bits hold the address of the hub to which this device is connected.	

MPn Transmit Hub Port Register

The USB_MP[n]_TXHUBPORT register specifies the hub port for full-speed and low-speed endpoints in host mode. This register is not used in device mode. The USB_MP[n]_TXHUBPORT register lets the USB controller support SPLIT transactions. EP0 only uses the USB_MP[n]_TXHUBPORT register. (The USB_MP[n]_RXHUBPORT register does not exist for EP0.)

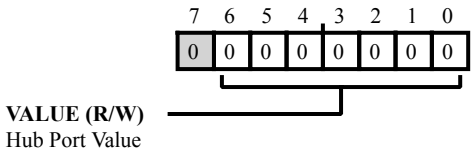

Figure 28-105: USB_MP[n]_TXHUBPORT Register Diagram

Table 28-80: USB_MP[n]_TXHUBPORT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	VALUE	Hub Port Value. The USB_MP[n]_TXHUBPORT.VALUE bits hold the hub port value of the target device for this endpoint.

PHY Control Register

The USB_PHY_CTL register provides access to PHY control features.

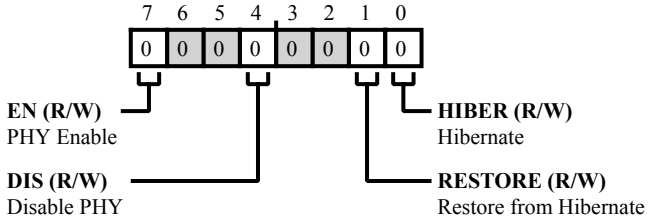

Figure 28-106: USB_PHY_CTL Register Diagram

Table 28-81: USB_PHY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	EN	PHY Enable. The USB_PHY_CTL.EN bit enables the USB controller PHY. This bit enables the schmitt-trigger inputs on D+ and D- to detect session request protocol. The bit also enables the bias circuits and VBUS comparators to detect when a host is connected. This bit should be set for all USB controller operations.	
4 (R/W)	DIS	Disable PHY. The USB_PHY_CTL.DIS bit disables the PHY, so it draws minimal power.	
		0	Enable USB PHY and 5V protection on USB signals.
		1	Disable USB PHY and 5V protection on USB signals. Caution: When 5V protection is disabled, the absolute max voltage on USB signals is reduced. See the data sheet for details.
1 (R/W)	RESTORE	Restore from Hibernate. The USB_PHY_CTL.RESTORE bit causes the PHY to come out of hibernate and release its latches.	
0 (R/W)	HIBER	Hibernate. The USB_PHY_CTL.HIBER bit causes the PHY to prepare for hibernate. Latches hold the pullup/pulldown state when the core power is removed.	

PLL and Oscillator Control Register

The USB_PLL_OSC register provides access to PLL and oscillator-related control features.

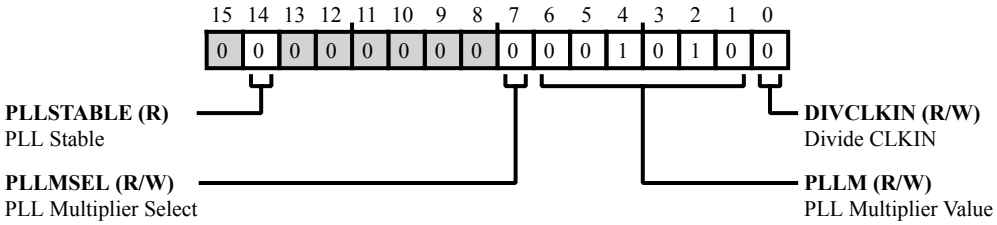

Figure 28-107: USB_PLL_OSC Register Diagram

Table 28-82: USB_PLL_OSC Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
14 (R/NW)	PLLSTABLE	PLL Stable. The USB_PLL_OSC.PLLSTABLE status bit indicates that the oscillator and PLL clock are stable.
7 (R/W)	PLLMSEL	PLL Multiplier Select. The USB_PLL_OSC.PLLMSEL bit directs the PLL to use the PLL multiplier value stored in the USB_PLL_OSC.PLLM bits.
6:1 (R/W)	PLLM	PLL Multiplier Value. The USB_PLL_OSC.PLLM bit field contains the PLL multiplier. This field should be set such that CLKIN * USB_PLL_OSC.PLLM value = 480MHz.
0 (R/W)	DIVCLKIN	Divide CLKIN. The USB_PLL_OSC.DIVCLKIN bit enables a divide CLKIN by 2 function for the PLL.

Power and Device Control Register

The USB_POWER register controls suspend and resume signaling and some operational aspects of the USB controller.

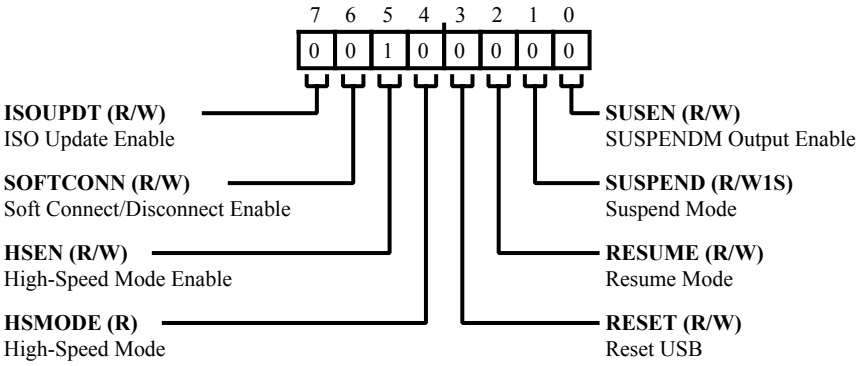

Figure 28-108: USB_POWER Register Diagram

Table 28-83: USB_POWER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	ISOUPDT	ISO Update Enable. The USB_POWER.ISOUPDT bit directs the USB controller to waits for an SOF token from the time the USB_EP[n]_TXCSR_P.TXPKTRDY bit is set before sending the packet. If an IN token is received before an SOF token, the USB controller send a zero length data packet. This USB_POWER.ISOUPDT bit only affects endpoints performing isochronous transfers. This bit only is valid in peripheral mode (USB_DEV_CTL.HOSTMODE = 0).	
		0	Disable ISO Update
		1	Enable ISO Update
6 (R/W)	SOFTCONN	Soft Connect/Disconnect Enable. In peripheral mode, the D+/- lines default to disconnected. Setting this bit enables the D+/- termination resistors. This bit is automatically set when the USB_DEV_CTL.SESSION bit is written with 1. The USB_POWER.SOFTCONN bit enables USB controller soft connect/disconnect, enabling the termination resistors for USB_DP (Data +) and USB_DM (Data -) pins. When disabled, these pins are three-stated. Note that USB_POWER.SOFTCONN only is valid in peripheral mode (USB_DEV_CTL.HOSTMODE = 0).	
		0	Disable Soft Connect/Disconnect
		1	Enable Soft Connect/Disconnect

Table 28-83: USB_POWER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	HSEN	High-Speed Mode Enable. The USB_POWER.HSEN bit enables USB controller negotiation for high speed (on devices supporting high-speed mode) when the device is reset by the hub/host. If disabled, the USB controller only operates in full-speed mode. When operating in full-speed mode, this bit should be cleared.	
		0	Disable Negotiation for HS Mode
		1	Enable Negotiation for HS Mode
4 (R/NW)	HSMODE	High-Speed Mode. The USB_POWER.HSMODE bit indicates whether or not the USB controller successfully negotiated high-speed mode during a USB controller reset. In peripheral mode (USB_DEV_CTL.HOSTMODE = 0), this bit has valid data when the USB controller completes reset. In host mode (USB_DEV_CTL.HOSTMODE = 1), this bit has valid data when the USB_IRQ.RSTBABBLE bit is cleared, remaining valid for the duration of the session.	
		0	Full-Speed Mode (HS fail during reset)
		1	High-Speed Mode (HS success during reset)
3 (R/W)	RESET	Reset USB. The USB_POWER.RESET bit indicates (in both host and peripheral modes) that the USB controller has detected that reset signaling is present on the bus. In peripheral mode (USB_DEV_CTL.HOSTMODE = 0), this bit is read only, but in host mode (USB_DEV_CTL.HOSTMODE = 1), this bit is read/write, permitting the processor core to set the bit and initiate a USB controller reset.	
		0	No Reset
		1	Reset USB
2 (R/W)	RESUME	Resume Mode. The USB_POWER.RESUME bit directs the USB controller to generate resume signaling when the function is in suspend mode (USB_POWER.SUSPEND =1). The processor core should clear this bit after 10 ms (a maximum of 15 ms) to end resume signaling. When the USB controller is in host mode (USB_DEV_CTL.HOSTMODE = 1), the USB controller automatically sets the USB_POWER.RESUME bit when resume signaling from the target is detected while the USB controller is suspended.	
		0	Disable Resume Signaling
		1	Enable Resume Signaling

Table 28-83: USB_POWER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1S)	SUSPEND	Suspend Mode. When the USB controller is in host mode (USB_DEV_CTL.HOSTMODE = 1), the USB_POWER.SUSPEND bit enables suspend mode. When the USB controller is in peripheral mode (USB_DEV_CTL.HOSTMODE = 0), the USB controller sets the USB_POWER.SUSPEND bit on entry to suspend mode and clears the bit when the processor reads the USB_IRQ register. Note that the USB controller automatically clears this bit if the USB_POWER.RESUME bit is set.	
		0	Disable Suspend Mode (Host)
		1	Enable Suspend Mode (Host)
0 (R/W)	SUSEN	SUSPENDM Output Enable. The USB_POWER.SUSEN bit enables the SUSPENDM output (internal USB controller signal). When enabled, the SUSPENDM output signal is used by the USB controller PHY to power-down its drivers when the USB controller is not active.	
		0	Disable SUSPENDM Output
		1	Enable SUSPENDM Output

RAM Information Register

The USB_RAMINFO register provides information about the width of the USB controller RAM.

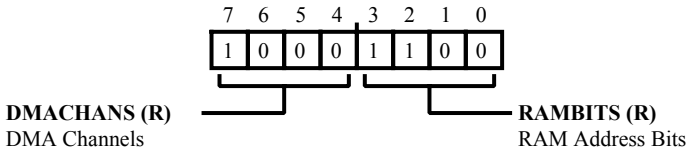

Figure 28-109: USB_RAMINFO Register Diagram

Table 28-84: USB_RAMINFO Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	DMACHANS	DMA Channels. The USB_RAMINFO.DMACHANS bits indicate the number of DMA channels.
3:0 (R/NW)	RAMBITS	RAM Address Bits. The USB_RAMINFO.RAMBITS bits indicate the number of RAM address bits. The USB controller FIFO RAM is 32-bits wide. The number of bytes in the FIFO RAM may be calculated from the formula: RAM_bytes = $2^{(RAM\_Bits+2)}$

EPn Request Packet Count Register

The USB_RQPKTCNT[n] register specifies (in host mode) the number of packets to request in a block transfer of one or more bulk packets of size USB_EP[n]_RXMAXP from a receive endpoint. This register only applies for receive endpoints 1 through 11 in host mode.

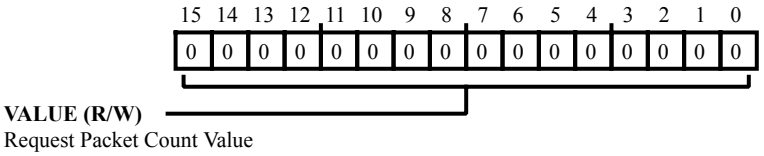

Figure 28-110: USB_RQPKTCNT[n] Register Diagram

Table 28-85: USB_RQPKTCNT[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Request Packet Count Value. The USB_RQPKTCNT[n].VALUE bits specify the number of bulk packets to request in a block transfer from a receive endpoint. This field is used with the auto request feature (USB_EP[n]_RXCSR_H.AUTOREQ).

Receive FIFO Address Register

The USB_RXFIFOADDR sets the start address for the selected Rx FIFO for endpoints 1-11. There is one receive FIFO address register for each endpoint, except endpoint 0. The USB_RXFIFOADDR register is indexed and selected by the USB_INDEX register. Note that the endpoint 0 FIFO has a fixed 64-byte size and is always located at address 0.

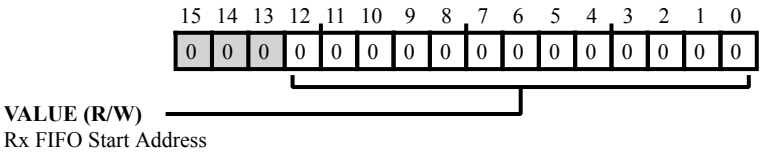

Figure 28-111: USB_RXFIFOADDR Register Diagram

Table 28-86: USB_RXFIFOADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	VALUE	Rx FIFO Start Address. The USB_RXFIFOADDR.VALUE bits hold the start address of the selected endpoint FIFO (selected with the USB_INDEX register) in units of 8 bytes, according to the formula: FIFO address = USB_RXFIFOADDR.VALUE * 8

Receive FIFO Size Register

The USB_RXFIFOSZ register defines the maximum amount of data that can be transferred through the selected receive endpoint in a single frame. When setting this value, consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. This register provides indexed-access to the FIFO (packet) size selection for each Rx endpoint (except endpoint 0).

Note that a value greater than the maximum allowed of 1023 for full-speed USB controller operation produces unpredictable results.

Also, note that the value written to this register should match the programmed maximum individual packet size (MaxPktSize) of the standard endpoint descriptor for the associated endpoint. (See the Universal Serial Bus Specification Revision 2.0, Chapter 9). A mismatch could cause unexpected results. The total amount of data represented by the value written to this register must not exceed the Rx FIFO size, and should not exceed half the FIFO size if double-buffering is required.

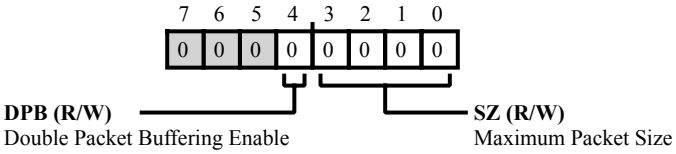

Figure 28-112: USB_RXFIFOSZ Register Diagram

Table 28-87: USB_RXFIFOSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DPB	Double Packet Buffering Enable. The USB_RXFIFOSZ.DPB bit enables double packet buffering, doubling the FIFO (packet) size selected with the USB_RXFIFOSZ.SZ field.	
		0	Single Packet Buffering
		1	Double Packet Buffering
3:0 (R/W)	SZ	Maximum Packet Size. The USB_RXFIFOSZ.SZ bits select the maximum FIFO (packet) size according to the formula: FIFOSZ= $2^{(SZ+3)}$ If the USB_RXFIFOSZ.DPB is cleared, the FIFO size is FIFOSZ from this formula. If the USB_RXFIFOSZ.DPB is set, the FIFO is twice this size. For each enumeration value, the enumerations descriptions show the packet size (PktSz=), the FIFO size if DPB=0 (DPB0=), and the FIFO size if DPB=1 (DPB1=); these values are in bytes.	
		0	PktSz=8, DPB0=8, DPB1=16
		1	PktSz=16, DPB0=16, DPB1=32

Table 28-87: USB_RXFIFOSZ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		2	PktSz=32, DPB0=32, DPB1=64
		3	PktSz=64, DPB0=64, DPB1=128
		4	PktSz=128, DPB0=128, DPB1=256
		5	PktSz=256, DPB0=256, DPB1=512
		6	PktSz=512, DPB0=512, DPB1=1024
		7	PktSz=1024, DPB0=1024, DPB1=2048
		8	PktSz=2048, DPB0=2048, DPB1=4096
		9	PktSz=4096, DPB0=4096, DPB1=8192

Software Reset Register

The USB_SOFT_RST register provides reset controls for the USB controller CLK domain and XCLK domain. The USB controller PHY operates in the controller's XCLK domain, and the USB controller interface to the processor core operates in the controller's CLK domain. Note that for correct operation, both of the reset control bits (USB_SOFT_RST.RST and USB_SOFT_RST.RSTX) should always be asserted simultaneously.

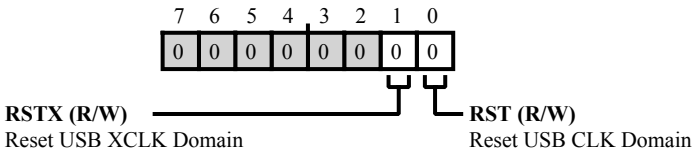

Figure 28-113: USB_SOFT_RST Register Diagram

Table 28-88: USB_SOFT_RST Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	RSTX	Reset USB XCLK Domain. The USB_SOFT_RST.RSTX bit resets logic in the USB XCLK domain. This bit is self-clearing. Note that this bit should always be asserted simultaneously with the USB_SOFT_RST.RST bit.	
		0	No Reset
		1	Reset USB XCLK Domain
0 (R/W)	RST	Reset USB CLK Domain. The USB_SOFT_RST.RST bit resets logic in the USB CLK domain. This bit is self-clearing. Note that this bit should always be asserted simultaneously with the USB_SOFT_RST.RSTX bit.	
		0	No Reset
		1	Reset USB CLK Domain

Testmode Register

The USB_TESTMODE register places the USB controller into the test mode state and can also put the USB controller into one of the test modes for high-speed operation. For more information about these modes, see the USB 2.0 specification.

Note that the USB_TESTMODE register is not used in normal operation. Only one of the test mode (bits 0-6) selection bits may be set at a time.

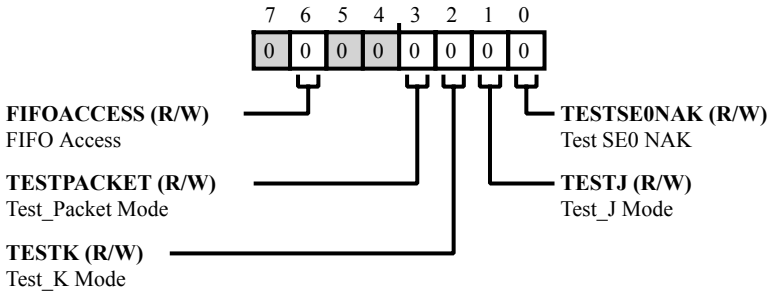

Figure 28-114: USB_TESTMODE Register Diagram

Table 28-89: USB_TESTMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6 (R/W)	FIFOACCESS	FIFO Access. The USB_TESTMODE bit directs the USB controller to transfer the packet in the endpoint 0 Tx FIFO to the endpoint 0 Rx FIFO. The bit is cleared automatically.
3 (R/W)	TESTPACKET	Test_Packet Mode. The USB_TESTMODE.TESTPACKET bit selects Test_Packet test mode, which applies only when the USB controller is in high-speed mode. In this mode, the USB controller repetitively transmits on the bus a 53-byte test packet, whose form is defined in the USB 2.0 Specification, Section 7.1.20. Note that the test packet has a fixed format and must be loaded into the endpoint 0 FIFO before this test mode is entered.
2 (R/W)	TESTK	Test_K Mode. The USB_TESTMODE.TESTK bit selects Test_K test mode. In this mode, the USB controller transmits a continuous K on the bus.
1 (R/W)	TESTJ	Test_J Mode. The USB_TESTMODE.TESTJ bit selects Test_J test mode. In this mode, the USB controller transmits a continuous J on the bus.
0 (R/W)	TESTSE0NAK	Test SE0 NAK. The USB_TESTMODE.TESTSE0NAK bit selects Test_SE0_NAK test mode, which applies only when the USB controller is in high-speed mode. In this mode, the USB controller remains in high-speed mode, but responds to any valid IN token with a NAK.

Transmit FIFO Address Register

The USB_TXFIFOADDR register sets the start address for the selected Tx FIFO for endpoints 1-11. There is one transmit FIFO address register for each endpoint, except endpoint 0. The USB_TXFIFOADDR register is indexed and selected by the USB_INDEX register. Note that the endpoint 0 FIFO has a fixed 64-byte size and is always located at address 0.

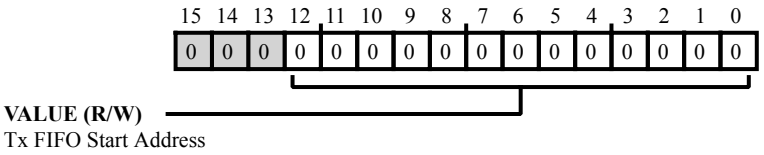

Figure 28-115: USB_TXFIFOADDR Register Diagram

Table 28-90: USB_TXFIFOADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
12:0 (R/W)	VALUE	Tx FIFO Start Address. The USB_TXFIFOADDR.VALUE bits hold the start address of the selected endpoint FIFO (selected with the USB_INDEX register) in units of 8 bytes, according to the formula: FIFO address = USB_TXFIFOADDR.VALUE * 8

Transmit FIFO Size Register

The USB_TXFIFOSZ register defines the maximum amount of data that can be transferred through the selected transmit endpoint in a single frame. When setting this value, consider the constraints placed by the USB specification on packet sizes for bulk, interrupt and isochronous transactions in full-speed operations. This register provides indexed access to the FIFO (packet) size selection for each Tx endpoint (except endpoint 0).

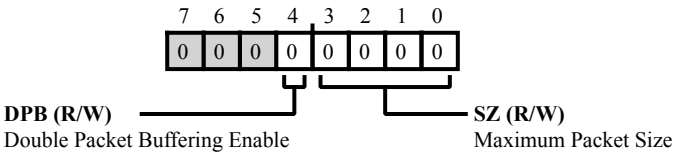

Figure 28-116: USB_TXFIFOSZ Register Diagram

Table 28-91: USB_TXFIFOSZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DPB	Double Packet Buffering Enable. The USB_TXFIFOSZ.DPB bit enables double packet buffering, doubling the FIFO (packet) size selected with the USB_TXFIFOSZ.SZ field.	
		0	Single Packet Buffering
		1	Double Packet Buffering
3:0 (R/W)	SZ	Maximum Packet Size. The USB_TXFIFOSZ.SZ bits select the maximum FIFO (packet) size according to the formula: FIFOSZ= $2^{(SZ+3)}$ If the USB_TXFIFOSZ.DPB is cleared, the FIFO size is FIFOSZ from this formula. If the USB_TXFIFOSZ.DPB is set, the FIFO is twice this size. For each enumeration value, the enumerations descriptions show the packet size (PktSz=), the FIFO size if DPB=0 (DPB0=), and the FIFO size if DPB=1 (DPB1=); these values are in bytes.	
		0	PktSz=8, DPB0=8, DPB1=16
		1	PktSz=16, DPB0=16, DPB1=32
		2	PktSz=32, DPB0=32, DPB1=64
		3	PktSz=64, DPB0=64, DPB1=128
		4	PktSz=128, DPB0=128, DPB1=256
		5	PktSz=256, DPB0=256, DPB1=512
		6	PktSz=512, DPB0=512, DPB1=1024
		7	PktSz=1024, DPB0=1024, DPB1=2048

Table 28-91: USB_TXFIFOSZ Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		8	PktSz=2048, DPB0=2048, DPB1=4096
		9	PktSz=4096, DPB0=4096, DPB1=8192

VBUS Control Register

The USB_VBUS_CTL controls USB controller VBUS-related features.

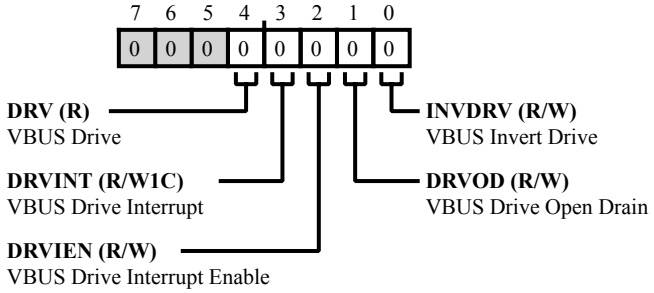

Figure 28-117: USB_VBUS_CTL Register Diagram

Table 28-92: USB_VBUS_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
4 (R/NW)	DRV	VBUS Drive. The USB_VBUS_CTL.DRV bit indicates the state of the UTMI+ DrvVBUS signal from the USB controller.
3 (R/W1C)	DRVINT	VBUS Drive Interrupt. The USB_VBUS_CTL.DRVINT bit indicates the state of the DrvVBUSInt interrupt.
2 (R/W)	DRVIEN	VBUS Drive Interrupt Enable. The USB_VBUS_CTL.DRVIEN bit enables the DrvVBUS interrupt.
1 (R/W)	DRVOD	VBUS Drive Open Drain. The USB_VBUS_CTL.DRVOD selects whether the DrvVBUS output is open drain.
0 (R/W)	INVDRV	VBUS Invert Drive. The USB_VBUS_CTL.INVDRV bit selects whether the DrvVBUS output is inverted.

VBUS Pulse Length Register

The USB_VPLEN register defines the duration of the VBUS pulsing charge for SRP initiation.

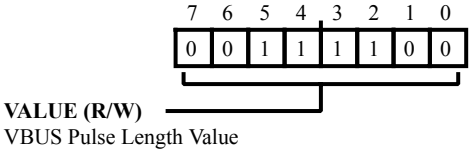

Figure 28-118: USB_VPLEN Register Diagram

Table 28-93: USB_VPLEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	VALUE	VBUS Pulse Length Value. The USB_VPLEN.VALUE bits sets the duration of the VBUS pulsing charge in units of 546.1us. The default setting corresponds to 32.77ms. Note that VBUS pulsing was removed in the OTG specification v2.0, section 5.1.4.

29 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.

ADSP-BF70x 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 29-1: ADSP-BF70x 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_MMRDH	Memory Mapped Read Header
SPI_MMTOP	SPI Memory Top Address
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

ADSP-BF70x SPI Interrupt List

Table 29-2: ADSP-BF70x SPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
36	SPI0_ERR	SPI0 Error	Level	
37	SPI0_STAT	SPI0 Status	Level	
38	SPI0_TXDMA	SPI0 TX DMA Channel	Level	4
39	SPI0_RXDMA	SPI0 RX DMA Channel	Level	5
40	SPI1_ERR	SPI1 Error	Level	
41	SPI1_STAT	SPI1 Status	Level	
42	SPI1_TXDMA	SPI1 TX DMA Channel	Level	6

Table 29-2: ADSP-BF70x SPI Interrupt List (Continued)

Interrupt ID	Name	Description	Sensitivity	DMA Channel
43	SPI1_RXDMA	SPI1 RX DMA Channel	Level	7
44	SPI2_ERR	SPI2 Error	Level	
45	SPI2_STAT	SPI2 Status	Level	
46	SPI2_TXDMA	SPI2 TX Channel	Level	8
47	SPI2_RXDMA	SPI2 RX Channel	Level	9

ADSP-BF70x SPI Trigger List

Table 29-3: ADSP-BF70x SPI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
18	SPI0_TXDMA	SPI0 TX DMA Channel	Edge
19	SPI0_RXDMA	SPI0 RX DMA Channel	Edge
20	SPI1_TXDMA	SPI1 TX DMA Channel	Edge
21	SPI1_RXDMA	SPI1 RX DMA Channel	Edge
22	SPI2_TXDMA	SPI2 TX Channel	Edge
23	SPI2_RXDMA	SPI2 RX Channel	Edge

Table 29-4: ADSP-BF70x SPI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
28	SPI0_TXDMA	SPI0 TX DMA Channel	Pulse
29	SPI0_RXDMA	SPI0 RX DMA Channel	Pulse
30	SPI1_TXDMA	SPI1 TX DMA Channel	Pulse
31	SPI1_RXDMA	SPI1 RX DMA Channel	Pulse
32	SPI2_TXDMA	SPI2 TX Channel	Pulse
33	SPI2_RXDMA	SPI2 RX Channel	Pulse

ADSP-BF70x SPI DMA Channel List

Table 29-5: ADSP-BF70x SPI DMA Channel List

DMA ID	DMA Channel Name	Description
DMA4	SPI0_TXDMA	SPI0
DMA5	SPI0_RXDMA	SPI0

Table 29-5: ADSP-BF70x SPI DMA Channel List (Continued)

DMA ID	DMA Channel Name	Description
DMA6	SPI1_TXDMA	SPI1
DMA7	SPI1_RXDMA	SPI1
DMA8	SPI2_TXDMA	SPI2
DMA9	SPI2_RXDMA	SPI2

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

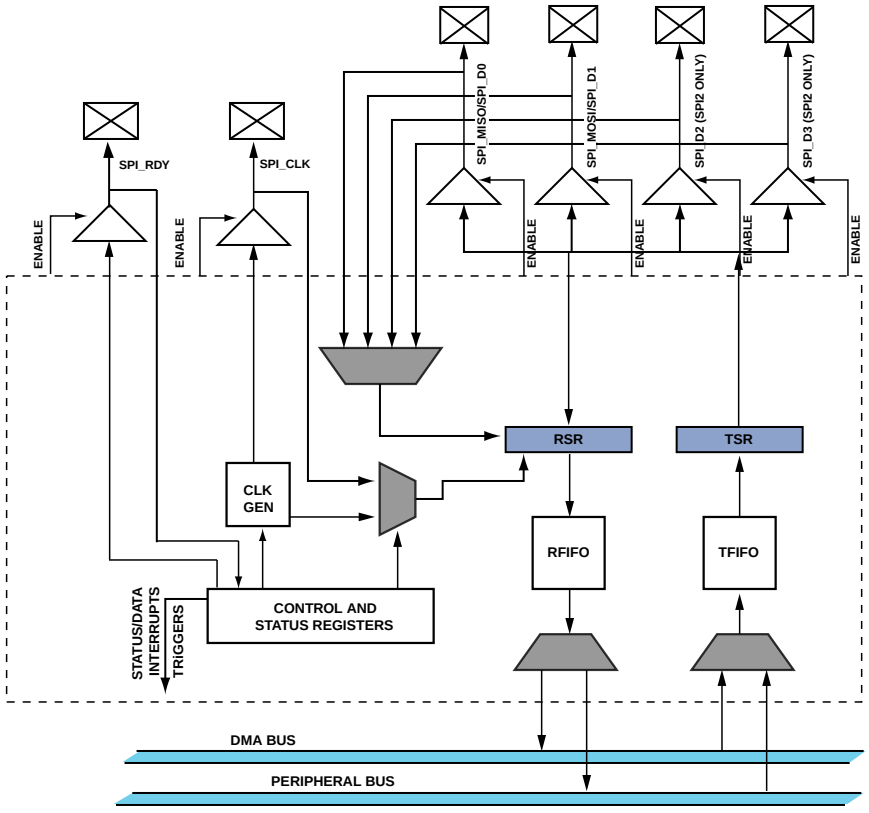

Figure 29-1: SPI Controller Block Diagram, Quad Mode

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.

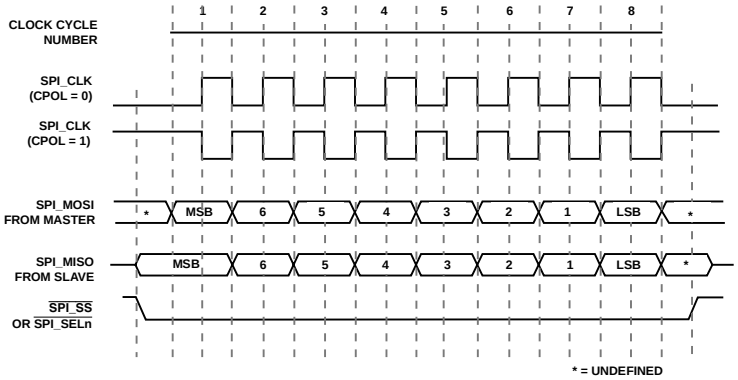

Figure 29-2: SPI Transfer Protocol for CPHA=0

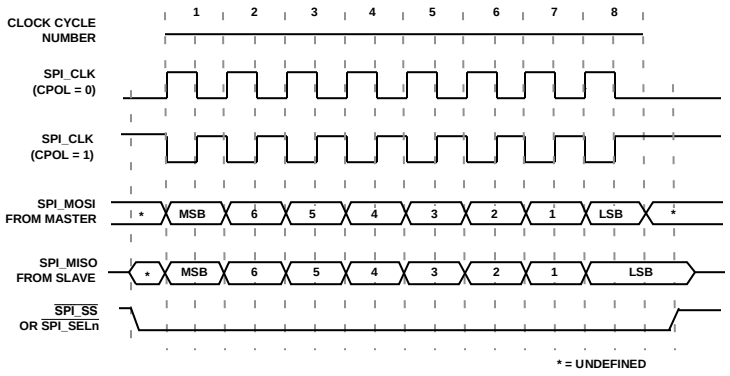

Figure 29-3: SPI Transfer Protocol for CPHA=1

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 SCLK0 rate, and both even and odd dividers from SCLK0 are supported. 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.

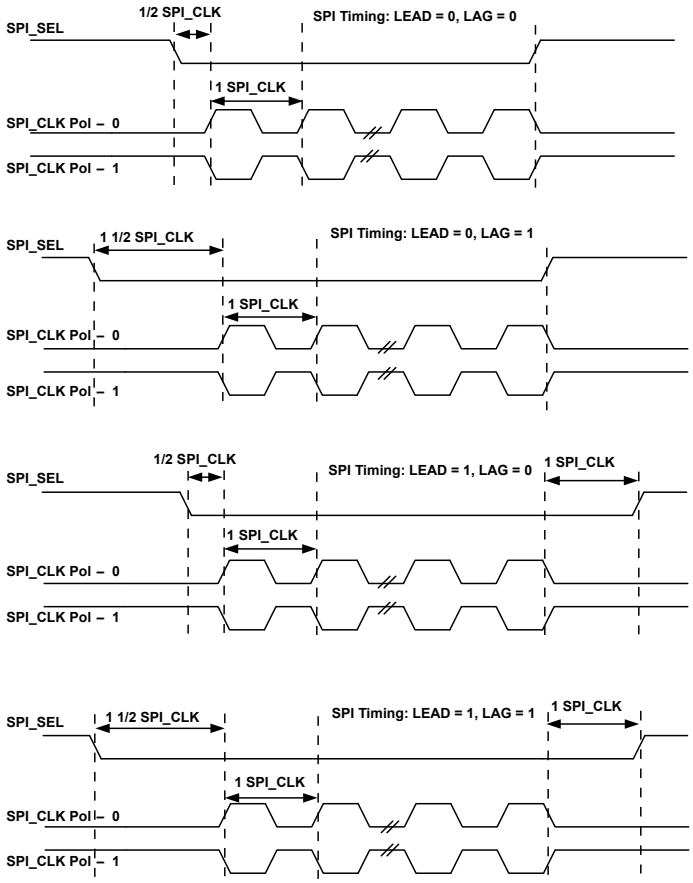

Figure 29-4: SPI Timing with Lead and Lag Programming (Independent of SPI_CTL.CPHA Setting)

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.

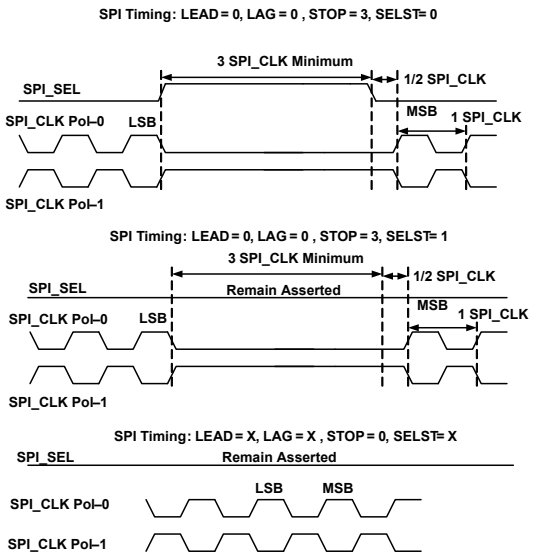

Figure 29-5: SPI Timing with SPI_DLY.STOP Programming (Independent of SPI_CTL.CPHA Setting)

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.

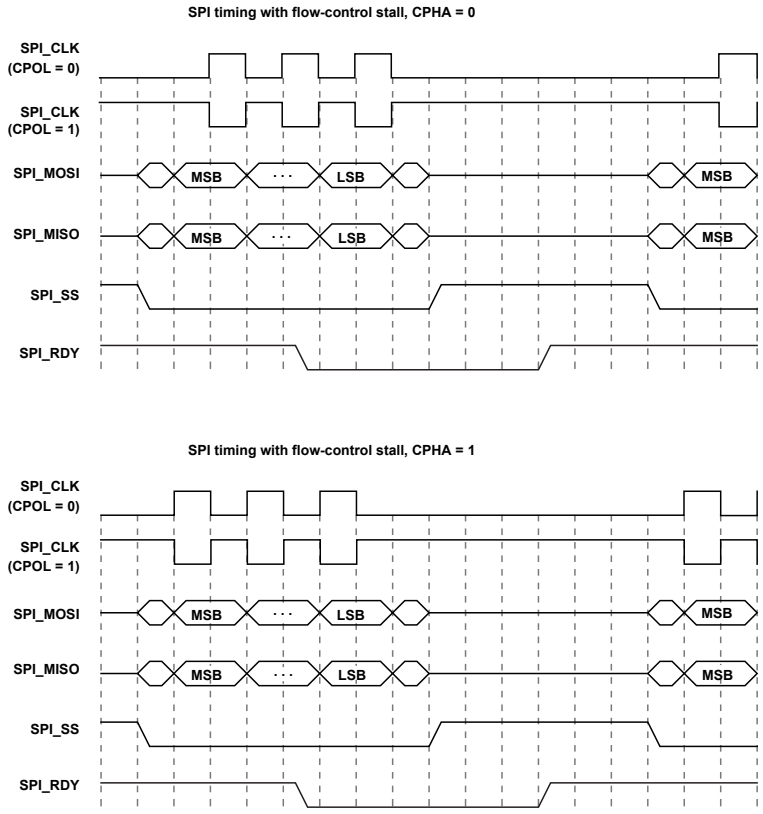

Figure 29-6: SPI Flow Control Timing in Master Mode.

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.SSE1 – 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.SSE1 – 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 EMISO feature can be available in some other microcontrollers. Therefore, it is possible to use the EMISO feature with any other SPI device that includes this functionality.

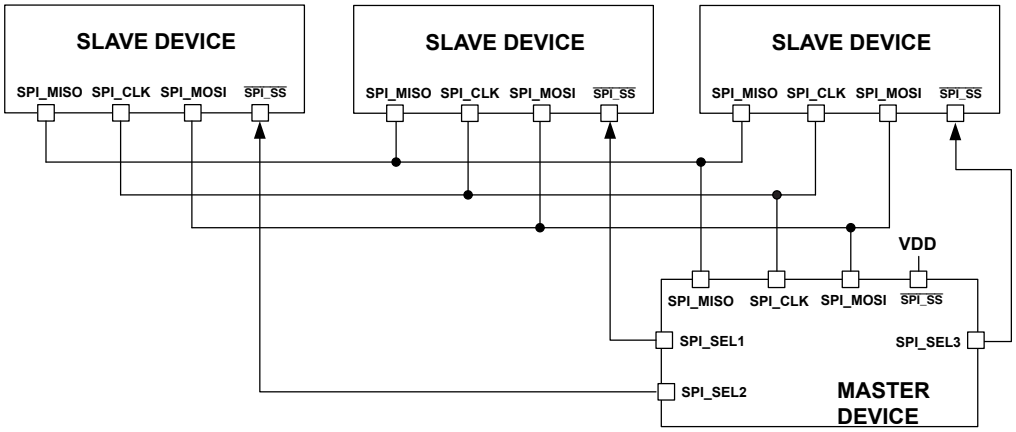

Figure 29-7: Single-Master, Multiple-Slave Configuration

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 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.

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 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.

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 generation is unaffected by dual I/O mode. However, the gap between successive interrupts 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.

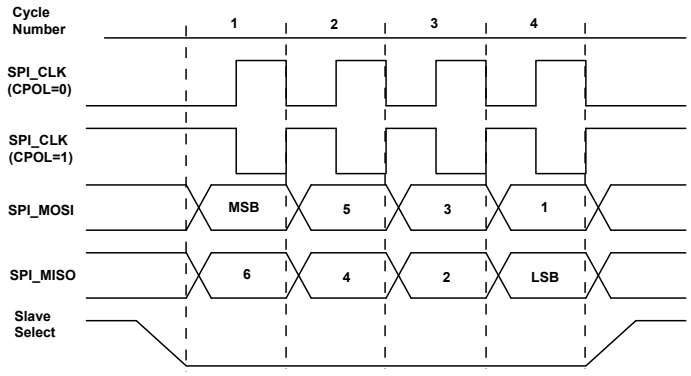

Figure 29-8: Dual I/O Mode Transfer Protocol for CPHA=0, SOSI=1, 8-Bit Transfer, LSBF=0.

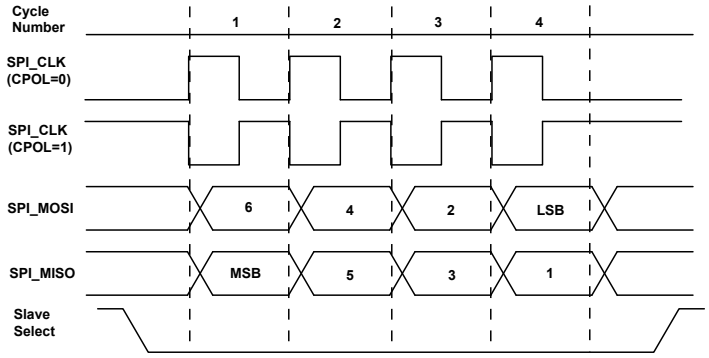

Figure 29-9: Dual I/O Mode Transfer Protocol for CPHA=1, SOSI=0, 8-Bit Transfer, LSBF=0.

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. Interrupt generation is unaffected by quad SPI mode.

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.

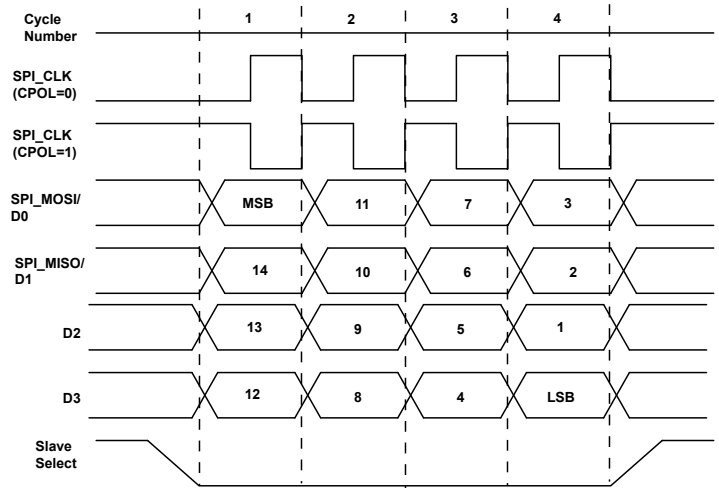

Figure 29-10: Quad Mode Timing for CPHA=0, SOSI=1, 16-Bit Transfer, LSBF=0.

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.

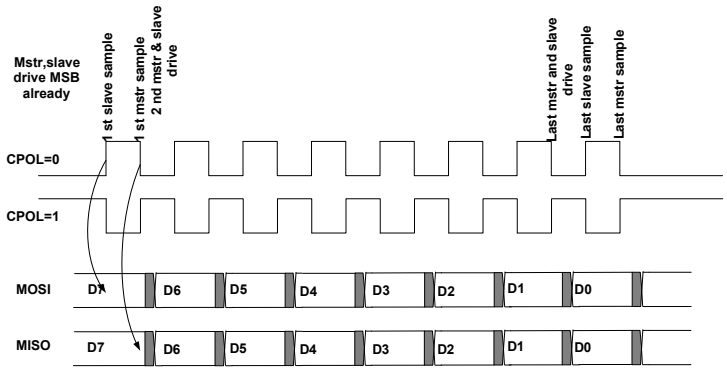

Figure 29-11: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 0

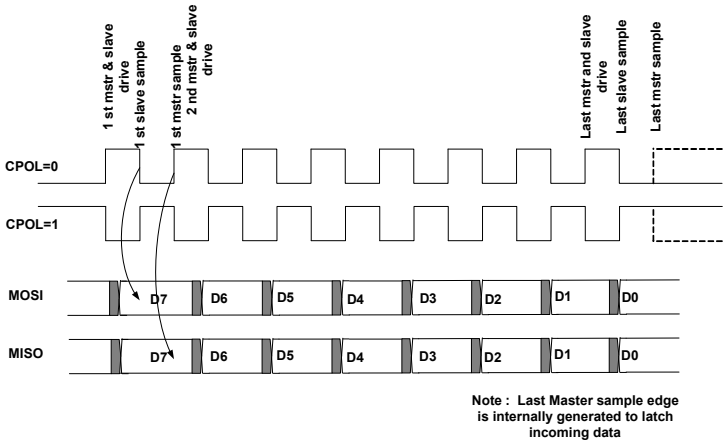

Figure 29-12: SPI Transfer Protocol in Fast Mode for SPI_CTL.CPHA = 1

Memory-Mapped Mode (SPI2 only)

The SPI supports direct memory-mapped read accesses from a SPI memory device, enabled by setting the `SPI_CTL.MMSE` bit. This mode allows for direct execution of instructions from a SPI memory device without the need for a low-level software driver, as hardware handles all overhead tasks (e.g., transmission of the read header, pin turnaround timing, and receive data sizing). The SPI features configurable options in the memory-mapped read header register (`SPI_MMRDH`) to provide compatibility with a wide range of SPI memory devices.

In non-memory-mapped mode, the software is responsible for providing the command and required dummy words for the read response, whereas this is all handled by hardware when the SPI is in memory-mapped mode. The memory of the SPI device is accessible directly through reads of the processor address space. The read accesses can be code or data accesses in core mode or when using memory DMA (MDMA). These accesses allow code to execute directly from SPI memory devices (true eXecute-In-Place operations), and the contents can be cached to improve performance. It is not necessary to access the SPI data buffer registers nor poll status bits; however, the hardware does not support peripheral DMA accesses nor write operations to the SPI memory space.

The ***Types of Operations*** table is a comparison of the permitted operations in the non-memory-mapped and memory-mapped modes supported by the SPI controller.

Table 29-6: Types of Operations

SPI Operation	Non-Memory-Mapped Mode	Memory-Mapped Mode
Core data write	Yes	No
Core data read	Yes	Yes
Code fetch: Execute-In-Place (XIP)	No	Yes
Read/Write accesses using SPI Peripheral DMA	Yes	No
Read/Write accesses by other peripheral DMA channels	No	No

Table 29-6: Types of Operations (Continued)

SPI Operation	Non-Memory-Mapped Mode	Memory-Mapped Mode
MDMA read	No	Yes
MDMA write	No	No

Memory-Mapped Description of Operation

Memory-mapped mode is enabled by setting the SPI_CTL.MMSE bit. When enabled, the SPI (if ready) accepts the read requests through a dedicated on-chip slave interface. The memory subsystem master drives this dedicated interface through the SCB fabric.

In a typical scenario, the memory subsystem master issues read requests to the fabric, and the fabric routes these requests to the slave port of the SPI peripheral. The master describes the read access using a number of parameters such as starting address, transfer size, and burst type. The SPI responds to this read access request when it is ready for a new transfer. It loads the opcode, a specified number of address bytes, and an optional mode byte into the transmit FIFO. The SPI memory state machine begins when both the transmit and receive channels of the SPI are enabled:

- the transmit transfer initiation bit is set (SPI_TXCTL.TTI=1), *and*
- the receive initiation bit is cleared (SPI_RXCTL.RTI=0)

The SPI memory read sequence starts with the assertion of SPI_SEL1. If the SPI memory state machine is in the reset state, it looks for a command. The SPI hardware then sends the specific 8-bit read command (which can be optionally skipped), followed by the SPI memory read address. After this, a dummy period is inserted, in which a mode byte is optionally sent and the pins are held or three-stated during the dummy clocking period.

NOTE: This read header is transmitted over the SPI standard protocol pins (SPI_CLK, SPI_MOSI, SPI_MISO, SPI_SEL1) or over the extended SPI protocol pins (SPI_CLK, SPI_MOSI, SPI_MISO, SPI_D2, SPI_D3, SPI_SEL1), based on the SPI_MMRDH.CMDPINS, SPI_MMRDH.ADRPINS, and SPI_CTL.MIOM bit settings. SPI memory devices usually support communication in MSB-first mode. In dual mode, the SPI typically uses SPI_MISO as IO1 and SPI_MOSI as IO0. In quad mode, the SPI typically uses SPI_D3 pin as IO3, SPI_D2 as IO2, SPI_MISO as IO1, and SPI_MOSI as IO0.

When all I/O data pins are three-stated, the SPI continues clocking the SPI memory device, which drives out the data bits at the addressed location, until all bytes are received. The SPI hardware reads the data as configured by the SPI_CTL.MIOM bit setting. Upon reception of the last byte, the SPI typically deasserts SPI_SEL1 to prepare for the next requested read header.

Application code must ensure that the opcode sent is consistent with multiple I/O programming and that the parameters specified in the memory-mapped read header register are consistent with flash read access timing.

The ***SPI Memory-Mapped Register Operations Flow*** diagram shows how the fields of the SPI_MMRDH register determine the read header while initiating transfers in memory-mapped mode.

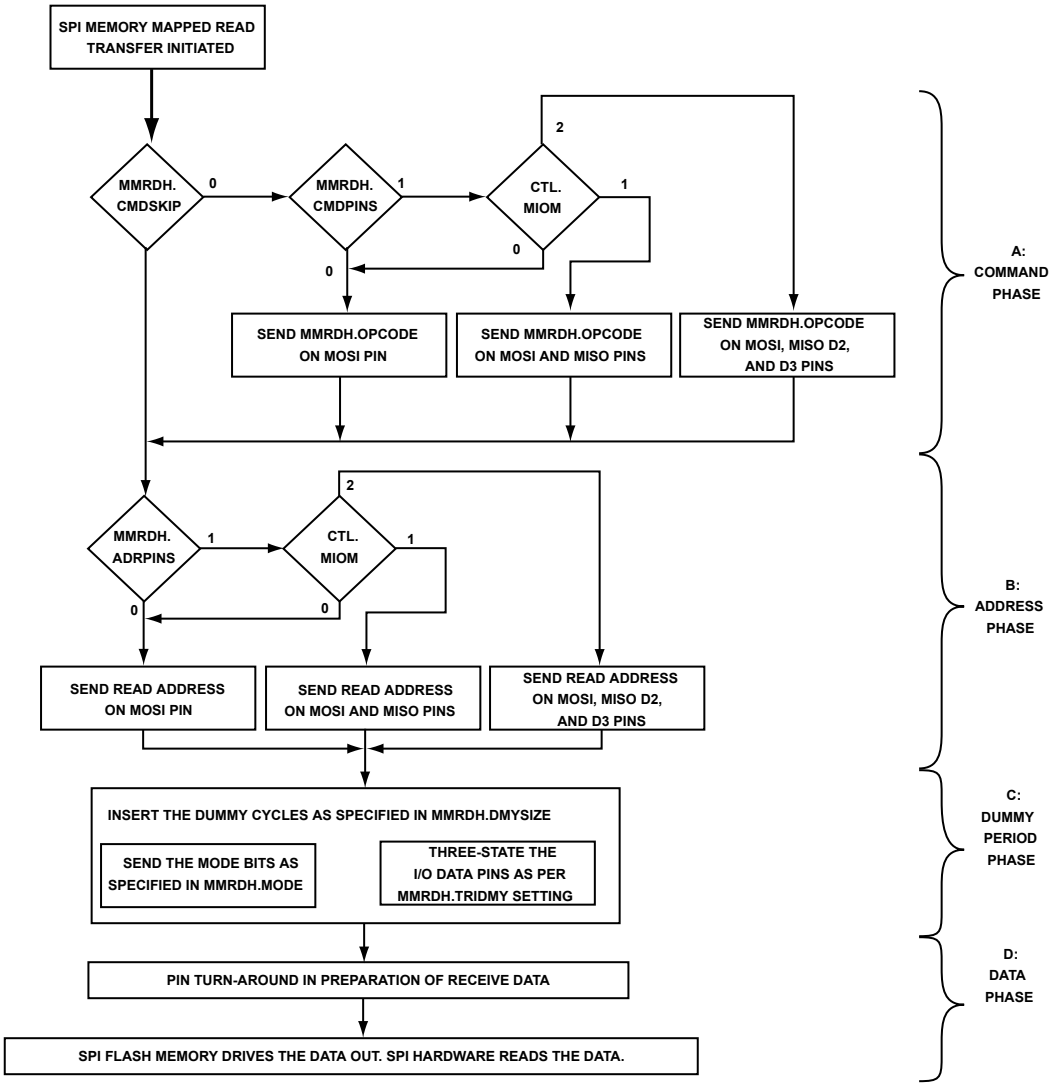

Figure 29-13: SPI Memory-Mapped Register Operations Flow

Memory-Mapped Architectural Concepts

In memory-mapped mode, the SPI accepts read requests through a dedicated on-chip slave interface. The SPI (if ready) accepts these requests and begins the process of assembling the read header based on access attributes described in both the `SPI_MMRDH` register and the internal bus request. After the read header transmission is complete, a pin turnaround period is timed and the receiver is enabled. The SPI continues clocking the SPI memory device until all bytes are received.

The SPI memory-mapped hardware accommodates various memory devices with different read timing. The capabilities include extra mode bits, flexible dummy period timing, and three-state control, as configured in the `SPI_MMRDH` register.

The ***Memory-Mapped Protocol*** figure shows the protocol for the SPI controller in memory-mapped mode.

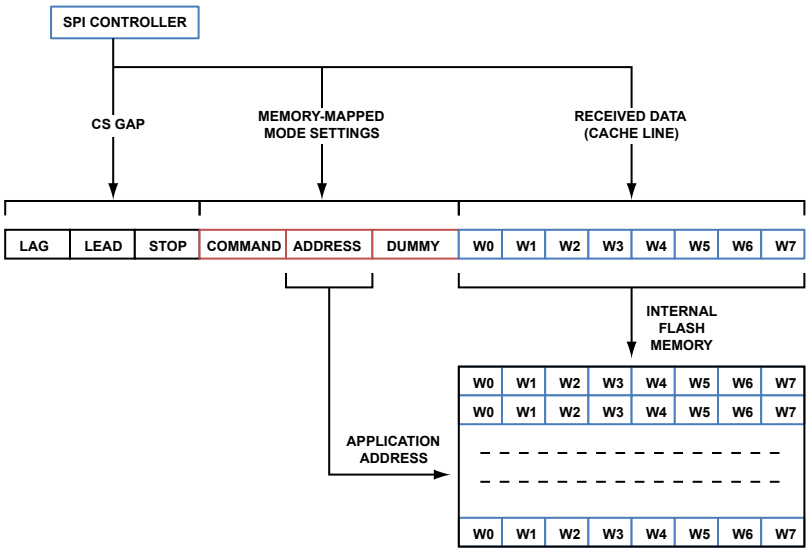

Figure 29-14: Memory-Mapped Protocol

As shown in the figure, the COMMAND field (`SPI_MMRDH.OPCODE`) is transmitted upon assertion of the `SPI_SEL[n]` signal. The SPI memory interprets this 8-bit value as a read command. Any 8-bit read opcode whose timing is compliant with the processor SPI and features provided by memory-mapped hardware is allowed, the most common being:

- Standard Read (0x03)
- Fast Read (0x0B)
- Fast Read Dual Output (0x3B)
- Fast Read Dual I/O (0x6B)
- Fast Read Quad Output (0xBB)
- Fast Read Quad I/O (0xEB)
- Word Read Quad I/O (0xE7)
- Octal Word Read Quad I/O (0xE3)

NOTE: The SPI hardware does not validate the content of the `SPI_MMRDH.OPCODE` field prior to transmitting.

DMYSIZE (Number of Dummy Bytes)

When operating at a high clock frequency in multi-IO modes, most flash devices require some dummy clocks after the address bits. These dummy clock cycles allow the internal circuits of the device extra time for setting up the initial address. These bits specify the number of bytes separating address transmission and read data return.

The number of dummy cycles required varies per manufacturer, the read command used, and the SPI access time. The SPI hardware allows dummy cycles to be programmed in bytes in the `SPI_MMRDH.DMYSIZE` field, the value of

which is a function of the number of pins used to transmit the address (SPI_MMRDH.ADRPINS), as shown in the ***Pins Used to Transmit the Address (ADRPINS)*** table.

Table 29-7: Pins Used to Transmit the Address (ADRPINS)

	Dummy clock cycles		
SPI_MMRDH.DMYSIZE	(**SPI_MMRDH.ADRPINS=0, SPI_CTL.MIOM=x**) **Dummy bytes elapse over 1-pin**	(**SPI_MMRDH.ADRPINS=1, SPI_CTL.MIOM=1**) **Dummy bytes elapse over 2-pins**	(**SPI_MMRDH.ADRPINS=1, MIOM=2**) **Dummy bytes elapse over 4-pins**
000	0	0	0
001	8	4	2
010	16	8	4
011	24	12	6
100	32	16	8
101	40	20	10
110	48	24	12
111	56	28	14

This dummy clocking period allows the mode bits to be sent, the pins to be three-stated, and the pins to be turned around in preparation for the receive data.

Memory-Mapped Read Accesses

The SPI hardware supports the most commonly used read operations.

- Two standard SPI reads (read and read fast), which use the unidirectional SPI_MOSI and SPI_MISO pins in addition to SPI_SEL[n] and SPI_CLK
- Four extended SPI multiple I/O reads: dual output, quad output, dual I/O, and quad I/O reads

The ***SPI Read Operations*** table and ***SPI Flash Fast Read Sequence*** figures summarize the types of read operations. Program each read operation in a way that is compatible with the description given in the SPI flash data sheet.

Table 29-8: SPI Read Operations

Operation	Read Command (Opcode)	CMDPIN	ADRPIN	DMYSIZE	Three-state	Multiple I/O Mode	Data Pins
Read	0x03	1	1	Zero	No	No	1
Fast Read	0x0B	1	1	Non-Zero	Yes	No	1
Dual Output Read	0x3B	1	1	Non-Zero	Yes	Yes(IO0-1)	2

Table 29-8: SPI Read Operations (Continued)

Operation	Read Command (Opcode)	CMDPIN	ADRPIN	DMYSIZE	Three-state	Multiple I/O Mode	Data Pins
Quad Output Read	0x6B	1	1	Non-Zero	Yes	Yes(IO0-3)	4
Dual I/O Read	0xBB	1, 2	2	Non-Zero	Yes	Yes (IO0-1)	2
Quad I/O Read	0xEB	1, 4	4	Non-Zero	Yes	Yes (IO0-3)	4

Some memory devices also support word quad I/O read (0xE7) and octal quad I/O read (0xE3) operations. These operations require fewer dummy cycles than normal quad I/O read operations.

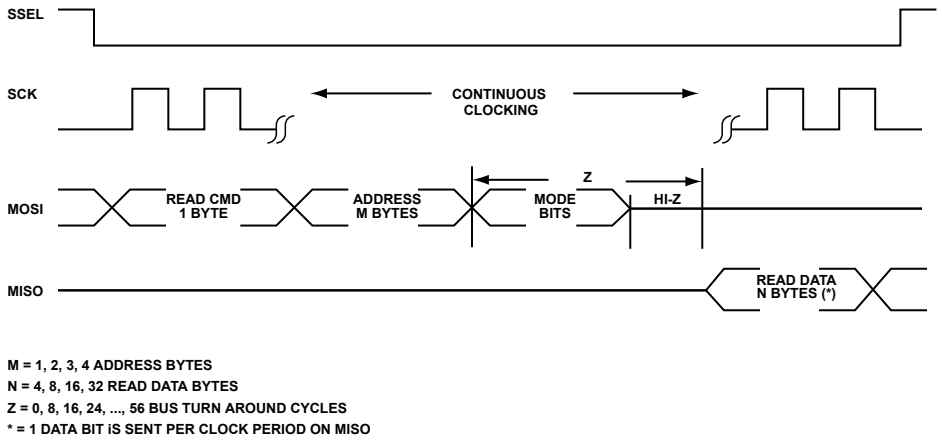

Figure 29-15: SPI Flash Fast Read Sequence

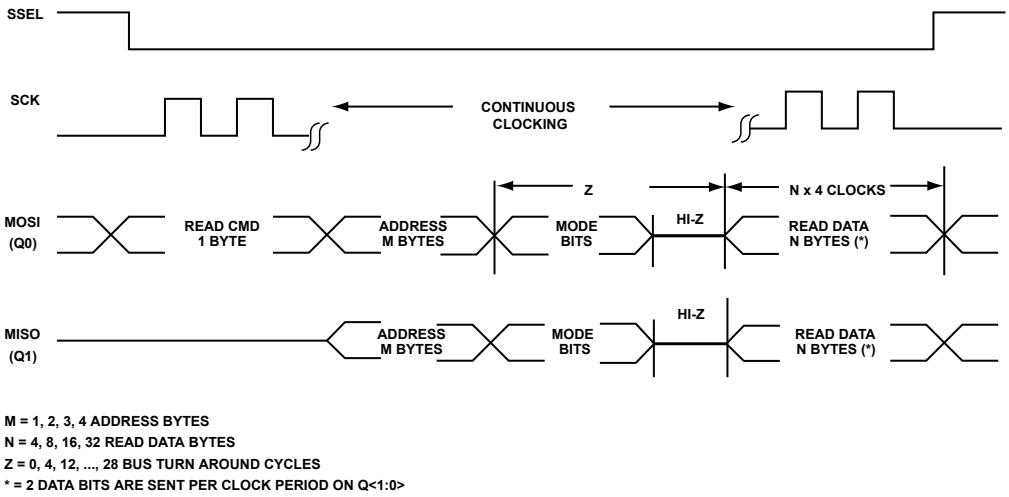

Figure 29-16: SPI Flash Fast Read (Dual Output) Sequence

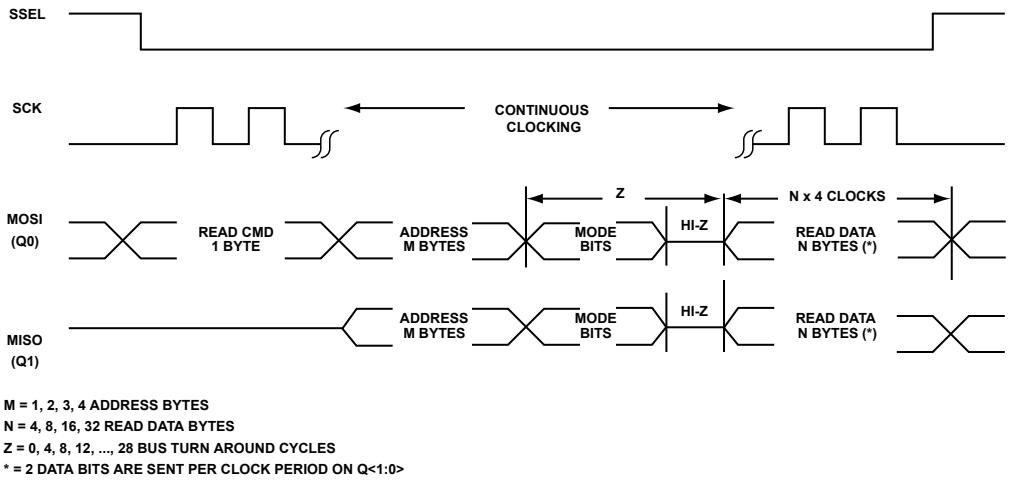

Figure 29-17: SPI Flash Fast Read (Dual I/O) Sequence

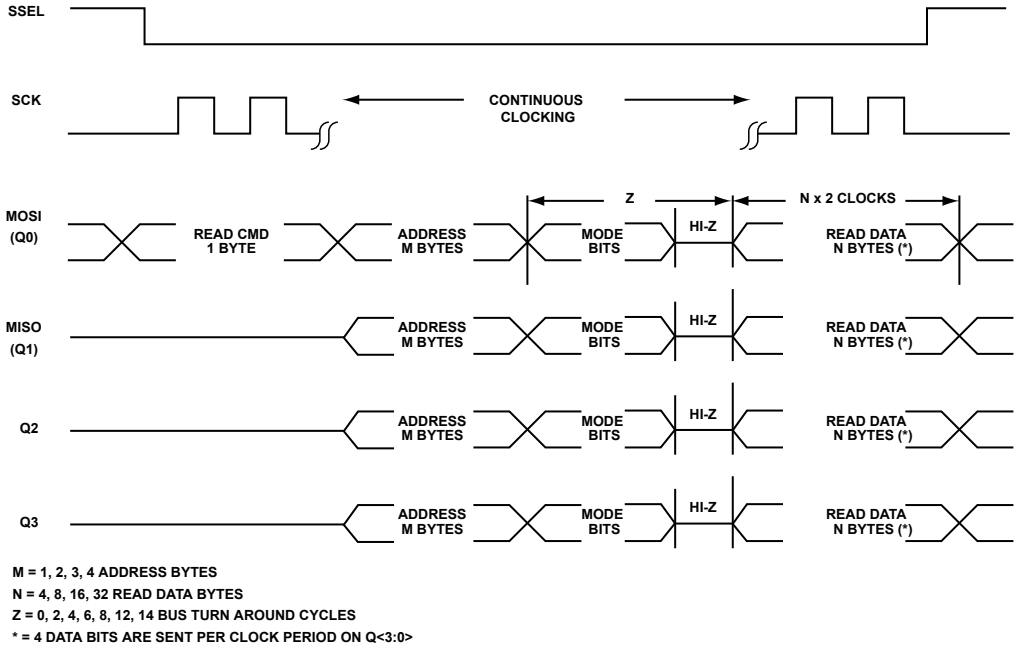

Figure 29-18: SPI Flash Quad Output Read Sequence

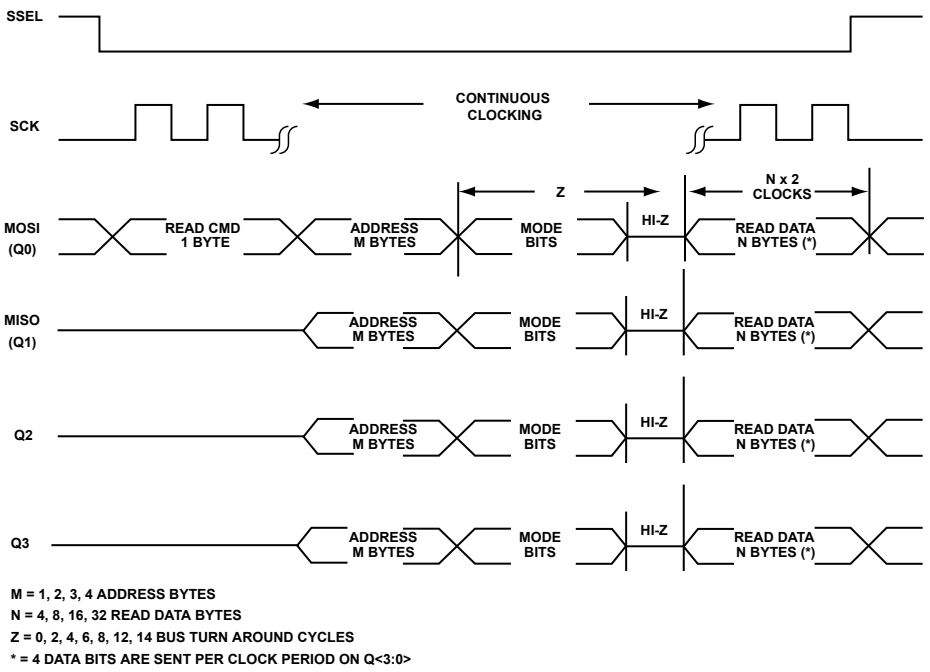

Figure 29-19: SPI Flash Quad I/O Read Sequence

SPI memory-mapped reads can be made cacheable in the core's internal memory by properly configuring the region as cacheable memory without bypass (see the related core's cache configuration documentation for details). In the figures, the number of read data bytes (N) is based on the following:

- For an instruction fetch by core (when in XIP mode); the number of instruction bytes to fetch depends the cache line size of the cache.
- For a data fetch by the core (data read), the number of data bytes to fetch depends on the cache line size of the cache.

Although the minimum size of a memory-mapped data read transfer is 4 bytes, applications can fetch a single byte or a 2-byte data. (For example, it can fetch an unsigned char or short access in C code). In this case, only the required bytes are provided to the core and the other bytes are cached.

The on-chip memory subsystem master provides a starting address for the burst and the SPI hardware issues this address as part of the read header. The address provided is N-byte aligned. For example, to read the 30th byte from SPI memory, then the typical address to provide is:

- 28 (0x0000_001C) for a 32-bit cache line
- 24 (0x0000_0018) for a 64-bit cache line
- 16 (0x0000_0010) for a 128-bit cache line
- 0 (0x0000_0000) for a 256-bit cache line

The read data is returned to the memory subsystem in the order provided by the SPI memory. There can be considerable delay for the expected data provided to the master.

- For MDMA reads, the number of read data bytes (N) is always equal to 4 bytes. The MDMA read does not depend on the cache setting. For MDMA reads, limit the DMA_CFG.MSIZE field to 1, 2 or 4 bytes. The address provided by the memory subsystem master to the SPI hardware is always 4 byte-aligned.

Memory-Mapped High-Performance Features

In addition to automating the SPI memory read accesses, the memory-mapped hardware also provides some features to improve SPI memory fetches and increase the system performance. The following sections describe these features.

Execute-In-Place (XIP, SPI2 only)

Execute-In-Place, most commonly known as XIP, allows software code to execute directly from an SPI flash device rather than downloading the code and executing it out of RAM. XIP, also known as Command Skip mode, is a general term and can be applied to fetching data as well.

There is a difference between XIP mode and standard mode. In XIP mode, after the SPI memory device is selected (CS# =LOW), the memory device does not decode the first input byte as command code. Instead, it expects the read header to directly start with address bytes. In standard mode, the memory decodes the first input byte it receives as a command code.

The XIP mode dramatically reduces random access time for applications that require fast code execution without shadowing the memory content on a RAM. The SPI memory-mapped hardware provides a control bit, SPI_MMRDH.CMDSKIP to skip the command from read header.

Some SPI memory devices require configuration of their control register to enable the XIP mode of operation, using the non-memory-mapped mode of the processor SPI. Typically, during the dummy cycle period, the mode bits are used to confirm the XIP operation and the SPI_MMRDH.MODE field must be set appropriately. A dummy memory-mapped access may be needed before setting the SPI_MMRDH.CMDSKIP bit in order to set the SPI memory device in Command Skip mode.

For more details about how to configure SPI memories into XIP mode, refer to the device data sheet.

Memory-Mapped Mode Error Status Bits

The SPI memory-mapped hardware provides bits in the SPI_STAT register to report errors. It provides these bits for notification only and their state has no effect on SPI operations. The status register bits are sticky. A W1C (write-1-to-clear) operation clears the bits.

- Memory-Mapped Write Error (SPI_STAT.MMWE). This bit is set (=1) if an attempt is made to write to address space that is reserved for memory-mapped SPI memory. The SPI memory-mapped hardware does not support automated write access to SPI memory space.
- Memory-Mapped Read Error (SPI_STAT.MMRE). This bit is set (=1) if an attempt is made to read address space reserved for memory-mapped SPI memory while memory mapping is disabled (SPI_CTL.MMSE =0).

- Memory-Mapped Access Error (SPI_STAT.MMAE). This bit is set (=1) if an attempt is made to access either the TX or RX FIFO while memory-mapped access of SPI memory is enabled. In this case, attempts to communicate with the SPI device using legacy methods are blocked and receive fabric error responses. Legacy methods include any direct access made to the TX and RX FIFOs, whether by DMA or processor MMR.
- Memory-Mapped Write Error Mask (SPI_CTL.MMWEM) bit specifies whether an error response is returned to the fabric on write attempts to address space that is reserved for memory-mapped SPI memory reads. Regardless of whether a write error response is masked using this bit, the memory-mapped write error (SPI_STAT.MMWE) sticky notification bit is still set.

  NOTE: Unlike other bits in the SPI_STAT register, these memory-mapped mode error bits do not have associated bits in the SPI interrupt mask (SPI_IMSK) and SPI interrupt condition (SPI_ILAT) registers.

The memory-mapped top register (SPI_MMTOP) is used to specify the upper limit of the SPI memory address. The memory-mapped accesses to SPI memory addresses equal to or above this range are considered illegal. The accesses are blocked and a bus error response is generated.

This register is useful to block the invalid SPI memory address accesses. Some SPI memory vendors do not clearly specify (guarantee) that overrange address bits are ignored (address spaces can be wrapped).

Memory-Mapped Programming Guidelines

Seting the SPI_CTL.MMSE bit enables SPI memory-mapped mode. When enabled, the SPI interface is forced to be consistent with SPI memory requirements regardless the settings of certain control bits. The following tables specify typical settings for configuring the SPI in memory-mapped mode:

Table 29-9: SPI Control (SPI_CTL) Register

Bits	Typical values to set	Description	Comments
SPI_CTL.MSTR	1	Master mode enable	
SPI_CTL.PSSE	0	Protected slave select enable	
SPI_CTL.ODM	0	Open-drain mode enable	
SPI_CTL.CPHASPI_CTL.CPOL	0–0 or 1–1	SPI mode of communication	Flash dependent, usually SPI flash supports mode-0 (CPHA=CPOL=0) and mode-3 (CPHA=CPOL=1)
SPI_CTL.ASSEL	1	Hardware slave select pin control	
SPI_CTL.SELST	1	Slave select asserted between transfers	
SPI_CTL.EMISO	1	MISO pin enable	
SPI_CTL.SIZE	2	32-bit transfer size	

Table 29-9: SPI Control (SPI_CTL) Register (Continued)

Bits	Typical values to set	Description	Comments
SPI_CTL.LSBF	0	MSB bit first mode	Flash dependent, usually SPI flash communicates in MSB bit first mode
SPI_CTL.FCEN SPI_CTL.FCCH SPI_CTL.FCPL SPI_CTL.FCWM	0	Hardware flow control related bits	
SPI_CTL.FMODE	1	FAST mode enable	Typically set to 1 for full cycle timing, 0 only works at low speed
SPI_CTL.SOSI	0	Treat SPI_MOSI pin as IO0 pin.	

Table 29-10: SPI Receive Control Register

Bits	Typical values to set	Description
SPI_RXCTL.REN	1	Receive channel enable
SPI_RXCTL.RTI	0	Receive transfer initiation disable
SPI_RXCTL.RWCEN	0	Receive word counter disable
SPI_RXCTL.RDR	0	Receive data request disable
SPI_RXCTL.RDO	0	Discard incoming data if RFIFO is full
SPI_RXCTL.RRWM	0	Receive FIFO regular watermark
SPI_RXCTL.RUWM	0	Receive FIFO urgent watermark disable

Table 29-11: SPI Transmit Control Register

Bits	Typical values to set	Description
SPI_TXCTL.TEN	1	Transmit channel enable
SPI_TXCTL.TTI	1	Transmit transfer initiation disable
SPI_TXCTL.TWCEN	0	Transmit word counter disable
SPI_TXCTL.TDR	0	Transmit data request disable
SPI_TXCTL.TDU	0	Send last word when TFIFO is empty
SPI_TXCTL.TRWM	0	Transmit FIFO regular watermark
SPI_TXCTL.TUWM	0	Transmit FIFO urgent watermark disable

Table 29-12: SPI DLY Control Register

Bits	Typical values to set	Description	Comments See Flash data sheet for CS (for example. SSEL) timing specs
SPI_DLY.LAGX	1	Extended LAG timing	
SPI_DLY.LEADX	1	Extended LEAD timing	
SPI_DLY.STOP	3	STOP bit between the transfers	Can be set to 1 at lower SPI clock frequencies.

The multiple I/O mode (SPI_CTL.MIOM) bits are partially ignored:

- The command (opcode) is transmitted using either just one or the number of pins specified by the SPI_CTL.MIOM bits, depending on SPI_MMRDH.CMDPINS bit setting.
- The address is then transmitted using either just one or the number of pins specified by the SPI_CTL.MIOM bits, depending on SPI_MMRDH.ADRPINS bit setting.
- The data is always read with the number of pins specified by the SPI_CTL.MIOM bits.

NOTE: Set the SPI module enable bits SPI_CTL.EN last after configuring all registers.

Use the following programming guidelines for memory-mapped mode:

- The SPI memory-mapped hardware does not check the flash status before initiating the access. It assumes that SPI memory is always able to respond to a read access. Befor enabling memory-mapped mode (for example, setting the SPI_CTL.MMSE bit) ensure that SPI flash is ready for a read access. When using non-memory-mapped mode, a write-complete status can be examined prior to enabling the SPI in memory-mapped mode. (See the write in progress bit in the SPI flash memory status register.) Also, immediately after initial power-up, SPI memory devices can be inaccessible for a vendor-specified period.
- When SPI is enabled in memory-mapped mode, attempts to communicate with the SPI device using legacy methods are blocked. Legacy methods include any direct access made to the transmit or receive FIFOs, whether initiated by DMA or by a processor MMR access.
- To use some of the features offered by SPI memory devices, programs can first configure the SPI memory device by setting its control word or sending some commands. Since SPI memory-mapped hardware does not allow any type of SPI write operations, configure the SPI in non-memory-mapped mode prior to enabling memory-mapped mode.
- The memory-mapped hardware does not interpret the opcode. It does not check the validity of the timing that is specified in the SPI_MMRDH register for a particular opcode. Programs must set the fields of the SPI_MMRDH register to be consistent with the read-type selected.
- When the core requests the data or code fetch, the memory-mapped transfer depends on cache settings. The cache configuration register in the SPI memory device must be appropriately configured before enabling memory-mapped mode.

- SPI memory-mapped MDMA reads do not support wrapping. For MDMA reads, limit the DMA_CFG.MSIZE field to 1 byte, 2 bytes or 4 bytes.
- There is not always tool support to change the SPI memory-mapped hardware setting or cache settings on-the-fly. Changing these settings can optimize the performance of code that accesses SPI memory in memory-mapped mode. It is expected that the SPI memory, SPI peripheral, and cache are programmed to one specific set of control settings for the whole application. Profiling or benchmarking of the actual application can be done to find the setting that works best.

Programming Example for Configuring SPI Memory Mapped Mode (Winbond W25Q32)

This example shows how to set XIP for Quad I/O mode of Winbond W25Q32 SPI flash (assuming SPI memory device is connected externally to SPI0 module):

• Setting XIP mode (Volatile approach)

```
/* Configure the SPI Flash device for XIP mode and Quad mode by configuring the
non-volatile QE bit of the flash status register.

/* Enable the SPI module in memory-mapped mode as shown in listing2. Make sure that Mode
bits are 0x00*/

/* To enable the command skip (XIP) mode, set the MODE bits of SPI Memory mapped
Read Header register and perform a dummy access to SPI flash. */

    *pREG_SPI2_MMRDH |=    ((0x20     << BITP_SPI_MMRDH_MODE) & BITM_SPI_MMRDH_MODE    );
    unsigned int* pSPI_MEM;
    pSPI_MEM = (unsigned int*) 0x40000000;
    int temp = *pSPI_MEM;

/* After the Dummy access, set the Command Skip bit of SPI Memory Mapped Read Header
register */
*pREG_SPI0_MMRDH |= BITM_SPI_MMRDH_CMDSKIP;

/* All the SPI flash read accesses after this, would be in XIP mode*/
```

• SPI Initialization

```
/* Configure the system clock */

/* Initialise the Pin Mux logic of processor to enable SPI pins*/
    Init_SPI_PinMux();

/* Configure the SPI Baud and SPI clock setting....*/
      /* SPI clock register: SPI_CLK = SCLK / (BAUD + 1)*/
    *pREG_SPI0_CLK = (    (0x01 << BITP_SPI_CLK_BAUD) & BITM_SPI_CLK_BAUD );

    /* SPI Delay register: STOP = 3; LEAD = 1; LAG = 1*/
```

```
    *pREG_SPI0_DLY = (((3 << BITP_SPI_DLY_STOP) & BITM_SPI_DLY_STOP) |
            ((1 << BITP_SPI_DLY_LEADX ) & BITM_SPI_DLY_LEADX)|
            ((1 << BITP_SPI_DLY_LAGX  ) & BITM_SPI_DLY_LAGX ));

/* Enable the Memory Mapped mode of SPI with following settings:
Master mode, 32-bit transfer size, HW Slave select, MSB bit first, Quad Multi-IO mode, FAST
enable. CPHA-CPOL settings depends on FLASH device*/
    *pREG_SPI0_CTL =(ENUM_SPI_CTL_MM_EN |
            ENUM_SPI_CTL_MASTER |
            ENUM_SPI_CTL_SIZE32 |
            ENUM_SPI_CTL_HW_SSEL|
            ENUM_SPI_CTL_ASSRT_SSEL |
            ENUM_SPI_CTL_MSB_FIRST  |
            ENUM_SPI_CTL_FAST_EN  |
            ENUM_SPI_CTL_MIO_QUAD |
            ((CPHA << BITP_SPI_CTL_CPHA) & BITM_SPI_CTL_CPHA)|
            ((CPOL << BITP_SPI_CTL_CPOL) & BITM_SPI_CTL_CPOL));

/* Enable the Transmit part of SPI with TTI = 1; other bits should be kept to default
values*/
    *pREG_SPI0_TXCTL = (BITM_SPI_TXCTL_TTI | BITM_SPI_TXCTL_TEN);

/* Enable the Receive part of SPI with RTI = 0; other bits should be kept to default
values*/
    *pREG_SPI0_RXCTL = (BITM_SPI_RXCTL_REN );

/* Configure the Memory Mapped Read Header as per read mode selected*/
    *pREG_SPI0_MMRDH =(
        ((0xE7<< BITP_SPI_MMRDH_OPCODE)& BITM_SPI_MMRDH_OPCODE)|
        ((3 << BITP_SPI_MMRDH_ADRSIZE) & BITM_SPI_MMRDH_ADRSIZE)|
        ((1 << BITP_SPI_MMRDH_ADRPINS) & BITM_SPI_MMRDH_ADRPINS)|
        ((2 << BITP_SPI_MMRDH_DMYSIZE) & BITM_SPI_MMRDH_DMYSIZE)|
        ((0 << BITP_SPI_MMRDH_MODE )   & BITM_SPI_MMRDH_MODE)|
        ((1 << BITP_SPI_MMRDH_TRIDMY ) & BITM_SPI_MMRDH_TRIDMY)    |
        ((0 << BITP_SPI_MMRDH_CMDSKIP) & BITM_SPI_MMRDH_CMDSKIP)|
        ((0 << BITP_SPI_MMRDH_CMDPINS) & BITM_SPI_MMRDH_CMDPINS));

/* Top addr of Flash device*/
    *pREG_SPI0_MMTOP = 0x40000000 + (FLASH_BLOCK_SIZE * FLASH_BLOCK_COUNT);

/* Slave Select Control reg*/
    *pREG_SPI0_SLVSEL= ENUM_SPI_SLVSEL_SSEL1_HI | ENUM_SPI_SLVSEL_SSEL1_EN;

/* Enable SPI*/
 *pREG_SPI0_CTL |= BITM_SPI_CTL_EN;
```

SPI Interrupt Signals

The SPI controller supports three types of interrupt 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.

Each of the data interrupts 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 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 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 interrupts apart from the usual data request interrupts. Transmit interrupts operate independently from the word counter-value in the SPI_TWC register.

Status Interrupts

The SPI controller supports several status interrupts to indicate different conditions of the receiver and transmitter. All status interrupts can be masked. Status interrupts 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 interrupts that are available for the SPI controller.

Table 29-13: SPI Status Interrupts

SPI_STAT Bit	Description
SPI_STAT.RUWM	Receive FIFO urgent watermark interrupt. 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	Receive FIFO urgent watermark interrupt. 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. 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. 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. Issued when a transmit burst completes (SPI_TWC decrements to zero).
SPI_STAT.RF	Receive finish interrupt. Issued when a receive burst completes (SPI_RWC decrements to zero).

Error Conditions

The SPI controller supports interrupts upon several different error conditions. All interrupts are maskable. The individual interrupt 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 and interrupts 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 and interrupts from both channels are enabled.

Table 29-14: 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 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.

Take precautions to 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 slaves are 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 until the SPI_RWC counts to zero. 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 until the SPI_TWC register value counts down to 0. 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).

ADSP-BF70x SPI Register Descriptions

Serial Peripheral Interface (SPI) contains the following registers.

Table 29-15: ADSP-BF70x 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_MMRDH	Memory Mapped Read Header
SPI_MMTOP	SPI Memory Top Address
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

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 (SCLK0).

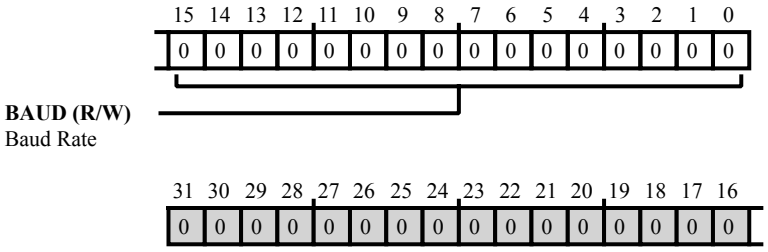

Figure 29-20: SPI_CLK Register Diagram

Table 29-16: 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 = (SCLK0 / SPI Clock) - 1

Control Register

The SPI_CTL register enables the SPI and configures settings for operating modes, communication protocols, and buffer operations.

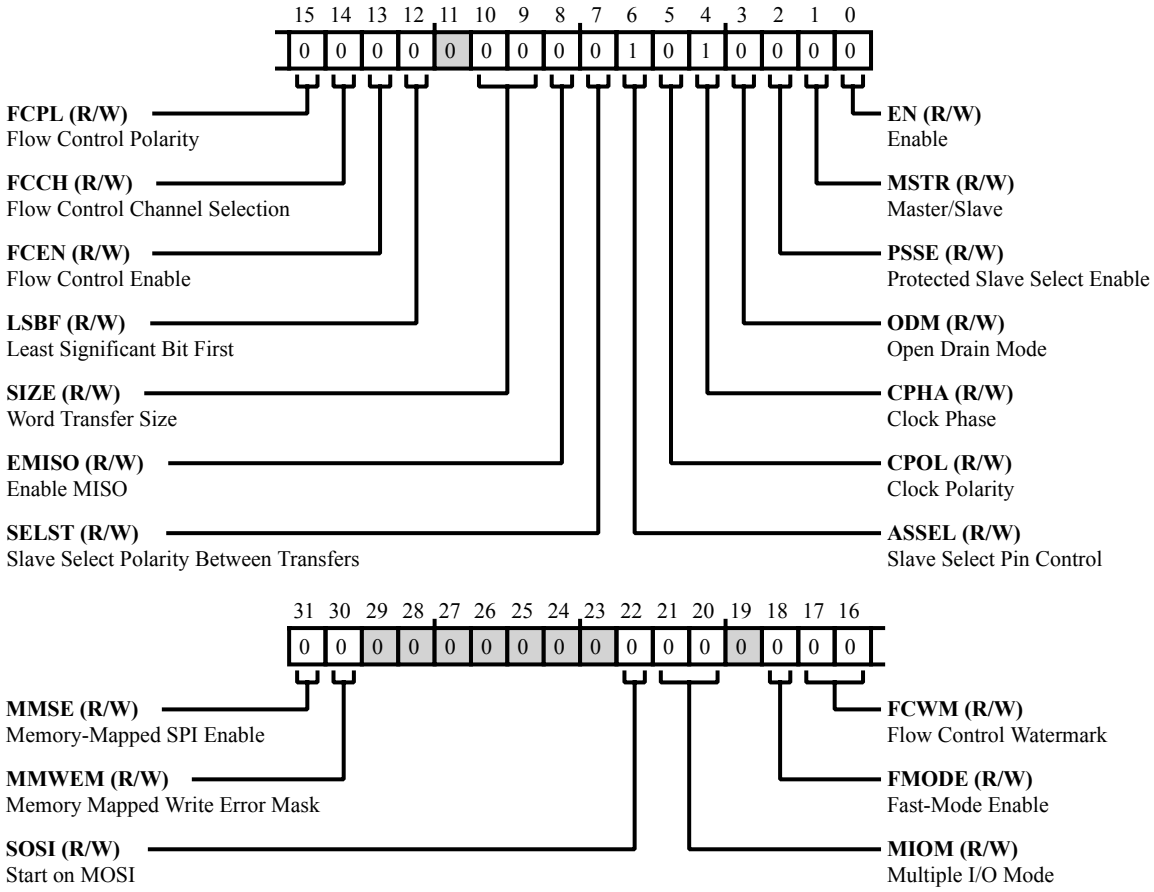

Figure 29-21: SPI_CTL Register Diagram

Table 29-17: SPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	MMSE	Memory-Mapped SPI Enable. When the SPI_CTL.MMSE bit is asserted, communication to an SPI memory device is automated such that the memory it contains is accessible directly through the read of processor address space assigned to it. (As far as the SPI peripheral is concerned, this includes all read accesses received by the SPI peripherals system crossbar slave port.) Note that when memory-mapped access of SPI memory is enabled, attempts to communicate with the SPI device using legacy methods are blocked and receive fabric error responses are generated. Legacy methods include any direct access made to the Tx and Rx FIFOs, whether initiated by DMA or processor MMR access.	
		0	Hardware automated access of memory-mapped SPI memory disabled.
		1	Hardware-automated access of memory-mapped SPI memory enabled.
30 (R/W)	MMWEM	Memory Mapped Write Error Mask. The SPI_CTL.MMWEM bit specifies whether an error response is returned to the fabric upon write attempts to address space reserved for memory-mapped reads of SPI memory.	
		0	Write error response returned upon write attempts to memory-mapped SPI memory
		1	Write error response masked (not returned) upon write attempts to memory-mapped SPI memory
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

Table 29-17: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

Table 29-17: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

Table 29-17: SPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	SELST	Slave Select Polarity Between Transfers. The SPI_CTL.SELST bit selects the state (polarity) for the 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)
6 (R/W)	ASSEL	Slave Select Pin Control. The SPI_CTL.ASSEL bit selects whether the SPI hardware sets the SPI_SEL[n] pin output value (ignoring the slave select SPI_SLVSEL.SSEL1 - SPI_SLVSEL.SSEL7 bits) or whether software control of the slave select bits set the SPI_SEL[n] pin output value. This feature is applicable only when the SPI is a master. When hardware control is enabled, the 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 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 29-17: 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.

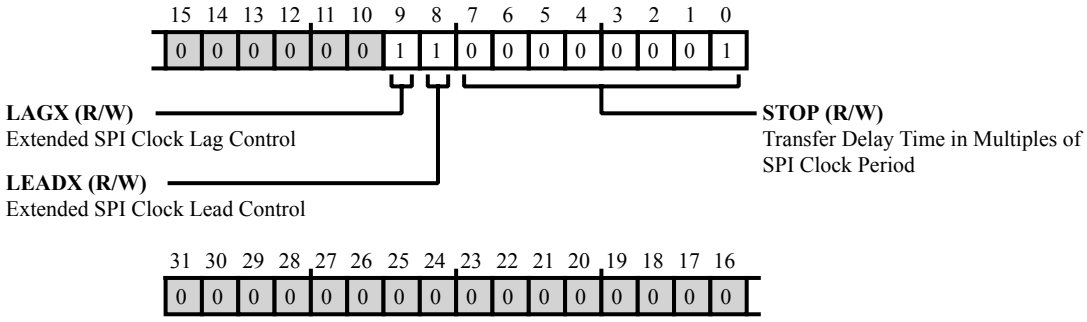

Figure 29-22: SPI_DLY Register Diagram

Table 29-18: 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 interrupts for service. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt is unmasked in SPI_IMSK, the SPI latches the interrupt's bit in SPI_ILAT.

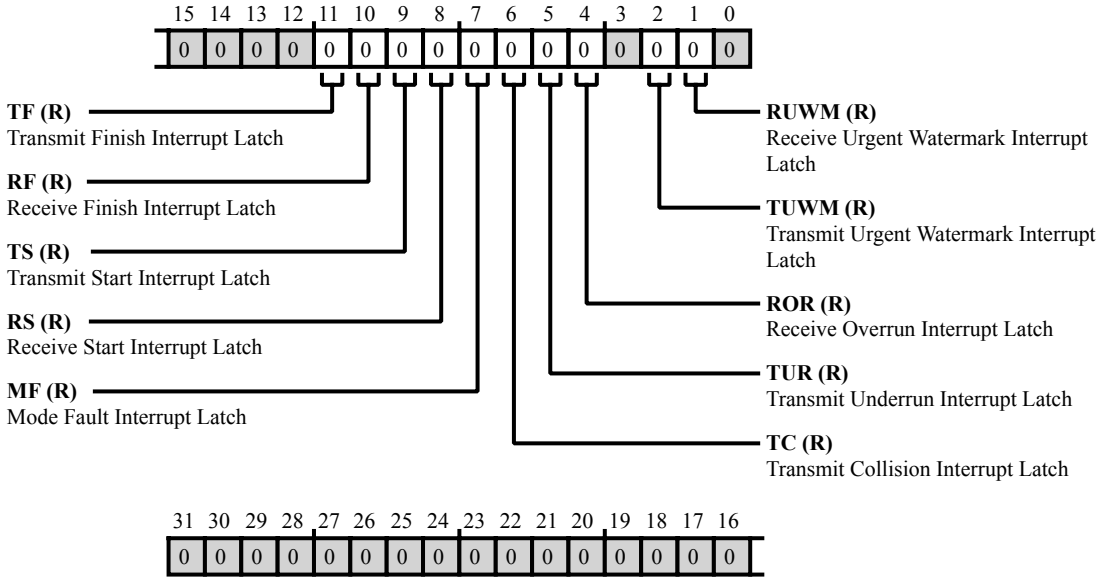

Figure 29-23: SPI_ILAT Register Diagram

Table 29-19: SPI_ILAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/NW)	TF	Transmit Finish Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
10 (R/NW)	RF	Receive Finish Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
9 (R/NW)	TS	Transmit Start Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
8 (R/NW)	RS	Receive Start Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt

Table 29-19: SPI_ILAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/NW)	MF	Mode Fault Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
6 (R/NW)	TC	Transmit Collision Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
5 (R/NW)	TUR	Transmit Underrun Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
4 (R/NW)	ROR	Receive Overrun Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
2 (R/NW)	TUWM	Transmit Urgent Watermark Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt
1 (R/NW)	RUWM	Receive Urgent Watermark Interrupt Latch.	
		0	No interrupt
		1	Latched interrupt

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.

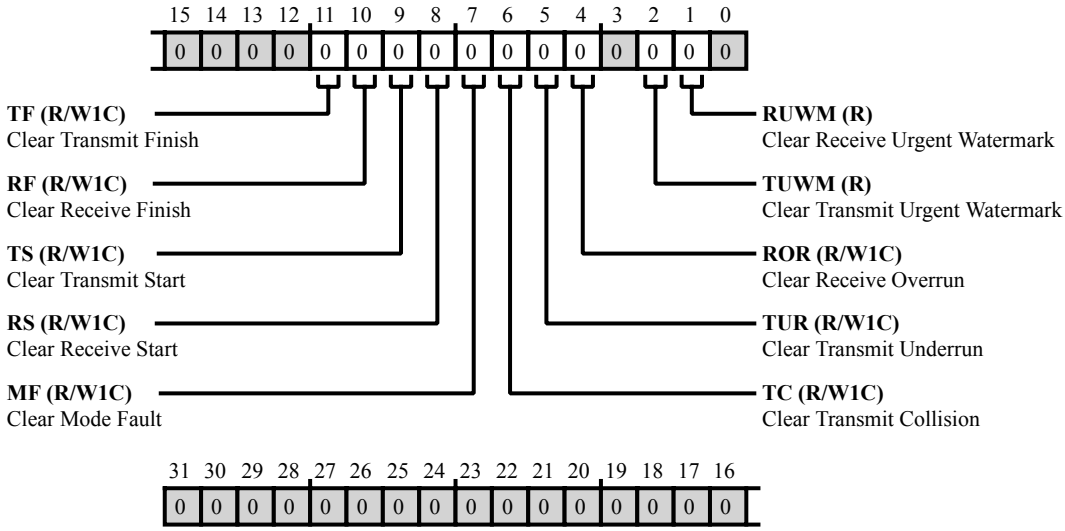

Figure 29-24: SPI_ILAT_CLR Register Diagram

Table 29-20: 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 29-20: 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 29-20: 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 interrupts. When a condition is indicated by a bit in the SPI_STAT register and the corresponding interrupt is unmasked in SPI_IMSK, the SPI latches the interrupt's bit in the SPI_ILAT register, queuing the interrupt for service.

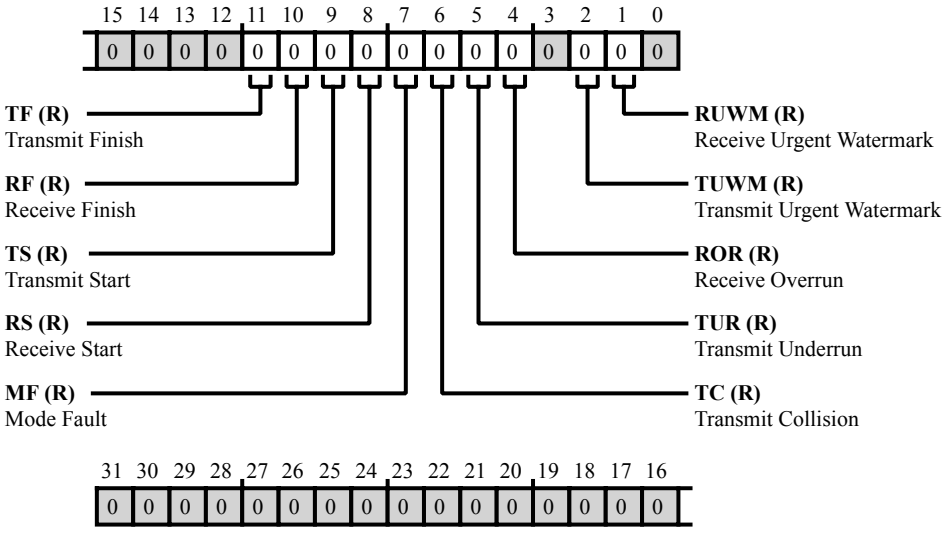

Figure 29-25: SPI_IMSK Register Diagram

Table 29-21: 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
		1	Enable (unmask) interrupt
10 (R/NW)	RF	Receive Finish. The SPI_IMSK.RF bit unmasks (enables) or masks (disables) the RF interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
9 (R/NW)	TS	Transmit Start. The SPI_IMSK.TS bit unmasks (enables) or masks (disables) the TS interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

Table 29-21: 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
		1	Enable (unmask) interrupt
7 (R/NW)	MF	Mode Fault. The SPI_IMSK.MF bit unmasks (enables) or masks (disables) the MF interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
6 (R/NW)	TC	Transmit Collision. The SPI_IMSK.TC bit unmasks (enables) or masks (disables) the TC interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
5 (R/NW)	TUR	Transmit Underrun. The SPI_IMSK.TUR bit unmasks (enables) or masks (disables) the TUR interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
4 (R/NW)	ROR	Receive Overrun. The SPI_IMSK.ROR bit unmasks (enables) or masks (disables) the ROR interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
2 (R/NW)	TUWM	Transmit Urgent Watermark. The SPI_IMSK.TUWM bit unmasks (enables) or masks (disables) the TUWM interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt
1 (R/NW)	RUWM	Receive Urgent Watermark. The SPI_IMSK.RUWM bit unmasks (enables) or masks (disables) the RUWM interrupt.	
		0	Disable (mask) interrupt
		1	Enable (unmask) interrupt

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.

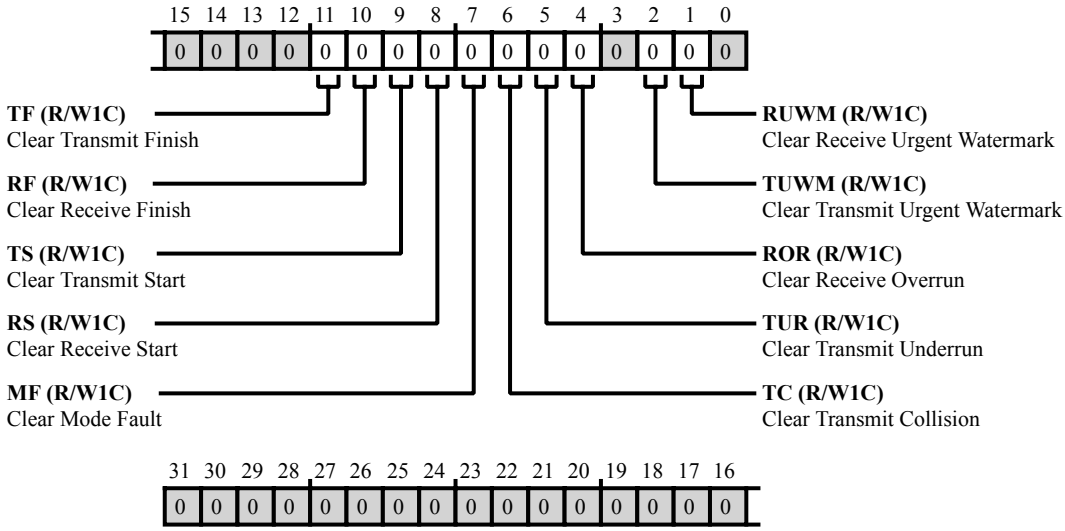

Figure 29-26: SPI_IMSK_CLR Register Diagram

Table 29-22: 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 29-22: 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 29-22: 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.

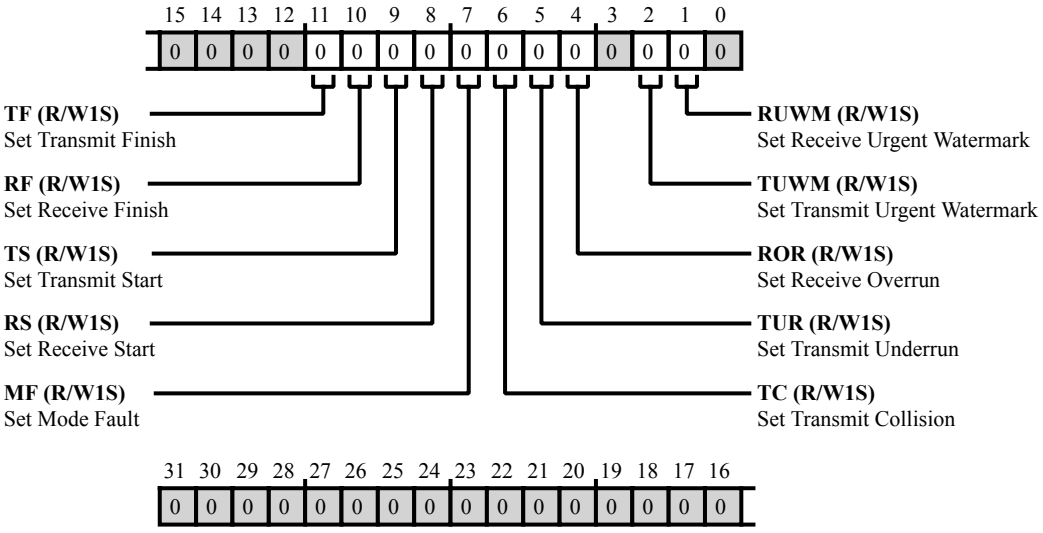

Figure 29-27: SPI_IMSK_SET Register Diagram

Table 29-23: 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 29-23: 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 29-23: 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

Memory Mapped Read Header

The SPI_MMRDH register enables the use of memory-mapped mode. This mode allows direct memory-mapped read accesses of an SPI memory device and is primarily used to directly execute instructions from an SPI FLASH memory without using a low-level software driver. All overhead tasks such as transmission of the read header, pin turnaround timing and receive data sizing are handled in hardware.

The memory-mapped access mode is enabled by setting the SPI_CTL.MMSE bit. The features within the SPI_MMRDH register include a command skip mode, variable length byte addressing, and independent multi-pin support for command transmission, address transmission and data reception. In addition, the command opcode and mode bytes are fully programmable.

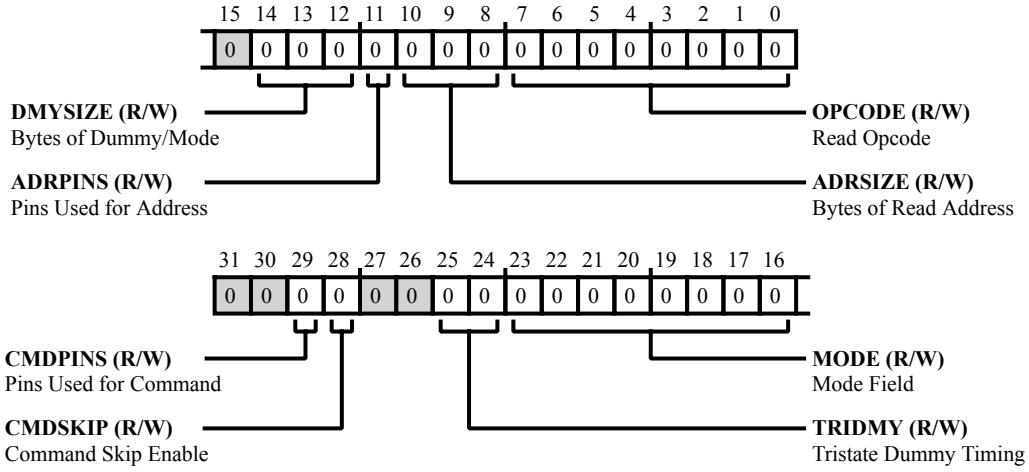

Figure 29-28: SPI_MMRDH Register Diagram

Table 29-24: SPI_MMRDH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	CMDPINS	Pins Used for Command. The SPI_MMRDH.CMDPINS bit specifies the number of pins to be used for command transmission. This bit must be set consistent with the expectations established by the read opcode. Hardware does not interpret SPI_MMRDH.OPCODE, but rather relies on this bit to specify behavior. When cleared, it overrides the SPI_CTL.MIOM bits. When set, it uses bits specified by the SPI_CTL.MIOM bit setting.	
		0	Use only one pin: MOSI (overrides SPI_CTL.MIOM bits)
		1	Use pins specified by SPI_CTL.MIOM bits

Table 29-24: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
28 (R/W)	CMDSKIP	Command Skip Enable. The SPI_MMRDH.CMDSKIP bit enables command skip mode where the address is sent first and the OPCODE field is not sent (SPI_MMRDH.CMDSKIP bit =1). This mode is useful for supporting XIP (Execute-In-Place) operation where only the address is sent and the same read command is assumed. The SPI flash device must be primed with an initial read command, before the SPI_MMRDH.CMDSKIP bit is set.	
		0	OPCODE field is sent first followed by address
		1	OPCODE field is not sent; address is sent first
25:24 (R/W)	TRIDMY	Tristate Dummy Timing. The SPI_MMRDH.TRIDMY bits specify whether and when output pins are three-stated during the interval of time specified by the SPI_MMRDH.DMYSIZE bits. Output pins potentially three-stated include all pins which were used to transmit the address	
		0	Tristate outputs immediately
		1	Tristate outputs after 4 bits of dummy/mode are transmitted
		2	Tristate outputs after 8 bits of dummy/mode are transmitted
		3	Never tristate outputs (previously specified output state is held)
23:16 (R/W)	MODE	Mode Field. These bits specify up to a leading byte to be transmitted during the interval of time specified by the SPI_MMRDH.DMYSIZE bit field. This first byte, or a portion of it, is interpreted as mode bits when certain opcodes are used in conjunction with certain SPI memory devices. Mode bits are sent using the same number of pins which were used to transmit the address. Once sent, output pins will be held in their final resultant state until the conclusion of all dummy byte periods, unless three-stating the outputs is specified first by the SPI_MMRDH.TRIDMY bits.	
14:12 (R/W)	DMYSIZE	Bytes of Dummy/Mode. The SPI_MMRDH.DMYSIZE bit field specifies the number of bytes separating address transmission and read data return. Dummy bytes elapse assuming dummy bits are transmitted using the same number of pins which were used to transmit address.	
		0	0 Bytes
		1	1 Bytes
		2	2 Bytes
		3	3 Bytes
		4	4 Bytes

Table 29-24: SPI_MMRDH Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		5	5 Bytes
		6	6 Bytes
		7	7 Bytes
11 (R/W)	ADRPINS	Pins Used for Address. The SPI_MMRDH.ADRPINS bit specifies the number of pins to be used for address transmission. This bit must be set consistent with expectations established by read opcode. Hardware does not interpret the SPI_MMRDH.OPCODE, but rather relies on this bit to specify behavior.	
		0	Use only one pin: MOSI (overrides SPI_CTL.MIOM bits)
		1	Use pins specified by SPI_CTL.MIOM bits
10:8 (R/W)	ADRSIZE	Bytes of Read Address. The SPI_MMRDH.ADRSIZE bit field defines the number of bytes used to specify the read address. The read address is sent immediately following the transmission of opcode. Unlike opcode bits, address bits may be sent using either one or multiple pins. The number of pins is selected using the SPI_MMRDH.ADRPINS bit. The address sent to a connected SPI memory device is an echo of the read address received by the SPI peripheral slave port. The least significant bytes of address are sent when the entire address is not sent.	
		0	1 Byte
		1	1 Byte
		2	2 Bytes
		3	3 Bytes
		4	4 Bytes
7:0 (R/W)	OPCODE	Read Opcode. The SPI_MMRDH.OPCODE bit field specifies the initial bits transmitted in response to a read request of SPI memory. Although any opcode may be sent, values 0x03, 0x0B, 0x3B, 0x6B, 0xBB, and 0xEB are likely to be the most commonly used. SPI_MMRDH.OPCODE is sent by the SPI without interpretation; the states of these bits have no effect beyond specifying what is initially shifted across the SPI interface.	

SPI Memory Top Address

The SPI_MMTOP register specifies the top populated address of a connected SPI memory device.

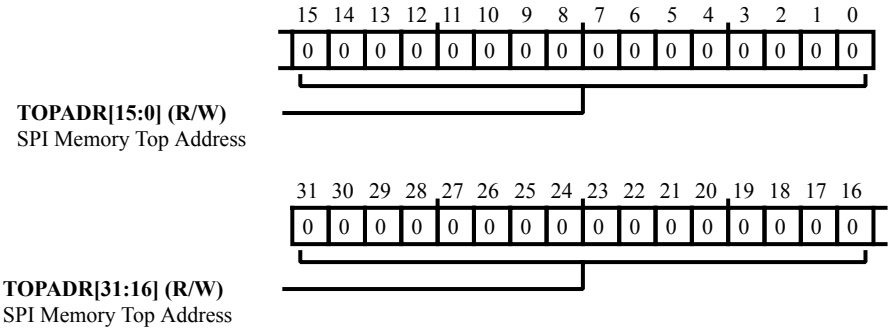

Figure 29-29: SPI_MMTOP Register Diagram

Table 29-25: SPI_MMTOP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	TOPADR	SPI Memory Top Address. The SPI_MMTOP.TOPADR bit field specifies the top populated address of a connected SPI memory device. Attempts to access SPI memory are not blocked if this address is exceeded and an error is generated as part of the read response.

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.

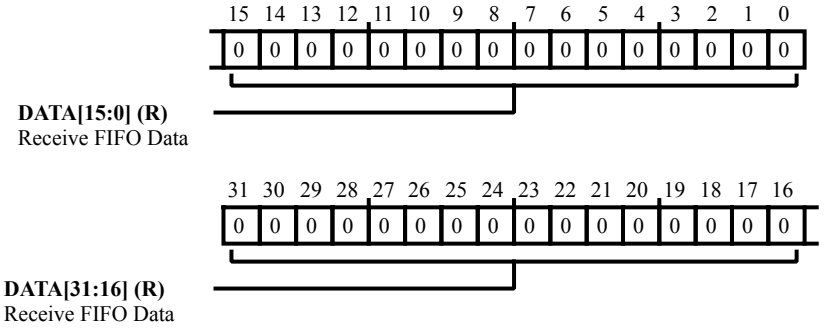

Figure 29-30: SPI_RFIFO Register Diagram

Table 29-26: 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.

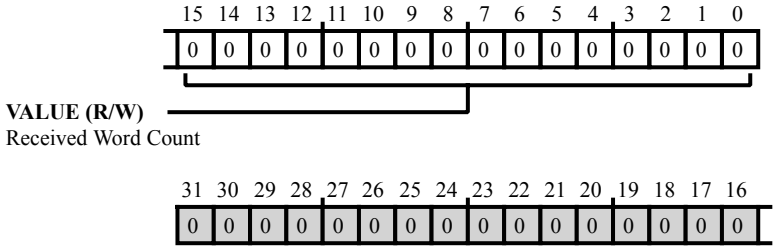

Figure 29-31: SPI_RWC Register Diagram

Table 29-27: 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.

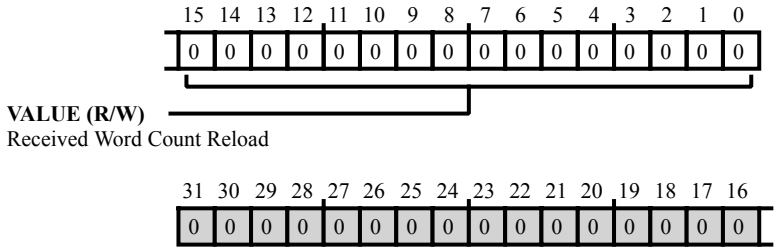

Figure 29-32: SPI_RWCR Register Diagram

Table 29-28: 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.

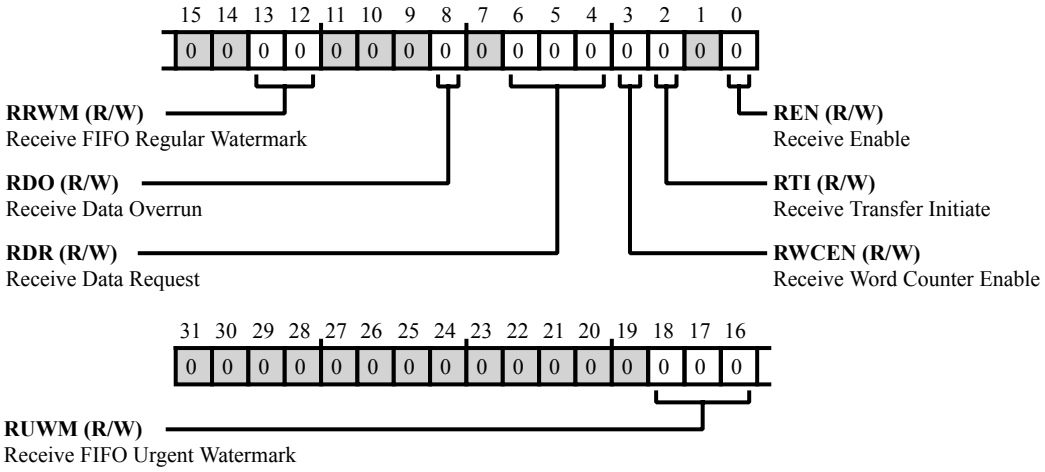

Figure 29-33: SPI_RXCTL Register Diagram

Table 29-29: 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 29-29: 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 29-29: 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.

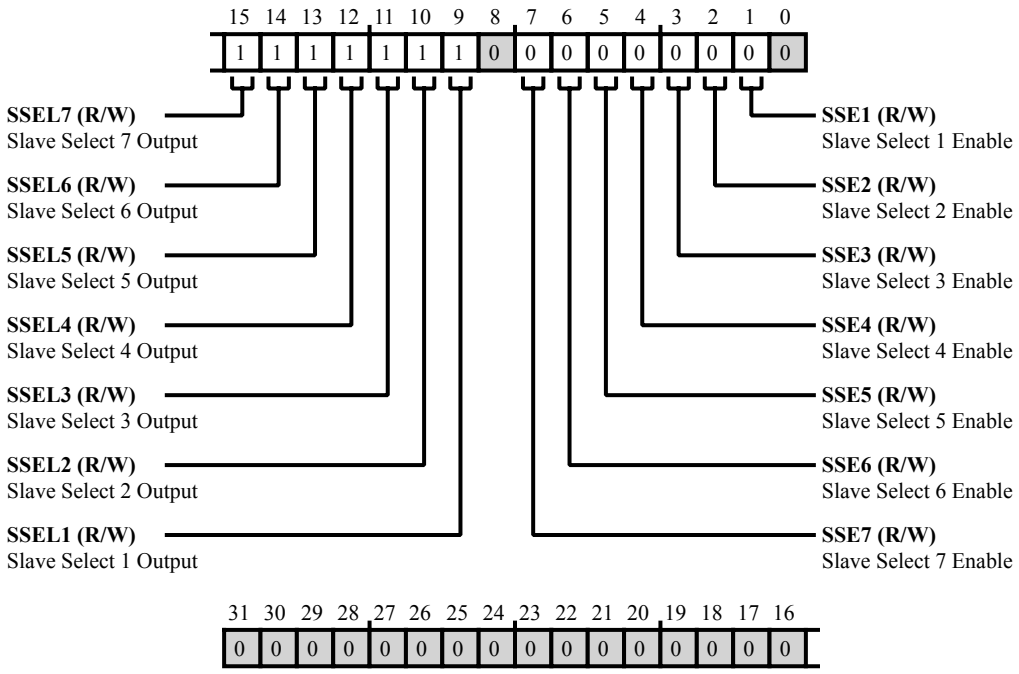

Figure 29-34: SPI_SLVSEL Register Diagram

Table 29-30: 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 29-30: 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 29-30: 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 $\overline{\texttt{SPI\_SEL[n]}}$ 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 $\overline{\texttt{SPI\_SEL[n]}}$ 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 $\overline{\texttt{SPI\_SEL[n]}}$ 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 $\overline{\texttt{SPI\_SEL[n]}}$ 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 $\overline{\texttt{SPI\_SEL[n]}}$ 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 $\overline{\texttt{SPI\_SEL[n]}}$ 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 is latched into the corresponding bit in the SPI_ILAT register.

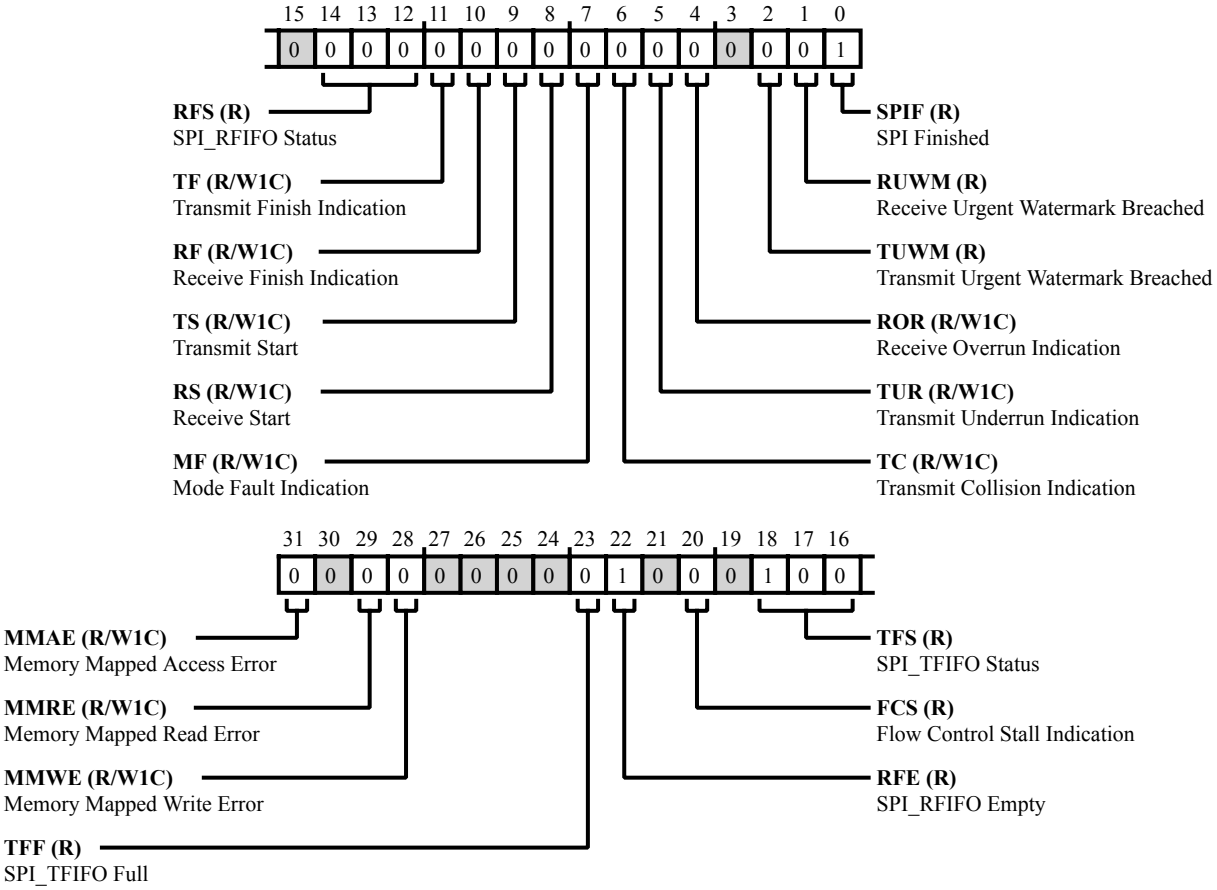

Figure 29-35: SPI_STAT Register Diagram

Table 29-31: SPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W1C)	MMAE	Memory Mapped Access Error. The SPI_STAT.MMAE bit =1 if an attempt is made to access either the Tx or Rx FIFO while memory-mapped access of SPI memory is enabled (see the SPI_CTL.MMSE bit). The SPI_STAT.MMAE bit =0 when a 1 is written to it. The SPI_STAT.MMAE bit is provided for software notification only. Its state has no further effect.
29 (R/W1C)	MMRE	Memory Mapped Read Error.

Table 29-31: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The SPI_STAT.MMRE bit =1 if an attempt is made to read address space reserved for memory-mapped SPI memory while memory mapping is disabled (see the SPI_CTL.MMSE bit). The SPI_STAT.MMRE bit =0 when a 1 is written to it. This bit is provided for software notification only. Its state has no further effect.	
28 (R/W1C)	MMWE	Memory Mapped Write Error. The SPI_STAT.MMWE bit =1 if an attempt is made to write address space reserved for memory-mapped SPI memory. The SPI_STAT.MMWE bit =0 when a 1 is written to it. This bit is provided for software notification only. Its state has no further effect.	
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
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.	

Table 29-31: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		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
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

Table 29-31: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

Table 29-31: SPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.

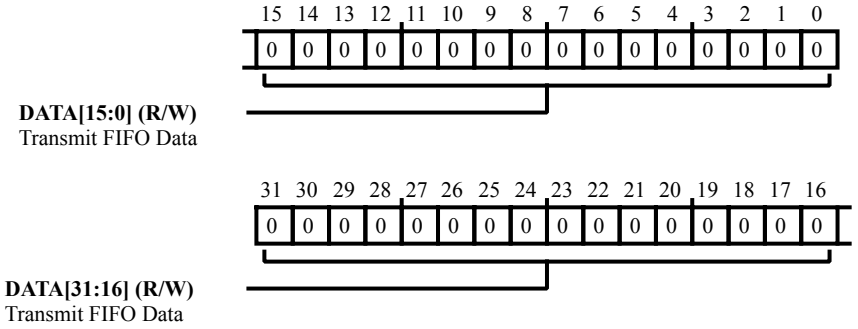

Figure 29-36: SPI_TFIFO Register Diagram

Table 29-32: 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. 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.

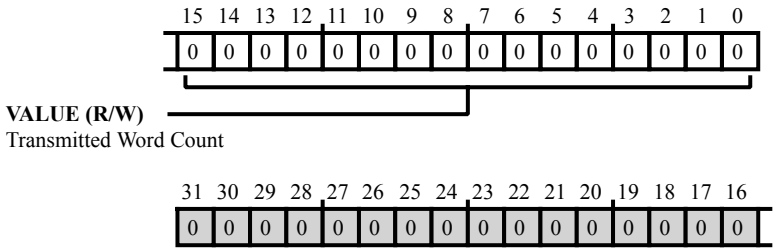

Figure 29-37: SPI_TWC Register Diagram

Table 29-33: 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.

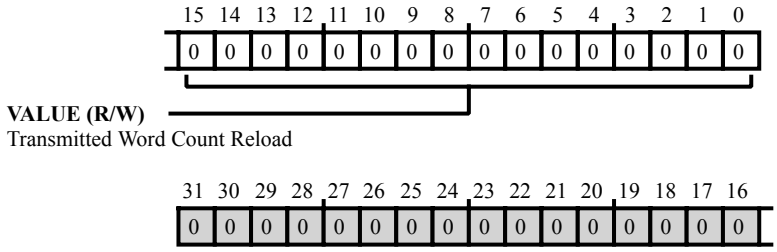

Figure 29-38: SPI_TWCR Register Diagram

Table 29-34: 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.

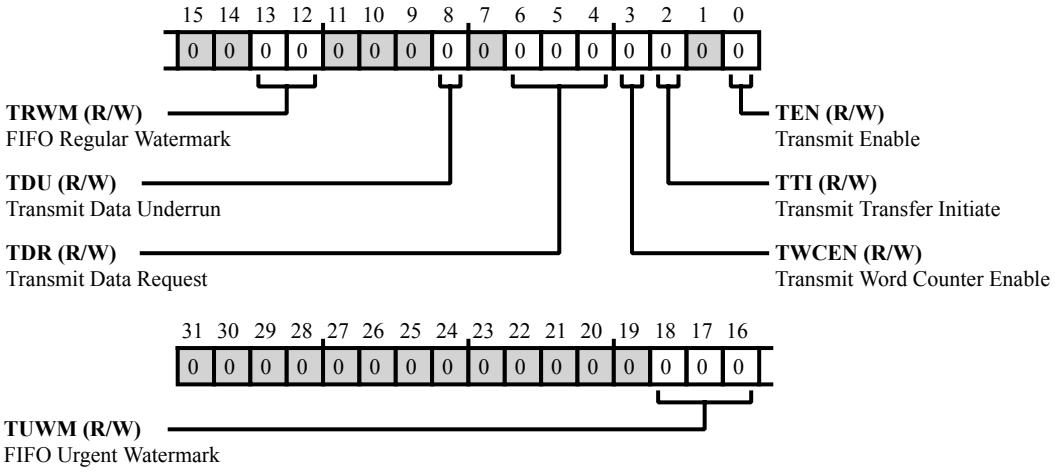

Figure 29-39: SPI_TXCTL Register Diagram

Table 29-35: 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. 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 interrupts that are latched.	
		0	Disabled
		1	25% empty TFIFO
		2	50% empty TFIFO
		3	75% empty TFIFO
		4	Empty TFIFO

Table 29-35: 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 interrupts 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.	
		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 29-35: 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

30 Serial Peripheral Interface Host Port (SPIHP)

The SPI host port provides the slave support for host to access the memory mapped resources of the processor through the SPI SRAM or Flash style protocol. The SPI host port functionality can be enabled to all the available instances of the SPI module of the processor.

The host port facilitates a host device external to the processor to be a SPI master and transfers data back and forth. The host device always masters the transactions and the Blackfin processor is always a slave device.

NOTE: The registers for the SPIHP module use the prefix SPIHP_. The registers for the SPI module use SPIn_. The SPIHP is a peripheral on the processor, which is referred to as the slave processor or processor SPI slave. The host processor is also referred to as the host, master, external host, or external master.

SPIHP Features

The SPIHP supports the following features.

- Direct read and write of on-chip and off-chip memories and memory mapped registers.
- Support for SPI controllers that implement hardware-based SPI memory protocol.
- Error capture and reporting for protocol errors, bus errors, and over or underflow (captured as sticky status and reported on dedicated signal).
- Support for hardware and software flow control.
- Support for various opcodes as applicable to flash devices.
- Write stride opcode support to write to non-contiguous addresses.

SPIHP Functional Description

The SPI must be enabled along with the host port functionality to ensure that the SPI operates in slave mode. In this mode, the SPI expects and interprets the opcodes from the host and responds appropriately for each supported opcode.

The SPI host device can use a software or hardware-driven SPI memory command protocol to access the memory-mapped resources of the processor. Flow control can be managed through software or hardware depending on the

host. The software flow control is managed by checking the SPIHP status prior to read/write operations and using the prefetch to perform reads from SPI slave memory space. The hardware flow control uses the SPI_RDY signal to perform the read/write operations. The following sections describe more functional descriptions.

- SPIHP Block Diagram
- SPIHP Signal Descriptions
- SPIHP Architectural Concepts

ADSP-BF70x SPIHP Trigger List

Table 30-1: ADSP-BF70x SPIHP Trigger List Masters

Trigger ID	Name	Description	Sensitivity
48	SPIHP0_EVT	SPIHP0 SPI Host Port Ready	Level

Table 30-2: ADSP-BF70x SPIHP Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

ADSP-BF70x SPIHP Interrupt List

Table 30-3: ADSP-BF70x SPIHP Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
85	SPIHP0_ERR	SPIHP0 Error	Level	

ADSP-BF70x SPIHP Register List

The Serial Peripheral Interface Host Port SPIHP provides a full-duplex, synchronous serial host interface, intended for communication with a SPI-compatible port on a master processor. A set of registers governs SPIHP operations. For more information on SPIHP functionality, see the SPIHP register descriptions.

Table 30-4: ADSP-BF70x SPIHP Register List

Name	Description
SPIHP_AUX[n]	Auxiliary Register
SPIHP_BAR[n]	Base Address Register
SPIHP_CTL	Control Register
SPIHP_RDPF	Read Prefetch Register
SPIHP_STAT	Status Register

SPIHP Block Diagram

The *SPIHP Block Diagram* shows the SPIHP module block diagram.

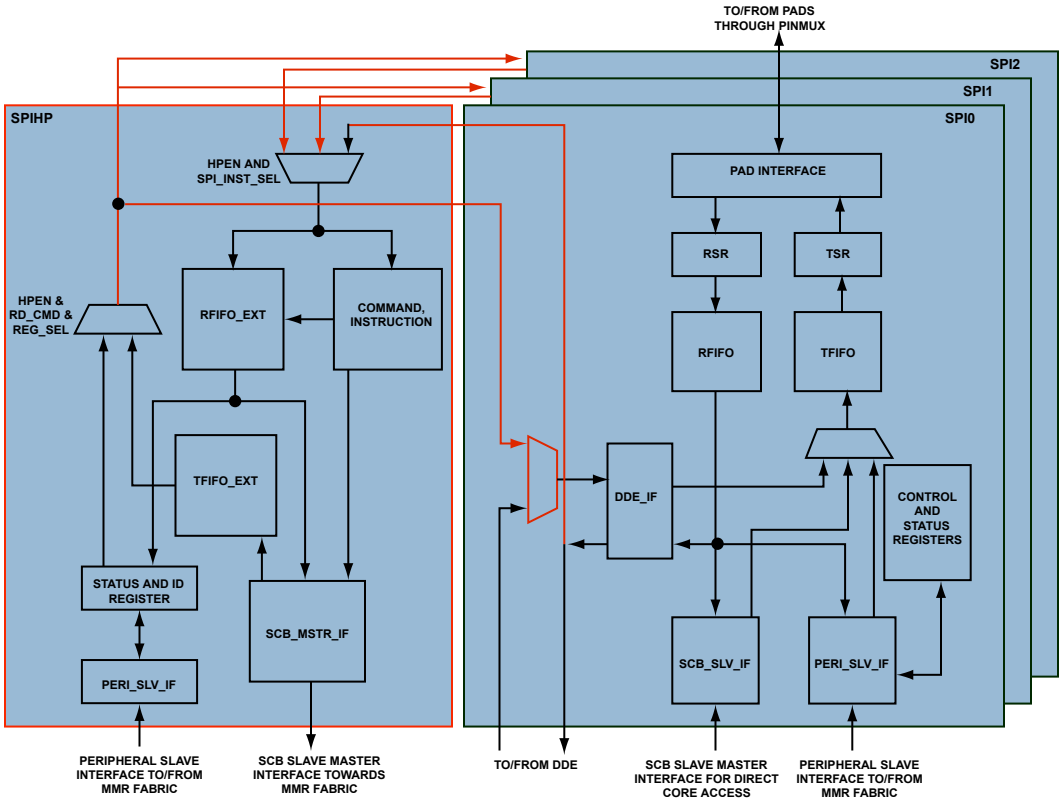

Figure 30-1: SPIHP Block Diagram

The SPIHP has a 128 deep FIFO for core bus transactions. Depending on the `SPIHP_CTL.MSIZE` value, the number of bytes supported by the interface varies as shown in the following list.

- 512 bytes for `SPIHP_CTL.MSIZE` =2
- 256 bytes for `SPIHP_CTL.MSIZE` =1
- 128 bytes for `SPIHP_CTL.MSIZE` =0

The BF70x processor has 3 SPI instances. To configure one of the instances as a host port slave, program the `SPIHP_CTL.SPISEL`[3:0] register bits as shown in the *SPI HP Slave Selections* table:

Table 30-5: SPI HP Slave Selections

SPIHP_CTL.SPISEL[3:0]	SPI Instance Selected as HP Slave
0x0	SPI0
0x1	SPI1
0x2	SPI2

Table 30-5: SPI HP Slave Selections (Continued)

SPIHP_CTL.SPISEL[3:0]	SPI Instance Selected as HP Slave
0x3 -0xF	Reserved

Also, program the `SPIHP_RDPF.CNT` bit field according to the appropriate limit.

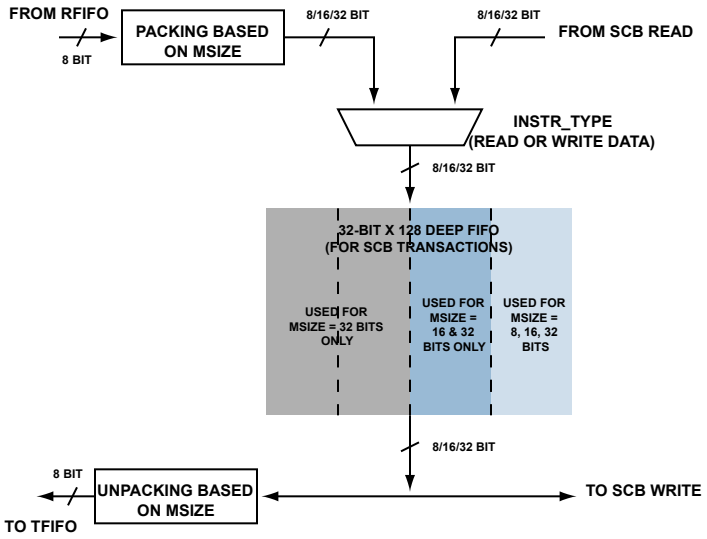

Figure 30-2: SPIHP FIFO Block Diagram

SPIHP Signal Descriptions

In addition to the supported SPI signals, the SPIHP supports SPIHPIRQ and SPIHPTO signals. The function of these signals is described below.

- SPIHP Interrupt Request (SPIHPIRQ). This signal is an output signal from the SPIHP to the system. It is intended to be connected to the SEC and its assertion is controlled by masks in the `SPIHP_CTL` register.
- SPIHP Trigger Out (SPIHPTO). This signal is intended to be connected to the TRU and functions as described in the `SPIHP_CTL.TRGM` bit description.

The ***Host to Slave Signal Connections*** figure shows the host to slave signal connection model.

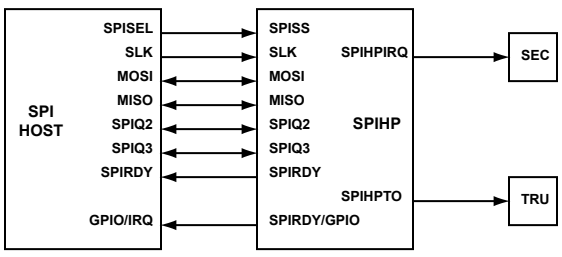

Figure 30-3: Host to Slave Signal Connections

If the host is capable of using the hardware flow control option, the SPI_RDY signal drives a flow control signal capable of stalling the SPI host. The SPI host then needs to wait for a SPI_RDY signal assertion to latch the new data during a read operation or to send the new data for a write operation.

SPIHP Architectural Concepts

The following concepts provide general information on the architecture of the SPIHP.

System Master Interface

The system master interface provides a master interface on the system interconnect fabric that can directly access the memory and memory-mapped registers of the device.

Address Translation

In order to provide unconstrained access to the full memory space of the SPI slave, the module supports address translation. The SPI host provides the 24-bit address to the SPIHP. The SPIHP base address register holds the 8-bit address offset to be pre-pended to the address supplied by the host. The base address offset (BAO) value of the selected SPIHP_BAR[n] register is used as bits [31:24] for the local address.

The SPIHP_CTL.BARSEL bit field selects the SPIHP_BAR[n] register to allow the host to switch BARs quickly (locally as opposed to an sMMR write). The host or slave configures the values of all the SPIHP_BAR[n] registers before an operation. The host can simply switch pages dynamically by writing to the SPIHP_CTL.BARSEL bit field to select the correct SPIHP_BAR[n] register.

Speculative Read Enable

The SPIHP_CTL.SPRDEN bit enables the system master interface to perform speculative reads beyond the current address to avoid interrupted flow of read data. This configuration is intended to be used for memory regions as opposed to MMR access to improve performance and avoid underflow or RDY deassertion. It applies to all read data operations.

When set, read data operations are limited to the value set in the SPIHP_CTL.MSIZE bit field. Any read data operation that attempts to access more than SPIHP_CTL.MSIZE results in the bus error (SPIHP_STAT.BERR) bit being set. This feature allows reads of contiguous memory locations without interruptions as long as SLK toggles.

Hardware Flow Control

Hardware flow control uses the SPIHP_STAT.RDY bit to indicate a need to stall. The host can use this bit as a throttle to hold off subsequent data transfer or instructions as necessary until the slave is ready.

Software Flow Control

Software flow control requires the host to place a limit on the size of transfers and check the SPIHP_STAT register after each instruction completion (read data, write data, or prefetch) to confirm the previous operation and that the

SPIHP is ready for the next operation. For instance, write data instruction payloads can be limited to SPIHP_RFIFO depth as a maximum. After each write data instruction, the host can check SPIHP_STAT.RDY to confirm that the SPIHP is ready for the next operation.

Software flow control frequently suffers from read underflow errors as a result of variable and potentially large latency due to system topology, system traffic, clock domain crossings, and other issues. To help solve this problem, the host port read prefetch support (configured using the SPIHP_RDPF register) can prime the SPIHP_TFIFO with the data the host wants to read. The host initially writes to the SPIHP_RDPF register to initiate the prefetch and then polls the SPIHP_STAT.RDY bit (or SPIHP_STAT.RDYSTKY) to determine when it is safe to read from the SPIHP_TFIFO.

Host Port Slave Operation

The following are the details for SPIHP slave operation.

- Endianess – Bytes are shifted out or in MSB first.
- Signals
  - MOSI/SPIQ0 = IO0
  - MISO/SPIQ1 = IO1
  - SPIQ2 = IO2
  - SPIQ3 = IO3
- Opcodes/Instructions
  - Opcodes, addresses, and dummy bytes are always received by single input signal MOSI/SPIQ0/IO0, MSB first.
  - Instructions begin on the falling edge of $\overline{\text{SPI\_SELn}}$ ($\overline{\text{SPI\_SS}}$), the first byte received is the opcode.
  - Instructions end on the rising edge of $\overline{\text{SPI\_SELn}}$ ($\overline{\text{SPI\_SS}}$).
  - Read Data Instructions do not work for SPI configurations with CPHA=0
- Dual Input and Dual Output Data Alignment
  - IO0 = D6, D4, D2, D0
  - IO1 = D7, D5, D3, D1
- Quad Input and Quad Output Data Alignment
  - IO0 = D4, D0,…
  - IO1 = D5, D1,…
  - IO2 = D6, D2,…
  - IO3 = D7, D3,…

SPIHP Event Control

The following section provides information on status and error signals.

SPIHP Status and Error Signals

The host port status register has the flexibility to reflect certain error conditions when triggered. The following list describes these conditions.

- Overflow (OVF). `SPIHP_STAT.OVF` is W1C. A host-initiated write that overflows the SPIHP_RFIFO sets the bit.
- Underflow (UVF). `SPIHP_STAT.UVF` is W1C. A host-initiated read that underflows the SPIHP_TFIFO sets the bit.
- Unsupported Opcode (UOP). `SPIHP_STAT.UOP` is W1C. A host-initiated command or opcode that is invalid or unsupported including access violations for local read/write register instructions (for example, write register with index pointing to NW register or write register with more than 4-byte payload).
- Bus Error (BERR). `SPIHP_STAT.BERR` is W1C and is set by any of the following:
  - SLVERR or DECERR returned for any transfer over system bus
  - Misaligned address errors (32b xfer with A1:0 != 00 or 16b xfer with A0 != 0)
  - Payload error (write data instruction terminates with non-integer number of `SPIHP_CTL.MSIZE` wide transfers in SPIHP_RFIFO)
  - Speculative read error (read data operation larger than `SPIHP_CTL.MSIZE` when `SPIHP_CTL.SPRDEN` is not set)

SPIHP Programming Model

The following sections provide programming information for the SPIHP module.

SPIHP Protocol

The ***SPIHP Programming Protocol*** table shows the instructions supported by the SPIHP module. The host must follow these instructions for proper operation. The table uses the following conventions:

- A = Address
- D = Data in to SPIHP
- (D)= Data out from SPIHP
- R = SPIHP Register address
- S = Stride
- Dummy = Unused or discard

- * = Read register instructions continue to read the same register (without dummy cycles) until /SPIxSS is deasserted. Writes are limited to 4 bytes maximum.

Table 30-6: SPIHP Programming Protocol

Instruction	Byte 0 (Opcode)	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Read Data	0x03	A23:16	A15:8	A7:0	D7:0	....
Fast Read	0x0B	A23:16	A15:8	A7:0	Dummy	(D7:0)
Fast Read Dual Output	0x3B	A23:16	A15:8	A7:0	Dummy	(D7:0,...)
Fast Read Quad Output	0x6B	A23:16	A15:8	A7:0	Dummy	(D7:0,...)
Write Data	0x02	A23:16	A15:8	A7:0	D7:0	...
Write Data Dual Input	0xA2	A23:16	A15:8	A7:0	Dummy	D7:0,...
Write Data Quad Input	0x32	A23:16	A15:8	A7:0	Dummy	D7:0,...
Write Data Stride*	0x0A	A23:16	A15:8	A7:0	S7:0	D7:0
Write Data Dual Input Stride*	0xAA	A23:16	A15:8	A7:0	Dummy	S7:0, D7:0
Write Data Quad Input Stride*	0x3A	A23:16	A15:8	A7:0	Dummy	S7:0, D7:0,...
Read Register*	0x55	R7:0	Dummy	D7:0	D15:8	D23:16
Read Register Dual Output*	0x75	R7:0	Dummy	(D7:0,...)	...	...
Read Register Quad Output*	0x95	R7:0	Dummy	(D7:0,...)	...	...
Write Register*	0x11	R7:0	D7:0	D15:8	D23:16	D31:24

Programming Guidelines

Observe the following guidelines for correct SPIHP operation.

1. Always enable SPIHP before enabling SPI.
2. These `SPI_CTL` bits affect SPIHP operation. Programs cannot change these bits while SPIHP and SPI are enabled: `SPI_CTL.ODM`, `SPI_CTL.CPOL`, `SPI_CTL.CPHA`, `SPI_CTL.FCEN`, and `SPI_CTL.FCPL`.
3. The rest of the `SPI_CTL` bits are do-not-care in SPIHP mode. Programs can change them, but that can in turn violate SPI programming guidelines. It is advised that programs not interact with the `SPI_CTL` register while the SPIHP is enabled.
4. The SPI connected to SPIHP can be disabled (`SPI_CTL.EN` =0) only if `SPIHP_STAT.RDY` =1 and ONLY after SPIHP is disabled (`SPIHP_CTL.EN` =0).

SPIHP Programming Sequence

The following pseudo code programs the host port. It assumes the condition of a host accessing the L2 memory space.

1. Program SPIHP with appropriate SPIHP_CTL.BWCTL|SPIHP_CTL.SPRDEN|SPIHP_CTL.MSIZE|SPIHP_CTL.BARSEL bit settings.
2. Program the SPIHP_BAR[n] register with the upper 8 bits of the memory space to be read

   ADDITIONAL INFORMATION: Example: `pREG_SPIHP0_BAR0=(0x08>>24) // For L2 memory space`
3. Enable the SPIHP module by setting the SPIHP_CTL.EN bit
4. Configure the associated SPI instance with SPI_CTL.EMISO enabled, SPI_CTL.CPHA =1 and enable the SPI (SPI_CTL.EN =1).

SPIHP Programming Concepts

Using the features, operating modes, and event control for the SPIHP to their greatest potential requires an understanding of some SPIHP-related concepts.

Issuing Instructions

Each instruction begins with an assertion of $\overline{\text{SPI\_SS}}$ and ends with de-assertion of $\overline{\text{SPI\_SS}}$. The host can check the SPIHP_STAT register to be sure that there are no errors and that the SPI is ready for a new command. Issue read data, write data, and prefetch operations only when SPIHP_STAT.RDY==1.

SPIHP Read Register and Write Register Operations

The host can read and write registers from the SPIHP locally through dedicated register read and register write instructions. All locally addressable registers are 4 bytes wide and aligned to 4-byte boundaries. Each local address (R7:0) corresponds to an individual, 4-byte aligned, 4-byte wide register.

Register Read

Register read instructions return data from least significant byte to the most significant byte and then wrap. For example, a register read instruction with 5 data cycles produces the register contents in the following order: (D7:0), (D15:8), (D23:16), (D31:24), (D7:0). Register read instructions can be terminated after any number of transferred bytes. The ***SPIHP Address Offsets*** table shows the local address and offset for the SHIHP read and write registers.

Table 30-7: SPIHP Address Offsets

Register	Description	Local Address (R7:0)	Address Offset
SPIHP_CTL	SPIHP Control	0x00	0x000
SPIHP_STAT	SPIHP Status	0x01	0x004
SPIHP_RDPF	SPIHP Read Prefetch	0x04	0x008
SPIHP_BARn	SPIHP Base Address Register	0x05-0x14	0x010-0x04C
SPIHP_AUXn	SPIHP Auxiliary	0x15-0x24	0x050-0x08C

NOTE: Register read operations from the SPIHP_TFIFO are exceptions to the above as continuous reads access the next byte location in the FIFO until complete (or until underflow).

NOTE: Register read operations can be performed regardless of the state of `SPIHP_STAT.RDY` bit.

Register Write Register write instructions expect data from least significant byte to most significant byte and MUST be 4 bytes or less. A register write operation with more than 4 bytes does not update the register and sets the `SPIHP_STAT.UOP` bit. A register write operation with less than 4 bytes is 0 extended.

NOTE: Register write operations performed when `SPIHP_STAT.RDY` bit ==0 result in unpredictable behavior. Only the `SPIHP_STAT` register can be safely written when the `SPIHP_STAT.RDY` bit ==0.

Write Data Stride Instructions

The host can write data to memory to discontinuous addresses using the write data stride instructions. These instructions expect a stride byte value prior to each data element of the payload for example: OPCODE (write data stride), ADDRESS_a, STRIDE_a, DATA_a, STRIDE_b, DATA_b, STRIDE_c, DATA_c results in the following memory arrangement.

Table 30-8: OPCODE Write Data Stride

Address	Data
ADDRESS_a	DATA_a
ADDRESS_a + STRIDE_a	DATA_b
ADDRESS_a + STRIDE_a + STRIDE_b	DATA_c

The `SPIHP_CTL.MSIZE` bit field defines the data element width. The following instruction examples show write data stride instructions with the `SPIHP_CTL.MSIZE` bit field set to 1 byte:

Table 30-9: Example Write Data Stride

Instruction	Byte 0 (Code)	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Write Data Stride*	0x0A	A23:16	A15:8	A7:0	S7:0	D7:0	S7:0	D7:0	...
Write Data Dual Input Stride*	0xAA	A23:16	A15:8	A7:0	S7:0, D7:0	S7:0, D7:0	...	...	
Write Data Quad Input Stride*	0x3A	A23:16	A15:8	A7:0	S7:0, D7:0, S7:0, D7:0	...	...	...	

* Read register instructions continue to read the same register (without dummy cycles) until /SPIxSS is de-asserted. Writes are limited to 4 bytes maximum.

ADSP-BF70x SPIHP Register Descriptions

SPI Host Port (SPIHP) contains the following registers.

Table 30-10: ADSP-BF70x SPIHP Register List

Name	Description
SPIHP_AUX[n]	Auxiliary Register
SPIHP_BAR[n]	Base Address Register
SPIHP_CTL	Control Register
SPIHP_RDPF	Read Prefetch Register
SPIHP_STAT	Status Register

Auxiliary Register

The SPIHP auxiliary registers (SPIHP_AUX[n]) are software-definable registers for communication between the SPIHP and the SPI host. These registers may be used for message passing, identification, other items as defined by the requirements of the application.

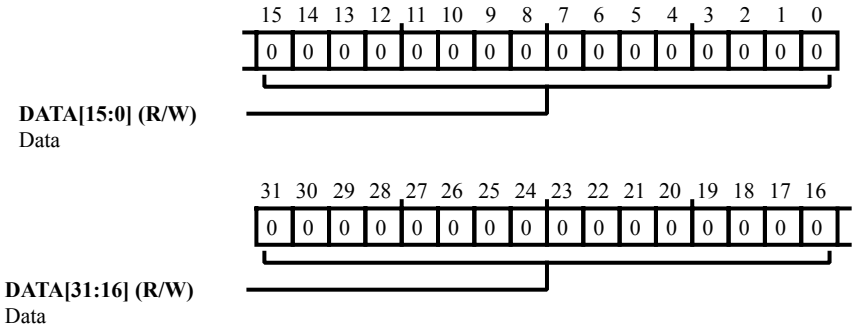

Figure 30-4: SPIHP_AUX[n] Register Diagram

Table 30-11: SPIHP_AUX[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	DATA	Data.

Base Address Register

The SPIHP base address registers (SPIHP_BAR[n]) hold an 8-bit address offset to be pre-pended to the address supplied by the host. The Base Address Offset (BAO) value of the selected SPIHP_BAR[n] is used as bits 31-24 for the local address.

The SPIHP_BAR[n] is selected by the SPIHP_CTL.BARSEL bit field. The host or slave may configure the values of all the SPIHP_BAR registers before operation and the host can simply switch pages on-the-fly by writing to the SPIHP_CTL.BARSEL bit field to select the correct SPIHP_BAR[n] register.

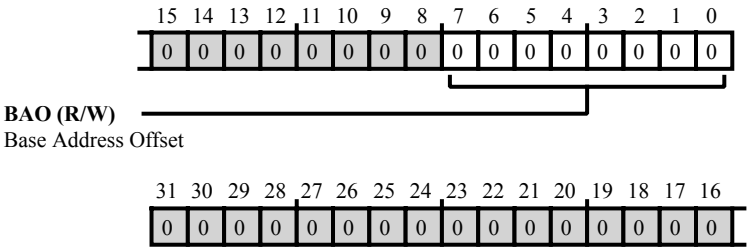

Figure 30-5: SPIHP_BAR[n] Register Diagram

Table 30-12: SPIHP_BAR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	BAO	Base Address Offset. The SPIHP_BAR[n].BAO bit field is set to the 8 MSBs of the base address of the local memory space that the SPI host targets. For example, if the host targets L2 of the SPI slave, it must set one of the SPIHP_BAR[n] registers to hold the value of the 8 MSBs of the base address for L2.

Control Register

The SPIHP_CTL register provides control fields for SPIHP operation.

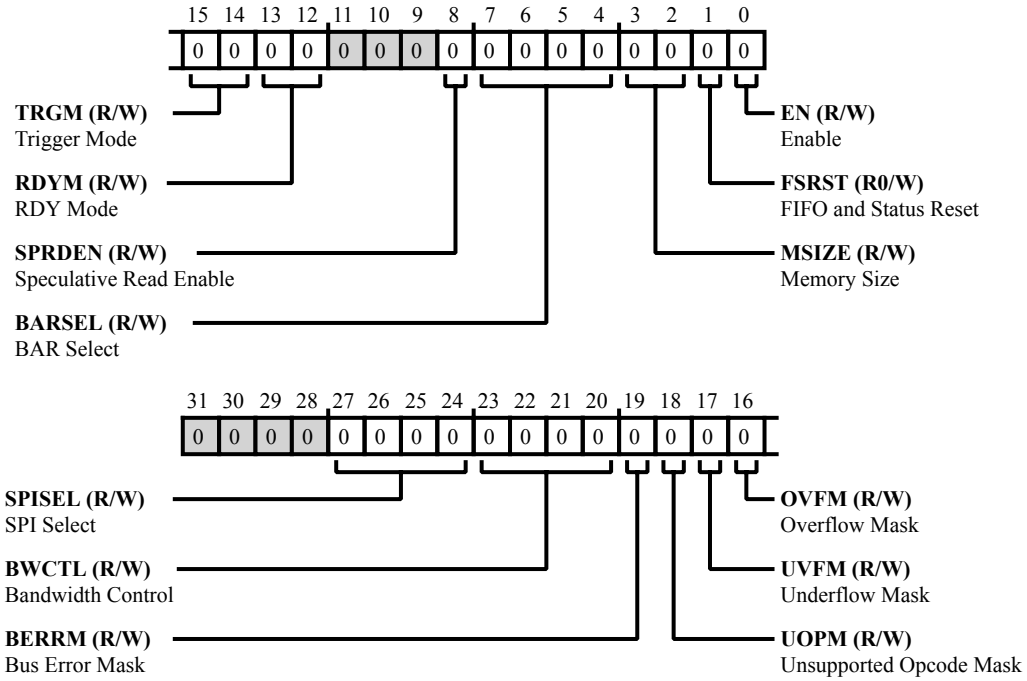

Figure 30-6: SPIHP_CTL Register Diagram

Table 30-13: SPIHP_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
27:24 (R/W)	SPISEL	SPI Select. The SPIHP_CTL.SPISEL bit field selects the SPI used for the physical interface for the SPIHP connection to the SPI host. These bits must not be modified while the SPIHP_CTL.EN bit ==1.	
		0	Select SPI0
		1	Select SPI1
		2	Select SPI2
		3-15	Reserved

Table 30-13: SPIHP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
23:20 (R/W)	BWCTL	Bandwidth Control. The SPIHP_CTL.BWCTL bit field sets the limit for the total number of outstanding transactions allowed by the system master interface. The value represents the total number of outstanding burst requests so the data transferred is a function of burst length and burst size. Burst length depends on fabric burst length support, address alignment, fifo capacity, etc. Burst size also depends on the SPIHP_CTL.MSIZE setting.	
		0	1 Transaction
		1	2 Transactions
		2	4 Transactions
		3-15	Reserved
19 (R/W)	BERRM	Bus Error Mask. The SPIHP_CTL.BERRM bit controls whether BERR status asserts the interrupt output.	
		0	Disable
		1	Enable
18 (R/W)	UOPM	Unsupported Opcode Mask. The SPIHP_CTL.UOPM bit controls whether UOPF status asserts the interrupt output.	
		0	Disable
		1	Enable
17 (R/W)	UVFM	Underflow Mask. The SPIHP_CTL.UVFM bit controls whether UVF status asserts the interrupt output.	
		0	Disable
		1	Enable
16 (R/W)	OVFM	Overflow Mask. The SPIHP_CTL.OVFM bit controls whether OVF status asserts the interrupt output.	
		0	Disable
		1	Enable

Table 30-13: SPIHP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15:14 (R/W)	TRGM	Trigger Mode. The SPIHP_CTL.TRGM bit field controls the source for the trigger output (SPIHPTO) signal. "RDYSTKY results in SPIHPTO reflecting the state of the SPIHP_STAT.RDYSTKY status bit.	
		0	RDYSTKY
		1	Reserved
		2	Reserved
		3	Reserved
13:12 (R/W)	RDYM	RDY Mode. The SPIHP_CTL.RDYM bit field controls the source for the SPIRDY signal. RDY results in SPIRDY reflecting the state of the SPIHP_STAT.RDY bit. RDYSTKY results in SPIRDY reflecting the state of the SPIHP_STAT.RDYSTKY bit.	
		0	RDY
		1	RDYSTKY
		2	Reserved
		3	Reserved
8 (R/W)	SPRDEN	Speculative Read Enable. The SPIHP_CTL.SPRDEN bit enables the system master interface to perform speculative reads beyond the current address to avoid interrupted flow of read data. It is intended to be used for memory regions (as opposed to sparsely populated address space such as sMMRs) to improve performance and avoid underflow and/or RDY deassertion. Applies to all read data operations. When the SPIHP_CTL.SPRDEN bit is set, read data operations must be limited to SPIHP_CTL.MSIZE. Any read data operation that attempts to access more than SPIHP_CTL.MSIZE results in the SPIHP_STAT.BERR bit being set.	
		0	Disable
		1	Enable
7:4 (R/W)	BARSEL	BAR Select. The SPIHP_CTL.BARSEL bit field selects the SPIHP_BAR[n] register used for address translation.	
		0-15	BAR 00 to 15

Table 30-13: SPIHP_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/W)	MSIZE	Memory Size. The SPIHP_CTL.MSIZE bit field controls the transfer size for the system master interface (read data, write data, and prefetch instructions). The SPIHP_CTL.MSIZE bit field must not be modified while the SPIHP_STAT register ==0 or non-deterministic behavior may result.	
		0	1-Byte
		1	2-Bytes
		2	4-Bytes
		3	Reserved
1 (R0/W)	FSRST	FIFO and Status Reset. The SPIHP_CTL.FSRST bit resets the SPIHP status including SPIHP_TFIFO/ SPIHP_RFIFO flush.	
		0	No Action
		1	Reset
0 (R/W)	EN	Enable. The SPIHP_CTL.EN bit controls the functional state of the SPIHP.	
		0	Disable
		1	Enable

Read Prefetch Register

The SPIHP read prefetch register (SPIHP_RDPF) holds a 24-bit address and an 8-bit transfer count for read prefetching. If the SPIHP_RDPF.CNT bit field == N and the SPIHP_CTL.MSIZE bit field == M, the system master interface reads NxM bytes from consecutive locations and writes them to the TFIFO. These locations begin at the address specified in the SPIHP_BAR[n].BAO bit field.

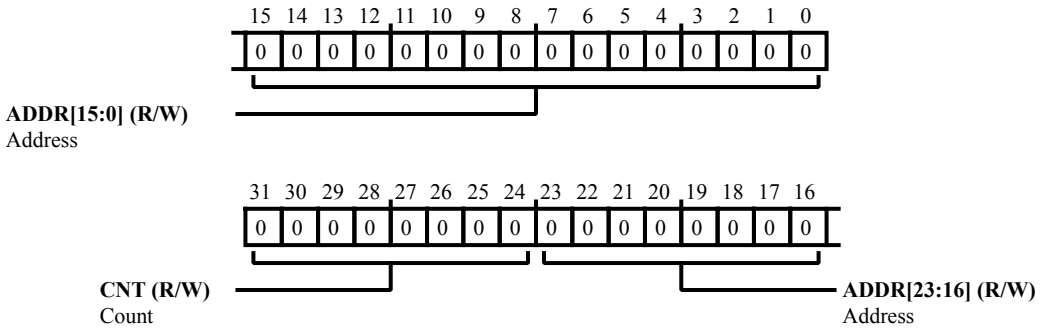

Figure 30-7: SPIHP_RDPF Register Diagram

Table 30-14: SPIHP_RDPF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	CNT	Count. The SPIHP_RDPF.CNT bit field specifies the transfer count for the prefetch operation. Each transfer consists of SPIHP_CTL.MSIZE bytes.
23:0 (R/W)	ADDR	Address. The SPIHP_RDPF.ADDR bit field specifies the address target for the prefetch operation. The full 32-bit address used by the system master interface for the prefetch is ADDR + (SPIHP_BAR[n] 24).

Status Register

The SPIHP status register (SPIHP_STAT) provides status information for SPIHP operation.

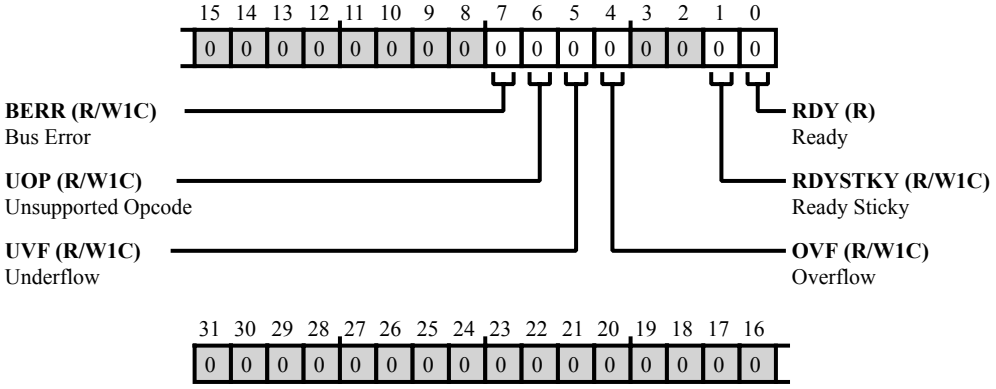

Figure 30-8: SPIHP_STAT Register Diagram

Table 30-15: SPIHP_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	BERR	Bus Error. The SPIHP_STAT.BERR bit is set by any of the following: • SLVERR/DECERR returned for any transfer over system bus • Misaligned address errors (32b xfer with A1:0 != 00 or 16b xfer with A0 != 0) • Payload error (write data instruction terminates with non-integer number of SPIHP_CTL.MSIZE wide transfers in the SPIHP_RFIFO) • Speculative read error (read data operation larger than SPIHP_CTL.MSIZE when the SPIHP_CTL.SPRDEN bit is not set)	
		0	Inactive
		1	Active
6 (R/W1C)	UOP	Unsupported Opcode. The SPIHP_STAT.UOP bit is set by a host-initiated command/opcode that is invalid/unsupported including access violations for local read/write register instructions (for example, a write register with an index pointing to a NW register or a write register with more than a 4 byte payload).	
		0	Inactive
		1	Active

Table 30-15: SPIHP_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W1C)	UVF	Underflow. The SPIHP_STAT.UVF bit is set by a host-initiated read that underflows the SPIHP_TFIFO	
		0	Inactive
		1	Active
4 (R/W1C)	OVF	Overflow. The SPIHP_STAT.OVF bit is set by a host-initiated read that underflows the SPIHP_TFIFO.	
		0	Inactive
		1	Active
1 (R/W1C)	RDYSTKY	Ready Sticky. The SPIHP_STAT.RDYSTKY bit indicates that the SPIHP is ready for a read data, write data, or prefetch operation. The SPIHP_STAT.RDYSTKY bit is set by SPIHP_STAT.RDY transition from 0 to 1 and is cleared only by a W1C operation.	
		0	Not Ready
		1	Ready
0 (R/NW)	RDY	Ready. The SPIHP_STAT.RDY bit indicates that the SPIHP is ready for a read data, write data, or prefetch operation. New read data, write data, or prefetch instructions must not be issued while SPIHP_STAT.RDY==0. The SPIHP_STAT.RDY bit deasserts under the following conditions: • SPIHP_RFIFO is almost full during a write data instruction • SPIHP_TFIFO is almost empty during a read (or fast read) data instruction • SPIHP_STAT.RDY is low between instructions • Any of the SPIHP_STAT.OVF, SPIHP_STAT.UVF, SPIHP_STAT.UOP, or SPIHP_STAT.BERR bits are set	
		0	Not Ready
		1	Ready

31 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 SCLK0 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 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 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 31-1: 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 refer explicitly to 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 (SCLK0) 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 SCLK0 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 = [SCLK0 ÷ (SPORT_DIV_A.CLKDIV + 1)]

From this, the following equation can be used to determine the value of SPORT_DIV_A.CLKDIV, given the SCLK0 frequency and the desired frequency of the SPORT clock:

SPORT_DIV_A.CLKDIV= [(SCLK0 ÷ SPORT_ACLK) - 1]

The half SPORT also supports a 1:1 SPORT_ACLK to SCLK0 ratio (per the equations above, program the clock divisor field to zero). In this case, the resulting SPORT clock frequency is equal to SCLK0.

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), then 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

ADSP-BF70x 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 31-2: ADSP-BF70x 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

Table 31-2: ADSP-BF70x SPORT Register List (Continued)

Name	Description
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

ADSP-BF70x SPORT Interrupt List

Table 31-3: ADSP-BF70x SPORT Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
28	SPORT0_A_STAT	SPORT0 Channel A Status	Level	
29	SPORT0_A_DMA	SPORT0 Channel A DMA	Level	0
30	SPORT0_B_STAT	SPORT0 Channel B Status	Level	
31	SPORT0_B_DMA	SPORT0 Channel B DMA	Level	1
32	SPORT1_A_STAT	SPORT1 Channel A Status	Level	
33	SPORT1_A_DMA	SPORT1 Channel A DMA	Level	2
34	SPORT1_B_STAT	SPORT1 Channel B Status	Level	
35	SPORT1_B_DMA	SPORT1 Channel B DMA	Level	3

ADSP-BF70x SPORT DMA Channel List

Table 31-4: ADSP-BF70x SPORT DMA Channel List

DMA ID	DMA Channel Name	Description
DMA0	SPORT0_A_DMA	SPORT0
DMA1	SPORT0_B_DMA	SPORT0
DMA2	SPORT1_A_DMA	SPORT1
DMA3	SPORT1_B_DMA	SPORT1

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.

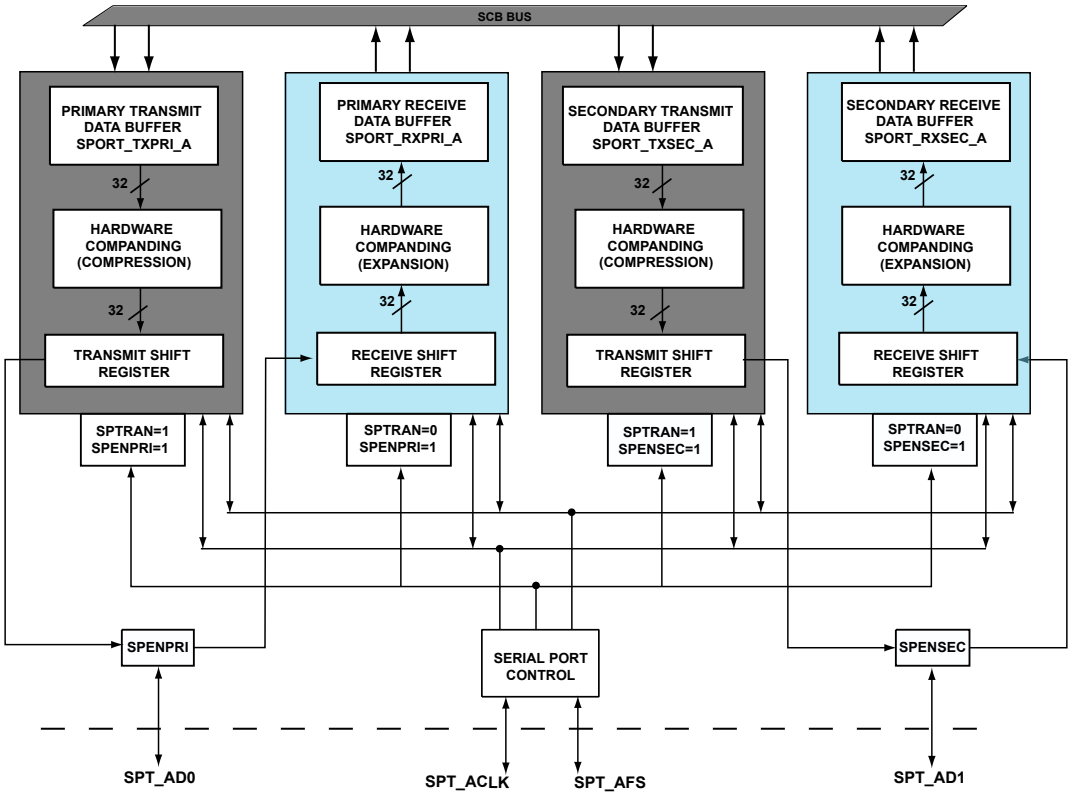

Figure 31-1: Half Serial Port Block Diagram

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.

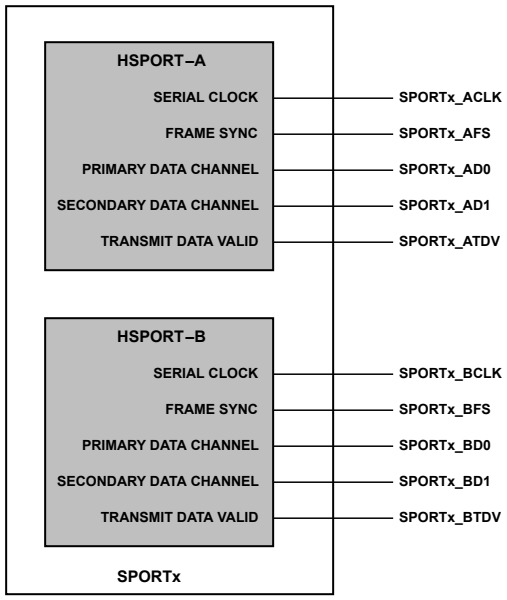

Figure 31-2: Top-Level SPORT Diagram

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 31-5: 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 31-5: 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 31-6: 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
13	0	0	1	0	HSA_CI ≤ SPB_CLK;

Table 31-6: Clock Combinations (Continued)

CLK Combination	HSA_ICLK	HSB_ICLK	CKAMUX	CKBMUX	Routing
					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 31-7: 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 via the peripheral bus or in DMA mode via 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 via the peripheral bus or in DMA mode via 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 deinterleave 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/Mutichannel modes and the various I^2S 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 31-8: Control Bits for SPORT Operating Modes

Name (Bit #)	Standard DSP Serial Mode	I^2S and Left-Justified Mode	Right-Justified Mode	Multichannel (TDM) Mode	Packed I^2S Mode
SPENPRI (0)	Yes				
DTYPE (2:1)	Yes	Reserved		Yes	
LSBF (3)	Yes	Reserved		Yes	
SLEN (8:4)	Yes				
PACK (9)	Yes				
ICLK (10)	Yes				
OPMODE (11)	= 0	= 1	= 1	= 0	= 1
CKRE (12)	Yes				
FSR (13)	Yes	Reserved			
IFS (14)	Yes				
DIFS (15)	Yes			Reserved	
LFS (16)	Yes	L_FIRST/PLFS		Yes	L_FIRST/PLFS
LAFS (17)	Yes	OPMODE2	Yes	Reserved	
RJUST (18)	Reserved		= 1	Reserved	
FSED (19)	Yes	Reserved		Yes	Reserved
TFIEN (20)	Yes				
GCLKEN (21)	Yes		Reserved		
SPENSEC (24)	Yes				
SPTRAN (25)	Yes				

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 31-9: 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.

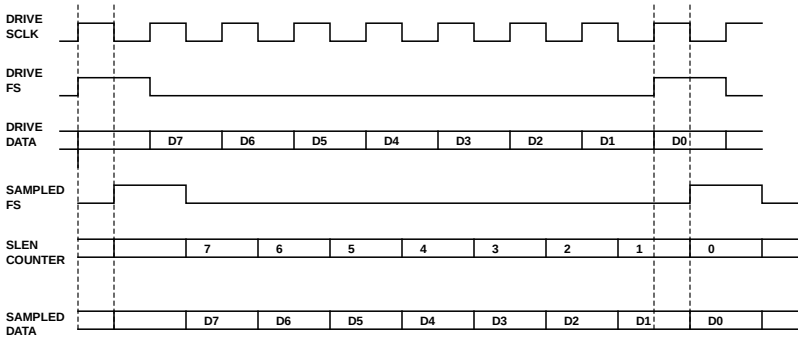

Figure 31-3: Frame Sync and Data Driven on Rising Edge

NOTE: The SCLK in the ***Frame Sync and Data Driven on Rising Edge*** figure is SCLK0.

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.

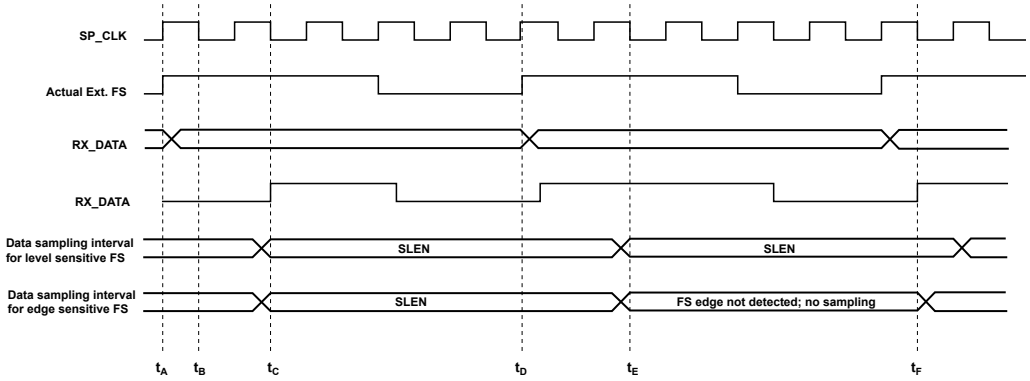

Figure 31-4: Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync

NOTE: SCLK in the ***Level-Sensitive Frame Sync versus Edge Sensitive Frame Sync*** figure is SCLK0.

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.

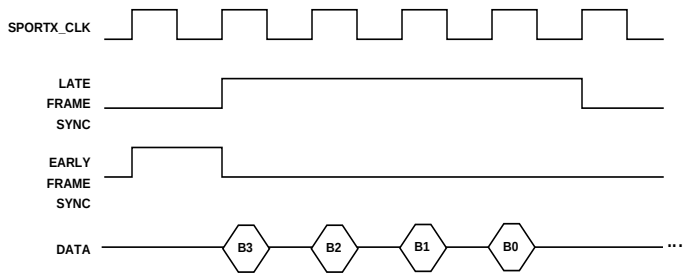

Figure 31-5: Normal Framing (Early Frame Sync) Versus Alternate Framing (Late Frame Sync)

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.

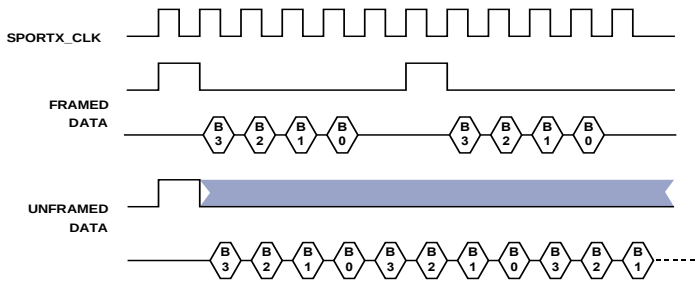

Figure 31-6: Framed versus Unframed Data Stream

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 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).

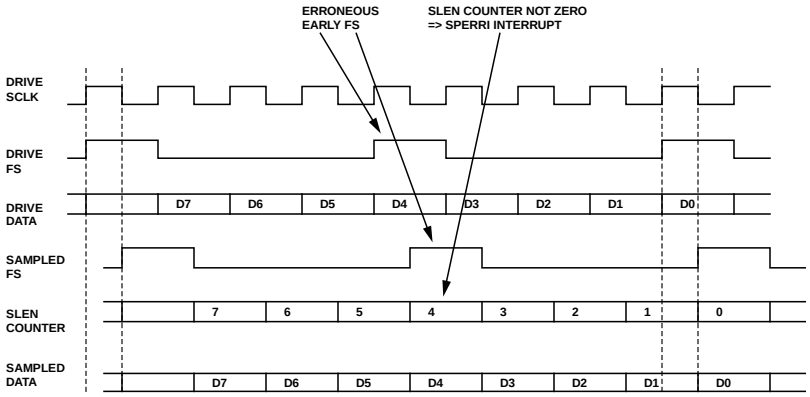

Figure 31-7: Frame Sync Error Detection

NOTE: SCLK in the ***Frame Sync Error Detection*** figure is SCLK0.

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 31-10: 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

Table 31-10: SPORT Operating Modes (Continued)

Operating Modes	SPORT_CTL_A.OPMODE	SPORT_CTL_A.LAFS	SPORT_CTL_A.RJUST	SPORT_MCTL_A.MCE
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 31-11: 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

I^2S Mode

I^2S 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 SCLK0 cycle, and the transfer begins with the left channel data word first.

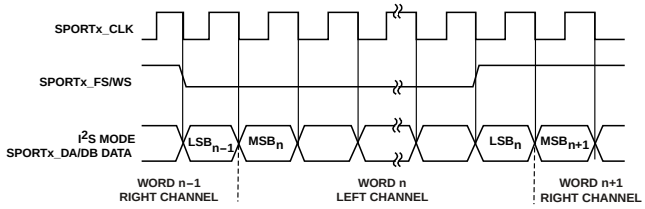

Figure 31-8: Word Select Timing in I²S Mode

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.

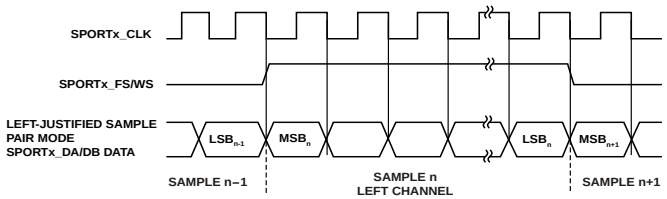

Figure 31-9: Word Select Timing in Left-Justified Mode

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 I^2S 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.

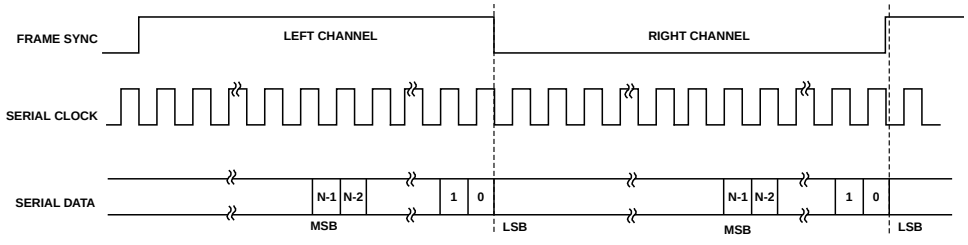

Figure 31-10: Word Select Timing in Right-Justified Mode

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.

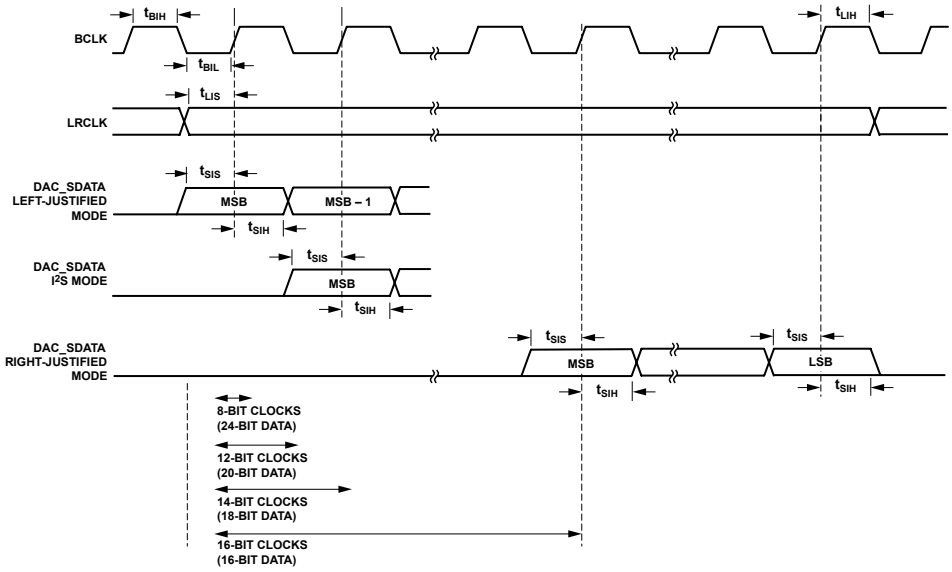

Figure 31-11: Timing Comparison Between Different Stereo Modes

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 multichannel 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 (i.e., 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 (i.e., 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.

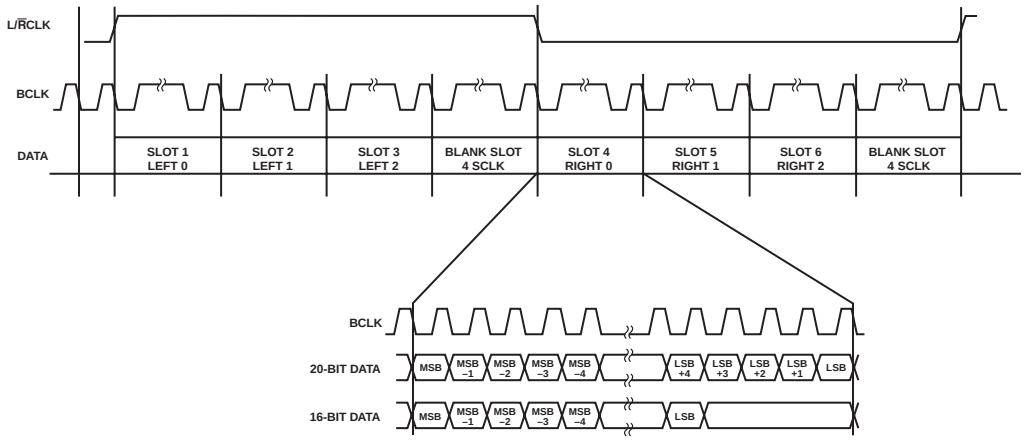

Figure 31-12: Packed I²S Mode 128 Operation

Serial Bit Clock Options

In packed I^{2}S 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 I^{2}S 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 (i.e., 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 via 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 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 when a work unit or a series of work units completes, thus signaling via the SEC to the 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, resulting in a core processor hang. 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 interrupts 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 interrupts occurring as each data word is transferred. When a SPORT is enabled with the corresponding DMA channel disabled, interrupts 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 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 will alternately load 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 will alternately read 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 will be 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 vector for data completion, as well as an error interrupt. The DMA controller can generate an interrupt 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 (e.g., 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 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 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, 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 interrupts are generated for 32-bit packed words, not for each 16-bit word.

In both cases, the interrupt 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 same data transfer interrupt 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 signal is asserted when the word count reaches zero (i.e., a DMA work unit has finished).

For transmit DMA, the interrupt is raised when the last word in the DMA work unit is loaded from the source memory to the HSPORT FIFO. This interrupt 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 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 is raised when the last word is loaded to the destination memory. In addition to that described for transmit DMA, this interrupt 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 interrupts 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.

Transfer Finish Interrupt (TFI)

When configured for transmit DMA (SPORT_CTL_A.SPTRAN = 1), the data transfer interrupt 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 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 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.

Error Detection (Status) Interrupt

In addition to the dedicated data transfer interrupt, each half SPORT also features an optional error status interrupt 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 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 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 will be 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, I^2S 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 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 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 below steps to properly initialize the SPORT for multichannel modes of operation. 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 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 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).

ADSP-BF70x SPORT Register Descriptions

Serial Port (SPORT) contains the following registers.

Table 31-12: ADSP-BF70x 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

Table 31-12: ADSP-BF70x SPORT Register List (Continued)

Name	Description
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
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).

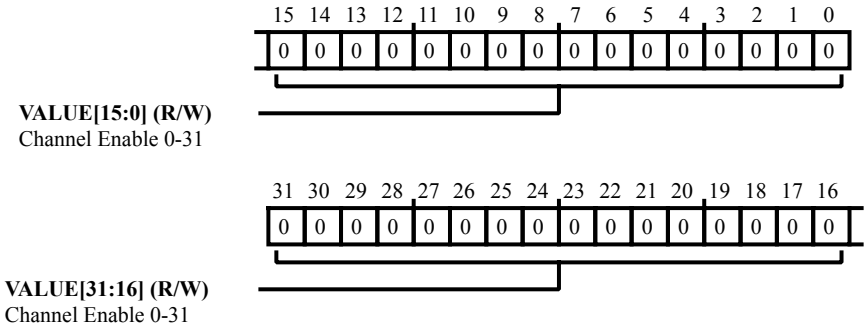

Figure 31-13: SPORT_CS0_A Register Diagram

Table 31-13: 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).

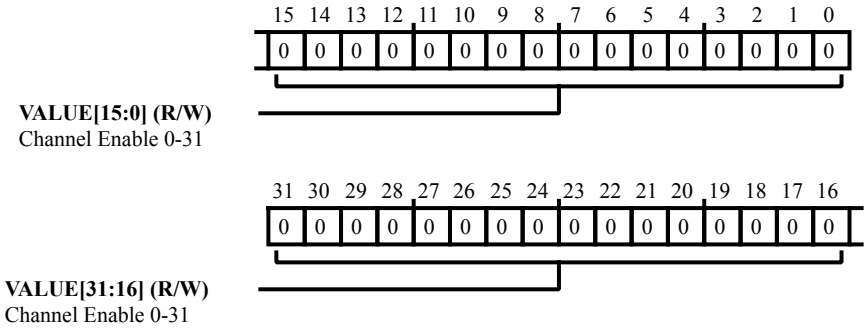

Figure 31-14: SPORT_CS0_B Register Diagram

Table 31-14: 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).

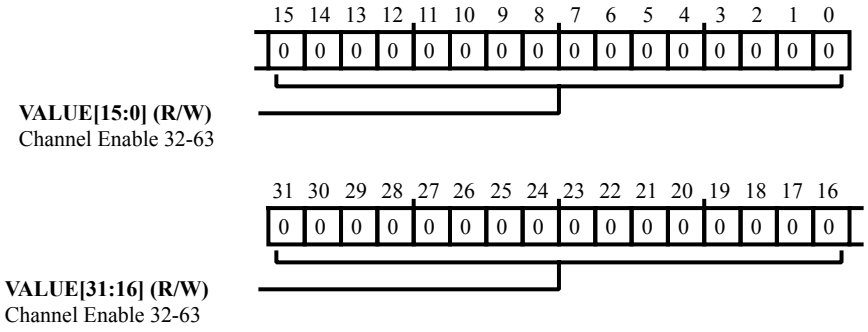

Figure 31-15: SPORT_CS1_A Register Diagram

Table 31-15: 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).

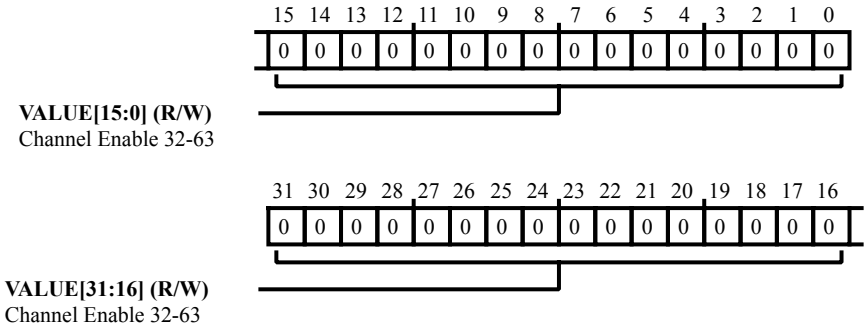

Figure 31-16: SPORT_CS1_B Register Diagram

Table 31-16: 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).

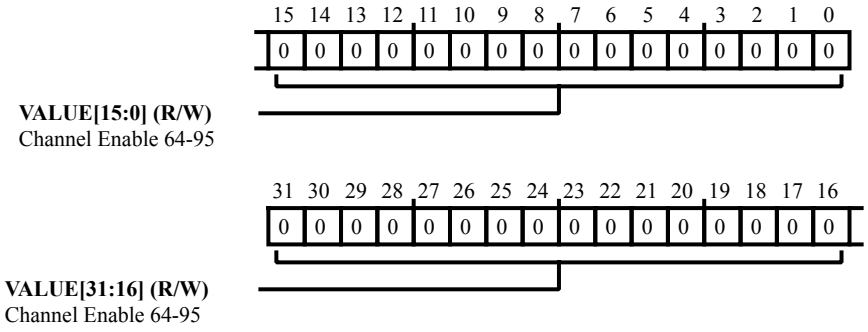

Figure 31-17: SPORT_CS2_A Register Diagram

Table 31-17: 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).

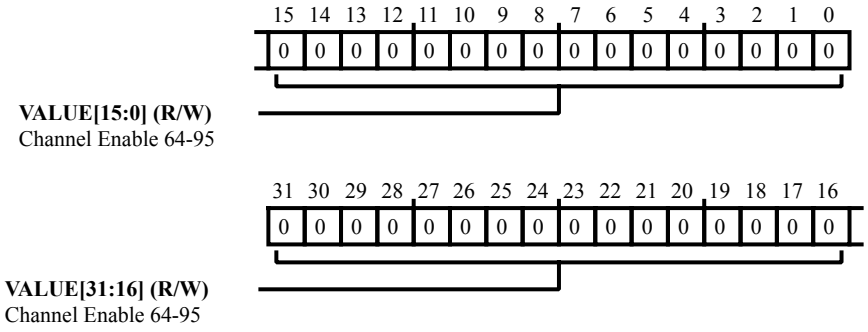

Figure 31-18: SPORT_CS2_B Register Diagram

Table 31-18: 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).

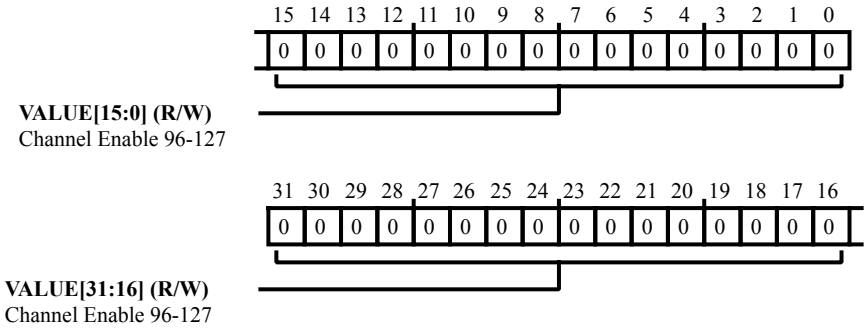

Figure 31-19: SPORT_CS3_A Register Diagram

Table 31-19: 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).

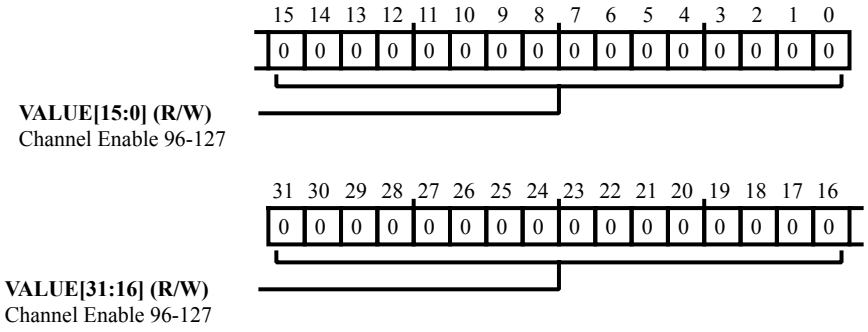

Figure 31-20: SPORT_CS3_B Register Diagram

Table 31-20: 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.

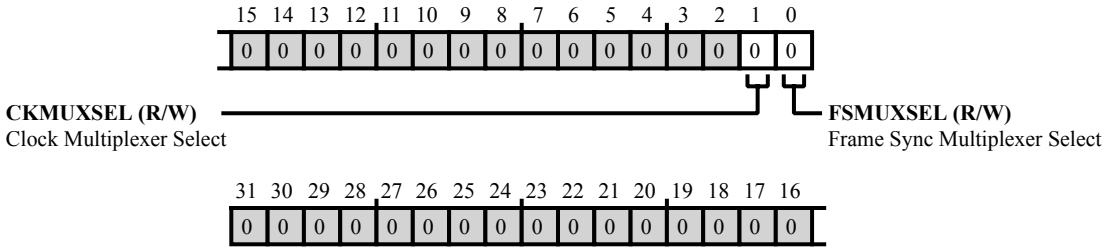

Figure 31-21: SPORT_CTL2_A Register Diagram

Table 31-21: 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.

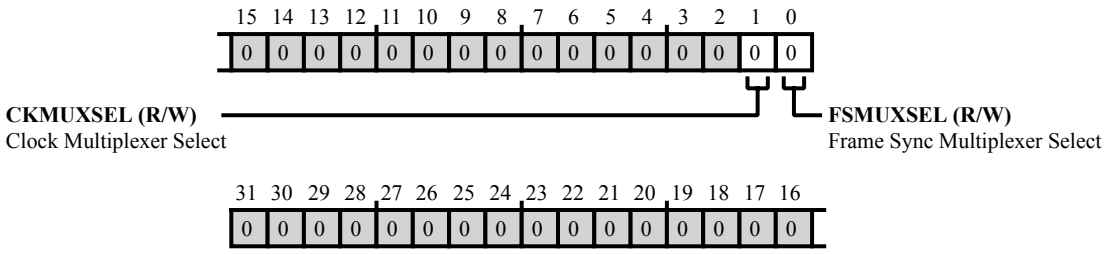

Figure 31-22: SPORT_CTL2_B Register Diagram

Table 31-22: 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.

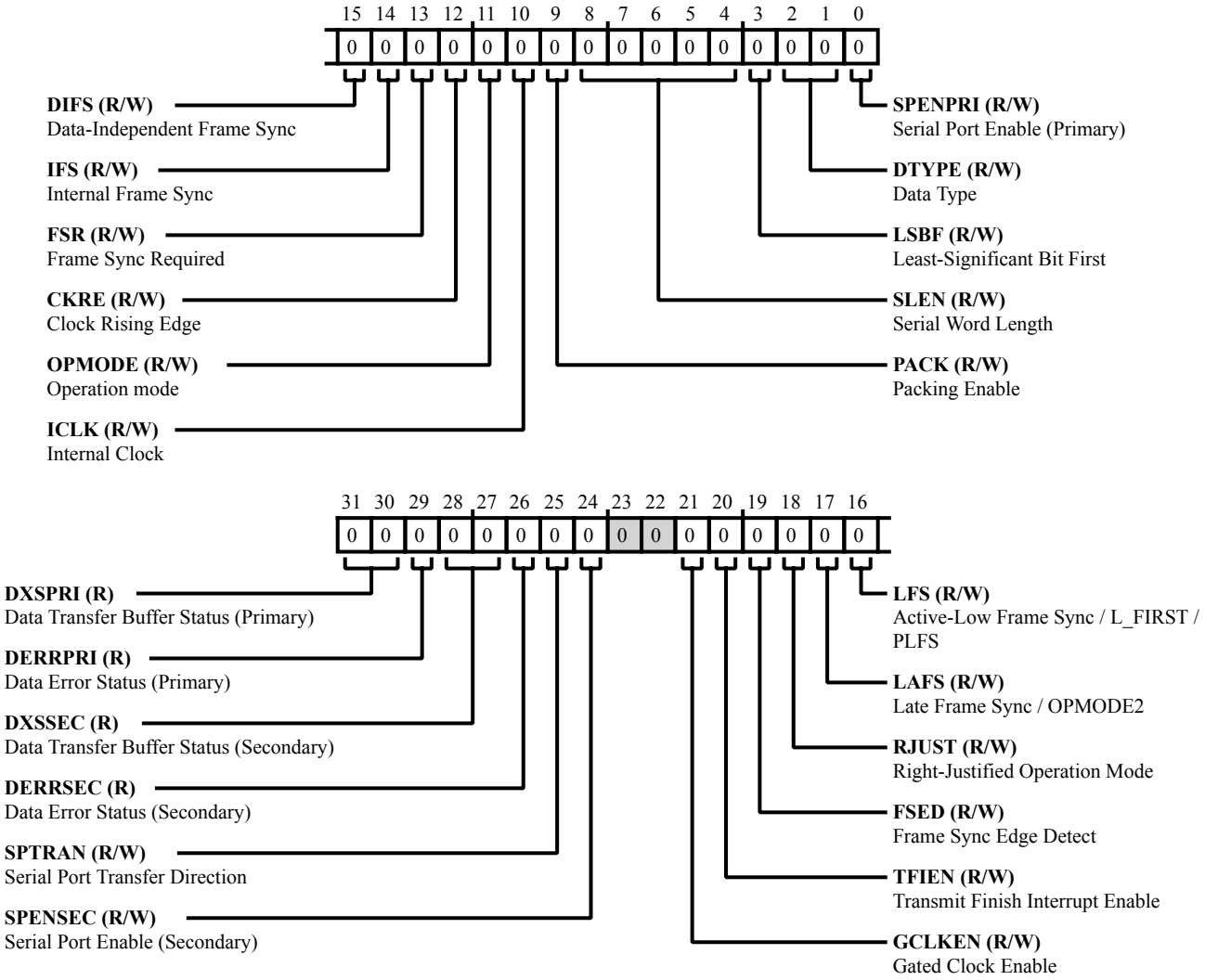

Figure 31-23: SPORT_CTL_A Register Diagram

Table 31-23: 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 31-23: 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.DERRSTAT 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 31-23: 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, if a DMA complete interrupt is enabled by the DMA_CFG.INT configuration. When enabled (SPORT_CTL_A.TFIEN =1), the DMA complete peripheral interrupt 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 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 31-23: 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 31-23: 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 31-23: 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
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	

Table 31-23: SPORT_CTL_A Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		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.

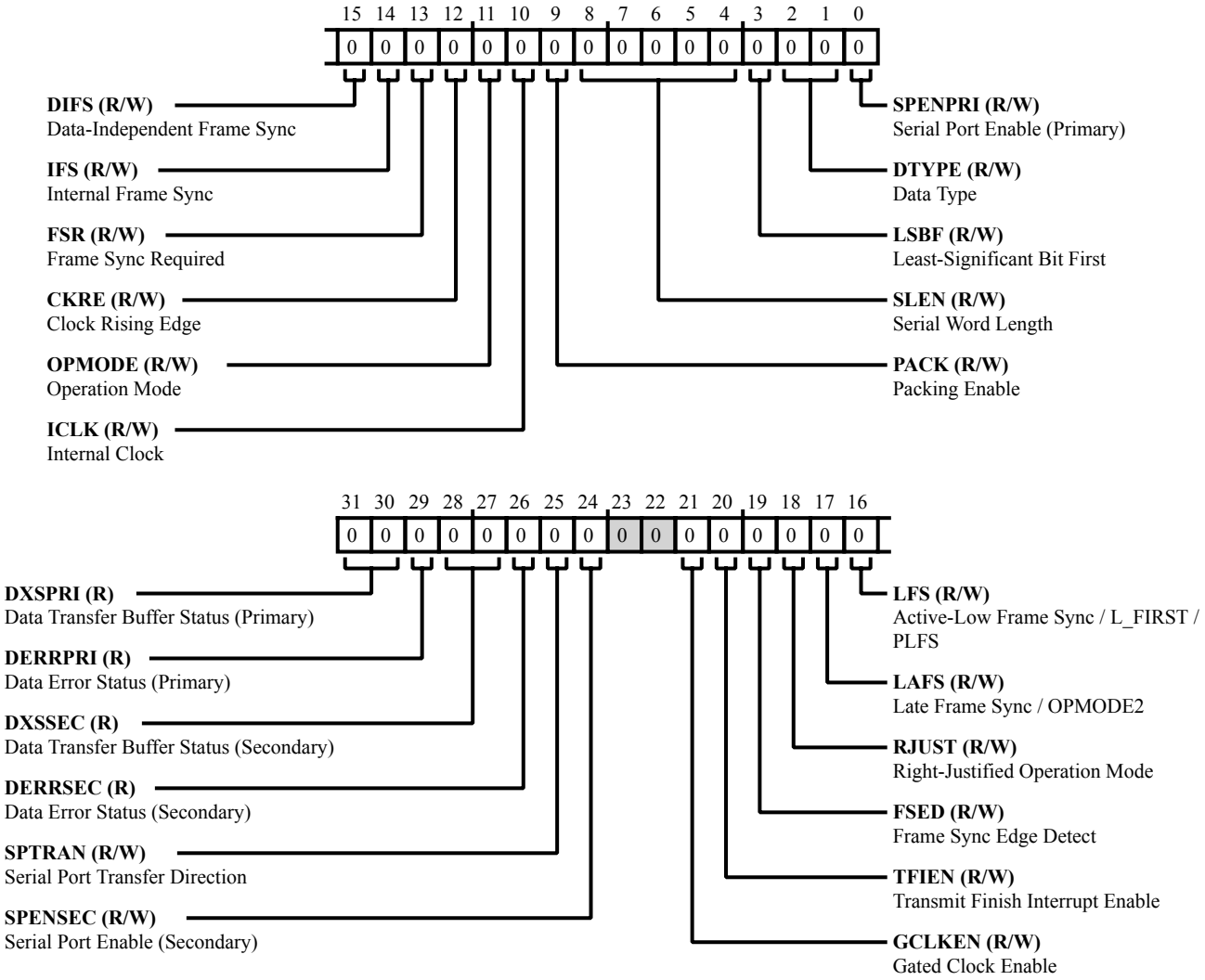

Figure 31-24: SPORT_CTL_B Register Diagram

Table 31-24: 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 31-24: 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 31-24: 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, if a DMA complete interrupt is enabled by the DMA_CFG.INT configuration. When enabled (SPORT_CTL_B.TFIEN =1), the DMA complete peripheral interrupt 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 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 31-24: 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 31-24: 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 31-24: 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 I²S / 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 I²S / 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	I²S/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 31-24: 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 31-24: 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'.

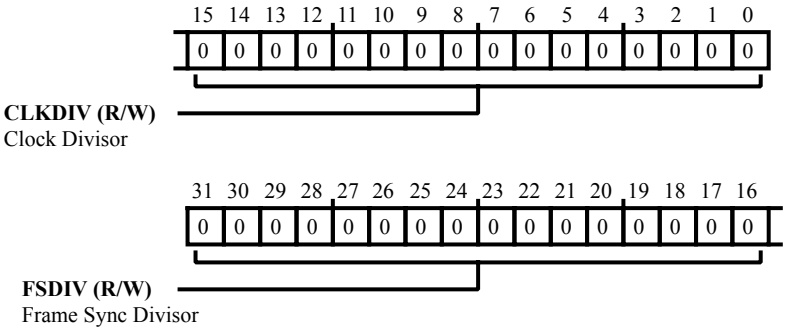

Figure 31-25: SPORT_DIV_A Register Diagram

Table 31-25: 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 (SCLK0). 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 = (SCLK0 / 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'.

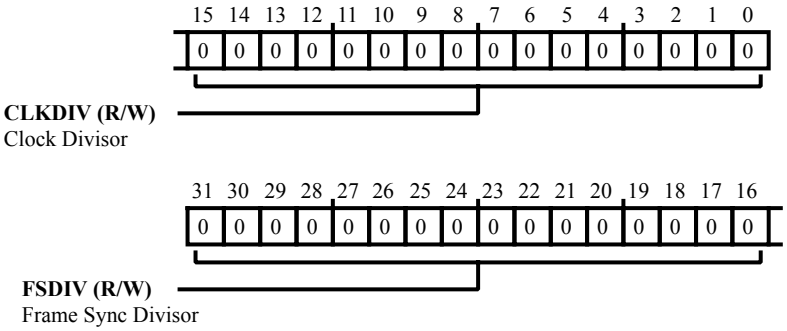

Figure 31-26: SPORT_DIV_B Register Diagram

Table 31-26: 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 (SCLK0). 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 = (SCLK0 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.

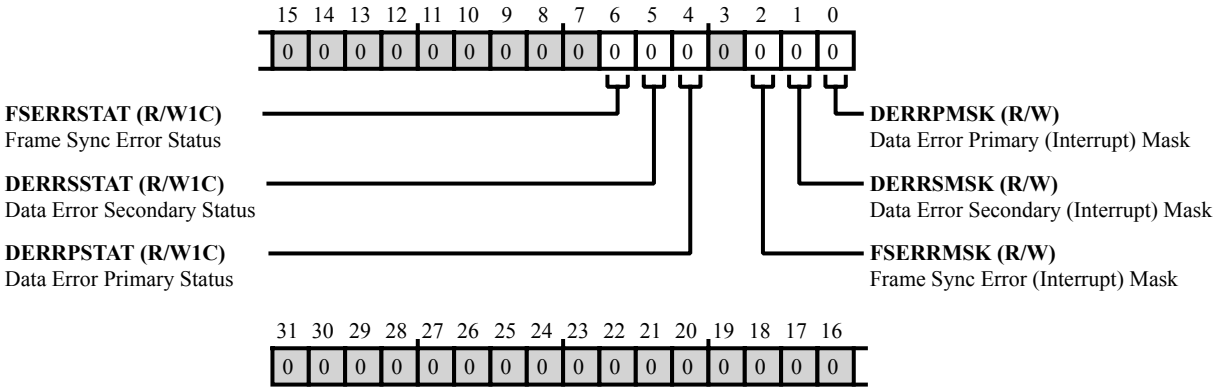

Figure 31-27: SPORT_ERR_A Register Diagram

Table 31-27: 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 31-27: 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 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 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.	
		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 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 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.

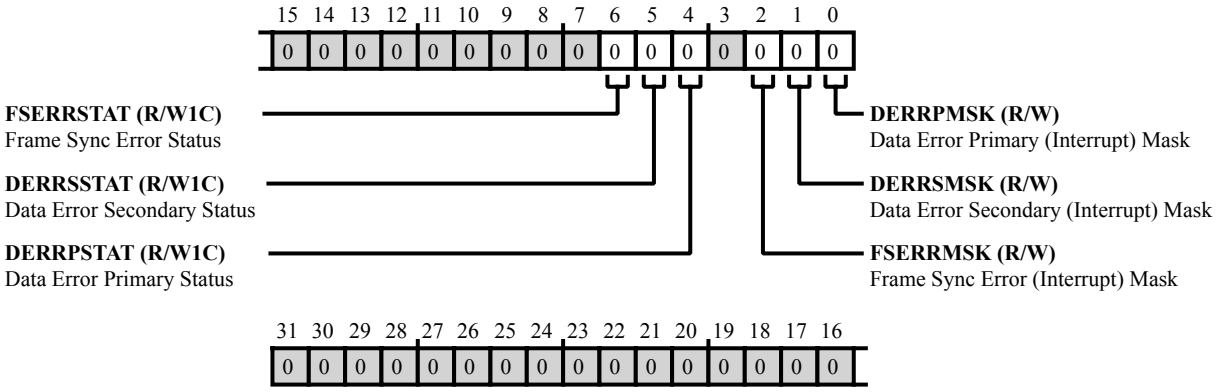

Figure 31-28: SPORT_ERR_B Register Diagram

Table 31-28: 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 31-28: 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 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 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.	
		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 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 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.

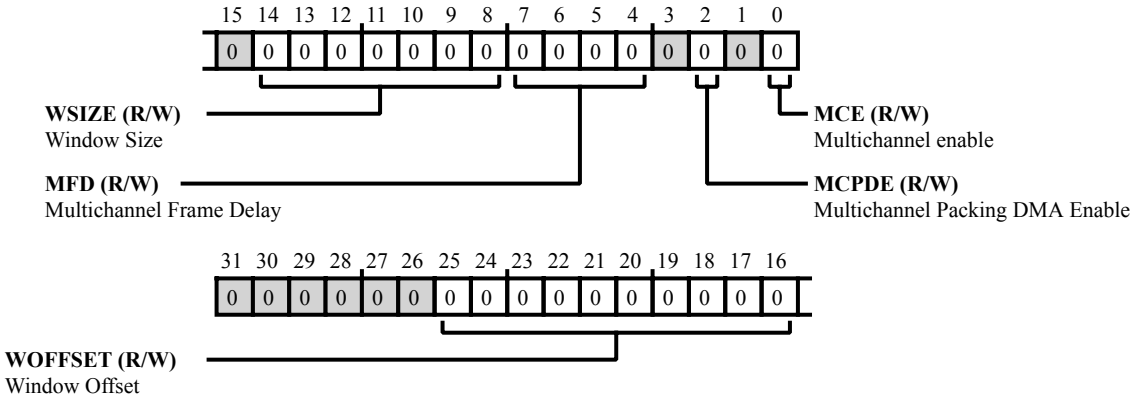

Figure 31-29: SPORT_MCTL_A Register Diagram

Table 31-29: 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 31-29: 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.

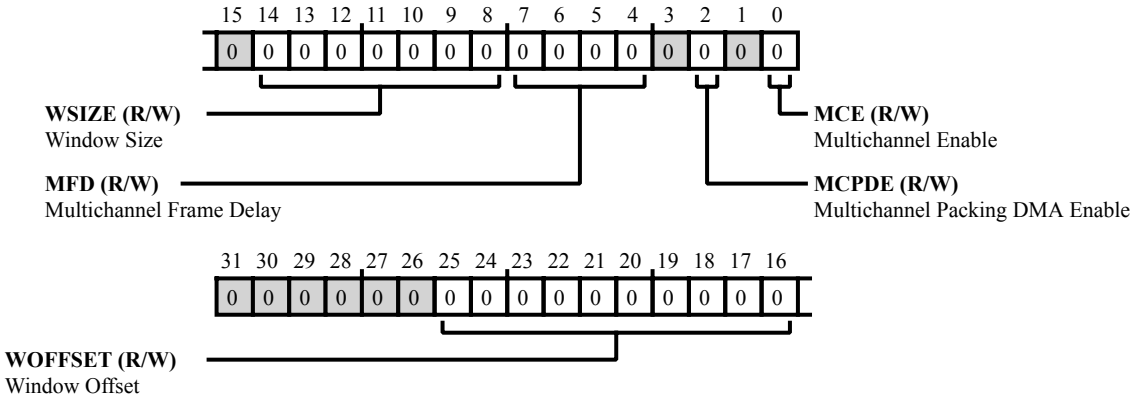

Figure 31-30: SPORT_MCTL_B Register Diagram

Table 31-30: 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 31-30: 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.

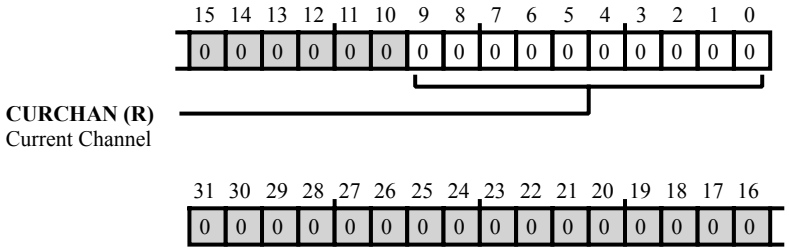

Figure 31-31: SPORT_MSTAT_A Register Diagram

Table 31-31: 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.

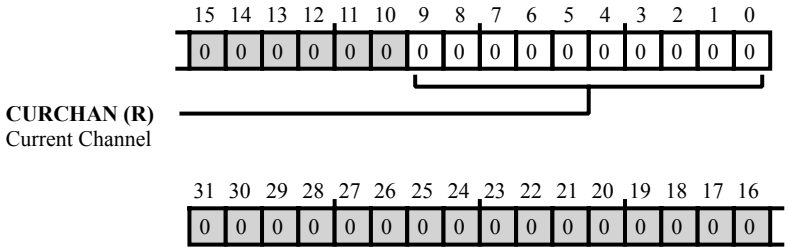

Figure 31-32: SPORT_MSTAT_B Register Diagram

Table 31-32: 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 (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_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).

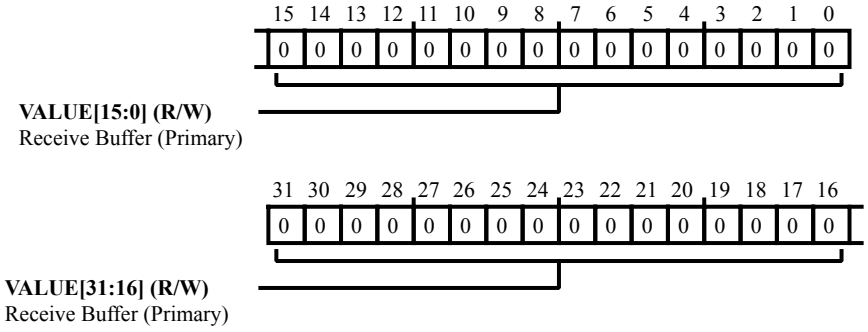

Figure 31-33: SPORT_RXPRI_A Register Diagram

Table 31-33: 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 (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_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).

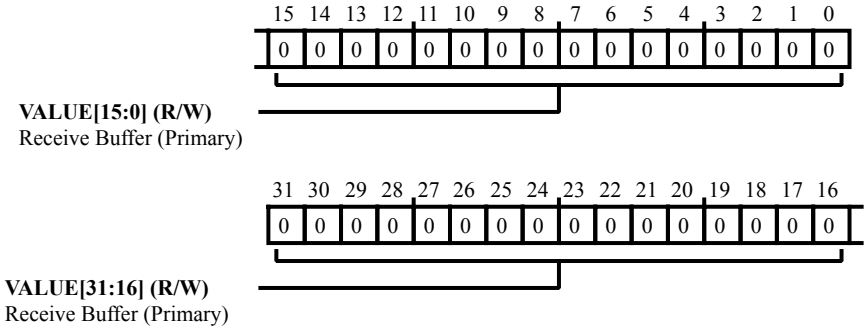

Figure 31-34: SPORT_RXPRI_B Register Diagram

Table 31-34: 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).

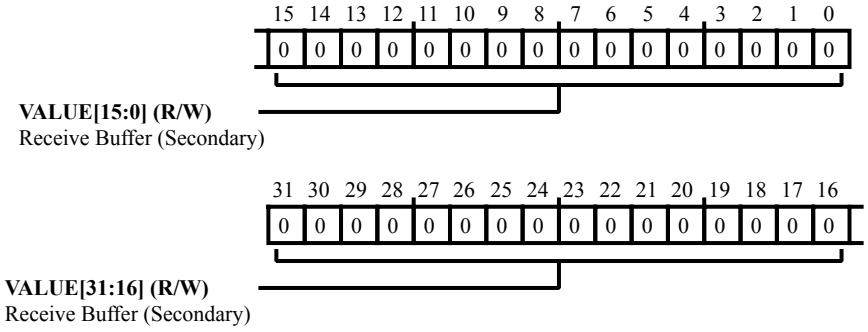

Figure 31-35: SPORT_RXSEC_A Register Diagram

Table 31-35: 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).

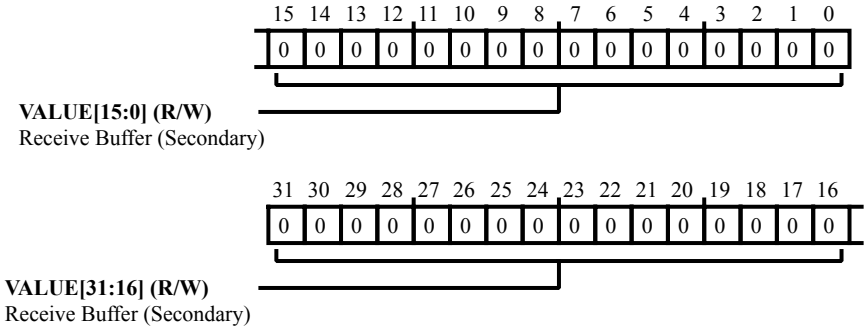

Figure 31-36: SPORT_RXSEC_B Register Diagram

Table 31-36: 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 (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 does not occur when the half SPORT is executing a DMA-based transfer.

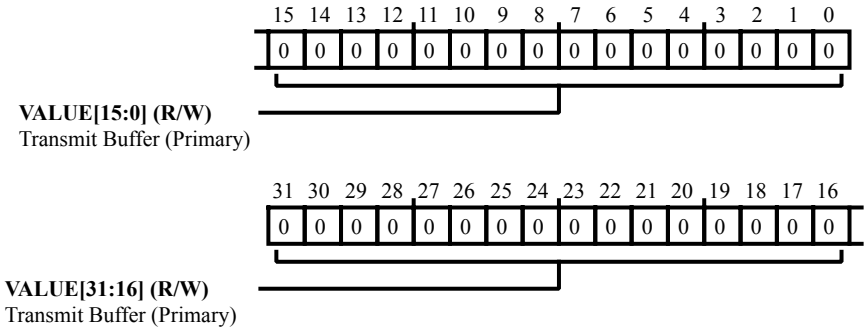

Figure 31-37: SPORT_TXPRI_A Register Diagram

Table 31-37: 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 (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 does not occur when the half SPORT is executing a DMA-based transfer.

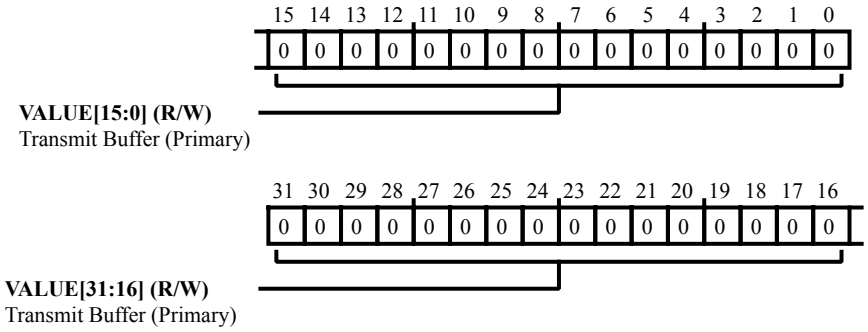

Figure 31-38: SPORT_TXPRI_B Register Diagram

Table 31-38: 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 (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 does not occur when the half SPORT is executing a DMA-based transfer.

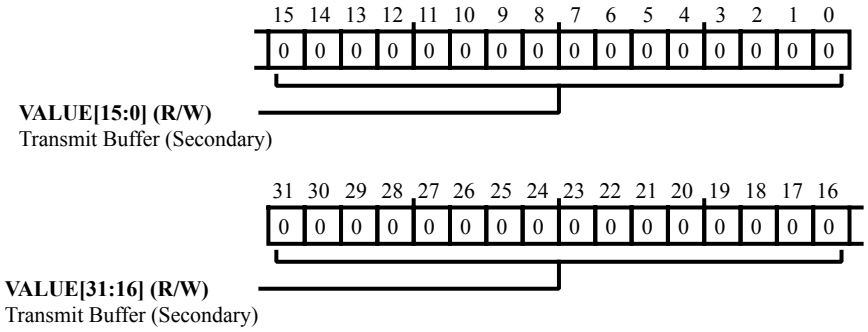

Figure 31-39: SPORT_TXSEC_A Register Diagram

Table 31-39: 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 (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 does not occur when the half SPORT is executing a DMA-based transfer.

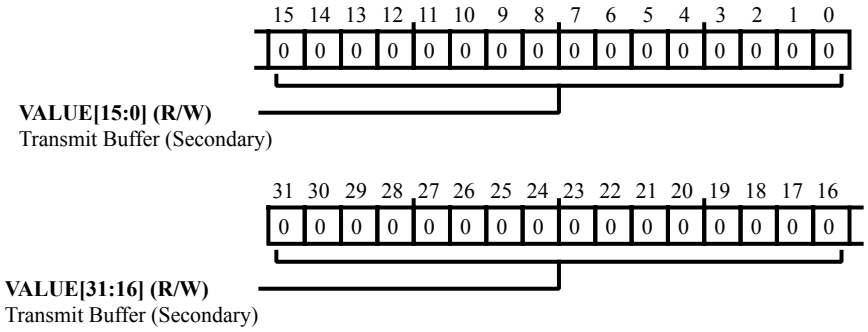

Figure 31-40: SPORT_TXSEC_B Register Diagram

Table 31-40: 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.

32 Enhanced Parallel Peripheral Interface (EPPI)

The Enhanced Parallel Peripheral Interface (EPPI) is a half-duplex, bidirectional port with a dedicated clock pin and three frame sync (FS) pins. It can support direct connections to active TFT LCDs, parallel A/D and D/A converters, video encoders and decoders, image sensor modules and other general-purpose peripherals. Each EPPI has two DMA channels associated with it. Moreover, in some modes, an EPPI can use an extra DMA channel.

EPPI Features

The EPPI module supports the following features.

- Programmable data length from 8 bits to up to 24 bits per clock cycle (depending on the product model)
- Bidirectional and half-duplex port
- Internal or external clock source
- Clock gating by an external device asserting the clock gating control signal
- Various framed and non-framed operating modes, as well as internal or external frame syncs
- Various general-purpose modes with 0, 1, 2, and 3 frame sync modes for both receive and transmit
- Ignores premature external frame syncs for data consistency
- SMPTE274M and SMPTE 296M high definition format support
- ITU-656, SMPTE 296M and SMPTE 274M status word error detection and correction for ITU-656 receive modes
- ITU-656, SMPTE 296M and SMPTE 274M receive modes – active video only, vertical blanking only, and entire field
- ITU-656, SMPTE 296M and SMPTE 274M preamble and status word decode
- Optional packing and unpacking of data to or from 32 bits from or to 8, 16 bits and 24 bits. If packing or unpacking is enabled, endianness can be altered to change the order of packing or unpacking of bytes/words
- Optional sign extension or zero-fill and alternate even or odd data sample filter for receive modes

- RGB888 to RGB666 or RGB565 conversion for transmit modes
- 4:2:2 YCrCb data Tx/Rx interleaving or de-interleaving modes
- Configurable LCD data enable (DEN) output available on frame sync 3
- Delayed start of PPI frame syncs
- Data clipping and mirroring
- Horizontal and vertical windowing for general purpose 2 and 3 frame sync modes
- Preamble, blanking and stripping support
- Multiplexing dual input

EPPI Functional Description

The EPPI has the following functionality.

RGB data formats

For transmit modes, the EPPI can convert RGB888 data in memory to either RGB565 or RGB666 at the output using bits in the Control register.

Data clipping

The EPPI contains two registers to define the lower and upper limits for the Luma and Chroma components. This functionality is used for clipping data values during 8-bit, 10-bit, 12-bit or 16-bit transmit modes.

Data mirroring

A data mirroring feature is available which mirrors the EPPI data bits 15–0. This functionality is available in both transmit and receive modes.

Windowing

The EPPI supports windowing for general-purpose input modes.

Preamble, blanking and stripping support

The EPPI can embed blanking information and clip active data to be transmitted. This functionality is available for single channel data, interleaved data, and parallel data and supports data lengths equal to 16 bits, 20 bits or 24 bits.

ADSP-BF70x EPPI Register List

The EPPI is a half-duplex, bidirectional parallel port. It comprises a clock pin, 3 frame sync pins, and a set of data pins. For more information on EPPI functionality, see the EPPI register descriptions.

Table 32-1: ADSP-BF70x EPPI Register List

Name	Description
EPPI_CLKDIV	Clock Divide Register
EPPI_CTL	Control Register
EPPI_CTL2	Control Register 2 Register
EPPI_EVENCLIP	Clipping Register for EVEN (Luma) Data Register
EPPI_FRAME	Lines Per Frame Register
EPPI_FS1_DLY	Frame Sync 1 Delay Value Register
EPPI_FS1_PASPL	FS1 Period Register / EPPI Active Samples Per Line Register
EPPI_FS1_WLHB	FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register
EPPI_FS2_DLY	Frame Sync 2 Delay Value Register
EPPI_FS2_PALPF	FS2 Period Register / EPPI Active Lines Per Field Register
EPPI_FS2_WLVB	FS2 Width Register / EPPI Lines Of Vertical Blanking Register
EPPI_HCNT	Horizontal Transfer Count Register
EPPI_HDLY	Horizontal Delay Count Register
EPPI_IMSK	Interrupt Mask Register
EPPI_LINE	Samples Per Line Register
EPPI_ODDCLIP	Clipping Register for ODD (Chroma) Data Register
EPPI_STAT	Status Register
EPPI_VCNT	Vertical Transfer Count Register
EPPI_VDLY	Vertical Delay Count Register

ADSP-BF70x EPPI Interrupt List

Table 32-2: ADSP-BF70x EPPI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
23	EPPI0_STAT	EPPI0 Status	Level	
24	EPPI0_CH0_DMA	EPPI0 Channel 0 DMA	Level	14
25	EPPI0_CH1_DMA	EPPI0 Channel 1 DMA	Level	15

ADSP-BF70x EPPI Trigger List

Table 32-3: ADSP-BF70x EPPI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
34	EPPI0_CH0_DMA	EPPI0 Channel 0 DMA	Edge
35	EPPI0_CH1_DMA	EPPI0 Channel 1 DMA	Edge

Table 32-4: ADSP-BF70x EPPI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
44	EPPI0_CH0_DMA	EPPI0 Channel 0 DMA	Pulse
45	EPPI0_CH1_DMA	EPPI0 Channel 1 DMA	Pulse

RGB Data Formats

For transmit modes, the EPPI can convert RGB888 data in memory to RGB666 at the output when the EPPI_CTL.RGBFMTEN bit is set and the EPPI_CTL.DLEN value is equal to 18 bits. Similarly, the EPPI can convert RGB888 data in memory to RGB565 at the output when the EPPI_CTL.RGBFMTEN bit is set and EPPI_CTL.DLEN is equal to 16 bits.

This conversion is performed as follows:

- If EPPI_CTL.PACKEN =1, the EPPI first unpacks according to the EPPI_CTL.SWAPEN bit setting, and the three 32-bit data words from the DMA are broken into four 24-bit data words to be transmitted out, as described earlier.
- If EPPI_CTL.PACKEN =0, the EPPI takes the lower 24 bits of the 32-bit DMA as the data to be transmitted. Then, the EPPI truncates this 24-bit data word to the required data width. It removes the lower 2 bits of G and the lower 2 bits or 3 bits of R and B.

Data Clipping

The EPPI contains two registers to define the lower and upper limits for the Luma and Chroma components. It uses these registers for clipping data values during 8-bit, 10-bit, 12-bit or 16-bit transmit modes. All data values for odd samples which are less than the value in the EPPI_ODDCLIP.LOWODD bit field are replaced with the value in the EPPI_ODDCLIP.LOWODD field. All data values for even samples which are less than the value in the EPPI_EVENCLIP.LOWEVEN field are replaced with the value in the EPPI_EVENCLIP.LOWEVEN field.

In the same manner, all data values for odd samples which are more than the value in the EPPI_ODDCLIP.HIGHODD bit field are replaced with the value in the EPPI_ODDCLIP.HIGHODD field. All data values for even samples which are more than the values in the EPPI_EVENCLIP.HIGHEVEN field are replaced with the values in the EPPI_EVENCLIP.HIGHEVEN field.

Depending on the programmed EPPI length, only the corresponding bits (least aligned) are considered for clipping. For example, if the EPPI is programmed in 10-bit mode, bits 9–0 and bits 25–16 constitute the clipping thresholds. The higher bits are ignored. The EPPI supports 8-bit, 10-bit, 12-bit, and 16-bit clipping thresholds.

For the 4:2:2 YCrCb color space, Luma and Chroma typically have different lower and upper thresholds. Separate thresholds can be required for even and odd data samples. For monochrome (Y only) or some non-video clipping applications, the value in the `EPPI_ODDCLIP.LOWODD` field can be the same as the value in the `EPPI_EVENCLIP.LOWEVEN` field. The value in the `EPPI_ODDCLIP.HIGHODD` field can be the same as the value in the `EPPI_EVENCLIP.HIGHEVEN` field.

In GP 0 FS mode with internal blanking generation, clipping is valid only for the active video part of the transmitted data. ITU-R 656 preambles, status words, and blanking data bypass the clipping logic.

If the EPPI is programmed in 16-bit mode with the `EPPI_CTL.SPLTWRD` bit set, the YDATA (luma data) gets the clipping threshold levels of the `EPPI_EVENCLIP` register. The CDATA (chroma data) gets the clipping threshold levels of the `EPPI_ODDCLIP` register.

The clipping registers are ignored when the `EPPI_CTL.RGBFMTEN` bit is set.

Data Mirroring

To increase the pin multiplexing options for the EPPI data pins, a data mirroring feature is available which mirrors the EPPI data bits 15–0. This feature is available in both transmit and receive modes. It is enabled by setting the `EPPI_CTL.DMIRR` bit.

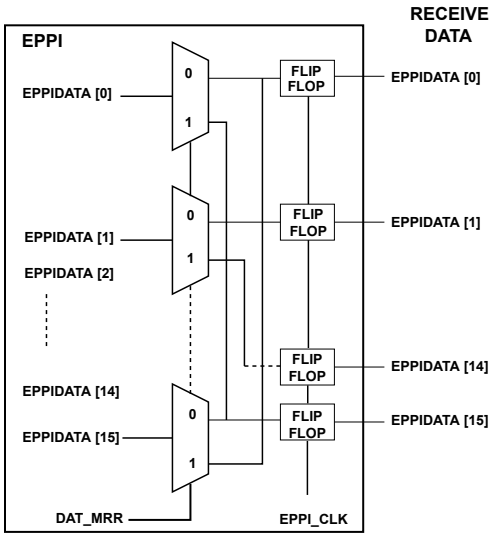

Figure 32-1: Data Mirroring Receive

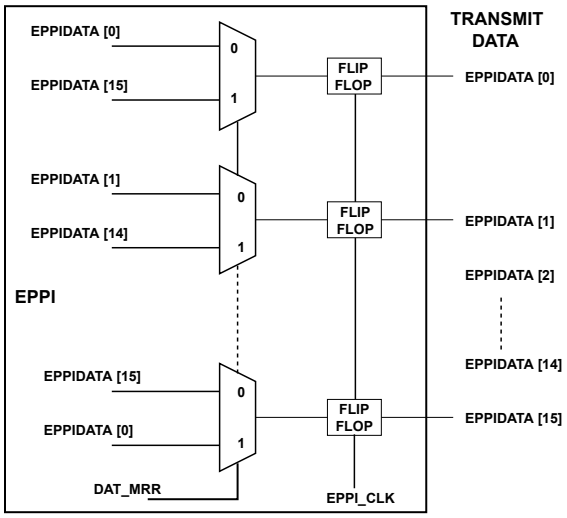

Figure 32-2: Data Mirroring Transmit

Windowing

The EPPI supports windowing for general-purpose input modes. The module can be configured to bring in a region of interest instead of the entire frame of data, which helps reduce bandwidth requirements.

Preamble, Blanking and Stripping Support

The EPPI supports embedding blanking information and clipping of active data for transmission. This functionality is available for single-channel data, interleaved data, and parallel data and supports a data length (`EPPI_CTL.DLEN`) of 16 bits.

Support of preamble generation or detection and stripping of blanking information is also provided for ITU656 mode and the SMPTE 274M and 296M HD formats. The ***Video Mode Comparison*** table shows the SMPTE standards in comparison with the ITU656 modes. Preambles for SMPTE and ITU656 modes are identical.

Table 32-5: Video Mode Comparison

Video Mode	Frame Rate	Frame Resolution	Active Video Resolution	Sampling Frequency (MHz)	Remarks
ITU656 (NTSC)	30	1716x525	720x480	27.02	Y-C interleaved
ITU656 (PAL)	25	1728x625	720x576	27.00	Y-C interleaved
SMPTE 296M	30	3300x750	1280x720	74.25	Y,C separate
	60	1650x750	1280x720	74.25	Y,C separate

Table 32-5: Video Mode Comparison (Continued)

Video Mode	Frame Rate	Frame Resolution	Active Video Resolution	Sampling Frequency (MHz)	Remarks
SMPTE 274M	30	2200x1125	1920x1080	74.25	Y,C separate
	60	2200x1125	1920x1080	148.50	Y,C separate
	25	2640x1125	1920x1080	74.25	Y,C separate
	50	2640x1125	1920x1080	148.50	Y,C separate
	24	2750x1125	1920x1080	74.25	Y,C separate

See the clock operating conditions section of the data sheet for the maximum sampling frequency for this product.

EPPI Definitions

The following definitions are helpful when using the EPPI module.

ITU-R BT.-656

Description of a digital video protocol for interfaces and data stream format required to send uncompressed PAL or NTSC standard definition TV (525 or 625 lines) signals.

YUV422

YUV is a color space where luminance (Y) and chrominance (UV) components define the pixels. The suffix signifies how the chrominance components have been decimated and provide formatting information. In this case, the YUV422 format has the chrominance decimated by two, meaning only half of each chrominance component is available. Typical YUV422 formatting interleaves the luminance and chrominance (for example, U1Y1V1Y2U2Y3V2Y4).

RGB888

RGB is a color space where three color values, one red (R), one green (G) and one blue (B), define the pixels. The suffix signifies the bit widths for these color components. In this case, RGB888 means that each red, green, and blue value is 8 bits.

RGB565

RGB is a color space where three color values, one red (R), one green (G) and one blue (B) define the pixels. The suffix signifies the bit widths for these color components. In this case, RGB565 means that the red (R) and blue (B) are 5 bits each while the green (G) is 6 bits. When packed together, each RGB565 pixel can be represented in a 16-bit data word. LCD display panels commonly use this format.

SMPTE 274M

An HD standard defining the spatial resolution (image sample structure) and frame rates for 1920x1080.

SMPTE 296M

An HD standard for defining the spatial resolution (image sample structure) and frame rates fro 1280x720.

EPPI Block Diagram

The ***EPPI Block Diagram*** figure shows the functional blocks within the EPPI.

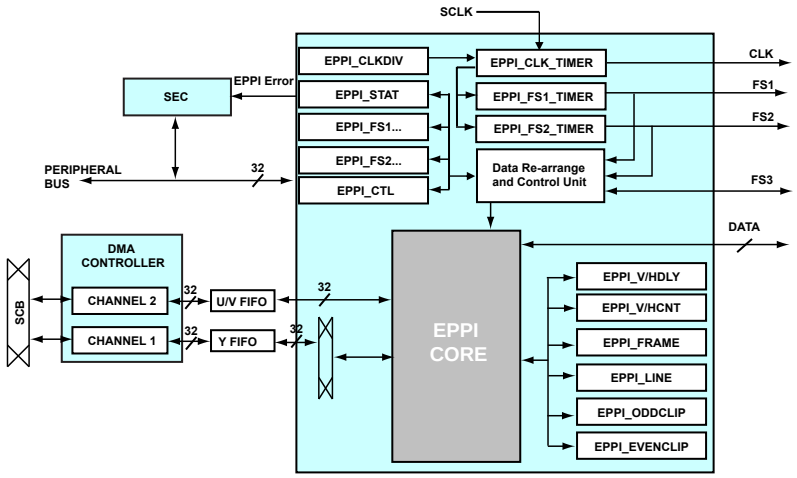

Figure 32-3: EPPI Block Diagram

EPPI Architectural Concepts

The following sections describe the architectural concepts.

- EPPI Interface
- Reset Operation
- Frame Sync Polarity and Sampling Edge
- Direct Memory Access (DMA)
- EPPI Clock

EPPI Interface

A block diagram of the architecture for the EPPI interface is shown in the ***EPPI DMA Interface*** figure.

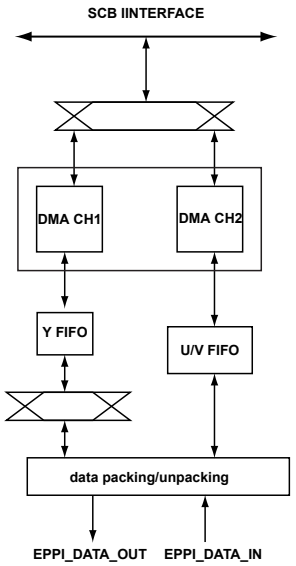

Figure 32-4: EPPI DMA Interface

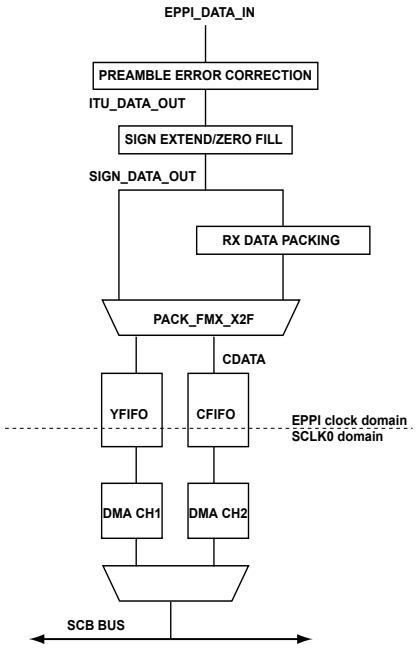

Figure 32-5: Receive Data Path

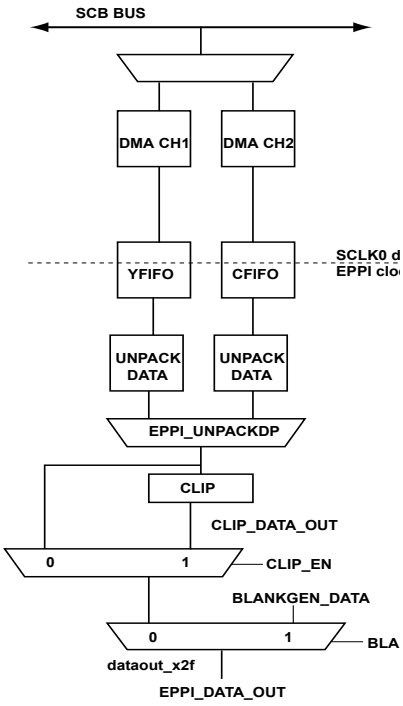

Figure 32-6: Transmit Data Path

Reset Operation

On a hardware reset, the entire EPPI is reset. All MMRs return to their default values. EPPI interrupt and DMA requests become inactive and internally generated `EPPI_CLK` and frame syncs are aborted.

In software, write 0 to the `EPPI_CTL.EN` bit to reset and reconfigure the EPPI. When disabled in this manner, only the `EPPI_STAT` register is cleared to its reset value. Interrupts and DMA requests become inactive and internally generated clock and frame syncs are aborted.

Frame Sync Polarity and Sampling Edge

The `EPPI_CTL.POLS` and `EPPI_CTL.POLC` bits provide a mechanism to select the active level of the frame syncs and the sampling or driving edge of the EPPI clock, respectively. This functionality allows the EPPI to connect to data sources and receivers with a wide array of control signal polarities. Often, the remote data source or receiver also offer configurable signal polarities. In these cases, the `EPPI_CTL.POLS` and `EPPI_CTL.POLC` bits add flexibility.

Table 32-6: Frame Sync Polarity Selections and Frame Sync Pin States

Bit Setting	Frame Sync 2	Frame Sync 1
POLS = b#00	Active high	Active high
POLS = b#01	Active high	Active low
POLS = b#10	Active low	Active high
POLS = b#11	Active low	Active low

Table 32-7: Clock Polarity Selections and Receive/Transmit Pin States

	Receive		Transmit	
Bit Setting	**Sample Data**	**Sample/Drive Syncs**	**Drive Data**	**Sample/Drive Syncs**
POLC = b#00	Falling edge	Falling edge	Rising edge	Rising edge
POLC = b#01	Falling edge	Rising edge	Rising edge	Falling edge
POLC = b#10	Rising edge	Falling edge	Falling edge	Rising edge
POLC = b#11	Rising edge	Rising edge	Falling edge	Falling edge

Direct Memory Access (DMA)

The EPPI has a native DMA controller with two channels. A local arbiter arbitrates between these channels and requests are forwarded to the system crossbar. The EPPI has one connection to the fabric.

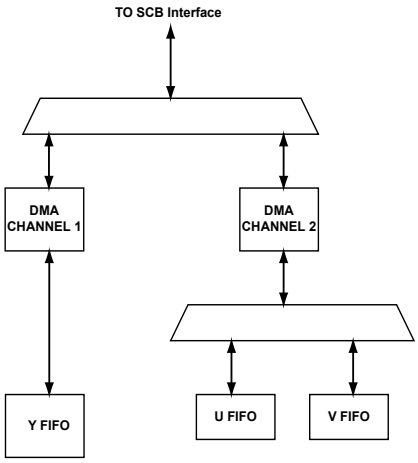

Figure 32-7: EPPI DMA Interface

The EPPI must be used with DMA. Configuring the EPPI DMA channels is a necessary step toward using the EPPI interface. The channels can be configured for either transmit or receive operation, and have a maximum throughput of (EPPI_CLK) x (32 bits/transfer). In modes where data lengths permit, packing can increase transfer bandwidth.

The DMA engine generates interrupts at the completion of a row, frame, or partial-frame transfer. The DMA engine also coordinates the source or destination point for the data that is transferred through the EPPI.

The 2D DMA capability allows the processor to be interrupted at the end of a line or after a frame of video is transferred, or if a DMA error occurs. The DMA_XCNT and DMA_YCNT registers allow for flexible data interrupt points. For example, assume the DMA_XMOD = DMA_YMOD =1. If a data frame contains 320 × 240 bytes (240 rows of 320 bytes each), the following conditions hold.

- Setting DMA_XCNT =320, DMA_YCNT = 240, and DMA_CFG.INT =1 interrupts on every row transferred, for the entire frame.
- Setting DMA_XCNT =320, DMA_YCNT =240, and DMA_CFG.INT =2 interrupts only on the completion of the frame (when 240 rows of 320 bytes have been transferred).

- Setting DMA_XCNT =38,400 (320 x 120), DMA_YCNT =2, and DMA_CFG.INT =1 causes an interrupt when half of the frame is transferred, and again when the whole frame is transferred.

The following is the general procedure for setting up DMA operation with the EPPI.

1. Configure the DMA registers as appropriate for the desired DMA operating mode.
2. Enable the DMA channel for operation.
3. Configure appropriate EPPI registers.
4. Enable the EPPI by writing 1 to the EPPI_CTL.EN bit.

EPPI Clock

The EPPI can be supplied with an external clock, or the clock can be generated internally and supplied to external devices. For information on the maximum PPI_CLK specification in internal and external clock modes, see the product-specific data sheet.

When using an external EPPI_CLK, there can be up to two cycles latency before valid data is received or transmitted.

The internal clock can be generated from SCLK0 when the EPPI_CTL.ICLKGEN bit is set. The value in the EPPI_CLKDIV register determines the generated clock frequency. The internally generated EPPI clock frequency is:

$f_{PCLK} = f_{SCLK0}/(EPPI\_CLKDIV + 1)$

where:

f_{PCLK} – frequency of internally generated EPPI clock

f_{SCLK} – frequency of SCLK0

EPPI_CLKDIV – Clock division value programmed in the EPPI_CLKDIV register.

The ***Relationship Between CLKDIV and the Ratio of SCLK0 to EPPI Clock*** table gives a few examples.

Table 32-8: Relationship Between CLKDIV and the Ratio of SCLK0 to EPPI Clock

CLKDIV15–0	EPPI/SCLK0 Clock Ratio
0x0002	1:3
0x0003	1:4
0x0004	1:5
0x0005	1:6
...	...

EPPI Operating Modes

The EPPI supports various receive and transmit modes of operation which include the detection and generation of preamble data. Specifically, the EPPI supports data formats described in the specifications ITU656, SMPTE 274M and SMPTE 296M. In addition to these modes, the EPPI also supports general purpose receive and transmit using up to three frame syncs (FS).

The control register (`EPPI_CTL`) includes most of the bits used for configuring operating modes. The "Register Descriptions" section of this chapter provides complete descriptions of these bits.

ITU-R 656 Modes

The EPPI supports three input modes and one output mode for ITU-R 656 framed data. This section describes these modes.

ITU-R 656 Background

In ITU-R 656 mode, the horizontal (H), vertical (V), and field (F) signals are sent as an embedded part of the video data stream. The signals are sent in a series of bytes that form a control word. ITU-R 656 was formerly known as CCIR-656.

The letter H is used to distinguish between the *start of active video* (SAV) and *end of active video* (EAV) signals. These signals indicate the beginning and end of active video data in each line. The SAV occurs on a 1-to-0 transition of H, and EAV occurs on a 0-to-1 transition of H. The space between EAV and SAV is filled with horizontal blanking data. Therefore, H = 1 during the horizontal blanking portion of the data stream, and H = 0 during the active video portion of the data stream.

The letter V is used to denote the vertical blanking portion of the data stream. A transition in V can occur only in the EAV sequence. When V = 1, the data stream contains vertical blanking data, and when V = 0, the data stream contains active video data.

The letter F is used to distinguish Field 1 from Field 2. Interlaced video has two fields in a frame of data. It requires each field to be handled uniquely, and alternate rows of each field combined to create the actual video image.

For interlaced video, F = 0 represents Field 1 (*Odd Field*) and F = 1 represents Field 2 (*Even Field*). Progressive video makes no distinction between Field 1 and Field 2, and F is always 0 for progressive video. Interlaced video requires each field to be handled uniquely because alternate rows of each field combine to create the actual video image.

An entire field of video is comprised of active video plus horizontal blanking (the space between an EAV and SAV code) and vertical blanking (the space where V = 1). A field of video commences on a transition of the F bit.

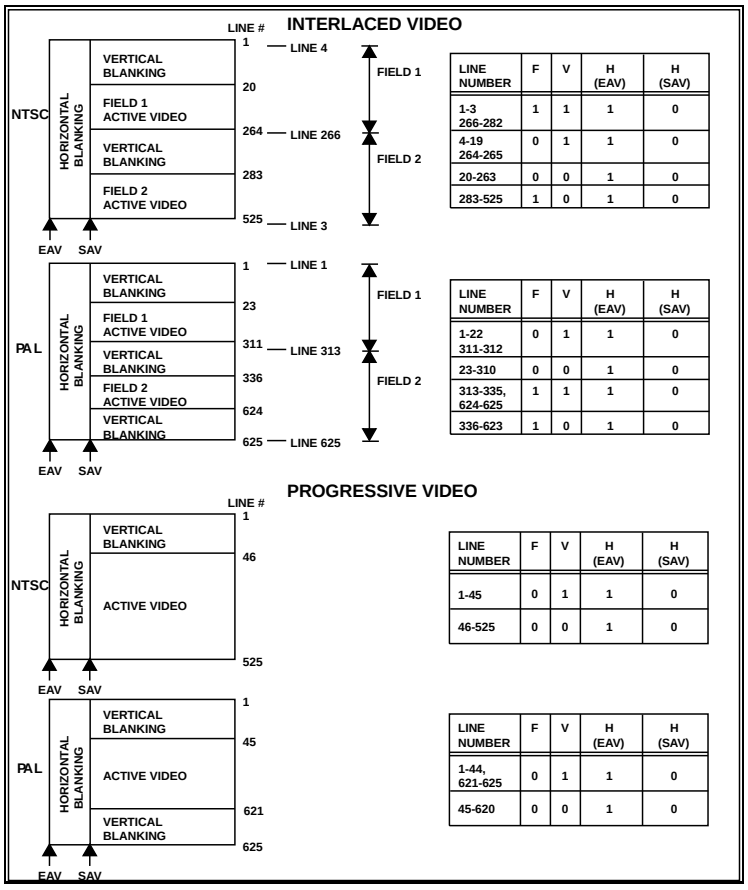

Figure 32-8: Typical Video Frame Partitioning for NTSC/PAL Systems in Interlaced and Progressive ITU-R BT.656 Systems

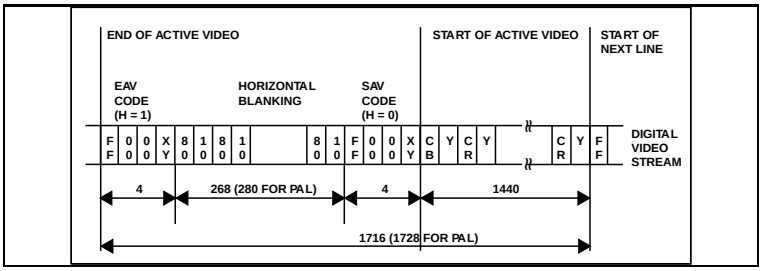

Figure 32-9: ITU-R 656 8-Bit Parallel Data Stream from NTSC (PAL) Systems

NOTE: Refer to the ***Control Sequences for 8-Bit and 10-Bit ITU-R 656 Video*** table. There is a defined preamble of three data elements (for example, in the case of 8-bit video: 0xFF, 0x00, 0x00), followed by the XY status word. The status word contains four protection bits for error detection and correction excluding the F (field), V (vertical blanking), and H (horizontal blanking) bits. F and V are only allowed to change as part of EAV sequences (that is, transition from H = 0 to H = 1).

The bit definitions are as follows:

- F = 0 for field 1

- F = 1 for field 2
- V = 1 during vertical blanking
- V = 0 when not in vertical blanking
- H = 0 at SAV
- H = 1 at EAV
- P3 = V XOR H
- P2 = F XOR H
- P1 = F XOR V
- P0 = F XOR V XOR H

P3–P0 are protection bits that enable 1-bit and 2-bit error detection, and 1-bit error correction at the receiver. The EPPI corrects the error if it detects 1-bit errors in F, V, or H. Errors in the protection bits themselves are detected but not corrected.

Table 32-9: Control Sequences for 8-Bit and 10-Bit ITU-R 656 Video

	8-Bit Data								**10-Bit Data**	
	D9 (MSB	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Preamble	1	1	1	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0
Control Byte	1	F	V	H	P3	P2	P1	P0	0	0

The `EPPI_STAT` register contains 2 bits, `EPPI_STAT.ERRDET` and `EPPI_STAT.ERRNCOR`, that are used to report the status of error detected and error not corrected, respectively.

The `EPPI_STAT.ERRDET` bit is set whenever an error is detected in the status word. However, this bit does not generate an interrupt. The `EPPI_STAT.ERRNCOR` bit is set when more than a 1-bit error is detected in the status word. An interrupt is generated when the `EPPI_STAT.ERRNCOR` bit is set. It can be cleared by clearing the `EPPI_STAT.ERRNCOR` and `EPPI_STAT.ERRDET` bits. Both bits are sticky and W1C.

In many applications, video streams other than the standard NTSC/PAL formats (for example, CIF, QCIF) can be employed. The processor interface is flexible enough to accommodate different row and field lengths. In general, as long as the incoming video has the proper EAV/SAV codes, the EPPI can read it in. A CIF image could be formatted to be 656-compliant, where EAV and SAV values define the range of the image for each line. The V and F codes are used to delimit fields and frames.

The following sections provide descriptions of EPPI operations.

Table 32-10: Operating Modes and Generic EPPI Operation

		How to configure	Useful for	How to configure in ITU R 656 Tx Mode
ITU-R BT.656 Rx	Entire field	DIR= 0 XFRTYPE = b#01		
	Active video	DIR = 0 XFRTYPE = b#00		
	Blanking only	DIR = 0 XFRTYPE = b#10		
GP 0 FS	Tx	DIR = 1 XFRTYPE = b#11 FSCFG = b#00	Applications where periodic frame syncs are not used to frame the data	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	Rx	DIR = 0 XFRTYPE = b#11 FSCFG = b#00		
GP 1 FS	Tx	DIR = 1 XFRTYPE = b#11 FSCFG =b#01	Interfacing with ADCs, DACs, and other general-purpose devices	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	Rx	DIR = 0 XFRTYPE = b#11 FSCFG = b#01		
GP 2 FS	Tx	DIR = 1 XFRTYPE = b#11 FSCFG = b#10	Video applications that use two hardware synchronization signals, HSYNC and VSYNC	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	Rx	DIR = 0 XFRTYPE = b#11 FS_CFG = b#10		
GP 3 FS	Tx	DIR = 1 XFRTYPE = b#11 FSCFG = b#11	Video applications that use three hardware sync signals, HSYNC, VSYNC, and FIELD	BLANKGEN = 1 DLEN = (b#000, b#001 or b#100)
	Rx	DIR = 0 XFRTYPE = b#11 FSCFG = b#11		

ITU-R 656 Input Modes

In the ITU-R 656 input modes, the video source provides the clock or the system supplies it externally.

As shown in the ***ITU-R 656 Input Submodes*** figure and described in the following sections, there are three submodes supported for ITU-R 656 inputs: entire field, active video only, and vertical blanking interval only.

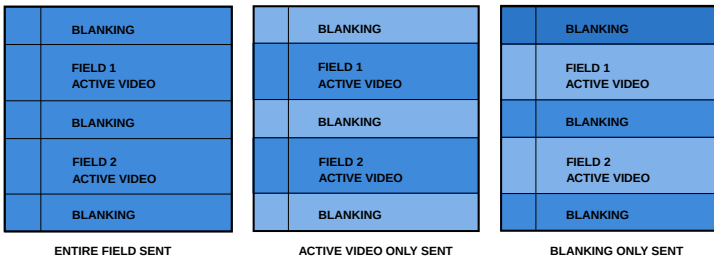

Figure 32-10: ITU-R 656 Input Submodes

Entire Field

In this mode, the EPPI reads the entire incoming bit stream. This stream includes active video as well as control byte sequences and ancillary data that can be embedded in horizontal and vertical blanking intervals.

Data transfer starts immediately after Field 1 synchronization occurs. The transfer does not include the first EAV code that contains the F = 0 assignment for interlaced video or the V = 0 assignment for progressive video.

Active Video

The EPPI uses this mode when only the active video portion of a field is of interest. The EPPI ignores (does not read in) all data between EAV and SAV, as well as all data present when V = 1. Furthermore, the control byte sequences are not stored to memory. The EPPI filters the sequences. After the start of Field 1 synchronizes, the EPPI ignores incoming samples until it sees an SAV.

In active video mode, programs must specify the number of total (active plus vertical blanking) lines per frame in the `EPPI_FRAME` register. Programs must specify the number of total (active plus horizontal blanking plus 8) samples per line in the `EPPI_LINE` register.

In this mode, any input data sequence that is considered part of the preamble is not sent to memory such as in 8-bit ITU mode. If 0xFF or 0x00 appear in the input data stream, these values are considered part of the preamble. The part of the preamble can appear individually and not be tagged along with the preamble sequence FF, 00, 00. This functionality also applies to vertical blanking interval mode.

Vertical Blanking Interval (VBI)

In this mode, data transfer is only active while V = 1 is in the control byte sequence. This functionality indicates that the video source is in the midst of the vertical blanking interval (VBI), which is sometimes used for ancillary data transmission. The ITU-R 656 recommendation specifies the format for these ancillary data packets, but the EPPI is not equipped to decode the packets themselves. Software must handle this task. Horizontal blanking data is logged where it coincides with the rows of the VBI.

The VBI is split into two regions within each field. The EPPI considers these two separate regions as one contiguous space. However, frame synchronization begins at the start of Field 1, which does not necessarily correspond to the start of vertical blanking. For instance, in 525/60 systems, the start of Field 1 (F = 0) corresponds to line 4 of the VBI.

In VBI mode, the program must specify the number of total (active plus vertical blanking) lines per frame in the EPPI_FRAME register. The program must specify the number of total (active plus horizontal blanking plus 8) samples per line in the EPPI_LINE register.

In this mode, any input data sequence that is considered as part of the preamble is not sent to memory such as in 8-bit ITU mode. If 0xFF or 0x00 appears in the input data stream, these values are considered part of the preamble. The part of the preamble can appear individually, and not be tagged along with the preamble sequence FF, 00, 00. This functionality applies to active video mode too.

ITU-R 656 Output in General-Purpose Transmit Modes

In GP transmit mode, the EPPI frames an ITU-R 656 output stream with the proper preambles and blanking intervals by setting the EPPI_CTL.BLANKGEN bit. The EPPI fetches active data from memory through the DMA channel, saving DMA bandwidth. The EPPI generates and embeds the proper preamble, status word (EAV and SAV sequences), and blanking data along with the active video from memory. Program the EPPI_FS1_PASPL, EPPI_FS2_WLVB, EPPI_FS2_PALPF, and EPPI_FS1_WLHB registers to perform the desired functions. The EPPI can also drive out the frame syncs using the EPPI_CTL.FSCFG bit setting.

The ***16-Bit Transmit with Internal Blanking Generation*** figure shows the bit stream format in 16-bit transmit modes with blanking generation (EPPI_CTL.BLANKGEN enabled). Each 16-bit data sample consists of 8-bit luma (Y) and 8-bit chroma (Cr or Cb) components. During transmission, the chroma data and blanking bytes of value 0x80 are placed on the upper half (MSBs) of the data lines. The luma data and blacking bytes of value 0x10 are placed on the lower half (LSBs) of the data lines.

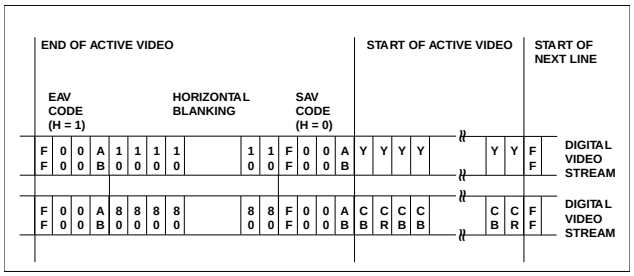

Figure 32-11: 16-Bit Transmit with Internal Blanking Generation

The ***Generated Blanking Preamble Sequence*** figure shows the data transmitted by the EPPI in this mode. After the EPPI is enabled, and if the EPPI FIFO is not empty, the transmission starts by sending out an EAV sequence for a vertical blanking line. For interlaced video, F starts at 1. For progressive video, F is always 0.

NOTE: Internal blanking generation functionality is valid only when the data length is 8, 10, or 16 bits and when the EPPI is in GP transmit modes. The EPPI_CTL.BLANKGEN bit generates preambles even in GP 2FS mode.

The internal blanking generation functionality of the ITU-R 656 output mode can also be bypassed by clearing the EPPI_CTL.BLANKGEN bit. (For example, if sending ancillary data in the blanking interval). The EPPI_CTL.BLANKGEN bit generates preambles even in GP 2FS mode.

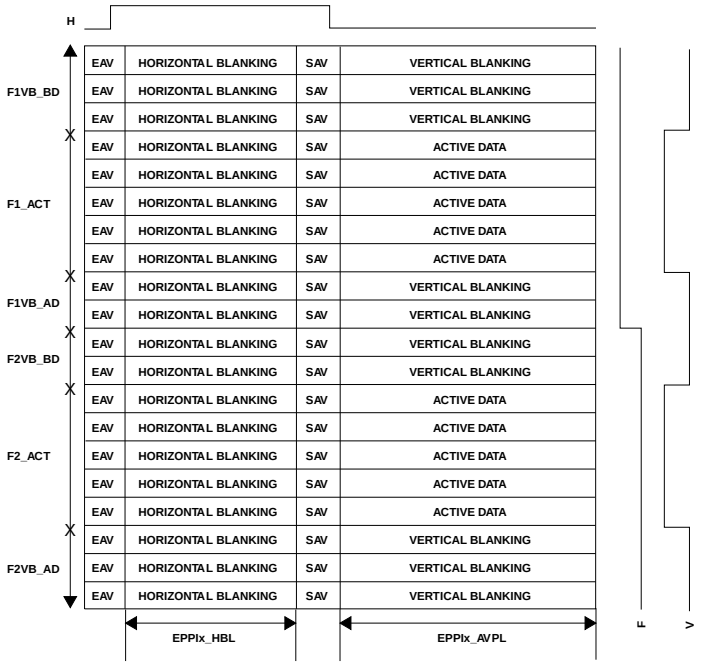

Figure 32-12: Generated Blanking Preamble Sequence

Frame Synchronization in ITU-R 656 Modes

For interlaced video, the start of frame synchronization occurs when a high-to-low transition is detected in F, the field indicator. For progressive video, the start of frame synchronization occurs when a high-to-low transition is detected in V, the vertical blanking indicator. These transitions in F and V can occur only in the EAV sequence. A start of line is detected on a low-to-high transition in H, the horizontal blanking indicator, which occurs in the EAV sequence as well.

For interlaced video, the start of frame corresponds to the start of field 1. Therefore, up to two fields can be ignored before the EPPI receives data. (For example, if field 1 started before the EPPI-to-camera channel was established). For progressive video, the start of frame corresponds to the start of active video.

Because all H and V signaling is embedded in the data stream in ITU-R 656 modes, the EPPI ignores the count registers (`EPPI_HCNT`, `EPPI_VCNT`). However, the EPPI still uses the `EPPI_FRAME` register to check for synchronization errors. Therefore, program this MMR with the number of lines expected in each frame of video.

The EPPI monitors the number of EAV-to-SAV transitions that occur from the start of a frame until it decodes the end of frame condition. (For example, a transition from F = 1 to F = 0 for interlaced video and a transition from V = 1 to V = 0 for progressive video).

At the end of frame condition, the actual number of lines processed is compared against the value in `EPPI_FRAME`. If there is a mismatch, a frame track error is asserted in the `EPPI_STAT` register. For example, if an SAV transition was missed, the current field only has NUM_ROWS – 1 rows. But, re-synchronization occurs at the start of the next frame. When the EPPI receives the entire field, the field status bit is toggled in the `EPPI_STAT` register. This way, an interrupt service routine (ISR) can discern which field was previously read in.

General-Purpose EPPI Modes

The general-purpose (GP) EPPI modes accommodate a wide variety of data capture and transmission applications.

Each EPPI has three bidirectional frame sync pins. The EPPI internally generates frame syncs, or an external device communicating with the EPPI generates them.

GP modes differ based on the number of frame syncs used and the EPPI supports GP 0 FS—GP 3 FS modes.

All the GP modes, except 0 FS mode, support horizontal windowing. GP modes with 2 and 3 frame syncs also support vertical windowing.

For GP transmit modes with internal clock or frame syncs, the EPPI starts generating the clock or frame syncs only when the EPPI FIFO is full for the first time. For GP 0 FS transmit mode, the EPPI only starts transmitting when the EPPI FIFO is full for the first time.

General-Purpose 0 Frame Sync Mode

This mode is useful for applications where periodic frame syncs are not used to frame the data.

After the initial trigger, the EPPI receives or transmits data samples on every clock cycle. However, if the `EPPI_CTL.SKIPEN` bit is set for receive mode, the EPPI receives only alternate data samples.

The `EPPI_LINE`, `EPPI_FRAME`, `EPPI_HCNT`, `EPPI_HDLY`, `EPPI_VCNT`, and `EPPI_VDLY` registers are not valid for GP 0 FS mode. Therefore, windowing is not possible in this mode. Also, line and frame track errors are not applicable in this mode.

GP 0 FS receive mode is further divided into two submodes; internal trigger (`EPPI_CTL.FLDSEL` bit =0) and external trigger (`EPPI_CTL.FLDSEL` bit =1). The submodes are based on how the processor initiates data transmission or reception. GP 0 FS transmit mode is always internally triggered. DMA handles all subsequent data manipulation.

- ***Frame synchronization in GP 0 FS external trigger mode.*** When the EPPI is programmed in external trigger mode, it does not generate the `EPPI_FS1` signal and the external device must provide a trigger. The EPPI starts receiving the data as soon as an `EPPI_FS1` signal assertion is detected. After that, the DMA handles all subsequent data manipulation and any activity on `EPPI_FS1` is ignored.
- ***Frame synchronization in GP 0 FS internal trigger mode.*** When the EPPI is programmed in internal trigger mode, it starts receiving or transmitting data as soon as the EPPI clock is enabled and synchronized. There can be up to four PPI clock cycles of latency before valid data is received or transmitted.

General-Purpose 1 Frame Sync Mode

This mode is useful for interfacing the EPPI with analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and other general-purpose devices. This mode works for both transmit and receive.

The `EPPI_FRAME`, `EPPI_VDLY`, and `EPPI_VCNT` registers have no effect in GP 1 FS mode. As a result, frame track errors and vertical windowing are not available.

General-Purpose 2 Frame Sync Mode

This mode is useful for video applications that use two hardware synchronization signals, HSYNC and VSYNC. The HSYNC signal can be connected to EPPI_FS1 and the VSYNC signal can be connected to EPPI_FS2.

Data Enable in General-Purpose 2 Frame Sync Transmit Mode

The EPPI_FS3 pin functions as a data enable (DEN) pin, when EPPI is configured in GP 2 FS transmit mode and generating the frame sync internally. The bits EPPI_CTL.MUXSEL and EPPI_CTL.CLKGATEN are not enabled. The functionality of the DEN pin is described in the following two cases.

Case 1

Blanking generation is configured using the EPPI_CTL.BLANKGEN bit. EPPI data length (EPPI_CTL.DLEN bit) is configured for 8, 10, or 16-bit transfers. The EPPI_FS3 pin asserts during the *active data* regions, aligned with EPPI_CLK according to the clock polarity (EPPI_CTL.POLC bit) settings. For this mode, the pin EPPI_FS3 is driven based on the EPPI_CTL.POLC setting. (The pin EPPI_FS3 is driven out on the same EPPI clock edge that drives out data). The frame sync polarity (EPPI_CTL.POLS) setting does not apply here—EPPI_FS3 is always active high in this mode.

Case 2

Blanking generation (EPPI_CTL.BLANKGEN =0) is disabled. Or blanking generation is enabled, but the EPPI data length (EPPI_CTL.DLEN bit) is configured for a transfer size other than 8, 10, or 16 bits. The EPPI_FS3 pin asserts at the start of the active data region on each line, aligned with EPPI_CLK according to the EPPI_CTL.POLC bit settings. For this mode, the pin EPPI_FS3 is driven based on the EPPI_CTL.POLC setting. (The EPPI_FS3 signal is driven out on the same EPPI clock edge that drives out data).

The EPPI_CTL.POLS bit setting does not apply for case 2. The EPPI_FS3 signal is always active high in this mode. Once asserted, EPPI_FS3 stays asserted for the number of clock cycles per line configured in the EPPI_HCNT register, then it deasserts. This behavior on each line continues for the total number of lines programmed in the EPPI_VCNT register per frame. The behavior repeats at the start of subsequent video frames.

In case 2, if transmission of valid data is held off due to delays programmed in the EPPI_HDLY or EPPI_VDLY registers, the assertion of EPPI_FS3 is also held off. The delay is on a per-line or per-frame basis.

General-Purpose 3 Frame Sync Mode

This mode is useful for video applications that use three synchronization signals for hardware: HSYNC, VSYNC, and FIELD. The HSYNC connects to the EPPI_FS1 pin, VSYNC connects to the EPPI_FS2 pin, and FIELD connects to the EPPI_FS3 pin.

GP 3 FS mode is similar in operation to GP 2 FS mode. However, the start of frame synchronization in GP 3 FS also considers the state of the EPPI_FS3 pin. All the windowing register settings (EPPI_FRAME, EPPI_LINE, EPPI_HDLY, EPPI_HCNT, EPPI_VDLY, and EPPI_VCNT registers), as well as data reception or transmission and

error generation are the same as for GP 2 FS mode. In addition, for GP 3 FS mode with internal frame syncs, the EPPI_CTL.FLDSEL bit setting specifies the condition under which the transfer begins.

The EPPI generates the EPPI_FS3 signal and toggles during every assertion of EPPI_FS2 or a combination of EPPI_FS2 and EPPI_FS1. The toggle depends on the EPPI_CTL.FLDSEL bit setting. The EPPI skips an EPPI_FS2 signal when the EPPI_FS3 value is high. Because of this condition, program the EPPI_FS2 period value to half of the total number of pixels in the frame as in GP 3 FS mode. When in GP 2 FS mode, program the EPPI_FS2 period with the value equal to the number of pixels per frame.

Supported Data Formats

The following sections describe EPPI receive and transmit data formats.

Receive Data Formats

The ***EPPI Receive Data Formats*** table provides information about EPPI configuration for specific use models for receive data.

Table 32-11: EPPI Receive Data Formats

Input Data Width	Use Model	Splitting/Packing Options
8	NTSC/PAL data	EPPI_CTL.SPLTEO =1 EPPI_CTL.SUBSPLTODD =1 if necessary to separate chroma components
	RGB sensor	No splitting possible. EPPI_CTL.PACKEN =1 – Four EPPI words are packed to 32-bit DMA data. EPPI_CTL.PACKEN =0 – Each EPPI word is sent as 8-bit data on the 32-bit DMA bus. This transfer consumes 4 times the DMA bandwidth of the 8-bit case with EPPI_CTL.PACKEN =1;
	ADCs	Gives I (in phase) and Q (quadrature) components. EPPI_CTL.SPLTEO =1 EPPI_CTL.SUBSPLTODD =0 since there are only two components.

Table 32-11: EPPI Receive Data Formats (Continued)

Input Data Width	Use Model	Splitting/Packing Options
10	NTSC/PAL data	Each EPPI word is zero filled or sign extended to 16 bits. EPPI_CTL.SPLTEO =1. EPPI_CTL.SUBSPLTODD =1 if necessary to separate chroma components.
	RGB sensor	No splitting possible. EPPI_CTL.PACKEN =1. Two EPPI words are zero filled or sign extended to 16 bits and packed to 32-bit DMA data. EPPI_CTL.PACKEN =0. Each EPPI word can be zero filled or sign extended to 16 bits and sent as a 16-bit data on the 32-bit DMA bus. This transfer consumes double the bandwidth of the 10-bit case with EPPI_CTL.PACKEN =1;
	ADCs	Each EPPI word is zero filled or sign extended to 16 bits. EPPI_CTL.SPLTEO =1 SEPPI_CTL.SUBSPLTODD = 0 since there are only two components.
12	RGB sensor	No splitting possible. EPPI_CTL.PACKEN =1. Two EPPI words are zero filled or sign extended to 16 bits and packed to 32-bit DMA data. EPPI_CTL.PACKEN =0. Each EPPI word can be zero filled or sign extended to 16 bits and sent as a 16-bit data on the 32-bit DMA bus. This transfer consumes double the bandwidth of the 12-bit case with EPPI_CTL.PACKEN =1;
	ADCs	Each EPPI word is zero filled or sign extended to 16 bits. EPPI_CTL.SPLTEO =1 EPPI_CTL.SUBSPLTODD =0 since there are only two components.
14	ADCs	Each EPPI word is zero filled or sign extended to 16 bits. EPPI_CTL.SPLTEO =1 EPPI_CTL.SUBSPLTODD =0 since there are only two components.

Table 32-11: EPPI Receive Data Formats (Continued)

Input Data Width	Use Model	Splitting/Packing Options
16	8-bit luma/chroma pair for NTSC or HD	EPPI_CTL.SPLTEO =1, EPPI_CTL.SPLTWRD =1, EPPI_CTL.SUBSPLTODD =1 if necessary to separate chroma components.
	16-bit luma/chroma pair for NTSC or HD	EPPI_CTL.SPLTEO =1, EPPI_CTL.SPLTWRD =0, EPPI_CTL.SUBSPLTODD =1 if necessary to separate chroma components.
	RGB565 sensor	No splitting possible. EPPI_CTL.PACKEN =1. Two EPPI words are packed to a 32-bit DMA data. EPPI_CTL.RGBFMTEN is valid only in transmit modes. So, RGB565 cannot be byte aligned in memory. EPPI_CTL.PACKEN =0. Each EPPI word is sent as a 16-bit data on the 32-bit DMA bus. This transfer consumes double the bandwidth of the 16-bit case with EPPI_CTL.PACKEN =1
	8-bit ADCs I/Q pair	EPPI_CTL.SPLTEO =1, EPPI_CTL.SPLTWRD =1, EPPI_CTL.SUBSPLTODD =0.
	16-bit ADCs I/Q pair	EPPI_CTL.SPLTEO =1, EPPI_CTL.SPLTWRD =0, EPPI_CTL.SUBSPLTODD =0.

Transmit Data Formats

The ***EPPI Transmit Data Formats*** table provides information about EPPI configuration for specific use models for transmit data.

Table 32-12: EPPI Transmit Data Formats

Output Data Width	Use Model	Splitting/Packing Options
8	NTSC/PAL data	EPPI_CTL.SPLTEO =1 EPPI_CTL.SUBSPLTODD =1 if the chroma components (U and V) come in separate DMA words.
	Serial RGB for lower-resolution LCDs	No splitting possible. EPPI_CTL.PACKEN =1. The 32-bit DMA data is unpacked to drive four EPPI words. EPPI_CTL.PACKEN =0. The lowest 8 bits of the DMA data is driven on the EPPI data and the rest of the 24 bits are discarded. This transfer consumes 4 times the DMA bandwidth of the 8-bit case with EPPI_CTL.PACKEN =1.
10	NTSC/PAL data	EPPI_CTL.SPLTEO =1 EPPI_CTL.SUBSPLTODD =1 if the chroma components (U and V) come in separate DMA words.
	DACs	EPPI_CTL.SPLTEO =1, EPPI_CTL.SUBSPLTODD =0.

Table 32-12: EPPI Transmit Data Formats (Continued)

Output Data Width	Use Model	Splitting/Packing Options
12	DACs	EPPI_CTL.SPLTEO =1, EPPI_CTL.SUBSPLTODD =0.
14	DACs	EPPI_CTL.SPLTEO =1, EPPI_CTL.SUBSPLTODD =0.
16	8-bit luma/chroma pair for NTSC or HD	EPPI_CTL.SPLTEO =1, EPPI_CTL.SPLTWRD =1, EPPI_CTL.SUBSPLTODD =1 if the chroma components (U and V) come in separate DMA words.
	16-bit luma/chroma pair for NTSC or HD	EPPI_CTL.SPLTEO =1, EPPI_CTL.SPLTWRD =0, EPPI_CTL.SUBSPLTODD =1 if the chroma components (U and V) come in separate DMA words.
	RGB565 LCD	No splitting possible. EPPI_CTL.RGBFMTEN =1. Takes RGB888 data from the memory and drops the LSBs from each component to drive out RGB565 data.
	8-bit ADCs I/Q pair	EPPI_CTL.SPLTEO =1, EPPI_CTL.SPLTWRD =1, EPPI_CTL.SUBSPLTODD = 0
	16-bit ADCs I/Q pair	EPPI_CTL.SPLTEO =1, EPPI_CTL.SPLTWRD =0, EPPI_CTL.SUBSPLTODD =1
18	RGB666 LCD	No splitting possible. EPPI_CTL.RGBFMTEN =1. Takes RGB888 data from the memory and drops the 2 LSBs from each component to drive out RGB666 data.

Data Transfer Modes

The following sections describe EPPI data transfer modes, including receive or transmit data packing, sign extension, zero fill, receive or transmit split modes, clock gating, delayed start, and data consistency management.

Data Packing for Receive Modes

For receive modes, if the EPPI_CTL.PACKEN bit =1 and the DMA is 32 bits, the EPPI packs the incoming data into 32-bit words based on the EPPI_CTL.DLEN and EPPI_CTL.SWAPEN bit settings. When EPPI_CTL.SWAPEN =0, the EPPI puts the first data in the least significant bits and when EPPI_CTL.SWAPEN =1, the EPPI puts the first data in the most significant bits. The packing options for the EPPI_CTL.DLEN bits are as follows.

- When EPPI_CTL.DLEN =8, four 8-bit words can be packed into one 32-bit word.
- When EPPI_CTL.DLEN =16, two 16-bit words can be packed into one 32-bit word.
- For EPPI_CTL.DLEN values that are more than 8 bits but less than 16 bits, two such words are either sign-extended or zero-filled to 16 bits, and packed into one 32-bit word.
- When EPPI_CTL.DLEN =18, the EPPI sign-extends or zero-fills the 18-bit data to 24 bits and packs four 24-bit words into three 32-bit words.

When EPPI_CTL.PACKEN =0, the EPPI receives the incoming data and sends it on the bus as-is. If EPPI_CTL.DLEN is less than or equal to 16 bits, the DMA is a 16-bit DMA; otherwise it is a 32-bit DMA.

Data Packing for Transmit Modes

For transmit modes, if the EPPI_CTL.DLEN bit =1 and the DMA is a 32-bit DMA, the EPPI unpacks the 32-bit word according to the EPPI_CTL.DLEN and EPPI_CTL.SWAPEN bit settings.

If EPPI_CTL.SWAPEN =1, the EPPI transmits the most significant bits as the first data, and if EPPI_CTL.SWAPEN =0, the EPPI transmits the least significant bits as the first data. The unpacking options for the EPPI_CTL.DLEN bits are as follows.

- When EPPI_CTL.DLEN =8, the EPPI transmits one 32-bit word from memory as four 8-bit data words.
- For EPPI_CTL.DLEN values greater than 8 bits but less than or equal to 16 bits, the EPPI transmits one 32-bit word from memory as two 16-bit data words.

Sign-Extended and Zero-Filled Data

The following list describes the bit settings and functionality for sign-extending and zero-filling data.

- For EPPI_CTL.DLEN equal to 10, 12 or 14, data is zero-filled or sign-extended to 16 bits.
- For EPPI_CTL.DLEN equal to 18 bits, data is zero-filled or sign-extended to 24 bits if packing is enabled, and zero-filled or sign-extended to 32 bits if packing is disabled.
- For EPPI_CTL.DLEN equal to 8 bits, data is zero-filled or sign-extended to 16 bits if packing is disabled.
- If EPPI_CTL.SIGNEXT =1, then the data is sign-extended, otherwise it is zero-filled.

Split Receive Modes

The control register has three control bits for split receive modes: EPPI_CTL.SPLTEO, EPPI_CTL.SUBSPLTODD, and EPPI_CTL.DMACFG. Packing is not valid in split modes.

- If EPPI_CTL.SPLTEO =1, the EPPI splits the incoming data stream into two substreams, an even stream, and an odd stream, and packs them separately.
- The EPPI_CTL.SUBSPLTODD bit is available only when EPPI_CTL.SPLTEO =1. When EPPI_CTL.SUBSPLTODD =1, the EPPI subsplits the odd substream, and packs the streams separately.
- The EPPI_CTL.DMACFG bit is also available only if EPPI_CTL.SPLTEO =1. If EPPI_CTL.DMACFG =1, the EPPI uses two DMA channels and if EPPI_CTL.DMACFG =0, the EPPI uses only one DMA channel.

Split Transmit Modes

The EPPI_CTL register has three control bits for split transmit modes: EPPI_CTL.SPLTEO, EPPI_CTL.SUBSPLTODD, and EPPI_CTL.DMACFG. The DMA is always a 32-bit DMA. Packing is not valid in split modes.

- If EPPI_CTL.SPLTEO =1, the EPPI receives the Luma (Y3Y2Y1Y0) and interleaved Chroma (Cr1Cb1Cr0Cb0) data as 32 bits from the DMA channel. The EPPI interleaves the data to form a 4:2:2 YCrCb data stream to transmit.
- The EPPI_CTL.SUBSPLTODD bit is available only when EPPI_CTL.SPLTEO =1. In this case, if EPPI_CTL.SUBSPLTODD =1, the EPPI receives the Luma (Y3Y2Y1Y0) and de-interleaved Chroma (Cb3Cb2Cb1Cb0 and Cr3Cr2Cr1Cr0). The EPPI interleaves the data to form a 4:2:2 YCrCb data stream to transmit. (The EPPI does not decimate the chroma data when formatting it into 4:2:2).
- The EPPI_CTL.DMACFG bit is also valid only if EPPI_CTL.SPLTEO =1. If EPPI_CTL.DMACFG =1, the EPPI uses two DMA channels and if EPPI_CTL.DMACFG =0, the EPPI uses only one DMA channel.

Clock Gating

In ITU-R BT.656 and GP 0/1/2 FS modes, EPPI_FS3 becomes a clock-gating input. This functionality is valid for both internally and externally sourced EPPI_CLK, in both receive and transmit modes. This clock gating signal must be synchronous with EPPI_CLK. The external device on the rising edge of EPPI_CLK must drive the clock gating signal. Its function is to hold the sync and data lines in their current state until EPPI_FS3 is driven low. There are no additional latency cycles upon coming out of clock gating mode.

If clock gating is not required, the EPPI_FS3 pin must either be tied to ground, or configured to operate as another of its multiplexed functions.

In GP 2 FS transmit mode with internally generated frame syncs, the EPPI_FS3 pin functions as a data enable signal.

Support for Delayed Start of EPPI Frame Syncs

The EPPI supports a delayed start of the EPPI_FS1 and EPPI_FS2 frame syncs. The EPPI_FS1_DLY and EPPI_FS2_DLY registers are programmable registers corresponding to EPPI_FS1 (HSYNC) and EPPI_FS2 (VSYNC).

The delay programmed in these registers applies to the first active edge of the internally generated frame sync. The delay starts from the first EPPI_CLK edge. The delay counter runs only for the first time and then shuts off until the EPPI is re-enabled. (The delay counter is the period counter itself, since they do not run together). Program the delay registers prior to the first EPPI_CLK edge (similar to the width and period registers). The ***EPPI Delayed Frame Sync Generation*** figure shows the functioning of EPPI_FS1 and EPPI_FS2.

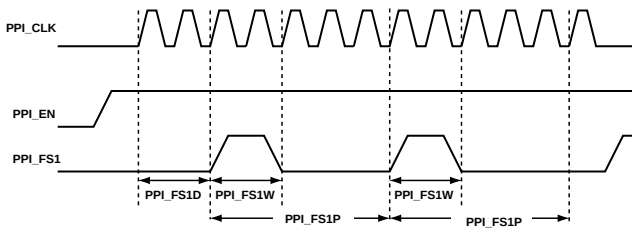

Figure 32-13: EPPI Delayed Frame Sync Generation

Ignoring Premature External Frame Syncs for Data Consistency

Once a frame has started with a VSYNC followed by an HSYNC (or both coming together), a line is tracked. When the count expires, the state machine waits at the end of line for an HSYNC to come. With the arrival of the HSYNC, the state machine starts tracking the next line, and so on.

The number of lines tracked is counted separately. Once the end of a frame is reached, the state machine waits there for the next VSYNC/HSYNC combination. The next frame starts once they are sampled. Unfortunately, every incoming FS (VSYNC or HSYNC) resets the respective counters and the tracking starts all over (even if the FS signals are premature). The result is incomplete data (or frames) to enter into memory through the PxP interface.

To correct this problem, the EPPI waits for a frame or line completion before considering any incoming FS as valid.

- Single FS mode and line tracking in dual FS mode – When a line is in progress, when `HSYNC` is detected prematurely, it is ignored. A line track underflow event is generated.
- Dual FS mode – If a `VSYNC` is received when a frame is in progress, it is ignored. A frame track underflow error (`EPPI_STAT.FTERRUNDR`) is generated.

Ignoring the FS ensures that once a frame starts, the amount of data that goes into the memory/PxP interface corresponds exactly to the programmed data size in a frame.

NOTE: Even if the premature FS is a valid FS, the state machine loses at most one frame and it recovers in the subsequent FS. The FS to number of data going into the memory relationship is always maintained as programmed.

When data underflow errors occur at the DMA interface, the EPPI does the following.

- If a premature line sync is detected, an LT underflow error is generated (`EPPI_STAT.LTERRUNDR` =1). All further line track errors are ignored until the EPPI detects the next valid line sync.
- If a premature frame sync is detected, an FT underflow error is generated (`EPPI_STAT.FTERRUNDR` =1). All further frame track and line track errors are ignored until the EPPI detects the next valid frame sync.

EPPI Event Control

The following sections describe how EPPI manages events.

EPPI Status, Error, and Interrupt Signals

The EPPI generates error interrupts (flagged in the `EPPI_STAT` register) when any one of the following error conditions occur.

- `EPPI_STAT.YFIFOERR` (YFIFO underflow or overflow)
- `EPPI_STAT.CFIFOERR` (CFIFO underflow or overflow)
- `EPPI_STAT.LTERROVR` (line track overflow error)
- `EPPI_STAT.LTERRUNDR` (line track underflow error)

- EPPI_STAT.FTERROVR (frame track overflow error)
- EPPI_STAT.FTERRUNDR (frame track underflow error)
- EPPI_STAT.ERRNCOR (ITU preamble error not corrected)

A W1C (write-1-to-clear) operation clears the error conditions. Each of the individual conditions which cause an EPPI error interrupt can be masked. The interrupt mask register (EPPI_IMSK) allows the masking of individual conditions which cause error interrupts.

There is only one interrupt line from each EPPI so all interrupts are internally OR'ed and sent as a single interrupt to the core. The EPPI_STATregister must then be read to discover specific errors. The following sections describe these errors in detail.

Frame and Line Track Errors

In external frame sync mode, the EPPI uses line track error (EPPI_STAT.LTERROVR and EPPI_STAT.LTERRUNDR) and frame track error (EPPI_STAT.FTERROVR and EPPI_STAT.FTERRUNDR) status bits to monitor the line and frame synchronization errors. The EPPI updates the bits when there is a mismatch detected in the HSYNC and VSYNC as compared to the programmed values in EPPI_LINE and EPPI_FRAME count registers.

Line Track Errors

The line track overflow (EPPI_STAT.LTERROVR) and underflow errors (EPPI_STAT.LTERRUNDR) generate a maskable interrupt as soon as the EPPI identifies them and not at the next frame sync.

- If the frame sync has not arrived when the EPPI_LINE counter expires, then the EPPI_STAT.LTERROVR error is generated.
- When the EPPI_LINE counter is running and a frame sync is detected, the EPPI_STAT.LTERRUNDR error is generated. A W1C operation clears boths interrupts.

Frame Track Errors

The frame track overflow (EPPI_STAT.FTERROVR) and underflow errors (EPPI_STAT.FTERRUNDR) generate a maskable interrupt as soon as the EPPI identifies them. When the EPPI_FRAME.VALUE counter expires, the EPPI_STAT.FTERROVR error is reported before the next frame sync arrives.

When the EPPI_FRAME counter is running and a frame sync is detected, then an EPPI_STAT.FTERRUNDR is reported.

Both errors generate an error interrupt. Perform a W1C operation to clear the interrupts at their respective locations in the status register.

A premature frame sync results in a frame track under run error. But, the error is logged (register bit set) only after the subsequent blanking period (if any) elapses.

Preamble Error Not Corrected Error

The EPPI supports data embedded frame syncs in ITU and SMPTE formats. In these formats, the module can receive an erroneous preamble which is not correctable. The EPPI_STAT.ERRNCOR error signals when this event occurs.

EPPI Programming Model

The following sections describe programming techniques, including receiving or transmitting ITU-R 656 frames; configuring transfers in GP0, GP1, GP2, and GP3 modes; and managing EPPI mode configurations.

Receiving ITU-R 656 Frames

The EPPI supports the reception of ITU-R 656 compliant frames.

1. Configure the EPPI to receive either full ITU-R 656 frame, active video, or blanking information by configuring the EPPI_CTL.XFRTYPE bits.
2. In both active video mode and in VBI (vertical blanking information) mode, specify the number of total (active plus vertical blanking) lines per frame in the EPPI_FRAME register. Specify the number of total (active plus horizontal blanking plus 8) samples per line in the EPPI_LINE register.
3. Configure DMA descriptors to move the data to memory.
4. Enable DMA.
5. Enable the EPPI.
6. To program the EPPI in internal clock mode, follow the procedure above with the EPPI_CTL.ICLKGEN bit =0. After enabling the EPPI, add a delay of 200 SCLK0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the EPPI_CTL.ICLKGEN bit =1.

Depending on the EPPI configuration, either the full ITU-R 656 frame is moved to memory or only the active video or only the blanking information.

Transmitting ITU-R 656 Frames in GP Transmit Modes

The EPPI can take active video from memory and generate the proper preambles and blanking information to produce valid ITU-R 656 video frames for transmission.

1. Provide active data frame in memory.
2. Set the EPPI_CTL.BLANKGEN bit so the EPPI generates blanking information.
3. Configure the EPPI_FS1_WLHB, EPPI_FS1_PASPL, EPPI_FS2_WLVB, EPPI_FS2_PALPF registers accordingly.
4. Configure the rest of the EPPI settings.

5. Configure DMA to fetch active frame data from memory buffers.
6. Enable DMA.
7. Enable the EPPI.
8. To program the EPPI in internal clock mode, follow the procedure above with the EPPI_CTL.ICLKGEN bit =0. After enabling the EPPI, add a delay of 200 SCLK0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the EPPI_CTL.ICLKGEN bit =1.

The EPPI takes the active data from memory, generates the blanking information, and transmits an ITU-R 656 frame

Configuring Transfers in GP 0 FS Mode

The EPPI can be configured to not use periodic frame syncs to frame the data.

1. Configure the EPPI to operate in GP 0 FS mode by setting EPPI_CTL.XFRTYPE = b#11 and EPPI_CTL.FSCFG = b#00.
2. When receiving, configure the EPPI to trigger on internally or externally by setting the EPPI_CTL.FLDSEL field appropriately. When transmitting, the EPPI always generates a trigger internally.
3. Configure DMA to move the data to or from memory.
4. Enable DMA.
5. Enable EPPI.
6. To program the EPPI in internal clock mode, follow the procedure above with the EPPI_CTL.ICLKGEN bit =0. After enabling the EPPI, add a delay of 200 SCLK0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the EPPI_CTL.ICLKGEN bit =1.

The DMA descriptions control the amount of data transferred. The frame syncs from the EPPI do not control the amount.

Configuring Transfers in GP 1 FS Mode

The GP 1 FS mode is useful for interfacing the EPPI with analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and other general-purpose devices. This mode works for both transmit and receive.

NOTE: The EPPI_FRAME, EPPI_VDLY, and EPPI_VCNT registers have no effect in GP 1 FS mode. As a result, frame track errors and vertical windowing are not possible in this mode.

1. Configure GP 1 FS mode by setting the EPPI_CTL.XFRTYPE bit =b#11 and the EPPI_CTL.FSCFG bit =b#01. An external device can provide the frame syncs or the EPPI can source the frame syncs.

2. Program the EPPI_LINE register to contain the number clock cycles expected between two assertions of the EPPI_FS1 signal to monitor the line track errors. Program the EPPI_LINE register before the EPPI_HCNT register.
3. Program the EPPI_HDLY register to contain the number of clock cycles to wait after the assertion of EPPI_FS1. For example, the start of frame.
4. Program the EPPI_HCNT register to contain the number of data samples to receive or transmit for each frame.
5. Configure DMA to move the data to or from memory.
6. Enable DMA.
7. Enable the EPPI.
8. To program the EPPI in internal clock mode, follow the procedure above with the EPPI_CTL.ICLKGEN bit =0. After enabling the EPPI, add a delay of 200 SCLK0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then, switch to internal clock mode by setting the EPPI_CTL.ICLKGEN bit =1.

Data moves in or out of memory. A frame sync frames the data for every line.

Configuring Transfers in GP 2 FS Mode

GP 2 FS mode is useful for video applications that use two hardware synchronization signals, HSYNC and VSYNC. The HSYNC connects to the EPPI_FS1 signal and VSYNC connects to the EPPI_FS2 signal.

1. Configure the EPPI to operate in GP 0 FS mode by setting the EPPI_CTL.XFRTYPE bit =b#11 and the EPPI_CTL.FSCFG bit =b#10. An external device can provide the frame syncs or the EPPI can source the frame syncs.
2. Program the EPPI_FRAME register to contain the number of expected lines per frame. The value can be equal to the number of EPPI_FS1 signal assertions expected between the start of each frame sync. The EPPI uses the value to monitor frame track errors. Program the EPPI_FRAME register before the EPPI_VCNT register.
3. Program the EPPI_LINE register to contain the number of clock cycles expected between two assertions of the EPPI_FS1 signal to monitor line track errors. Program the EPPI_LINE register before the EPPI_HCNT register.
4. Program the EPPI_HDLY register to configure the number of clock cycles to wait after the assertion of the EPPI_FS1 signal. (For example, the start of the line).
5. Program the EPPI_HCNT register to contain the number of data samples to receive or transmit for each line.
6. Program the EPPI_VDLY register to contain the number of lines to wait after the start of frame is detected.
7. Program the EPPI_VCNT register to contain the number of lines to receive or transmit.
8. If setting up the EPPI for transmit, the data enable (DEN) pin behaves according to the enabling of the blanking generation and the data length setting (DLEN). See Data Enable in General-Purpose 2 Frame Sync Transmit Mode for more details.

9. Enable DMA.
10. Enable the EPPI.
11. To program the EPPI in internal clock mode, follow the procedure above with the EPPI_CTL.ICLKGEN bit =0. After enabling the EPPI, add a delay of 200 SCLK0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the EPPI_CTL.ICLKGEN bit =1.

Data moves in or out of memory. A frame sync frames the data for every line and frame.

Configuring Transfers in GP 3 FS Mode

GP 3 FS mode is useful for video applications that use three synchronization signals for hardware: HSYNC, VSYNC, and FIELD. The HSYNC connects to EPPI_FS1, VSYNC connects to EPPI_FS2, and FIELD connects to EPPI_FS3.

1. Configure the EPPI to operate in GP 3 FS mode by setting the EPPI_CTL.XFRTYPE bit =b#11 and the EPPI_CTL.FSCFG bit =b#11. An external device can provide the frame syncs or the EPPI can source the frame syncs.
2. Configure the windowing registers according to steps in GP 2 FS mode.
3. Enable DMA.
4. Enable the EPPI.
5. To program the EPPI in internal clock mode, follow the procedure above with the EPPI_CTL.ICLKGEN bit =0. After enabling the EPPI, add a delay of 200 SCLK0 cycles (worst case) to ensure the EPPI FIFO becomes full. Then switch to internal clock mode by setting the EPPI_CTL.ICLKGEN bit =1.

Data moves in or out of memory. A frame sync frames the data for every line and frame. Operation and result are similar to operation in GP 2 FS mode but the EPPI also uses the EPPI_FS3 signal.

Configuring the EPPI to Use the Windowing Feature

Windowing is a useful feature for applications where the region of interest is smaller than the active video stream (for example, sensor calibration, auto-focusing, and others). It can result in significant DMA bandwidth reduction. The EPPI supports windowing for GP input modes.

1. Program the EPPI_FRAME register with the number of lines the frame contains.
2. Program the EPPI_LINE register with the number of samples per line in the frame.
3. Program the EPPI_VDLY register with the number of lines to wait after the start of a new frame before starting to read or transmit data.
4. Program the EPPI_VCNT register with the number of lines to read in or write out after EPPI_VDLY number of lines from the start of the frame.

5. Program the EPPI_HDLY register with the number of clock cycles to delay after the assertion of EPPI_FS1 is detected for the start of a new line.
6. Program the EPPI_HCNT register with the number of samples to read in or write out after EPPI_HDLY number of cycles have expired since the assertion of EPPI_FS1.

EPPI Mode Configuration

This section describes EPPI mode configurations, including support for all EPPI transmit and receive modes.

Configuring 8-Bit Receive Mode

For 8-bit non-split receive mode and if EPPI_CTL.PACKEN =1, the EPPI packs 4 bytes of incoming data into a 32-bit word. Alternate even or odd samples can be skipped based on the EPPI_CTL.SKIPEN and EPPI_CTL.SKIPEO bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the EPPI_CTL.SWAPEN bit setting.

Table 32-13: 8-Bit Receive Mode with Packing Enabled

Pin Data (8 bits)	DMA DATA SKIPEN=0 SKIPEO =X SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=X
0x11						
0x22						
0x33						
0x44	0x4433 2211	0x1122 3344				
0x55						
0x66						
0x77			0x7755 3311		0x1133 5577	
0x88	0x8877 6655	0x5566 7788		0x8866 4422		0x2244 6688
0x99						
0xAA						
0xBB						
0xCC	0xCCBB AA99	0x99AA BBCC				
0xDD						
0xEE						
0xFF			0xFFDD BB99		0x99BB DDFF	
0x00	0x00FF EEDD	0xDDE EFF00		0x00EE CCAA		0xAACC EE00

If EPPI_CTL.PACKEN=0, the DMA is a 16-bit DMA and the EPPI either sign-extends or zero-fills the bytes of incoming data into a 16-bit word. The EPPI_CTL.SWAPEN bit has no effect if EPPI_CTL.PACKEN=0.

Table 32-14: 8-Bit Receive Mode with Packing Disabled

Pin Data (8 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=1
0x44	0x0044	0x0044	0x0044	
0x55	0x0055	0x0055		0x0055
0x66	0x0066	0x0066	0x0066	
0x77	0x0077	0x0077		0x0077
0x88	0x0088	0xFF88	0x0088	
0x99	0x0099	0xFF99		0xFF99
0xAA	0x00AA	0xFFAA	0x00AA	
0xBB	0x00BB	0xFFBB		0xFFBB

Configuring 10/12/14-Bit Receive Modes

For 10, 12, or 14-bit non-split receive modes, the EPPI first either zero-fills or sign-extends the incoming data into a 16-bit word. The action depends on the setting of the EPPI_CTL.SIGNEXT bit. If EPPI_CTL.PACKEN =1, the EPPI then packs two of these words into one 32-bit word. Alternate even or odd samples can be skipped based on the EPPI_CTL.SKIPEN and EPPI_CTL.SKIPEO bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the EPPI_CTL.SWAPEN bit setting.

Table 32-15: 10-Bit Receive Mode with Sign Extension, with Packing Enabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=1	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=1
0x111	0			
0x222	1	0xFE22 0111	0x0111 FE22	
0x333	1			0xFF33 0111
0x044	0	0x0044 FF33	0xff33 0044	
0x155	0			
0x266	1	0xFE66 0155	0x0155 FE66	

Table 32-15: 10-Bit Receive Mode with Sign Extension, with Packing Enabled (Continued)

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=1	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=1
0x377	1			0xFF77 0155
0x088	0	0x0088 FF77	0xFF77 0088	

Table 32-16: 10-Bit Receive Mode with Sign Extension, with Packing Enabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=1
0x111	0			
0x222	1			
0x333	1		0x0011 FF33	
0x044	0	0x0044 FE22		0xFE22 0044
0x155	0			
0x266	1			
0x377	1		0x0155 FF77	
0x088	0	0x0088 FE66		0xFE66 0088

Table 32-17: 10-Bit Receive Mode, with Zero-Fill, with Packing Enabled

Pin Data (10 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=0	DMA DATA SKIP_EN=1 SKIP_EO=0 SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=0
0x111						
0x222	0x0222 0111	0x0111 0222				
0x333			0x0333 0111		0x0011 0333	
0x044	0x0044 0333	0x0333 0044		0x0044 0222		0x0222 0044
0x155						

Table 32-17: 10-Bit Receive Mode, with Zero-Fill, with Packing Enabled (Continued)

Pin Data (10 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=0	DMA DATA SKIP_EN=1 SKIP_EO=0 SWAPEN=0 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=0
0x266	0x0266 0155	0x0155 0266				
0x377			0x0377 0155		0x0155 0377	
0x088	0x0088 0377	0x0377 0088		0x0088 0266		0x0266 0088

The ***10-bit Receive Mode with Packing Disabled*** table shows a 10-bit receive mode example when `EPPI_CTL.PACKEN` =0:

Table 32-18: 10-bit Receive Mode with Packing Disabled

Pin Data (10 bits)	MSB	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=1	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=0
0x111	0	0x0111	0x0111	0x0111	
0x222	1	0xFE22	0x0222		0x0222
0x333	1	0xFF33	0x0333	0xFF33	
0x044	0	0x0044	0x0444		0x0444
0x155	0	0x0155	0x0155	0x0155	
0x266	1	0xFE66	0x0266		0x0266
0x377	1	0xFF77	0x0377	0xFF77	
0x088	0	0x0088	0x0088		0x088

Configuring 16-Bit Receive Mode

For 16-bit non-split receive mode, if `EPPI_CTL.PACKEN` =1, the EPPI packs two 16-bit incoming data into one 32-bit word. Alternate even or odd samples can be skipped based on the `EPPI_CTL.SKIPEN` and `EPPI_CTL.SKIPEO` bits. The first incoming data can be placed either in the least significant bit positions or in the most significant bit positions, based on the `EPPI_CTL.SWAPEN` bit setting.

Table 32-19: 16-Bit Receive Mode with Packing Enabled

Pin Data (16 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=0 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=1 SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=1 SIGNEXT=X
0x1111						
0x2222	0x2222 1111	0x1111 2222				
0x3333			0x3333 1111		0x1111 3333	
0x4444	0x4444 3333	0x3333 4444		0x4444 2222		0x2222 4444
0x5555						
0x6666	0x6666 5555	0x5555 6666				
0x7777			0x7777 5555		0x5555 7777	
0x8888	0x8888 7777	0x7777 8888		0x8888 6666		0x6666 8888

Table 32-20: 16-bit Receive Mode with Packing Disabled

Pin Data (16 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=X	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=X
0x1111	0x1111	0x1111	
0x2222	0x2222		0x2222
0x3333	0x3333	0x3333	
0x4444	0x4444		0x4444
0x5555	0x5555	0x5555	
0x6666	0x6666		0x6666
0x7777	0x7777	0x7777	
0x8888	0x8888		0x8888

Configuring 18-Bit Receive Mode

For 18-bit non-split receive mode, if `EPPI_CTL.PACKEN` =0, the EPPI zero-fills or sign-extends the incoming data into a 32-bit word. If `EPPI_CTL.PACKEN` =1, the EPPI first zero-fills or sign-extends the incoming data to 24 bits, and then packs four such 24-bit data words into three 32-bit words. Alternate even or odd samples can be skipped based on the `EPPI_CTL.SKIPEN` and `EPPI_CTL.SKIPEO` bits. The `EPPI_CTL.SWAPEN` bit has no effect.

Table 32-21: 18-bit Receive Mode with Packing Disabled

Pin Data (18 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=0
0x0 6666	0x0000 6666	0x0000 6666	
0x1 7777	0x0001 7777		0x0001 7777
0x2 8888	0x0002 8888	0x0002 8888	
0x3 9999	0x0003 9999		0x0003 9999

Table 32-22: 18-bit Receive Mode with Packing Enabled

Pin Data (18 bits)	DMA DATA SKIPEN=0 SKIPEO=X SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=1 SWAPEN=X SIGNEXT=0	DMA DATA SKIPEN=1 SKIPEO=0 SWAPEN=X SIGNEXT=0
0x0 1122			
0x1 3344	0x4400 1122		
0x2 5566	0x5566 0133	0x6600 1122	
0x3 7788	0x0377 8802		0x8801 3344
0x0 99AA		0x99AA 0255	
0x1 BBCC	0xCC00 99AA		0xBBCC 0377
0x2 DDEE	0xDDEE 01BB	0x02DD EE00	
0x3 FF12	0x03FF 122D		0x03FF 1201

Configuring 8-Bit Split Receive Mode

For 8-bit split receive mode, the `EPPI_CTL.PACKEN` and `EPPI_CTL.SIGNEXT` bits are not valid. The EPPI always packs 4 bytes of data into one 32-bit word.

Table 32-23: 8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 0

Pin Data (8 bits)	SPLTEO=1 SUBSPLTODD= 0 SWAPEN=0 SKIPEN=0 SKIPEO=X			SPLTEO=1 SUBSPLTODD= 1 SWAPEN=0 SKIPEN=0 SKIPEO=X		
	DMACFG=1		DMACFG=0	DMACFG=1		DMACFG=0
	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel	Primary DMA Channel	Secondary DMA Channel	Primary DMA Channel
V_0						
Y_0						
U_0						
Y_1						
V_1						
Y_2						
U_1		$U_1V_1U_0V_0$	$U_1V_1U_0V_0$			
Y_3	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$
V_2						
Y_4						
U_2						
Y_5						
V_3					$V_3V_2V_1V_0$	$V_3V_2V_1V_0$
Y_6						
U_3		$U_3V_3U_2V_2$	$U_3V_3U_2V_2$		$U_3U_2U_1U_0$	
Y_7	$Y_7Y_6Y_5Y_4$		$Y_7Y_6Y_5Y_4$	$Y_7Y_6Y_5Y_4$		$Y_7Y_6Y_5Y_4$
V_4						$U_3U_2U_1U_0$

Table 32-24: 8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 1

Pin Data (8 bits)	**SPLTEO=1 SUBSPLTODD=0 SWAPEN=1 SKIPEN=0 SKIPEO=X**			**SPLTEO=1 SUBSPLTODD=1 SWAPEN=1 SKIPEN=0 SKIPEO=X**		
	DMACFG=1		**DMACFG=0**	**DMACFG=1**		**DMACFG=0**
	PRIMARY DMA CHANNEL	**SECONDARY DMA CHANNEL**	**PRIMARY DMA CHANNEL**	**PRIMARY DMA CHANNEL**	**SECONDARY DMA CHANNEL**	**PRIMARY DMA CHANNEL**
V_0						
Y_0						
U_0						
Y_1						
V_1						
Y_2						
U_1		$V_0U_0V_1U_1$	$V_0U_0V_1U_1$			
Y_3	$Y_0Y_1Y_2Y_3$		$Y_0Y_1Y_2Y_3$	$Y_0Y_1Y_2Y_3$		$Y_0Y_1Y_2Y_3$
V_2						
Y_4						
U_2						
Y_5						
V_3					$V_0V_1V_2V_3$	$V_0V_1V_2V_3$
Y_6						
U_3		$V_2U_2V_3U_3$	$V_2U_2V_3U_3$		$U_0U_1U_2U_3$	
Y_7	$Y_4Y_5Y_6Y_7$		$Y_4Y_5Y_6Y_7$	$Y_4Y_5Y_6Y_7$		$Y_4Y_5Y_6Y_7$
V_4						$U_0U_1U_2U_3$

When the bits settings are `EPPI_CTL.SPLTEO` =1, `EPPI_CTL.SUBSPLTODD` =1 and `EPPI_CTL.DMACFG` =0, the EPPI packs the second Chroma component sent over the DMA bus completely before the Luma component. However, it is intentionally held until that previous word is moved out. This functionality allows the separation of Luma and Chroma values into individual buffers when using 2D-DMA. The second Chroma component is U0U1U2U3 in the ***8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 0*** and ***8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 1*** tables. The Luma component is Y4Y5Y6Y7 in the ***8-bit Split Receive Mode with SKIPEN = 0 and SWAPEN = 1*** table.

Configuring 10/12/14/16-Bit Split Receive Mode with SPLTWRD=0

For 16-bit split receive mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always packs two 16-bit words into one 32-bit word. For 10, 12, or 14-bit split receive modes, the EPPI first either sign-extends or zero-fills the incoming data into a 16-bit word. The EPPI then packs two of these words into one 32-bit word to send to the DMA.

Table 32-25: 16-bit Split Receive Mode with SPLTWRD = 0, SKIPEN = 0 and SWAPEN = 0

Pin Data (16 bits)	**SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X**			**SPLTEO=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X**		
	DMACFG=1		**DMACFG=0**	**DMACFG=1**		**DMACFG=0**
	Primary DMA Channel	**Secondary DMA Channel**	**Primary DMA Channel**	**Primary DMA Channel**	**Secondary DMA Channel**	**Primary DMA Channel**
V_0						
Y_0						
U_0		U_0V_0	U_0V_0			
Y_1	Y_1Y_0		Y_1Y_0	Y_1Y_0		Y_1Y_0
V_1					V_1V_0	V_1V_0
Y_2						
U_1		U_1V_1	U_1V_1		U_1U_0	
Y_3	Y_3Y_2		Y_3Y_2	Y_3Y_2		Y_3Y_2
V_2						U_1U_0

Table 32-26: 16-bit Split Receive Mode with SPLTWRD = 0, SKIPEN = 0 and SWAPEN = 1

Pin Data (16 bits)	**SPLTEO=1 SUBSPLTODD=0 SWAPEN=1 SKIPEN=0 SKIPEO=X**			**SPLTEO=1 SUBSPLTODD=1 SWAPEN=1 SKIPEN=0 SKIPEO=X**		
	DMACFG=1		**DMACFG=0**	**DMACFG=1**		**DMACFG=0**
	PRIMARY DMA CHANNEL	**SECONDARY DMA CHANNEL**	**PRIMARY DMA CHANNEL**	**PRIMARY DMA CHANNEL**	**SECONDARY DMA CHANNEL**	**PRIMARY DMA CHANNEL**
V_0						
Y_0						
U_0		V_0U_0	V_0U_0			
Y_1	Y_0Y_1		Y_0Y_1	Y_0Y_1		Y_0Y_1
V_1					V_0V_1	V_0V_1
Y_2						
U_1		V_1U_1	V_1U_1		U_0U_1	
Y_3	Y_2Y_3		Y_2Y_3	Y_2Y_3		Y_2Y_3
V_2						U_0U_1

Configuring 16-Bit Split Receive Mode with SPLTWRD=1

For 16-bit split receive mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always packs two 16-bit words into one 32-bit word. The `EPPI_CTL.SPLTWRD` bit is only valid when the `EPPI_CTL.DLEN` bit =16 bits.

Table 32-27: 16-bit Split Receive Mode with SPLTWRD = 1, SKIPEN = 0 and SWAPEN = 0

Pin Data (16 bits)	**SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X**			**SPLT_EVEN_ODD=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X**		
	DMACFG=1		**DMACFG=0**	**DMACFG=1**		**DMACFG=0**
	Primary DMA Channel	**Secondary DMA Channel**	**Primary DMA Channel**	**Primary DMA Channel**	**Secondary DMA Channel**	**Primary DMA Channel**
V_0Y_0						

Table 32-27: 16-bit Split Receive Mode with SPLTWRD = 1, SKIPEN = 0 and SWAPEN = 0 (Continued)

Pin Data (16 bits)	**SPLTEO=1 SUBSPLTODD=0 SWAPEN=0 SKIPEN=0 SKIPEO=X**			**SPLT_EVEN_ODD=1 SUBSPLTODD=1 SWAPEN=0 SKIPEN=0 SKIPEO=X**		
	DMACFG=1		**DMACFG=0**	**DMACFG=1**		**DMACFG=0**
	Primary DMA Channel	**Secondary DMA Channel**	**Primary DMA Channel**	**Primary DMA Channel**	**Secondary DMA Channel**	**Primary DMA Channel**
U_0Y_1						
V_1Y_2						
U_1Y_3	$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	$Y_3Y_2Y_1Y_0$	$Y_3Y_2Y_1Y_0$		$Y_3Y_2Y_1Y_0$
V_2Y_4			$U_1V_1U_0V_0$			
U_2Y_5						
V_3Y_6					$V_3V_2V_1V_0$	$V_3V_2V_1V_0$
U_3Y_7	$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	$Y_7Y_6Y_5Y_4$	$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	$Y_7Y_6Y_5Y_4$
V_4Y_8			$U_3V_3U_2V_2$			$U_3U_2U_1U_0$

Configuring 8-Bit Transmit Mode

For 8-bit non-split transmit mode, if the `EPPI_CTL.PACKEN` bit =1, the DMA is a 32-bit DMA and the EPPI unpacks the 32-bit word from memory into 4 bytes to transmit. The EPPI transmits either the MSBs or the LSBs as the first data, depending on the `EPPI_CTL.SWAPEN` bit setting. If `EPPI_CTL.PACKEN` =0, the DMA is a 16-bit DMA and the EPPI transmits the lower 8 bits. The `EPPI_CTL.SWAPEN` bit has no effect when `EPPI_CTL.PACKEN` =0.

Table 32-28: 8-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	**Pin Data when SWAPEN=0**	**Pin Data when SWAPEN=1**
0x11223344	0x44	0x11
0x55667788	0x33	0x22
	0x22	0x33
	0x11	0x44
	0x88	0x55
	0x77	0x66
	0x66	0x77

Table 32-28: 8-bit Transmit Mode with Packing Enabled (Continued)

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
	0x55	0x88

Table 32-29: Data Sent in 8-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x34
0x2345	0x45
0x3456	0x56

Configuring 10/12/14-Bit Transmit Modes

For 10, 12, or 14-bit non-split transmit modes, if the `EPPI_CTL.PACKEN` bit =1, the DMA is a 32-bit DMA. The EPPI unpacks the 32-bit word from memory into two 16-bit data words. The EPPI then transmits the required LSBs from each data word. The EPPI transmits either the most significant word or the least significant word as the first data, depending on the `EPPI_CTL.SWAPEN` bit setting. If `EPPI_CTL.PACKEN` =0, the DMA is a 16-bit DMA and the EPPI transmits the required LSBs. The `EPPI_CTL.SWAPEN` bit has no effect when the `EPPI_CTL.PACKEN` bit =0.

Table 32-30: 10-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
0x1111 2222	0x222	0x111
0x3333 4444	0x111	0x222
	0x044	0x333
	0x333	0x044

Table 32-31: 10-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x234
0x2345	0x345
0x3456	0x056
0x4567	0x167

Configuring 16-Bit Transmit Mode

For 16-bit non-split transmit mode, if the `EPPI_CTL.PACKEN` bit =1, the DMA is a 32-bit DMA. The EPPI unpacks the 32-bit word from memory into two 16-bit data words to transmit. The EPPI transmits either the MSBs or the LSBs as the first data, depending on the `EPPI_CTL.SWAPEN` bit setting. If the `EPPI_CTL.PACKEN` bit =0, the

DMA is a 16-bit DMA and the EPPI transmits the data as is. The EPPI_CTL.SWAPEN has no effect when EPPI_CTL.PACKEN bit =0.

Table 32-32: 16-bit Transmit Mode with Packing Enabled

DMA Data (32 bits)	Pin Data when SWAPEN=0	Pin Data when SWAPEN=1
0x1111 2222	0x2222	0x1111
0x3333 4444	0x1111	0x2222
	0x4444	0x3333
	0x3333	0x4444

Table 32-33: 16-bit Transmit Mode with Packing Disabled

DMA Data (16 bits)	Pin Data SWAPEN=X
0x1234	0x1234
0x2345	0x2345
0x3456	0x3456

Configuring 18-Bit Transmit Mode

For 18-bit transmit mode, if the EPPI_CTL.PACKEN bit =1, the DMA is a 32-bit DMA and the EPPI unpacks the 32-bit word from memory. In this mode, when EPPI_CTL.RGBFMTEN is set, the least significant 2 bits of R, G, and B are dropped.

Table 32-34: 18-bit Transmit Mode with Packing Enabled

DMA Data	Pin Data (18-bits)	
	RGBFMTEN=0	RGBFMTEN=1
0x0123 4567	0x3 4567	0x0 8459
0x89AB CDEF	0x1 EF01	0x3 3EC0
0x0123 4567	0x3 89AB	0x1 98AA
	0x1 2345	0x0 0211

Table 32-35: 18-bit Transmit Mode with Packing Disabled

DMA Data	Pin Data (18-bits)	
	RGBFMTEN=0	RGBFMTEN=1
0x0123 4567	0x3 4567	0x0 8459
0x89AB CDEF	0x3 CDEF	0x2 ACFB
0x0123 4567	0x3 4567	0x0 8459

Configuring 8-Bit Split Transmit Mode

For 8-bit split transmit mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always unpacks the 32-bit DMA data into 4 bytes to transmit.

Table 32-36: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=0

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	V_0	$U_1V_1U_0V_0$	V_0
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	Y_0	$Y_3Y_2Y_1Y_0$	Y_0
		U_0	$U_3V_3U_2V_2$	U_0
		Y_1	$Y_7Y_6Y_5Y_4$	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3
		V_2		V_2
		Y_4		Y_4
		U_2		U_2
		Y_5		Y_5
		V_3		V_3
		Y_6		Y_6
		U_3		U_3
		Y_7		Y_7

Table 32-37: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1 and SWAPEN=0

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_0	$V_3V_2V_1V_0$	V_0
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	Y_0	$Y_3Y_2Y_1Y_0$	Y_0
	$V_7V_6V_5V_4$	U_0	$U_3U_2U_1U_0$	U_0
	$U_7U_6U_5U_4$	Y_1	$Y_7Y_6Y_5Y_4$	Y_1
		V_1		V_1
		Y_2		Y_2

Table 32-37: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1 and SWAPEN=0 (Continued)

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
		U_1		U_1
		Y_3		Y_3
		V_2		V_2
		Y_4		Y_4
		U_2		U_2
		Y_5		Y_5
		V_3		V_3
		Y_6		Y_6
		U_3		U_3
		Y_7		Y_7

Table 32-38: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=1

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	U_1	$U_1V_1U_0V_0$	U_1
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	Y_3	$Y_3Y_2Y_1Y_0$	Y_3
		V_1	$U_3V_3U_2V_2$	V_1
		Y_2	$Y_7Y_6Y_5Y_4$	Y_2
		U_0		U_0
		Y_1		Y_1
		V_0		V_0
		Y_0		Y_0
		U_3		U_3
		Y_7		Y_7
		V_3		V_3
		Y_6		Y_6
		U_2		U_2
		Y_5		Y_5
		V_2		V_3

Table 32-38: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=0 and SWAPEN=1 (Continued)

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
		Y_4		Y_4

Table 32-39: 8-bit Split Transmit Mode with SPLTEO=1, SUBSPLTODD=1, and SWAPEN=1

DMACFG=1			DMACFG=0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (8 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (8 bits)**
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_3	$V_3V_2V_1V_0$	V_3
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	Y_3	$Y_3Y_2Y_1Y_0$	Y_3
	$V_7V_6V_5V_4$	U_3	$U_3V_3U_2V_2$	U_3
	$U_7U_6U_5U_4$	Y_2	$Y_7Y_6Y_5Y_4$	Y_2
		V_2		V_2
		Y_1		Y_1
		U_2		U_2
		Y_0		Y_0
		V_1		V_1
		Y_7		Y_7
		U_1		U_1
		Y_6		Y_6
		V_0		V_0
		Y_5		Y_5
		U_0		U_0
		Y_4		Y_4

Configuring 10/12/14/16-Bit Transmit Mode with SPLTWRD=0

For 16-bit split transmit mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always unpacks the 32-bit DMA data into two 16-bit words to transmit. For 10, 12, or 14-bit split transmit modes, the EPPI first unpacks the data in the same way as for 16-bit transmit mode. But, the EPPI transmits only the required number of LSBs.

Table 32-40: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_1Y_0	U_0V_0	V_0	U_0V_0	V_0

Table 32-40: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 0 (Continued)

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_3Y_2	U_1V_1	Y_0	Y_1Y_0	Y_0
		U_0	U_1V_1	U_0
		Y_1	Y_3Y_2	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3

Table 32-41: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 1, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_1Y_0	V_1V_0	V_0	V_1V_0	V_0
Y_3Y_2	U_1U_0	Y_0	Y_1Y_0	Y_0
	V_3V_2	U_0	U_1U_0	U_0
	U_3U_2	Y_1	Y_3Y_2	Y_1
		V_1		V_1
		Y_2		Y_2
		U_1		U_1
		Y_3		Y_3

Table 32-42: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 1

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_1Y_0	V_0U_0	V_0	V_0U_0	V_0
Y_3Y_2	V_1U_1	Y_1	Y_1Y_0	Y_1
		U_0	V_1U_1	U_0
		Y_0	Y_3Y_2	Y_0
		V_1		V_1
		Y_3		Y_3

Table 32-42: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 0, and SWAPEN = 1 (Continued)

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
		U_1		U_1
		Y_2		Y_2

Table 32-43: 16-bit Split Transmit Mode with SPLTEO = 1, SUBSPLTODD = 1, and SWAPEN = 1

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
Y_1Y_0	V_1V_0	V_1	V_1V_0	V_1
Y_3Y_2	U_1U_0	Y_1	Y_1Y_0	Y_1
	V_3V_2	U_1	U_1U_0	U_1
	U_3U_2	Y_0		Y_0
		V_0		V_0
		Y_3		Y_1
		U_0		U_0
		Y_2		Y_2

Configuring 16-Bit Split Transmit Mode with SPLTWRD=1

For 16-bit split transmit mode, the `EPPI_CTL.PACKEN` bit is not valid. The EPPI always unpacks the 32-bit DMA data into two 16-bit words to transmit. The `EPPI_CTL.SPLTWRD` bit is only valid when the `EPPI_CTL.DLEN` bit =16 bits.

Table 32-44: 16-bit Split Transmit Mode with SPLTWRD = 1, SUBSPLTODD = 0, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
DMA0 DATA (32 bits)	**DMA1 DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
$Y_3Y_2Y_1Y_0$	$U_1V_1U_0V_0$	V_0Y_0	$U_1V_1U_0V_0$	V_0Y_0
$Y_7Y_6Y_5Y_4$	$U_3V_3U_2V_2$	U_0Y_1	$Y_3Y_2Y_1Y_0$	U_0Y_1
		V_1Y_2	$U_3V_3U_2V_2$	V_1Y_2
		U_1Y_3	$Y_7Y_6Y_5Y_4$	U_1Y_3
		V_2Y_4		V_2Y_4
		U_2Y_5		U_2Y_5
		V_3Y_6		V_3Y_6
		U_3Y_7		U_3Y_7

Table 32-45: 16-bit Split Transmit Mode with SPLTWRD = 1, SUBSPLTODD = 1, and SWAPEN = 0

DMACFG = 1			DMACFG = 0	
PRIMARY DMA DATA (32 bits)	**SECONDARY DMA DATA (32 bits)**	**Pin Data (16 bits)**	**DMA0 DATA (32 bits)**	**Pin Data (16 bits)**
$Y_3Y_2Y_1Y_0$	$V_3V_2V_1V_0$	V_0Y_0	$V_3V_2V_1V_0$	V_0Y_0
$Y_7Y_6Y_5Y_4$	$U_3U_2U_1U_0$	U_0Y_1	$Y_3Y_2Y_1Y_0$	U_0Y_1
	$V_7V_6V_5V_4$	V_1Y_2	$U_3U_2U_1U_0$	V_1Y_2
	$U_7U_6U_5U_4$	U_1Y_3	$Y_7Y_6Y_5Y_4$	U_1Y_3
		V_2Y_4		V_2Y_4
		U_2Y_5		U_2Y_5
		V_3Y_6		V_3Y_6
		U_3Y_7		U_3Y_7

EPPI Programming Concepts

This section provides information on SMPTE programming.

SMPTE Modes Programming

The programming model of SMPTE modes is similar to ITU Modes. All programming modes pertaining to ITU modes like XFRTYPE, FSCFG, FLDSEL, and BLANKGEN hold true for SMPTE modes as well. The only difference is that since ITU modes use Y-Cr/Cb interleaved data and SMPTE use parallel Y-Cr/Cb data, SPLTWRD could be set while operating in SMPTE modes. The ***Programming Modes for SMPTE Formats*** table describes the programming modes for different SMPTE formats.

Table 32-46: Programming Modes for SMPTE Formats

SMPTE Format	SMPTE Channel Width	EPPI Input Bit Width	EPPI Mode	Remarks
296M	8	16 Cr/Cb - [15:8] Y - [7:0]	DLEN = 16 bits SPLTWRD = 1	SIGNEXT not supported
	8	16 Cr/Cb - [15:8] Y - [7:0]	DLEN = 16 bits SPLTWRD = 1	SIGNEXT not supported

ADSP-BF70x EPPI Register Descriptions

Enhanced Parallel Peripheral Interface (EPPI) contains the following registers.

Table 32-47: ADSP-BF70x EPPI Register List

Name	Description
EPPI_CLKDIV	Clock Divide Register
EPPI_CTL	Control Register
EPPI_CTL2	Control Register 2 Register
EPPI_EVENCLIP	Clipping Register for EVEN (Luma) Data Register
EPPI_FRAME	Lines Per Frame Register
EPPI_FS1_DLY	Frame Sync 1 Delay Value Register
EPPI_FS1_PASPL	FS1 Period Register / EPPI Active Samples Per Line Register
EPPI_FS1_WLHB	FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register
EPPI_FS2_DLY	Frame Sync 2 Delay Value Register
EPPI_FS2_PALPF	FS2 Period Register / EPPI Active Lines Per Field Register
EPPI_FS2_WLVB	FS2 Width Register / EPPI Lines Of Vertical Blanking Register
EPPI_HCNT	Horizontal Transfer Count Register
EPPI_HDLY	Horizontal Delay Count Register
EPPI_IMSK	Interrupt Mask Register
EPPI_LINE	Samples Per Line Register
EPPI_ODDCLIP	Clipping Register for ODD (Chroma) Data Register
EPPI_STAT	Status Register
EPPI_VCNT	Vertical Transfer Count Register
EPPI_VDLY	Vertical Delay Count Register

Clock Divide Register

The EPPI_CLKDIV register provides the divisor for EPPI internal clock generation. The generated clock frequency is given by following formula:

EPPI_CLK = (SCLK0) / (EPPI_CLKDIV + 1)

Note that a value of 0xFFFF is invalid for the EPPI_CLKDIV register.

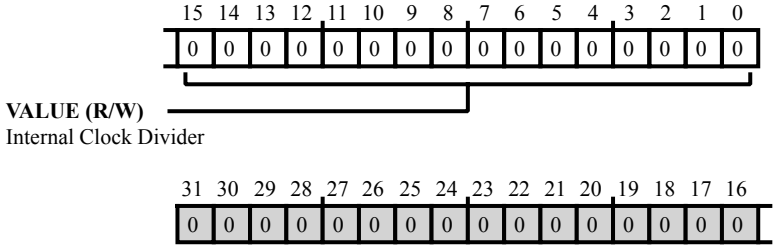

Figure 32-14: EPPI_CLKDIV Register Diagram

Table 32-48: EPPI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Internal Clock Divider.

Control Register

The EPPI_CTL register configures the EPPI for operating mode, control signal polarities, and data width of the port.

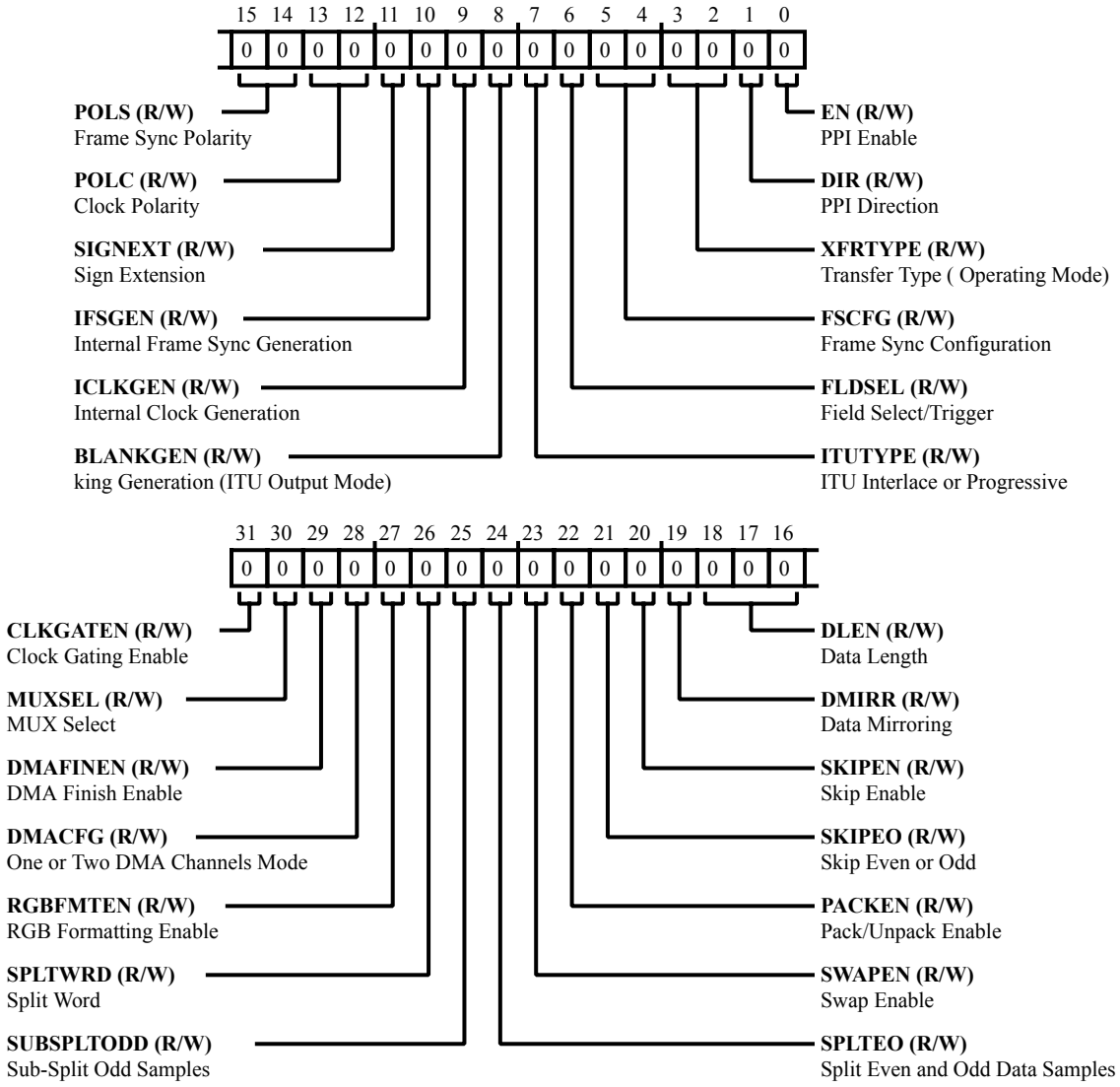

Figure 32-15: EPPI_CTL Register Diagram

Table 32-49: EPPI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CLKGATEN	Clock Gating Enable. The EPPI_CTL.CLKGATEN bit enables using the EPPI_FS3 pin as a clock gating pin. When EPPI_CTL.CLKGATEN is set, the EPPI_FS3 pin acts as a clock gating signal, and both the internal and external clock are gated. Note that the EPPI_FS3 pin gating signal is active low, and the EPPI_CTL.CLKGATEN selection is ignored if EPPI_CTL.MUXSEL is set or EPPI_CTL.FSCFG equals 0x3.	
		0	Disable
		1	Enable
30 (R/W)	MUXSEL	MUX Select. The EPPI_CTL.MUXSEL bit enables multiplexing of a primary and alternate camera using the EPPI main data and clock lines. For more information on this feature, see the EPPI functional description.	
		0	Normal Operation
		1	Multiplexed Operation
29 (R/W)	DMAFINEN	DMA Finish Enable. The EPPI_CTL.DMAFINEN bit selects whether or not the EPPI sends a finish command (010) through the DDE COMMAND line soon after a frame/line is received completely.	
		0	No Finish Command
		1	Enable Send Finish Command
28 (R/W)	DMACFG	One or Two DMA Channels Mode. The EPPI_CTL.DMACFG bit is valid only if EPPI_CTL.SPLTEO is set. If EPPI_CTL.DMACFG is set, the EPPI uses two DMA channels. And, if EPPI_CTL.DMACFG is cleared, the EPPI uses only one DMA channel.	
		0	PPI Uses One DMA Channel
		1	PPI Uses Two DMA Channels
27 (R/W)	RGBFMTEN	RGB Formatting Enable. For 16- or 18-bit transmit modes only, the EPPI_CTL.RGBFMTEN bit enables conversion of RGB888 from memory into RGB666 output data (18-bit transmit) or enables conversion of RGB888 from memory into RGB565 output data (16-bit transmit). Note that EPPI_CTL.SPLTEO and EPPI_CTL.RGBFMTEN should never be set simultaneously.	
		0	Disable RGB Formatted Output
		1	Enable RGB Formatted Output

Table 32-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/W)	SPLTWRD	Split Word. The EPPI_CTL.SPLTWRD bit selects split word data placement when the data length (EPPI_CTL.DLEN) selects 16-, 20-, or 24-bit data words. For all other EPPI_CTL.SPLTWRD values, the set or clear selections for EPPI_CTL.SPLTWRD produce the same result (act as though EPPI_CTL.SPLTWRD is cleared). For EPPI_CTL.SPLTWRD set, the EPPI_CTL.DLEN values below result in the following combinations of split words: DLEN Cr/Cb data Y data 16 PPI_DATA[15:8] PPI_DATA[7:0] 20 PPI_DATA[19:10] PPI_DATA[9:0] 24 PPI_DATA[23:12] PPI_DATA[11:0]	
		0	PPI_DATA has (DLEN-1) bits of Y or Cr or Cb
		1	PPI_DATA Contains 2 Elements per Word
25 (R/W)	SUBSPLTODD	Sub-Split Odd Samples. The EPPI_CTL.SUBSPLTODD bit is valid only if EPPI_CTL.SPLTEO is set. If EPPI_CTL.SUBSPLTODD is set, the EPPI sub-splits the odd sub-stream, and packs them separately.	
		0	Disable
		1	Enable
24 (R/W)	SPLTEO	Split Even and Odd Data Samples. If EPPI_CTL.SPLTEO is set, the EPPI splits the incoming data stream into two sub-streams, an even stream and an odd stream, and packs them separately.	
		0	Do Not Split Samples
		1	Split Even/Odd Samples
23 (R/W)	SWAPEN	Swap Enable. The EPPI_CTL.SWAPEN selects whether or not to swap the order of the first data (most-significant bits versus least-significant bits) of the DMA word. For receive modes, the EPPI puts the first data in the most significant bits (if set) or puts the first data in the least significant bits (if cleared) of the DMA word. For transmit modes, the EPPI transmits the most significant bits in the DMA word as the first data (if set) or transmits the least significant bits in the DMA word as the first data (if cleared).	
		0	Disable
		1	Enable

Table 32-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
22 (R/W)	PACKEN	Pack/Unpack Enable. The EPPI_CTL.PACKEN select whether or not packing is enabled (for receive modes) and unpacking is enabled (for transmit modes). When this bit is set, EPPI transfer DMA is 32-bits wide. When this bit is cleared and the EPPI_CTL.DLEN is less than or equal to 16 bits, EPPI transfer DMA is 16-bits wide. For receive modes, if this bit is set, then the EPPI packs the incoming data into 32-bit words. If this bit is cleared, then the EPPI does not do any packing. For transmit modes, if this bit is set, then the EPPI always unpacks the 32-bit data from DMA. If this bit is not set, the EPPI does not do any unpacking.	
		0	Disable
		1	Enable
21 (R/W)	SKIPEO	Skip Even or Odd. The EPPI_CTL.SKIPEO bit selects whether even (if set) or odd (if cleared) samples are skipped if sample skipping is enabled (EPPI_CTL.SKIPEN is set). This feature only is useful for receive modes.	
		0	Skip Odd Samples
		1	Skip Even Samples
20 (R/W)	SKIPEN	Skip Enable. The EPPI_CTL.SKIPEN bit enables skipping alternate samples. This feature only is useful for receive modes.	
		0	No Samples Skipping
		1	Skip Alternate Samples
19 (R/W)	DMIRR	Data Mirroring. The EPPI_CTL.DMIRR field enables mirroring (bit reversing) of the data coming in or going out on the EPPI data pins.	
		0	No Data Mirroring
		1	Data Mirroring
18:16 (R/W)	DLEN	Data Length. The EPPI_CTL.DLEN bits select the data length for the EPPI. Note that the 20 bits data length selection is valid only for SMPTE modes (EPPI_CTL.SPLTWRD set).	

```
Pin      PPI Data      PPI Data
Data   (DAT_MRR=0)   (DAT_MRR=1)
------------------------------
 15         15             0
 14         14             1
....       ....           ....
  1          1            14
  0          0            15
```

Table 32-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	8 bits
		1	10 bits
		2	12 bits
		3	14 bits
		4	16 bits
		5	18 bits
		6	20 bits
		7	24 bits
15:14 (R/W)	POLS	Frame Sync Polarity. The `EPPI_CTL.POLS` selects whether the frame syncs' polarity is active low versus active high.	
		0	FS1 and FS2 are active high
		1	FS1 is active low. FS2 is active high
		2	FS1 is active high. FS2 is active low
		3	FS1 and FS2 are active low
13:12 (R/W)	POLC	Clock Polarity. The `EPPI_CTL.POLC` selects the rising versus falling edge for sampling data and sampling/driving syncs.	
		0	Clock/Sync Polarity Mode 0. For receive mode: Sample data on falling edge and sample/drive syncs on falling edge. For transmit mode: Drive data on rising edge and sample/drive syncs on rising edge.
		1	Clock/Sync Polarity Mode 1. For receive mode: Sample data on falling edge and sample/drive syncs on rising edge. For transmit mode: Drive data on rising edge and sample/drive syncs on falling edge.
		2	Clock/Sync Polarity Mode 2. For receive mode: Sample data on rising edge and sample/drive syncs on falling edge. For transmit mode: Drive data on falling edge and sample/drive syncs on rising edge.
		3	Clock/Sync Polarity Mode 3. For receive mode: Sample data on rising edge and sample/drive syncs on rising edge. For transmit mode: Drive data on falling edge and sample/drive syncs on falling edge.

Table 32-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	SIGNEXT	Sign Extension. The EPPI_CTL.SIGNEXT select whether (for receive modes when EPPI_CTL.DLEN selecting 16 bit data length) the data is sign extended or zero filled. Not that EPPI_CTL.SPLTWRD is removed from this shared bit.	
		0	Zero Filled
		1	Sign Extended
10 (R/W)	IFSGEN	Internal Frame Sync Generation. The EPPI_CTL.IFSGEN bit selects whether the frame syncs are generated internally or are supplied by an external device.	
		0	External Frame Sync
		1	Internal Frame Sync
9 (R/W)	ICLKGEN	Internal Clock Generation. The EPPI_CTL.ICLKGEN bit selects whether the EPPI_CLK is generated internally or is supplied by an external device.	
		0	External Clock
		1	Internal Clock
8 (R/W)	BLANKGEN	king Generation (ITU Output Mode). The EPPI_CTL.BLANKGEN enables ITU output with internal blanking. In GP 8, 10 transmit bit modes (when EPPI_CTL.SPLTWRD is cleared) and 16-, 20-, and 24-bit transmit modes (when EPPI_CTL.SPLTWRD is set), EPPI_CTL.BLANKGEN selects whether or not the EPPI generates blanking and generates preamble and insertion with active data from memory.	
		0	Disable
		1	Enable
7 (R/W)	ITUTYPE	ITU Interlace or Progressive. The EPPI_CTL.ITUTYPE selects interlaced or progressive operation for ITU656 mode. This selection is valid for both TX and RX modes.	
		0	Interlaced
		1	Progressive

Table 32-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W)	FLDSEL	Field Select/Trigger. The EPPI_CTL.FLDSEL bits configure the EPPI field and trigger selection. These are valid for GP modes (EPPI_CTL.XFRTYPE =0x3) and ITU656 active video mode (EPPI_CTL.XFRTYPE cleared).	
		0	Field Mode 0. Read field 1 (for ITU656 active video mode). Set internal trigger (for GP RX mode). FS3 is toggled on FS2 assertion followed by FS1 assertion (when the EPPI_CTL.FSCFG bit selects sync mode 3 and the EPPI_CTL.IFSGEN bit selects internal frame sync).
		1	Field Mode 1 Read field 1 and field 2 (ITU656 active video mode). Set external trigger (GP RX mode). FS3 is toggled on FS2 assertion (when the EPPI_CTL.FSCFG bit selects sync mode 3 and the EPPI_CTL.IFSGEN bit selects internal frame sync).
5:4 (R/W)	FSCFG	Frame Sync Configuration. The EPPI_CTL.FSCFG bits configure the EPPI frame syncs. These are valid only for GP modes (EPPI_CTL.XFRTYPE =0x3). The output of the frames syncs also depends on whether the EPPI transfer direction is transmit and the EPPI is in ITU output mode (EPPI_CTL.BLANKGEN is set).	
		0	Sync Mode 0. FS0 driven in GP mode. FS0 not driven in ITU output mode.
		1	Sync Mode 1. FS1 driven in GP mode. HSYNC driven on FS1 in ITU output mode.
		2	Sync Mode 2. FS2 driven in GP mode. HSYNC driven on FS1 and VSYNC driven on FS1 in ITU output mode.
		3	Sync Mode 3. FS3 driven in GP mode. HSYNC driven on FS1, VSYNC driven on FS2, and FIELD driven on FS3 in ITU output mode.

Table 32-49: EPPI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3:2 (R/W)	XFRTYPE	Transfer Type (Operating Mode). The EPPI_CTL.XFRTYPE bits select the EPPI operating mode. In receive mode (EPPI_CTL.DIR cleared), the EPPI modes include ITU656 active video only mode, ITU656 entire field mode, ITU656 vertical blanking only mode, and non-ITU656 mode (GP mode). For transmit mode (EPPI_CTL.DIR set), the EPPI_CTL.XFRTYPE bits have no effect, and the EPPI (in transmit mode) is always in GP mode.	
		0	ITU656 Active Video Only Mode
		1	ITU656 Entire Field Mode
		2	ITU656 Vertical Blanking Only Mode
		3	Non-ITU656 Mode (GP Mode)
1 (R/W)	DIR	PPI Direction. The EPPI_CTL.DIR bit selects whether the EPPI is in receive mode (if cleared) or in transmit mode (if set).	
		0	Receive Mode
		1	Transmit Mode
0 (R/W)	EN	PPI Enable. The EPPI_CTL.EN bit enables or disables the EPPI.	
		0	Disable
		1	Enable

Control Register 2 Register

The EPPI_CTL2 register controls HSYNC finish signal generation.

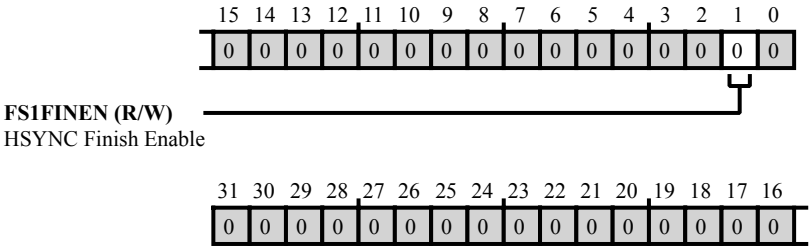

Figure 32-16: EPPI_CTL2 Register Diagram

Table 32-50: EPPI_CTL2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W)	FS1FINEN	HSYNC Finish Enable. The EPPI_CTL2.FS1FINEN bit selects whether (if set) the EPPI sends a finish command (010) through the DDE COMMAND line soon after a LINE is received completely or (if cleared) the EPPI sends a finish command (010) through the DDE COMMAND line soon after a FRAME is received completely. Note that the EPPI_CTL.DMAFINEN bit must be set for the EPPI to generate either of the finish commands.	
		0	Finish sent after frame RX done. PPI sends a finish command (010) through the DDE COMMAND line soon after a FRAME is received completely
		1	Finish sent after frame/line RX done. PPI sends a finish command (010) through the DDE COMMAND line soon after a frame/line is received completely.

Clipping Register for EVEN (Luma) Data Register

The EPPI_EVENCLIP register selects the clipping threshold for luma data, which provides clipping of individual video components.

The high even and low even spaces in EPPI_EVENCLIP are 16-bits wide and (depending on the EPPI_CTL.DLEN bit selection) only the corresponding video component bits are considered for clipping.

For example, if the EPPI is programmed in 10-bit mode, bits [9:0] and bits [25:16] constitute the clipping thresholds. The higher bits are (in this case) ignored.

Using the this method, 8-, 10-, 12- and 16-bit clipping thresholds can be set.

Note that when the EPPI is programmed in 16-, 20-, or 24-bit mode with the EPPI_CTL.SPLTWRD bit set, the luma data gets the clipping threshold levels of the EPPI_EVENCLIP register, and the chroma data gets the clipping threshold levels of the EPPI_ODDCLIP register.

Also, note that the EPPI_EVENCLIP and EPPI_ODDCLIP registers are ignored when the EPPI_CTL.RGBFMTEN bit is set.

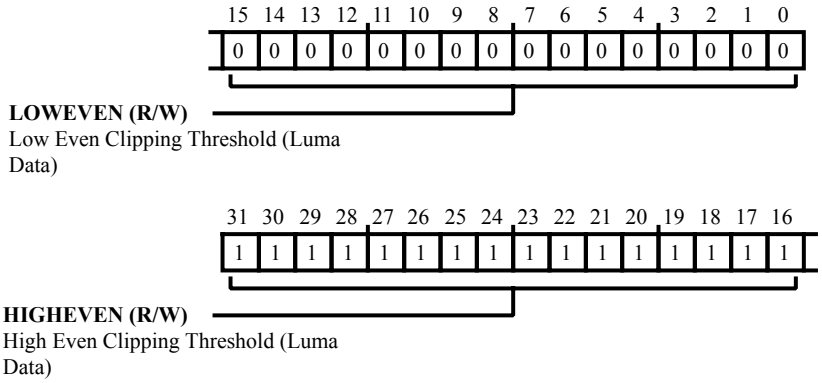

Figure 32-17: EPPI_EVENCLIP Register Diagram

Table 32-51: EPPI_EVENCLIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	HIGHEVEN	High Even Clipping Threshold (Luma Data). The EPPI_EVENCLIP.HIGHEVEN bit field selects the clipping threshold for luma data. The high even spaces are 16-bits wide and (depending on the EPPI_CTL.DLEN selection) only the corresponding video component bits are considered for clipping.
15:0 (R/W)	LOWEVEN	Low Even Clipping Threshold (Luma Data). The EPPI_EVENCLIP.LOWEVEN bit field selects the clipping threshold for luma data. The low even spaces are 16-bits wide and (depending on the EPPI_CTL.DLEN selection) only the corresponding video component bits are considered for clipping.

Lines Per Frame Register

The EPPI_FRAME register tracks the frame track overflow and underflow errors. This register should be programmed with the number of lines expected per frame. Any write to the EPPI_FRAME register will also write the same value to the EPPI_VCNT register. But, any write to the EPPI_VCNT register does not affect the EPPI_FRAME register value. So the EPPI_FRAME register should be programmed before the EPPI_VCNT register.

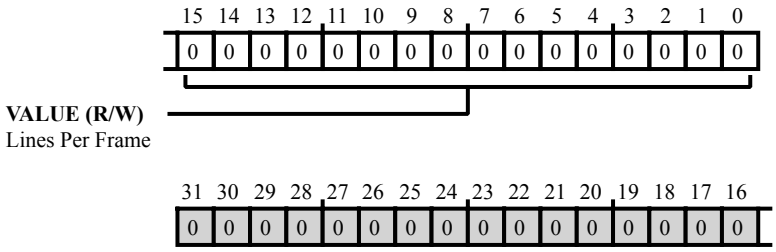

Figure 32-18: EPPI_FRAME Register Diagram

Table 32-52: EPPI_FRAME Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Lines Per Frame. The EPPI_FRAME.VALUE holds the number of lines expected per frame of data.

Frame Sync 1 Delay Value Register

The EPPI_FS1_DLY register selects the delay count (based on the period of the EPPI_CLK clock) between the first rising edge of EPPI_CLK after the EPPI is enabled and the first active edge of the associated frame sync when the internal frame sync is used.

Note that if the EPPI_FS1_DLY or EPPI_FS2_DLY registers are programmed with value 0, the EPPI operates as though 0 value is 1, and the first frame sync transition occurs after the completion of one period value of the respective counters.

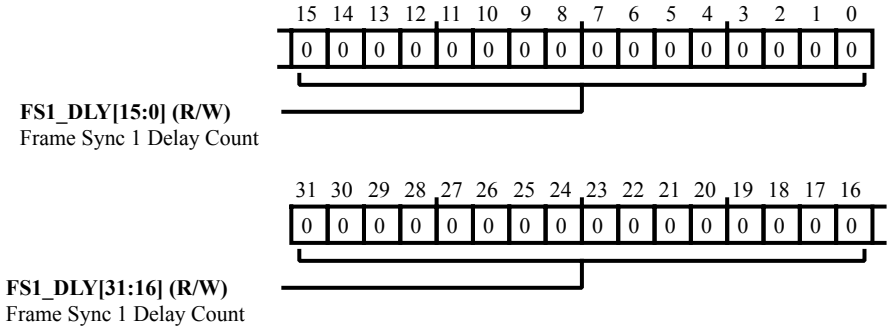

Figure 32-19: EPPI_FS1_DLY Register Diagram

Table 32-53: EPPI_FS1_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	FS1_DLY	Frame Sync 1 Delay Count. The EPPI_FS1_DLY.FS1_DLY bit field selects the delay count.

FS1 Period Register / EPPI Active Samples Per Line Register

The EPPI_FS1_PASPL register content varies depending on whether the EPPI is in GP1/2/3 FS modes or in GP transmit mode.

In GP 1, 2, or 3 FS modes, the EPPI_FS1_PASPL register is used for the generation of frame sync 1. The register contains the period required for EPPI_FS1 based on the EPPI_CLK clock.

In GP transmit mode with the EPPI_CTL.BLANKGEN bit set, this register contains the number of samples of active video or vertical blanking samples per line. When used for blanking generation, only the lower 16 bits are valid.

Note that a value of 0 for this register is illegal. If programmed as 0, the EPPI regards the EPPI_FS1_PASPL register as containing 1.

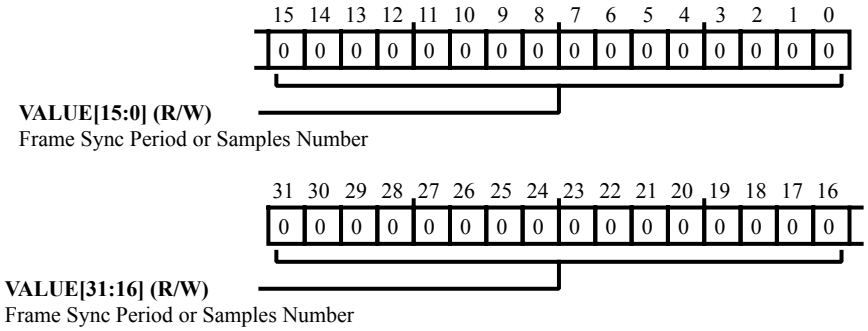

Figure 32-20: EPPI_FS1_PASPL Register Diagram

Table 32-54: EPPI_FS1_PASPL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Sync Period or Samples Number. In GP 1, 2, or 3 FS modes, the EPPI_FS1_PASPL.VALUE bit field is used for the generation of frame sync 1 and contains the period required for EPPI_FS1 based on the EPPI_CLK clock. In GP transmit mode with the EPPI_CTL.BLANKGEN bit set, this bit field contains the number of samples of active video or vertical blanking samples per line.

FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register

The EPPI_FS1_WLHB register's content varies depending on whether the EPPI is in GP1/2/3 FS modes or in GP transmit mode.

In GP 1, 2 or 3 FS modes, EPPI_FS1_WLHB is used for the generation of frame sync 1. The register contains the width required for EPPI_FS1 based on the EPPI_CLK clock.

In GP transmit mode with the EPPI_CTL.BLANKGEN bit set, this register contains the number of samples of horizontal blanking per line. When used for blanking generation, only the lower 16 bits are valid.

Note that a value of 0 for the EPPI_FS1_WLHB register is illegal. If programmed as 0, the EPPI regards the EPPI_FS1_WLHB register as containing 1.

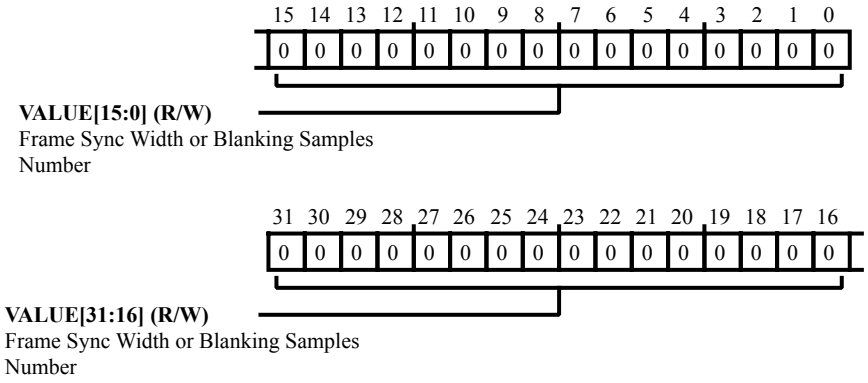

Figure 32-21: EPPI_FS1_WLHB Register Diagram

Table 32-55: EPPI_FS1_WLHB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Frame Sync Width or Blanking Samples Number. The EPPI_FS1_WLHB.VALUE bit field content varies depending on whether the EPPI is in GP1/2/3 FS modes or in GP transmit mode. In GP 1, 2 or 3 FS modes, the EPPI_FS1_WLHB.VALUE bit field is used for the generation of frame sync 1. The register contains the width required for EPPI_FS1 based on the EPPI_CLK clock. In GP transmit mode with EPPI_CTL.BLANKGEN set, this bit field contains the number of samples of horizontal blanking per line.

Frame Sync 2 Delay Value Register

The EPPI_FS2_DLY register selects the delay count (based on the period of the EPPI_CLK clock) between the first rising edge of EPPI_CLK after EPPI enabled and the first active edge of the associated frame sync when the internal frame sync is used.

Note that if the EPPI_FS1_DLY or EPPI_FS2_DLY registers are programmed with the value 0, the EPPI operates as though 0 value is 1, and the first frame sync transition occurs after the completion of one period value of the respective counters.

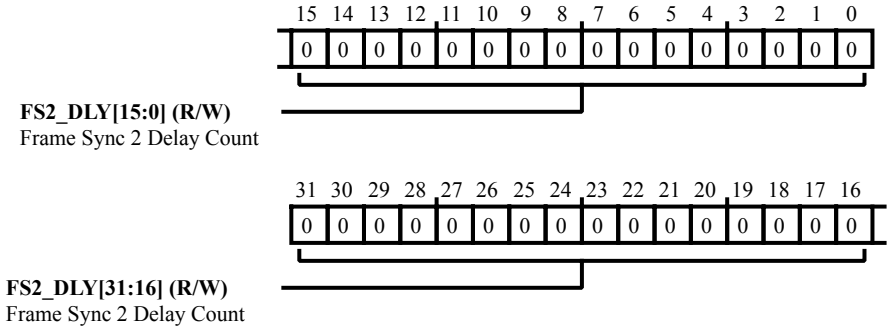

Figure 32-22: EPPI_FS2_DLY Register Diagram

Table 32-56: EPPI_FS2_DLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	FS2_DLY	Frame Sync 2 Delay Count. The EPPI_FS2_DLY.FS2_DLY bit field selects the delay count.

FS2 Period Register / EPPI Active Lines Per Field Register

The EPPI_FS2_PALPF register content varies depending on whether the EPPI is in GP2/3 FS modes or in GP transmit mode.

In GP 2 or 3 FS modes, EPPI_FS2_PALPF is used for the generation of frame sync 2. This register contains the period required for EPPI_FS2 based on the EPPI_CLK clock.

In GP transmit mode with the EPPI_CTL.BLANKGEN bit set, this register contains the number of lines of active video per field.

Note that a value of 0 for the EPPI_FS2_PALPF.F1ACT or EPPI_FS2_PALPF.F2ACT bits is illegal. If either is programmed as 0, the EPPI regards the 0 value fields as containing 1.

Also, note that for progressive video, the EPPI_FS2_PALPF.F2ACT bit is ignored.

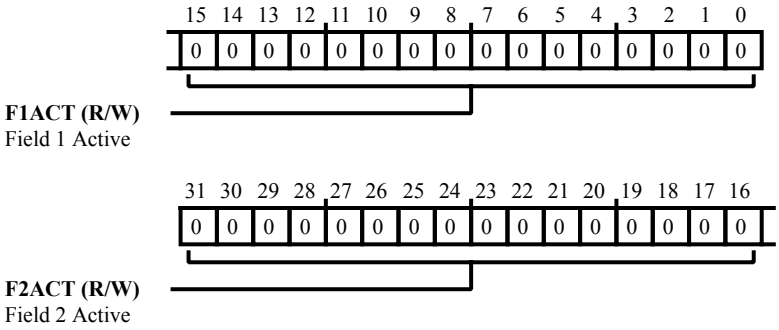

Figure 32-23: EPPI_FS2_PALPF Register Diagram

Table 32-57: EPPI_FS2_PALPF Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	F2ACT	Field 2 Active. The EPPI_FS2_PALPF.F2ACT bit field contains the number of lines of active data in field 2.
15:0 (R/W)	F1ACT	Field 1 Active. The EPPI_FS2_PALPF.F1ACT bit field contains the number of lines of active data in field 1.

FS2 Width Register / EPPI Lines Of Vertical Blanking Register

The EPPI_FS2_WLVB register content varies depending on whether the EPPI is in GP2/3 FS modes or in GP transmit mode.

In GP 2 or 3 FS modes, the EPPI_FS2_WLVB register is used for the generation of frame sync 2. The register contains the width required for EPPI_FS2 based on the EPPI_CLK clock.

In GP transmit mode with the EPPI_CTL.BLANKGEN bit set, this register contains the number or lines of vertical blanking.

Note that for progressive video, the EPPI_FS2_WLVB.F2VBBD and EPPI_FS2_WLVB.F2VBAD bits are ignored.

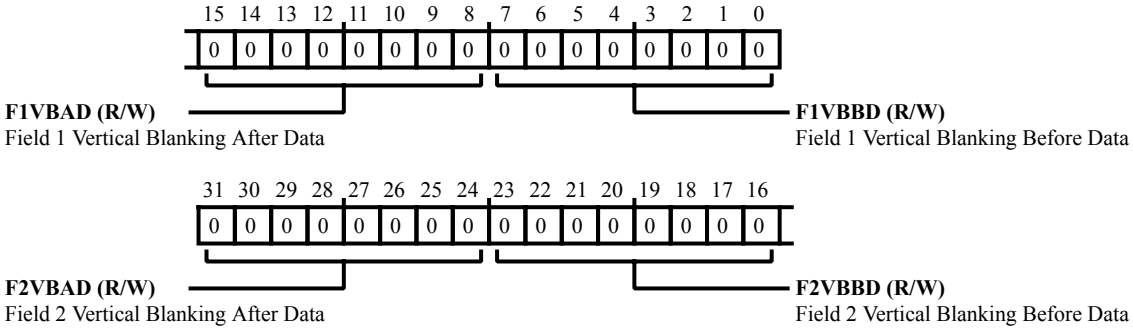

Figure 32-24: EPPI_FS2_WLVB Register Diagram

Table 32-58: EPPI_FS2_WLVB Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:24 (R/W)	F2VBAD	Field 2 Vertical Blanking After Data. The EPPI_FS2_WLVB.F2VBAD bit field contains the number of lines of vertical blanking after field 2.
23:16 (R/W)	F2VBBD	Field 2 Vertical Blanking Before Data. The EPPI_FS2_WLVB.F2VBBD bit field contains the number of lines of vertical blanking before field 2.
15:8 (R/W)	F1VBAD	Field 1 Vertical Blanking After Data. The EPPI_FS2_WLVB.F1VBAD bit field contains the number of lines of vertical blanking after field 1.
7:0 (R/W)	F1VBBD	Field 1 Vertical Blanking Before Data. The EPPI_FS2_WLVB.F1VBBD bit field contains the number of lines of vertical blanking before field 1.

Horizontal Transfer Count Register

The EPPI_HCNT register holds the number of samples to read in or write out per line, after EPPI_HDLY number of cycles have expired since the assertion of EPPI_FS1. Any write to the EPPI_LINE register modifies the EPPI_HCNT register. But, any write to the EPPI_HCNT register does not affect the EPPI_LINE register value. So the EPPI_HCNT register should be programmed after the EPPI_LINE register.

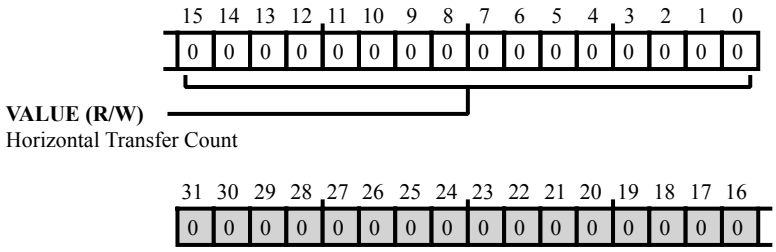

Figure 32-25: EPPI_HCNT Register Diagram

Table 32-59: EPPI_HCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Transfer Count. The EPPI_HCNT.VALUE holds the number of samples to read in or write out per line, after EPPI_HDLY number of cycles have expired since the last assertion of EPPI_FS1.

Horizontal Delay Count Register

The EPPI_HDLY register contains the number of clock cycles to delay after the assertion of EPPI_FS1 is detected before starting to read or write data.

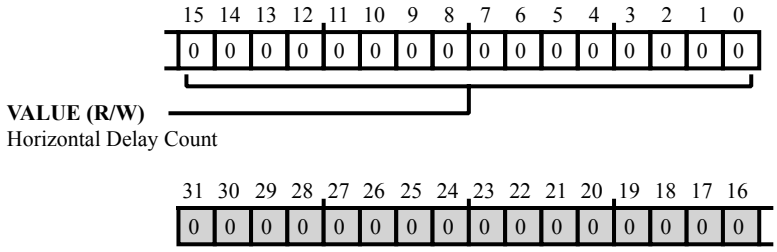

Figure 32-26: EPPI_HDLY Register Diagram

Table 32-60: EPPI_HDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Horizontal Delay Count. The EPPI_HDLY.VALUE holds the number of EPPI_CLK cycles to delay after assertion of EPPI_FS1 before starting to read or write data.

Interrupt Mask Register

The EPPI_IMSK register permits the masking (if associated bit is set) of EPPI error interrupts for YFIFO underflow or overflow, CFIFO underflow or overflow, line track overflow error, line track underflow error, frame track overflow error, frame track underflow error, and ERR_NCOR (ITU preamble error not corrected. These conditions are flagged in the EPPI_STAT register and cleared by write-1-to-clear.

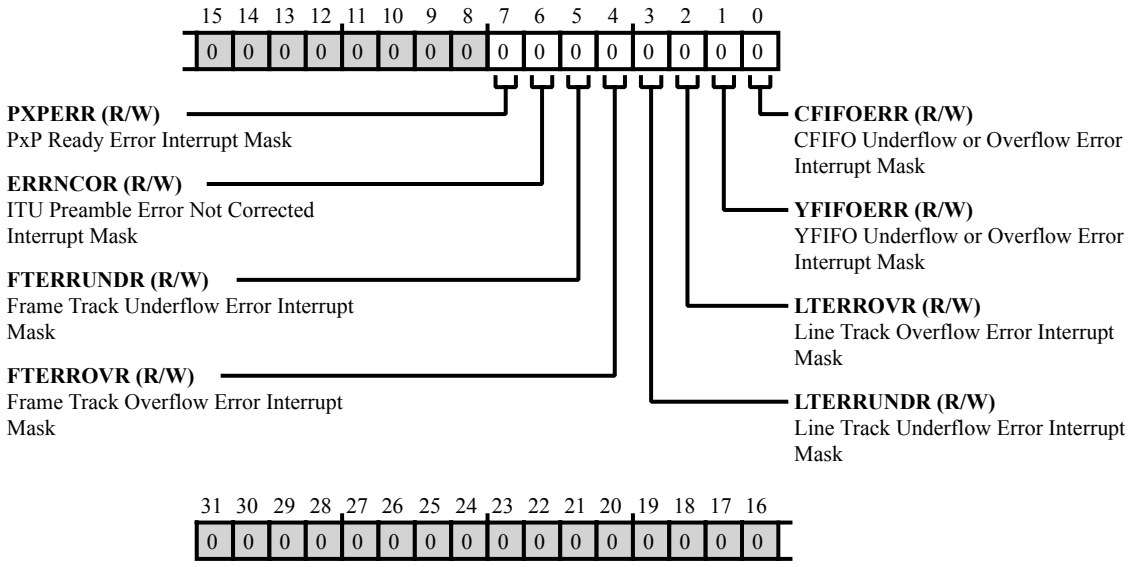

Figure 32-27: EPPI_IMSK Register Diagram

Table 32-61: EPPI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	PXPERR	PxP Ready Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
6 (R/W)	ERRNCOR	ITU Preamble Error Not Corrected Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
5 (R/W)	FTERRUNDR	Frame Track Underflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
4 (R/W)	FTERROVR	Frame Track Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt

Table 32-61: EPPI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/W)	LTERRUNDR	Line Track Underflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
2 (R/W)	LTERROVR	Line Track Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
1 (R/W)	YFIFOERR	YFIFO Underflow or Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt
0 (R/W)	CFIFOERR	CFIFO Underflow or Overflow Error Interrupt Mask.	
		0	Unmask Interrupt
		1	Mask Interrupt

Samples Per Line Register

The EPPI_LINE register tracks the line track overflow and underflow errors. This register should be programmed with the number of samples expected per line. Any write to the EPPI_LINE register will also write the same value to the EPPI_HCNT register. However, any write to the EPPI_HCNT register does not affect the EPPI_LINE register value. So the EPPI_LINE register should be programmed before the EPPI_HCNT register.

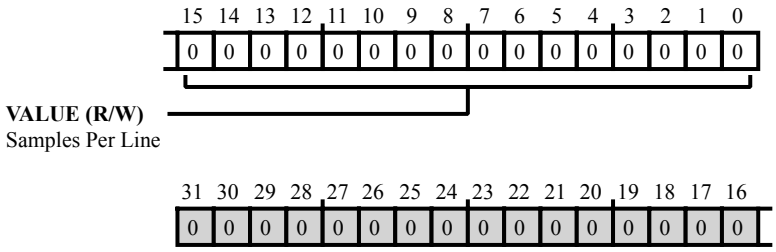

Figure 32-28: EPPI_LINE Register Diagram

Table 32-62: EPPI_LINE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Samples Per Line. The EPPI_LINE.VALUE holds the number of samples expected per line.

Clipping Register for ODD (Chroma) Data Register

The EPPI_ODDCLIP register selects the clipping threshold for chroma data, which provides clipping of individual video components.

The high odd and low odd spaces in EPPI_ODDCLIP are 16-bits wide and (depending on the EPPI_CTL.DLEN bit selection) only the corresponding video component bits are considered for clipping.

For example, if the EPPI is programmed in 10-bit mode, bits [9:0] and bits [25:16] constitute the clipping thresholds. The higher bits are (in this case) ignored.

Using the this method, 8-, 10-, 12- and 16-bit clipping thresholds can be set.

Note that when the EPPI is programmed in 16-, 20-, or 24-bit mode with the EPPI_CTL.SPLTWRD bit set, the luma data gets the clipping threshold levels of the EPPI_EVENCLIP register, and the chroma data gets the clipping threshold levels of the EPPI_ODDCLIP register.

Also, note that the EPPI_EVENCLIP and EPPI_ODDCLIP registers are ignored when the EPPI_CTL.RGBFMTEN bit is set.

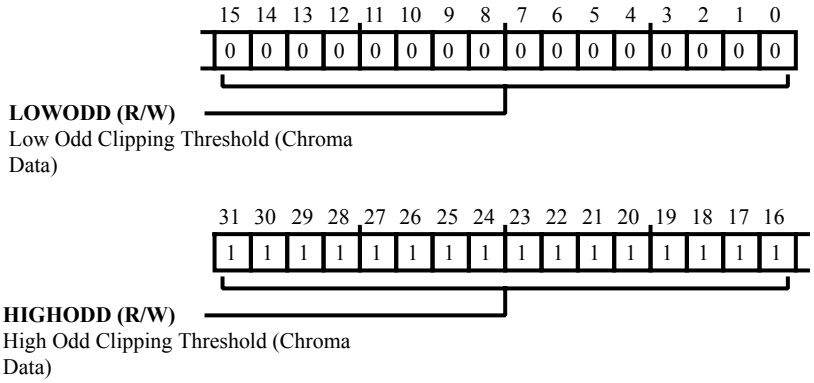

Figure 32-29: EPPI_ODDCLIP Register Diagram

Table 32-63: EPPI_ODDCLIP Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:16 (R/W)	HIGHODD	High Odd Clipping Threshold (Chroma Data). The EPPI_ODDCLIP.HIGHODD bit field selects the clipping threshold for luma data. The high odd spaces are 16-bits wide and (depending on the EPPI_CTL.DLEN selection) only the corresponding video component bits are considered for clipping.
15:0 (R/W)	LOWODD	Low Odd Clipping Threshold (Chroma Data). The EPPI_ODDCLIP.LOWODD bit field selects the clipping threshold for luma data. The low add spaces are 16-bits wide and (depending on the EPPI_CTL.DLEN selection) only the corresponding video component bits are considered for clipping.

Status Register

The EPPI_STAT register contains bits that provide information about the current operating state of the EPPI.

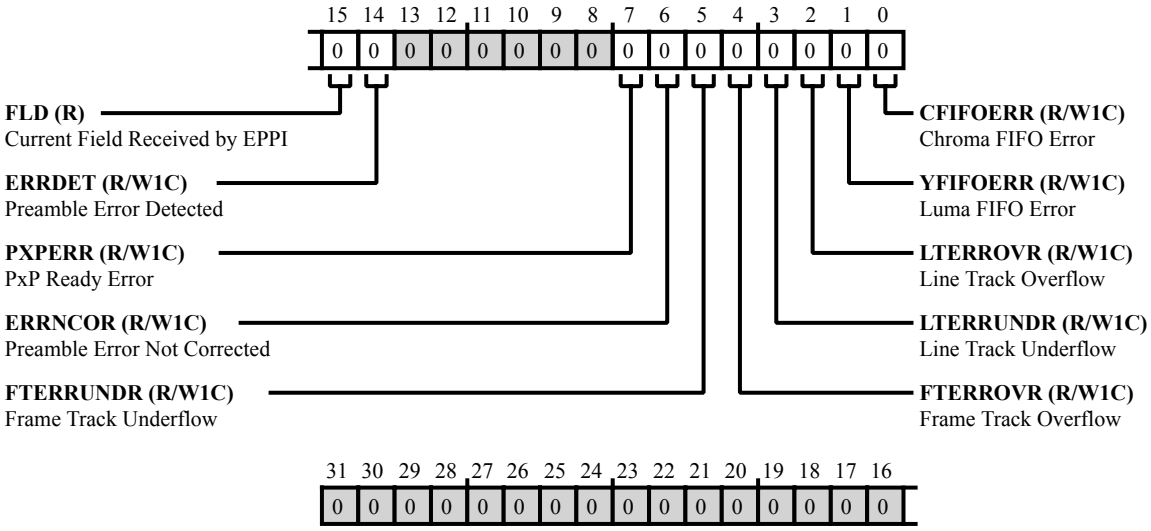

Figure 32-30: EPPI_STAT Register Diagram

Table 32-64: EPPI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	FLD	Current Field Received by EPPI. The EPPI_STAT.FLD bit indicates whether the current field being received by the PPI is field 1 (if clear) or field 2 (if set).	
		0	Field 1
		1	Field 2
14 (R/W1C)	ERRDET	Preamble Error Detected. The EPPI_STAT.ERRDET bit is useful only in ITU receive modes and indicates if an error has been detected in the status word of EAV or SAV sequences (if set) or not (if clear).	
		0	No Preamble Error Detected
		1	Preamble Error Detected
7 (R/W1C)	PXPERR	PxP Ready Error. The EPPI_STAT.PXPERR bit is valid only in the RX mode. This bit indicates whether the incoming PPI data overflows the PxP interface (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	

Table 32-64: EPPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
6 (R/W1C)	ERRNCOR	Preamble Error Not Corrected. The EPPI_STAT.ERRNCOR bit is useful only in the ITU receive modes and indicates if an error in the status word of EAV or SAV sequences can not be cleared (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No uncorrected preamble error has occurred
		1	Preamble error detected but not corrected
5 (R/W1C)	FTERRUNDR	Frame Track Underflow. The EPPI_STAT.FTERRUNDR bit indicates whether a frame track underflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
4 (R/W1C)	FTERROVR	Frame Track Overflow. The EPPI_STAT.FTERROVR bit indicates whether a frame track overflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
3 (R/W1C)	LTERRUNDR	Line Track Underflow. The EPPI_STAT.LTERRUNDR bit indicates whether a line track underflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
2 (R/W1C)	LTERROVR	Line Track Overflow. The EPPI_STAT.LTERROVR bit indicates whether a line track overflow error has occurred (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred

Table 32-64: EPPI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	YFIFOERR	Luma FIFO Error. For RX modes, the `EPPI_STAT.YFIFOERR` bit indicates whether the Luma FIFO has overflowed (if set) or not (if clear). For TX modes, this bit indicates whether the Luma FIFO has underflowed (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred
0 (R/W1C)	CFIFOERR	Chroma FIFO Error. For RX modes, the `EPPI_STAT.CFIFOERR` bit indicates whether the chroma FIFO has overflowed (if set) or not (if clear). For TX modes, this bit indicates whether the chroma FIFO has underflowed (if set) or not (if clear). This bit is sticky and must be cleared by software by writing 1 to it.	
		0	No Error Detected
		1	Error Occurred

Vertical Transfer Count Register

The EPPI_VCNT register holds the number of lines to read in or write out, after EPPI_VDLY number of lines from the start of frame. Any write to the EPPI_FRAME register modifies the EPPI_VCNT register. However, any write to the EPPI_VCNT register does not affect the EPPI_FRAME register value. So the EPPI_VCNT register should be programmed after the EPPI_FRAME register.

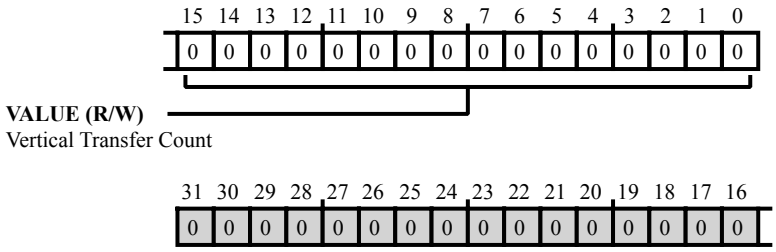

Figure 32-31: EPPI_VCNT Register Diagram

Table 32-65: EPPI_VCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Transfer Count. The EPPI_VCNT.VALUE holds the number of lines to read in or write out, after EPPI_VDLY number of lines from the start of frame.

Vertical Delay Count Register

The EPPI_VDLY register contains the number of lines to wait after the start of a new frame before starting to read/transmit data.

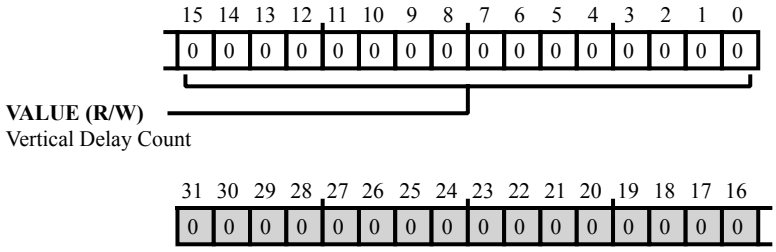

Figure 32-32: EPPI_VDLY Register Diagram

Table 32-66: EPPI_VDLY Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Vertical Delay Count. The EPPI_VDLY.VALUE holds the number of lines to wait after the start of a new frame before starting to read/transmit data.

33 Mobile Storage Interface (MSI)

The mobile storage interface (MSI) is a fast, synchronous controller that uses various protocols to communicate with MMC, SD, and SDIO cards. It addresses the growing storage need in embedded systems, handheld, and consumer electronics applications that require low power. The MSI is compatible with the following protocols.

- MMC (Multimedia Card) bus protocol
- SD (Secure Digital) bus protocol
- SDIO (Secure Digital Input Output) bus protocol

All of these storage solutions use similar interface protocols. The main difference between MMC and SD support is the initialization sequence. The main difference between SD and SDIO support is the use of interrupt and read wait signals for SDIO.

NOTE: The MSI does not support the SPI bus protocol.

MSI Features

The MSI includes the following features.

- Supports Secure Digital memory protocol (version 3.0)
- Supports Secure Digital I/O protocol (version 3.0)
- Supports Multimedia Card protocol (MMC version 4.41, eMMC version 4.5)
- Support for a single SD or SDIO card
- Support for a single MMC device (removable or embedded)
- Support for 1-bit and 4-bit SD modes (SPI mode is not supported)
- Support for 1-, 4-, and 8-bit MMC modes (SPI mode is not supported)
- Supports MMC boot operation
- Supports SDIO interrupts
- Supports Command Completion signal and interrupt to host processor

- Supports CRC generation and error detection
- 1024-byte transmit/receive FIFO
- Integrated DMA controller
- Card detection capabilities
- Programmable clock frequency
- Supports power management and clock control.

MSI Functional Description

This section provides information on the function of the MSI module.

MMC booting

Normal Boot operation is applicable to MMC4.3, MMC4.4, and MMC4.41 cards and is performed in push-pull mode. Also alternate Boot Operation; removable MMC4.3 card.

CRC generation and error detection

Cyclic redundancy codes (CRC) are used to protect commands, responses, and data transfers from transmission errors.

FIFO controller

Interfaces the internal FIFO to the host or DMA interface and the card controller unit. The FIFO depth is configured for 1024 bytes

Integrated DMA controller

Contains a single transmit or receive engine, which transfers data from host memory to the device port and conversely. The controller uses a descriptor to move data efficiently from source to destination with minimal core intervention

Power management and clock control

A low-power mode is available. A clock control block provides the clock frequencies required for SD/MMC cards

ADSP-BF70x MSI Register List

The Mobile Storage Interface (MSI) supports access to mobile memory and devices. A set of registers governs MSI operations. For more information on MSI functionality, see the MSI register descriptions.

Table 33-1: ADSP-BF70x MSI Register List

Name	Description
MSI_BLKSIZ	Block Size Register
MSI_BUFADDR	Current Buffer Descriptor Address Register
MSI_BUSMODE	Bus Mode Register
MSI_BYTCNT	Byte Count Register
MSI_CDETECT	Card Detect Register
MSI_CDTHRCTL	Card Threshold Control Register
MSI_CLKDIV	Clock Divider Register
MSI_CLKEN	Clock Enable Register
MSI_CMD	Command Register
MSI_CMDARG	Command Argument Register
MSI_CTL	Control Register
MSI_CTYPE	Card Type Register
MSI_DBADDR	Descriptor List Base Address Register
MSI_DEBNCE	Debounce Count Register
MSI_DSCADDR	Current Host Descriptor Address Register
MSI_ENSHIFT	Enable Phase Shift Register
MSI_FIFOTH	FIFO Threshold Watermark Register
MSI_IDINTEN	Internal DMA Interrupt Enable Register
MSI_IDSTS	Internal DMA Status Register
MSI_IMSK	Interrupt Mask Register
MSI_ISTAT	Raw Interrupt Status Register
MSI_MSKISTAT	Masked Interrupt Status Register
MSI_PLDMND	Poll Demand Register
MSI_RESP0	Response Register 0
MSI_RESP1	Response Register 1
MSI_RESP2	Response Register 2
MSI_RESP3	Response Register 3
MSI_STAT	Status Register
MSI_TBBCNT	Transferred Host to BIU-FIFO Byte Count Register
MSI_TCBCNT	Transferred CIU Card Byte Count Register
MSI_TMOUT	Timeout Register

Table 33-1: ADSP-BF70x MSI Register List (Continued)

Name	Description
MSI_UHS_EXT	Ultra High Speed Register Extension

ADSP-BF70x MSI Interrupt List

Table 33-2: ADSP-BF70x MSI Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
72	MSI0_STAT	MSI0 Status	Level	

ADSP-BF70x MSI Trigger List

Table 33-3: ADSP-BF70x MSI Trigger List Masters

Trigger ID	Name	Description	Sensitivity
47	MSI0_DONE	MSI0 Transfer Done	Level

Table 33-4: ADSP-BF70x MSI Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

MSI Block Diagram

The ***MSI Block Diagram*** shows the functional blocks within the MSI. These blocks are described in more detail in the MSI Architectural Concepts section.

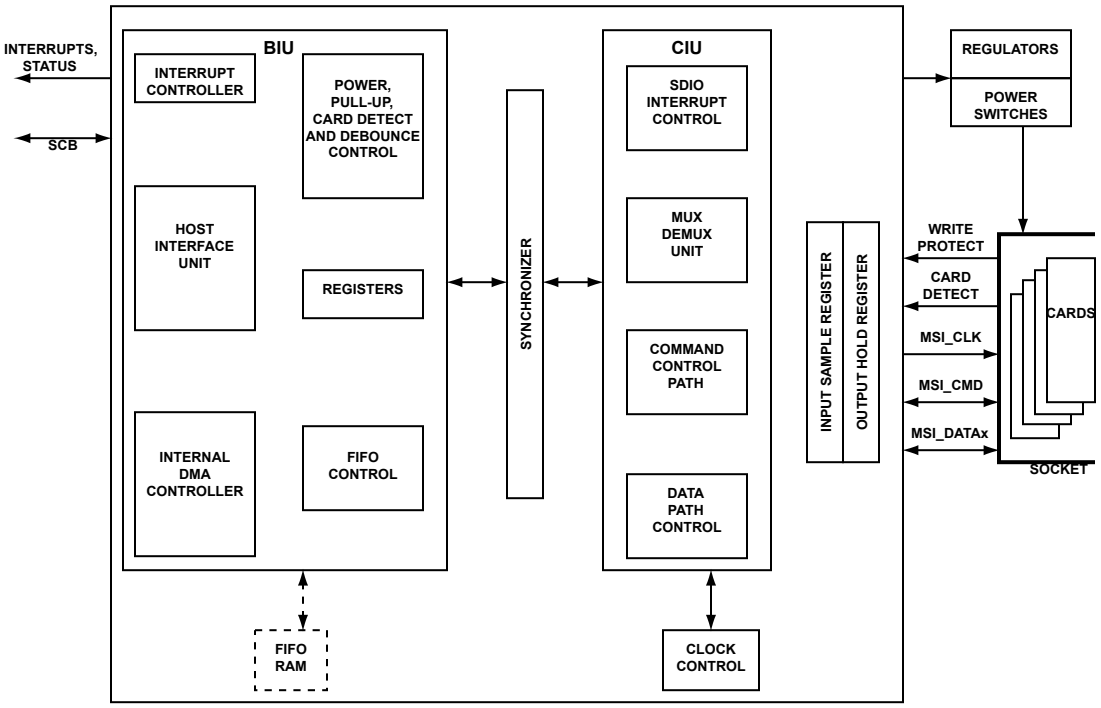

Figure 33-1: MSI Block Diagram

NOTE: The card-detect and write-protect signals are from the SD/MMC card socket and not from the SD/MMC card.

MSI Architectural Concepts

The following sections describe the functions and features of the MSI controller as well as the MMC, SD, and SDIO protocols.

Communication is through a master and slave configuration, where the MSI is the master device and the card is the slave device. The MSI communicates with the device using a message-based bus protocol in which the host sends commands serially using the `MSI_CMD` signal. Some commands require that the card provide a response back to the host. This response is also sent serially on the `MSI_CMD` signal.

Data transfers, both to and from the card, occur using the data signals. The number of data lines used for the data transfer can be configured to 1 (`MSI_D0`), 4 (`MSI_D3–MSI_D0`), or 8 (`MSI_D7–MSI_D0`). All `MSI_CMD` and `MSI_D7–MSI_D0` transfers are synchronous with `MSI_CLK`. Cyclic redundancy codes (CRC) are used to protect commands, responses, and data transfers from transmission errors. A CRC7 code is generated for every command sent by the host and for almost every response returned by the card on the `MSI_CMD` signal.

The MSI architecture can be described in terms of its submodules. The primary parts of the architecture are:

- Bus Interface Unit (BIU). This unit provides the host interface to the registers through the Host Interface Unit (HIU). Additionally, it provides independent data FIFO access through a DMA interface.

- Internal Direct Memory Access Controller (IDMAC). This unit is responsible for exchanging data between the FIFO and the host memory. A set of IDMAC registers is accessible to the host for controlling the IDMAC operation.
- Card Interface Unit (CIU). This unit controls the card-specific protocols. Within the CIU, the command path control unit and datapath control unit interface the controller to the command and data ports of the SD/MMC/SDIO cards. The CIU also provides clock control.

Bus Interface Unit (BIU)

The BIU provides the following functions:

- Host interface
- Interrupt control
- Register access
- FIFO access
- Power and pull-up control and card detection

Host Interface Unit (HIU)

The Host Interface Unit (HIU) is a slave bus interface, which provides the interface between the MSI and the processor system bus. It supports the burst accesses to the data FIFO address region only. The register address region is accessed through the standard accesses.

Interrupt Controller Unit

The interrupt controller unit generates an interrupt that depends on the controller interrupt status, the interrupt-mask register, and the global interrupt-enable register bit. Once an interrupt condition is detected, the software sets the corresponding interrupt bit in the interrupt status register. The interrupt status bit remains set until the software clears it by writing a 1 to the interrupt bit (a 0 leaves the bit unchanged).

NOTE: Before enabling the interrupt, programs must write 32'hFFFF_FFFF to the interrupt status register (`MSI_ISTAT`) in order to clear any pending unserviced interrupts. When clearing interrupts during normal operation, only clear the interrupt bits that are serviced.

Register Unit

The register unit is part of the bus interface unit (BIU). It provides read and write access to the registers.

All registers reside in the BIU clock domain. When a command is sent to a card by setting the `MSI_CMD.STARTCMD` bit, all relevant registers needed for the CIU operation are transferred to the CIU block. (The `MSI_CMD.STARTCMD` bit is bit[31] of the `MSI_CMD` register). During this time, software must not write to the registers that are transferred from the BIU to the CIU. The software must wait for the hardware to clear the start bit before writing to these registers again. The register unit has a hardware locking feature to prevent illegal writes to registers.

Once a command start is issued by setting the MSI_CMD.STARTCMD bit, the following registers cannot be reprogrammed until the card interface unit (CIU) accepts the command:

- MSI_CMD – Command
- MSI_CMDARG – Command Argument
- MSI_BYTCNT – Byte Count
- MSI_BLKSIZ – Block Size
- MSI_CLKDIV – Clock Divider
- MSI_CLKEN – Clock Enable
- MSI_TMOUT – Timeout
- MSI_CTYPE – Card Type

The hardware resets the MSI_CMD.STARTCMD bit once the CIU accepts the command. If a host writes to any of these registers during this locked time, then the write is ignored and the hardware lock error bit is set in the MSI_ISTAT (status) register. Additionally, if the interrupt is enabled and not masked for a hardware lock error, then an interrupt is sent to the host.

Once a command is accepted, software can send another command to the CIU-which has a one-deep command queue-under the following conditions:

- If the previous command was not a data transfer command, the new command is sent to the card once the previous command completes.
- If the previous command is a data transfer command and if the MSI_CMD.WTPRIVDATA bit is set for the new command, the new command is sent to the card only when the data transfer completes.
- If the MSI_CMD.WTPRIVDATA is 0, then the new command is sent to the card as soon as the previous command is sent. Typically, software uses this option only to stop or abort a previous data transfer or query the card status in the middle of a data transfer.

FIFO Controller Unit

The FIFO controller interfaces the internal FIFO to the host or DMA interface and the card controller unit. The FIFO depth is configured for 1024 bytes. A single shared FIFO is used for read and write operations because read and write transfers to the cards do not occur simultaneously.

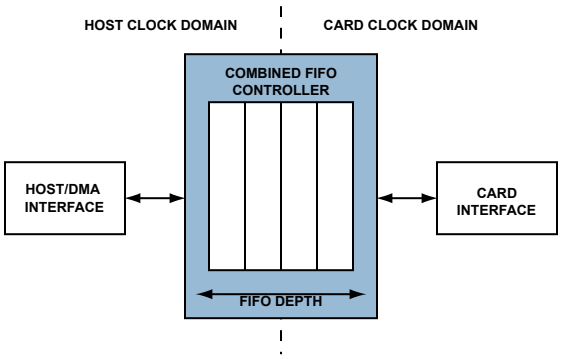

Figure 33-2: Combined Transmit and Receive FIFO

Power and Pull-up Control and Card Detection Unit

The card detection unit looks for any changes in the card-detect signal for card insertion or card removal. The unit filters out the debounce associated with mechanical insertion or removal, and generates one interrupt to the host. The debounce filter value is programmed through the `MSI_DEBNCE` register.

On power-on, the controller reads in the `MSI_CDETECT` register and stores the value in the memory. Upon receiving a card-detect interrupt, the controller again reads the `MSI_CDETECT` register to decide whether card was removed or inserted. The ***Card-Detect Signals*** figure shows the timing for the card-detect signal.

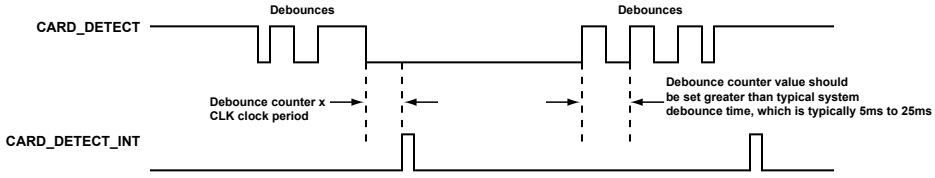

Figure 33-3: Card-Detect Signals

Internal Direct Memory Access Controller (IDMAC)

The Internal Direct Memory Access Controller (IDMAC) contains a single transmit or receive engine, which transfers data from the host memory to the device port and conversely. The controller uses a descriptor to move data efficiently from source to destination with minimal core intervention. Programs can configure the controller to interrupt the core in situations such as data transmit and receive transfer completion from the card, as well as other normal or error conditions.

The IDMAC and the core communicate through a single data structure. The IDMAC transfers the data received from the card to the data buffer in the host memory, and it transfers transmit data from the data buffer in the host memory to the MSI FIFO. Descriptors that reside in the memory act as pointers to these buffers.

A data buffer resides in physical memory space of the processor and consists of complete data or partial data. Buffers contain only data, while buffer status is maintained in the descriptor. Data chaining refers to data that spans multiple data buffers. However, a single descriptor cannot span multiple data.

A single descriptor is used for both reception and transmission. The base address of the list is written into descriptor list base address register (MSI_DBADDR). A descriptor list is forward-linked. The last descriptor can point back to the first entry creating a ring structure. The descriptor list resides in the physical memory address space of the host. Each descriptor can point to a maximum of two data buffers.

Programs can enable or disable the IDMAC using the MSI_CTL.INTDMAC bit of the BIU.

DMA Descriptors

The IDMAC uses two types of descriptor structures:

- Dual-Buffer Structure – The skip length value programmed in the MSI_BUSMODE.DSL bit field determines the distance between two descriptors.
- Chain Structure – Each descriptor points to a unique buffer and the next descriptor.

The ***Dual-Buffer Descriptor Structure*** and ***Chain Descriptor Structure*** figures show these descriptor structures.

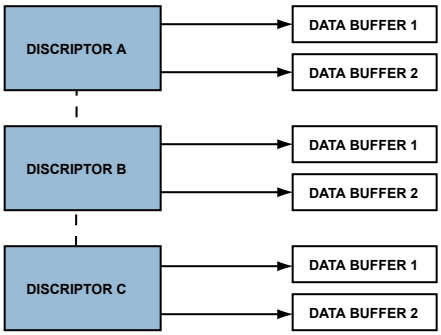

Figure 33-4: Dual-Buffer Descriptor Structure

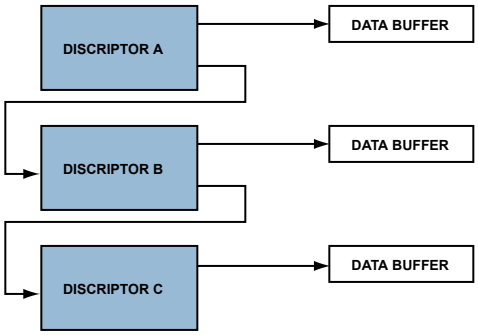

Figure 33-5: Chain Descriptor Structure

The ***Descriptor Format*** figure illustrates the internal format of a descriptor. The descriptor addresses must align with the 32-bit bus width. Each descriptor contains 16 bytes of control and status information. DES0 is a notation used to denote the [31:0] bits, DES1 to denote [63:32] bits, DES2 to denote [95:64] bits, and DES3 to denote [127:96] bits in a descriptor.

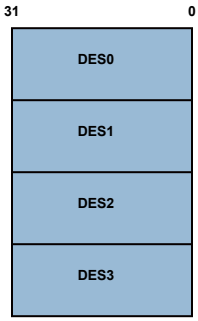

Figure 33-6: Descriptor Format

The following tables provide descriptor bit descriptions. Bits not shown are reserved. The DES0 descriptor in the IDMAC (described in the ***IDMAC DES0 Descriptor*** table) contains control and status information.

Table 33-5: IDMAC DES0 Descriptor

Bits	Name	Description
31	OWN	When set, this bit indicates that the IDMAC owns the descriptor. When this bit is reset, it indicates that the host owns the descriptor. The IDMAC clears this bit when it completes the data transfer.
30	Card Error Summary (CES)	This error bit indicates the status of the transaction to or from the card. This bit is also present in the `MSI_ISTAT` register and indicates the logical OR of the following bits: EBE: End Bit Error RTO: Response timeout RCRC: Response CRC SBE: Start Bit Error DRTO: Data Read timeout DCRC: Data CRC for receive RE: Response error
5	End of Ring (ER)	When set, this bit indicates that the descriptor list reached its final descriptor. The IDMAC returns to the base address of the list, creating a descriptor ring. This feature is meaningful for only a dual-buffer descriptor structure.
4	Second Address Chained (CH)	When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When this bit is set, BS2 (DES1[25:13]) is all zeros.
3	First Descriptor (FS)	When set, this bit indicates that this descriptor contains the first buffer of the data. If the size of the first buffer is 0, next descriptor contains the beginning of the data.
2	Last Descriptor (LD)	This bit is associated with the last block of a DMA transfer. When set, the bit indicates that the buffers pointed to by this descriptor are the last buffers of the data. After this descriptor is completed, the remaining byte count is 0. In other words, after the descriptor with the LD bit set is completed, the remaining byte count is 0.

Table 33-5: IDMAC DES0 Descriptor (Continued)

Bits	Name	Description
1	Disable Interrupt on Completion (DIC)	When set, this bit prevents the setting of the TI/RI bit of the IDMAC Status Register (IDSTS) for the data that ends in the buffer pointed to by this descriptor.

The DES1 descriptor (described in the ***IDMAC DES1 Descriptor*** table) contains the buffer size.

Table 33-6: IDMAC DES1 Descriptor

Bits	Name	Description
25:13	Buffer 2 Size (BS2)	These bits indicate the second data buffer byte size which must be a multiple of 4. In the case where the buffer size is not a multiple of 4, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and proceeds to the next buffer if there is a dual-buffer structure. This field is not valid for chain structure; that is, if DES0[4] is set.
12:0	Buffer 1 Size (BS1)	Indicates the data buffer byte size which must be a multiple of 4. In the case where the buffer size is not a multiple of 4, the resulting behavior is undefined. This field must not be zero. Note: If there is only one buffer to be programmed, use only the buffer 1, and not buffer 2.

The DES2 descriptor (described in the ***IDMAC DES2 Descriptor*** table) is the address pointer to the data buffer.

Table 33-7: IDMAC DES2 Descriptor

Bits	Name	Description
31:0	Buffer Address Pointer 1 (BAP1)	These bits indicate the physical address of the first data buffer. The IDMAC ignores DES2 [1:0], internally.

The DES3 descriptor (described in the ***IDMAC DES3 Descriptor*** table) is the address pointer to the next descriptor when the present descriptor is not

- the last descriptor in a chained descriptor structure, or
- the second buffer address for a dual-buffer structure

Table 33-8: IDMAC DES3 Descriptor

Bits	Name	Description
31:0	Buffer Address Pointer 2/ Next Descriptor Address (BAP2)	These bits indicate the physical address of the second buffer when the dual-buffer structure is used. If the Second Address Chained (DES0[4]) bit is set, then this address contains the pointer to the physical memory where the next descriptor is present. If this pointer is not the last descriptor, then the next descriptor address pointer must be bus-width aligned (DES3[1:0] = 0. Internally, the LSBs are ignored.

Initialization

IDMAC initialization occurs as follows.

1. Write to IDMAC bus mode register (`MSI_BUSMODE`) to set host bus access parameters.

2. Write to IDMAC interrupt enable register (MSI_IDINTEN) to mask unnecessary interrupt causes.
3. The software driver creates either the transmit or the receive descriptor list. Then it writes to IDMAC descriptor list base address register (MSI_DBADDR), which provides the IDMAC the starting address of the list.
4. The IDMAC engine attempts to acquire descriptors from the descriptor lists.

Host Bus Burst Access

The IDMAC executes fixed-length burst transfers on the bus interface when the MSI_BUSMODE.FB bit =1. The maximum burst length is indicated and limited by the MSI_BUSMODE.PBL bit field. The descriptors are always accessed in the maximum burst-size for the 16-bytes to be read (4-word burst).

The IDMAC initiates a data transfer only when sufficient space to accommodate the configured burst is available in the FIFO or the number of bytes to the end of data, when less than the configured burst-length. The IDMAC indicates the start address and the number of transfers required to the bus interface. When the bus interface is configured for fixed-length bursts, it transfers data using the best combination of INCR4/8/16 and SINGLE transactions. Otherwise, in no fixed-length bursts, it transfers data using INCR (undefined length) and SINGLE transactions.

The transmit and receive data buffers must be aligned to data bus width (32-bit).

NOTE: Due to the bus protocol limitation, the burst mode address should not cross the 1 KB boundary. Address [31:10] should not change during a burst.

Buffer Size Calculations

The software knows the amount of data to transmit or receive. For transmitting to the card, the IDMAC transfers the exact number of bytes to the FIFO, indicated by the buffer size field of DES1.

If a descriptor is not marked as last (LS bit of DES0), then the corresponding buffers of the descriptor are full. Its buffer size field indicates the amount of valid data in a buffer. If a descriptor is marked as last, then the buffer cannot be full, as indicated by the buffer size in DES1. The software is aware of the number of locations that are valid in this case.

Data Transmit/Receive

IDMAC transmission occurs as follows:

1. The software sets up the Descriptor (DES0–DES3) for transmission and sets the OWN bit (DES0[31]). The software also prepares the data buffer.
2. The software programs the write data command in the MSI_CMD register in the BIU.
3. The software also programs the required transmit threshold level (MSI_FIFOTH.TXWM bit field).
4. The IDMAC determines that a write data transfer must occur as a consequence of step 2.
5. The IDMAC engine fetches the descriptor and checks the OWN bit. If the OWN bit is not set, it means that the software owns the descriptor. In this case, the IDMAC enters the suspend state and asserts the MSI_IDSTS.DU bit. The software must release the IDMAC by writing any value to the poll demand register.

6. It then waits for command done (MSI_ISTAT.CMDDONE) bit and no errors from BIU which indicates that a transfer can occur.
7. The IDMAC engine waits for a DMA interface request from the BIU. This request is generated based on the programmed transmit threshold value. For the last bytes of data which cannot be accessed using a burst, SINGLE transfers are performed.
8. The IDMAC fetches the transmit data from the data buffer and transfers it to the FIFO for transmission to the card.
9. When data spans across multiple descriptors, the IDMAC fetches the next descriptor and continues with its operation with the next descriptor. The last descriptor bit in the descriptor indicates whether the data spans multiple descriptors or not.
10. When data transmission is complete, status information is updated in IDMAC status register (MSI_IDSTS) by setting the transmit interrupt, when enabled. Also, the IDMAC performs a write transaction to DES0 to clear the OWN bit.

IDMAC reception occurs as follows:

1. The software sets up the Descriptor (DES0–DES3) for reception, sets the OWN bit (DES0[31]).
2. The software programs the read data command in the MSI_CMD register in the BIU.
3. The software programs the required receive threshold level (MSI_FIFOTH.RXWM bit field).
4. The IDMAC determines that a read data transfer must occur as a consequence of step 2.
5. The IDMAC engine fetches the descriptor and checks the OWN bit. If the OWN bit is not set, it means that the host owns the descriptor. In this case, the DMA enters suspend state and asserts the release the IDMAC by writing any value to the poll demand register.
6. It then waits for the command done (MSI_ISTAT.CMDDONE) bit and no errors from BIU which indicates that a transfer can occur.
7. The IDMAC engine waits for a DMA interface request from the BIU. This request is generated based on the programmed receive threshold value. For the last bytes of data which cannot be accessed using a burst, SINGLE transfers are performed.
8. The IDMAC fetches the data from the FIFO and transfers to the memory.
9. When data spans across multiple descriptors, the IDMAC fetches the next descriptor and continues with its operation with the next descriptor. The last descriptor bit in the descriptor indicates whether the data spans multiple descriptors or not.
10. When data reception is complete, status information is updated in the IDMAC status register (MSI_IDSTS) by setting the receive interrupt, when enabled. Also, the IDMAC performs a write transaction to DES0 to clear the OWN bit.

Interrupts

Interrupts can be generated as a result of various DMAC events. The IDMAC status register (MSI_IDSTS) contains all the bits that can cause an interrupt. The IDMAC interrupt enable register (MSI_IDINTEN) contains an enable bit for each of the events that can cause an interrupt.

There are two groups of summary interrupts—normal and abnormal—as outlined in the MSI_IDSTS register. Interrupts are cleared by writing a 1 to the corresponding bit position. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared.

Interrupts are not queued and if the interrupt event occurs before the software has responded to it, no additional interrupts are generated. For example, the receive interrupt (MSI_IDSTS.RI) indicates that one or more data was transferred to the buffer. An interrupt is generated only once for simultaneous, multiple events. The software must scan the MSI_IDSTS register for the interrupt cause.

NOTE: The final interrupt signal from MSI is a logical OR of the interrupt from the BIU and the IDMAC.

Finite State Machine (FSM)

The IDMAC finite state machine can be in any one of the states reflected in the MSI_IDSTS.FSM bit field.

The FSM uses the following sequence.

1. IDMAC performs four accesses for fetching the descriptor.
2. Stores descriptors in internal register and also issues a FIFO reset when it is first descriptor.
3. Each bit is checked for correctness. In case of bit mismatches, appropriate error bit is set. If it is first descriptor, then issues a FIFO reset and wait until FIFO reset is complete. The error status indicates one of the following:
    - Response timeout
    - Response CRC error
    - Data receive timeout
    - Response error
4. The FSM waits in current state until DMA request is asserted, which implies that, FIFO:
    - For DMA write request wait – Holds the number of data, indicated by FIFO RX watermark. In case of error due to response timeout or error, FSM goes to DESC_CLOSE state to close descriptor.
    - For DMA read request wait – Holds number of data, indicated by FIFO TX watermark. In case of error, FSM goes to DESC_CLOSE state to close descriptor.
5. FSM performs the following:
    - For DMA write – Requests a write to SCB. If number of beats in one transfer is greater than PBL, then one of the following occurs:

      Burst count to SCB is PBL value

Single transfers are initiated

- For DMA read – Requests a read from SCB. If number of beats in one transfer is greater than PBL, then one of the following occurs:

  Burst count to SCB is PBL value

  Single transfers are initiated

6. After the programmed transfer count is accessed from the memory, the OWN bit in descriptor is closed. If a transfer spans more than one descriptor, the FSM fetches the next descriptor. If the transfer ends with the current descriptor, the FSM goes to the idle state after setting the receive or transmit interrupt. Depending on the descriptor structure—dual buffer or chained—the appropriate starting address of descriptor is loaded. If it is the second data buffer of the dual buffer, the descriptor is not fetched again.

Abort Operation

The host issues CMD12 when a data transfer on the card data lines is in progress. The FSM closes the present descriptor after completing the transfer of data until a DTO interrupt is asserted. Once an abort command is issued, the DMAC performs single burst transfers.

1. For a card write, the FSM keeps pushing data to the FIFO after fetching it from the memory until a DTO interrupt is asserted. The controller asserts a busy clear interrupt after the DTO interrupt. It ensures that the card has completed driving the busy signal. This sequence occurs to keep the card clock running so that CMD12 is reliably sent to the card.
2. For a card read, the FSM keeps popping data from the FIFO and writes to the memory until a DTO interrupt is generated. This sequence is required since the DTO interrupt is not generated until and unless all the FIFO data is emptied.

NOTE: If an FBE occurs, the IDMAC FSM does not close the current descriptor and the remaining unread descriptors. If a write abort occurs, the IDMAC FSM closes only the current descriptor during which the abort occurred. The IDMAC FSM does not close the current descriptor and the remaining unread descriptors. If a read abort occurs, the IDMAC FSM pops the data out of the FIFO and writes it to the corresponding descriptor data buffers. The remaining unread descriptors are not closed.

FIFO Overflow and Underflow

During normal data transfer conditions, FIFO overflow and underflow do not occur. If there is a programming error, then a FIFO overflow or underflow can result as shown in the following examples.

Transmit settings: `MSI_BUSMODE.PBL` =4, `MSI_FIFOTH.TXWM` =1

In this example, if the FIFO has only one location empty, it issues a DMA request to the IDMAC FSM. Because `MSI_BUSMODE.PBL` =4, the IDMAC FSM performs 4 pushes into the FIFO which results in a FIFO overflow interrupt.

Receive settings: `MSI_BUSMODE.PBL` =4, `MSI_FIFOTH.RXWM` =1

In this example, if the FIFO has only one location filled, it issues a DMA request to the IDMAC FSM. Because MSI_BUSMODE.PBL =4, the IDMAC FSM performs 4 pops to the FIFO which results in a FIFO underflow interrupt.

The software must ensure that the number of bytes to be transferred as indicated in the descriptor are a multiple of 4. For example, if the MSI_BYTCNT register =13, the number of bytes indicated in the descriptor must be 16.

Card Interface Unit

The Card Interface Unit (CIU) interfaces with the Bus Interface Unit (BIU) on one side and with the SD/MMC/SDIO cards or devices on other side. The software writes command parameters to the MSI BIU control registers, and these parameters are then passed to the CIU. Depending on control register values, the CIU generates card command and data traffic on a selected card bus according to card protocol.

As shown in the MSI block diagram, the CIU consists of the following primary functional blocks:

- Command path
- Datapath
- SDIO interrupt control
- Clock control
- Multiplex or demultiplex unit

The following software restrictions must be met for proper CIU operation:

- Only one data transfer command can be issued at a time.
- During an open-ended card write operation, if the card clock is stopped because the FIFO is empty, the software must first fill the data into the FIFO and start the card clock. It can then issue only a stop or abort command to the card.
- During an SDIO card transfer, if the card function is suspended and the software wants to resume the suspended transfer, it must first reset the FIFO and start the resume command as if it were a new data transfer command.
- When issuing card reset commands (CMD0, CMD15 or CMD52) while a card data transfer is in progress, the software must set the MSI_CMD.STPABORTCMD bit so that the MSI can stop the data transfer after issuing the card reset command.
- When the data end error bit (MSI_ISTAT.EBE) is set, the MSI does not guarantee SDIO interrupts. The software must ignore the SDIO interrupts and issue the stop or abort command to the card, so that the card stops sending the read data.
- If the card clock is stopped because the FIFO is full during a card read, the software or DMA must read at least two FIFO locations to start the card clock.

Command Path

The command path performs the following functions.

- Loads clock parameters
- Loads card command parameters
- Sends commands to card bus
- Receives responses from card bus
- Sends responses to BIU
- Drives the P-bit on command line

A new command is issued to the MSI by programming the BIU registers and setting the MSI_CMD.STARTCMD bit. The command path loads this new command (command, command argument, timeout) and sends acknowledge to the BIU.

Once the new command is loaded, the command path state machine sends a command to the SD/MMC bus (this includes the internally generated CRC7). The command path state machine receives a response if there is any. The state machine then sends the received response and signals to the BIU that the command is done, and then waits for eight clock cycles before loading a new command.

Load Command Parameters

One of the following commands or responses is loaded in the command path:

- New command from BIU when the MSI_CMD.STARTCMD bit is set.
- Internally generated auto-stop command when the data path ends, the stop command request is loaded.
- IRQ response with RCA =0x000 when the command path is waiting for an IRQ response from the MMC card, then the MSI_CTL.IRQRESP bit is set.

Loading a new command from the BIU in the command path depends on the following MSI_CMD register bit settings:

- Update clock registers only using the MSI_CMD.UCLKREGS bit. If the MSI_CMD.UCLKREGS bit =1, the command path updates only the clock enable and clock divider registers. If MSI_CMD.UCLKREGS =0, the command path loads the command, command argument, and timeout registers, then starts processing the new command.
- Wait for previous data to complete using the MSI_CMD.WTPRIVDATA bit. If the MSI_CMD.WTPRIVDATA bit =1, the command path loads the new command under one of the following conditions:
  - Immediately, if the data path is free (that is, there is no data transfer in progress), or if an open-ended data transfer is in progress (MSI_BYTCNT =0).
  - After completion of the current data transfer, if a predefined data transfer is in progress.

Send Command and Receive Response

Once a new command is loaded in the command path with MSI_CMD.UCLKREGS bit =0, the command path state machine sends out a command on the SD/MMC bus. The ***SD_MMC Command Path State Machine*** figure illustrates the command path state machine.

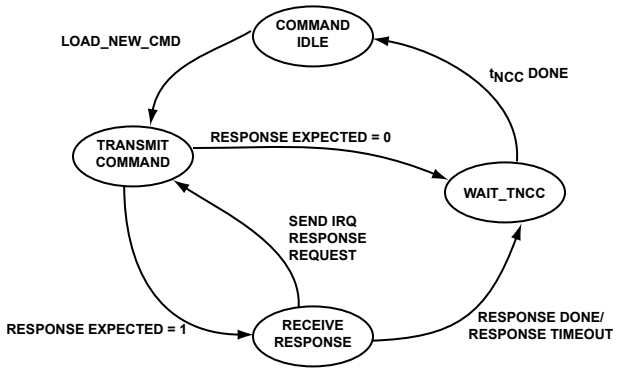

Figure 33-7: SD_MMC Command Path State Machine

The command path state machine performs the following functions, according to following command register bit values:

1. MSI_CMD.SENDINIT (send initialization). Initialization sequence of 80 clocks is sent before sending the command.
2. MSI_CMD.RXPECT (response expected). A response is expected for the command. After the command is sent out, the command path state machine receives a 48-bit or 136-bit response and sends it to the BIU. If the start bit of the card response is not received within the number of clocks programmed in the timeout register, then the MSI_ISTAT.RTO(response timeout) and MSI_ISTAT.CD (command done) bits are set as a signal to the BIU. If the MSI_CMD.RXPECT bit is not set, the command path sends out a command and signals a response done to the BIU, that is, the MSI_ISTAT.CD bit is set.
3. MSI_CMD.RLEN (response length). If =1, a 136-bit response is received; if =0, a 48-bit response is received.
4. MSI_CMD.CHKRESPCRC (check response CRC). If =1, the command path compares CRC7 received in the response with the internally-generated CRC7. If the two do not match, the response CRC error is signaled to the BIU, that is, the response CRC error bit (MSI_ISTAT.RCRC) is set.

Send Response to BIU

If the MSI_CMD.RXPECT bit =1, the received response is sent to the BIU. The MSI_RESP0 register is updated for a short response, and the MSI_RESP3, MSI_RESP2, MSI_RESP1, and MSI_RESP0 registers are updated on a long response, after which the MSI_ISTAT.CMDDONE bit is set. If the response is for an AUTO_STOP command sent by the CIU, the response is saved in the MSI_RESP1 register, after which the MSI_ISTAT.ACD (auto command done) bit is set.

Additionally, the command path checks for the following.

- Transmission bit =0
- Command index in response matches command index of the sent command
- End bit =1 in received card response

The command index is not checked for a 136-bit response or if the MSI_CMD.CHKRESPCRC bit is cleared. For a 136-bit response and reserved CRC 48-bit responses, the command index is reserved (=111111).

Driving a P-bit on CMD Line

The command path drives a P-bit =1 on the CMD line between two commands when a response is not expected. If a response is expected, the P-bit is driven after the response is received and before the start of the next command.

Datapath

The datapath block pops the data FIFO and transmits data on MSI data lines during a write data transfer. Or, it receives data from data lines and pushes it into the FIFO during a read data transfer. Whenever a data transfer command is not in-progress, the datapath loads new data parameters;

- data expected
- read/write data transfer
- stream or block transfer
- block size
- byte count
- card type
- timeout registers

If the MSI_CMD.DXPECT bit =1, the new command is a data transfer command and the datapath starts one of the following:

- Data transmit if the MSI_CMD.RDWR (read/write) bit =1
- Data receive if MSI_CMD.RDWR (read/write) bit 0

Auto-Stop

The MSI internally generates a stop command and is loaded in the command path when the MSI_CTL.AUTOSTOP bit is set. The auto-stop command helps to send an exact number of data bytes using a stream read or write for the MMC, and a multiple-block read or write for the SD memory transfer for SD cards.

The software must set the MSI_CTL.AUTOSTOP bit according to details listed in the ***Auto-Stop Generation*** table.

Table 33-9: Auto-Stop Generation

Card Type	Transfer Type	Byte Count	MSI_CTL. AUTOSTOP bit =1	Comments
MMC	Stream read	0	No	Open-ended stream
	Stream read	>0	Yes	Auto-stop after all bytes transfer
	Stream write	0	No	Open-ended stream
	Stream write	>0	Yes	Auto-stop after all bytes transfer
	Single-block read	>0	No	Byte count = 0 is illegal
	Single-block write	>0	No	Byte count = 0 is illegal
	Multiple-block read	0	No	Open-ended Multiple block
	Multiple-block read	>0	Yes*1	Pre-defined Multiple block
	Multiple-block write	0	No	Open-ended Multiple block
	Multiple-block write	>0	Yes	Pre-defined Multiple block
SDMEM	Single-block read	>0	No	Byte count = 0 is illegal
	Single-block write	>0	No	Byte count = 0 is illegal
	Multiple-block read	0	No	Open-ended Multiple block
	Multiple-block read	>0	Yes	Auto-stop after all bytes transfer
	Multiple-block write	0	No	Open-ended Multiple block
	Multiple-block write	>0	Yes	Auto-stop after all bytes transfer
SDIO	Single-block read	>0	No	Byte count = 0 is illegal
	Single-block write	>0	No	Byte count = 0 is illegal
	Multiple-block read	0	No	Open-ended Multiple block
	Multiple-block read	>0	No	Pre-defined Multiple block
	Multiple-block write	0	No	Open-ended Multiple block
	Multiple-block write	>0	No	Pre-defined Multiple block

*1 The condition under which the transfer mode blocks transfer and `byte_count` equals block size is treated as a single-block data transfer command for both MMC and SD cards. If byte_count = n × block_size (n = 2, 3, …), the condition is treated as a predefined multiple-block data transfer command. For an MMC card, the host software can perform a predefined data transfer in two ways:

1. Issue the CMD23 command before issuing CMD18/CMD25 commands to the card – in this case, issue CMD18/CMD25 commands without setting the `MSI_CTL.AUTOSTOP` bit.
2. Issue CMD18/CMD25 commands without issuing CMD23 command to the card, with the `MSI_CTL.AUTOSTOP` bit set. In this case, an internally generated auto-stop command after the programmed byte count terminates the multiple-block data transfer.

The following list explains different conditions for the auto-stop command.

- Stream read for MMC card with byte count greater than 0. The MSI generates an internal stop command and loads it into the command path. The end bit of the stop command is sent out when the last byte of data is read from the card and no extra data byte is received. If the byte count is less than 6 (48 bits), a few extra data bytes are received from the card before the end bit of the stop command is sent.
- Stream write for MMC card with byte count greater than 0. The MSI generates an internal stop command and loads it into the command path. The end bit of the stop command is sent when the last byte of data is transmitted on the card bus and no extra data byte is transmitted. If the byte count is less than 6 (48 bits), the datapath transmits the data last in order to meet the above condition.
- Multiple-block read memory for SD card with byte count greater than 0. If the block size is less than 4 (single-bit data bus), 16 (4-bit data bus), or 32 (8-bit data bus) bytes, the auto-stop command is loaded in the command path after all the bytes are read. Otherwise, the stop command is loaded in the command path so that the end bit of the stop command is sent after the last data block is received.
- Multiple-block write memory for SD card with byte count greater than 0. If the block size is less than 3 (single-bit data bus), 12 (4-bit data bus), or 24 (8-bit data bus), the auto-stop command is loaded in the command path after all data blocks are transmitted. Otherwise, the stop command is loaded in the command path so that the end bit of the stop command is sent after the end bit of the CRC status is received.
- Precaution for host software during auto-stop. Whenever an auto-stop command is issued, the host software must not issue a new command to the MSI until the MSI sends the auto-stop command and the data transfer completes. If the host issues a new command during a data transfer with the auto-stop in progress, an auto-stop command can be sent after the new command is sent and its response is received. This process can delay sending the stop command, which transfers extra data bytes. For a stream write, extra data bytes are erroneous data that can corrupt the card data. If the host wants to terminate the data transfer before the data transfer is complete, it can issue a stop or abort command. In this case, the MSI does not generate an auto-stop command.

Non-Data Transfer Commands that Use Datapath

Some non-data transfer commands (non-read/write commands) also use the datapath. The ***Non-Data Transfer Commands and Requirements*** table lists the commands and register programming requirements.

Table 33-10: Non-Data Transfer Commands and Requirements

	CMD27	CMD30	CMD42	ACMD13	ACMD22	ACMD51
Command Register Programming						
Cmd_index	6'h1B	6'h1E	6'h2A	6'h0D	6'h16	6'h33
Response_expect	1	1	1	1	1	1
Response_length	0	0	0	0	0	0
Check_response_crc	1	1	1	1	1	1
Data_expected	1	1	1	1	1	1
Read/write	1	0	1	0	0	0

Table 33-10: Non-Data Transfer Commands and Requirements (Continued)

	CMD27	CMD30	CMD42	ACMD13	ACMD22	ACMD51
Transfer_mode	0	0	0	0	0	0
Send_auto_stop	0	0	0	0	0	0
Wait_prevdata_complete	0	0	0	0	0	0
Stop_abort_cmd	0	0	0	0	0	0
Command Argument register programming						
	Stuff bits	32-bit write protect data address	Stuff bits	Stuff bits	Stuff bits	Stuff bits
Block Size register programming						
	16	4	Num_bytes*1	64	4	8
Byte Count register programming						
	16	4	Num_bytes*1	64	4	8

*1 Number of bytes specified as per the lock card data structure (Refer to the SD specification and the MMC specification).

SDIO Interrupt Control

Interrupts for SD cards are reported to the BIU by asserting an interrupt signal for two clock cycles. SDIO cards signal an interrupt by asserting `MSI0_D1` low during the interrupt period; The interrupt control state machine determines an interrupt period for the selected card. An interrupt period is always valid for cards in 1-bit data mode. An interrupt period for a wide-bus active or selected card is valid for the following conditions:

- Card is idle
- Non-data transfer command in progress
- Third clock after end bit of data block between two data blocks
- From two clocks after end bit of last data until end bit of next data transfer command

Bear in mind that, in the following situations, the MSI does not sample the SDIO interrupt of the selected card when the card data width is 4 bits. Since the SDIO interrupt is level-triggered, it is sampled in a further interrupt period and the host does not lose any SDIO interrupt from the card.

1. Read/write resume. The CIU treats the resume command as a normal data transfer command. SDIO interrupts during the resume command are handled similarly to other data commands. According to the SDIO specification, for the normal data command the interrupt period ends after the command end bit of the data command. For the resume command, it ends after the response end bit. For the resume command, the MSI stops the interrupt sampling period after the resume command end bit, instead of stopping after the response end bit of the resume command.
2. Suspend during read transfer. If the host suspends the read data transfer, the host sets the `MSI_CTL.RDABORT` bit to reset the data state machine. In the CIU, the SDIO interrupts are handled such that the interrupt

sampling starts after the host sets the MSI_CTL.RDABORT bit. In this case, the MSI does not sample SDIO interrupts between the period from response of the suspend command.

Clock Control

The clock control block provides the clock frequencies required for SD/MMC cards.

The source clock for the MSI block and the input clock for clock dividers of the clock control block can be selected from one of four options shown in the ***MSI Clock Sources*** table. The MSI_UHS_EXT register is used to control the selection of source clock to MSI module.

Table 33-11: MSI Clock Sources

UHS_REG_EXT[31:30]	MSI Clock
00	SCLK0/2
01	SCLK0
10	SCLK1/3
11	GND

The clock muxing increases the MSI flexibility.

The source clock is used to generate the card clock frequencies. The card clock can have different clock frequencies, since the SD card can be a low-speed SD card or a full-speed SD card. The MSI allows the card to operate at different clock frequencies.

The clock frequency of a card depends on the following clock control registers:

- Clock divider register (MSI_CLKDIV). An internal clock divider is used to generate different clock frequencies required for the card. The clock divider is an 8-bit value that provides a clock division factor from 1 to 510. A value of 0 represents a clock-divider bypass, a value of 1 represents a divide by 2, a value of 2 represents a divide by 4, and so on.
- Clock control register (MSI_CLKEN). The MSI module can enable or disable the card clock under the following conditions:
  - The clock for a card is enabled when the MSI_CLKEN.EN0 bit for a card is programmed (= 1) or disabled (= 0).
  - Setting the MSI_CLKEN.LP0 =1. enables the low-power mode of a card. If low-power mode is enabled to save card power, the clock signal is disabled when the card is idle for at least 8 card clock cycles. It is enabled when a new command is loaded and the command path goes to a non-idle state.

Additionally, the clock of the card is disabled when:

- An internal FIFO is full on a card read (no more data can be received from card)
- The FIFO is empty on a card write (no data is available for transmission).

Disabling the clock in these situations helps to avoid FIFO overrun and underrun conditions.

Under the following conditions, the card clock is stopped or disabled for the card:

- The clock can be disabled by writing to the clock enable MSI_CLKEN register.
- If low-power mode is selected and a card is idle, or not selected for 8 clocks.
- The FIFO is full and data path cannot accept more data from the card and data transfer is incomplete to avoid FIFO overrun.
- The FIFO is empty and data path cannot transmit more data to the card and data transfer is incomplete to avoid FIFO underrun.

NOTE: The host software must take care while changing the MSI_CLKDIV register values. The card clock must be disabled through the MSI_CLKEN register before changing the values of the MSI_CLKDIV register.

MSI Data Transfer Modes

The following sections provide information on data transfer using the MSI interface.

Data Transmit

The data transmit state machine starts data transmission two clocks after a response for the data write command is received. The transmission occurs even if the command path detects a response error or response CRC error. See the ***Data Transmit State Machine*** figure. If a response is not received from the card because of a response timeout, data is not transmitted. Depending upon the value of the MSI_CMD.XFRMODE bit, the data transmit state machine puts data on the card data bus in a stream or in blocks.

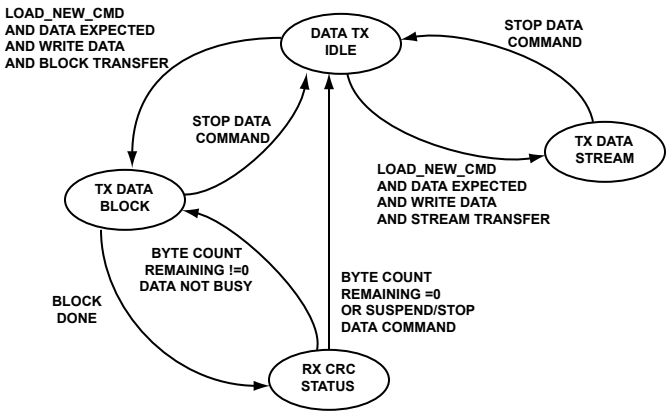

Figure 33-8: Data Transmit State Machine

Stream Data Transmit

If the MSI_CMD.XFRMODE bit =1, it is a stream-write data transfer. The data path pops the FIFO from the BIU and transmits in a stream to the card data bus. If the FIFO becomes empty, the card clock is stopped and restarted once data is available in the FIFO.

If the MSI_BYTCNT register is programmed to 0, it is an open-ended stream-write data transfer. During this data transfer, the data path continuously transmits data in a stream until the host software issues a stop command. A stream data transfer is terminated when the end bit of the stop command and end bit of the data match over two clocks.

If the MSI_BYTCNT register is programmed with a non-zero value and the MSI_CMD.SENDASTOP bit is set, the stop command is internally generated. The command is loaded in the command path when the end bit of the stop command occurs after a match with the last byte of the stream-write transfer. This data transfer can also terminate if the software issues a stop command before all the data bytes transfer to the card bus.

Single Block Data

If the MSI_CMD.XFRMODE bit =0 and the MSI_BYTCNT register value is equal to the value of the MSI_BLKSIZ register, a single-block write-data transfer occurs. The data transmit state machine sends data in a single block, where the number of bytes equals the block size, including the internally-generated CRC16.

If the MSI_CTYPE register bit is set for a 1-bit, 4-bit, or 8-bit data transfer, the data transmits on 1, 4, or 8 data lines, respectively. CRC16 is separately generated and transmitted for 1, 4, or 8 data lines, respectively.

After a single data block is transmitted, the data transmit state machine receives the CRC status from the card and signals a data transfer to the BIU. This operation happens when the MSI_ISTAT.DTO bit =1. If a negative CRC status is received from the card, the data path signals a data CRC error to the BIU by setting the MSI_ISTAT.DCRC bit.

Additionally, if the start bit of the CRC status is not received before two clocks after the end of the data block, a CRC status start bit error is signaled to the BIU. The MSI_ISTAT.EBE bit is set.

Multiple Block Data

A multiple-block write-data transfer occurs if the MSI_CMD.XFRMODE bit =0 and the value in the MSI_BYTCNT register is not equal to the value of the MSI_BLKSIZ register. The data transmit state machine sends data in blocks, where the number of bytes in a block equals the block size, including the internally-generated CRC16.

If the MSI_CTYPE register bit is set to 1-bit, 4-bit, or 8-bit data transfer, the data transmits on 1, 4, or 8 data lines, respectively. CRC16 is separately generated and transmitted on 1, 4, or 8 data lines, respectively.

After one data block is transmitted, the data transmit state machine receives the CRC status from the card. If the remaining MSI_BYTCNT becomes 0, the data path signals to the BIU that the data transfer is complete. This operation happens when the MSI_ISTAT.DTO bit =1. If the remaining data bytes are greater than 0, the data path state machine starts to transmit another data block.

If a negative CRC status is received from the card, the data path signals a data CRC error to the BIU by setting the MSI_ISTAT.DCRC bit. The block continues further data transmission until all the bytes are sent. Additionally, if the CRC status start bit is not received within two clocks after the end of a data block, a CRC status start bit error is signaled to the BIU. The MSI_ISTAT.EBE bit is set. Further data transfer is terminated.

If the MSI_CMD.SENDASTOP is set, the stop command is internally generated during the transfer of the last data block. No extra bytes are transferred to the card. The end bit of the stop command does not always exactly match the end bit of the CRC status in the last data block.

If the block size is less than 4, 16, or 32 for card data widths of 1 bit, 4 bits, or 8 bits, respectively, the data transmit state machine terminates the data transfer when all the data has transferred. The internally generated stop command is loaded in the command path.

If the MSI_BYTCNT register =0 (the block size must be greater than 0), it is an open-ended block transfer. The data transmit state machine for this type of data transfer continues the block-write data transfer until the host software issues a stop or abort command.

Data Receive

The data-receive state machine receives data two clock cycles after the end bit of a data read command, even if the command path detects a response error or response CRC error. See the ***Data Receive State Machine*** figure. If a response is not received from the card because a timeout error occurs, the BIU does not receive the signal that the data transfer is complete. This error happens when the command sent by the MSI is an illegal operation for the card. The error prevents the card from starting a read data transfer.

If data is not received before the data timeout, the data path signals a data timeout to the BIU and an end to the data transfer done. Based on the value of the MSI_CMD.XFRMODE bit, the data-receive state machine gets data from the card data bus in a stream or blocks.

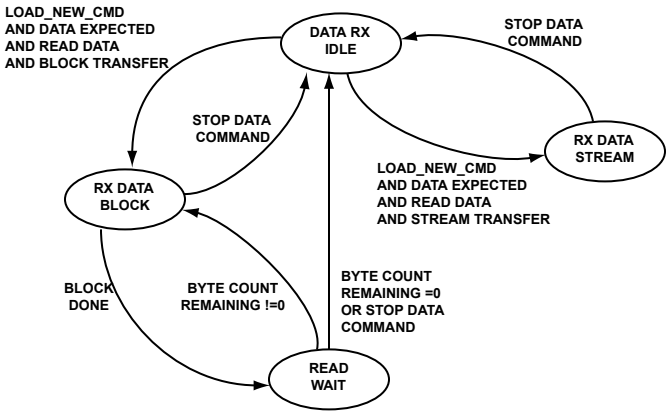

Figure 33-9: Data Receive State Machine

Stream Data Read

A stream-read data transfer occurs if the MSI_CMD.XFRMODE bit =1, at which time the data path receives data from the card and pushes it to the FIFO. If the FIFO becomes full, the card clock stops and restarts once the FIFO is no longer full.

An open-ended stream-read data transfer occurs when the MSI_BYTCNT register equals 0. During this type of data transfer, the data path continuously receives data in a stream until the host software issues a stop command. A stream data transfer terminates two clock cycles after the end bit of the stop command.

If the MSI_BYTCNT register contains a non-zero value and the MSI_CTL.AUTOSTOP bit =1, a stop command is internally generated and loaded into the command path. In the command path, the end bit of the stop command occurs after the last byte of the stream data transfer is received. This data transfer can terminate if the host issues a stop or abort command before all the data bytes are received from the card.

Single Block Data Read

A single-block read-data transfer occurs if the MSI_CMD.XFRMODE bit =0 and the value of the MSI_BYTCNT register is equal to the value of the MSI_BLKSIZ register. When a start bit is received before the data times out, data bytes equal to the block size and CRC16 are received and checked with the internally-generated CRC16.

If the MSI_CTYPE register bit for the card is set to a 1-bit, 4-bit, or 8-bit data transfer, data is received from 1, 4, or 8 data lines, respectively. CRC16 is separately generated and checked for 1, 4, or 8 data lines, respectively. If there is a CRC16 mismatch, the data path signals a data CRC error to the BIU. If the received end bit is not 1, the BIU receives an end-bit error.

Multiple Block Data Read

If the MSI_CMD.XFRMODE =0 and the value of the MSI_BYTCNTregister is not equal to the value of the MSI_BLKSIZ register, it is a multiple-block read-data transfer. The data-receive state machine receives data in blocks, where the number of bytes in a block is equal to the block size, including the internally-generated CRC16.

If the MSI_CTYPE register bit for the card is set to a 1-bit, 4-bit, or 8-bit data transfer, data is received from 1, 4, or 8 data lines, respectively. CRC16 is separately generated and checked for 1, 4, or 8 data lines, respectively. After a data block is received, if the remaining MSI_BYTCNT becomes 0, the data path signals a data transfer to the BIU.

If the remaining data bytes are greater than 0, the data path state machine causes another data block to be received. If CRC16 of a received data block does not match the internally-generated CRC16, a data CRC error to the BIU and data reception continue further data transmission until all bytes are transmitted. Additionally, if the end of a received data block is not 1, data on the data path signals terminate the bit error to the CIU. The data-receive state machine terminates data reception, waits for a data timeout, and signals to the BIU that the data transfer is complete.

If the MSI_CTL.AUTOSTOP bit =1, the stop command is internally generated when the last data block is transferred, where no extra bytes are transferred from the card. The end bit of the stop command does not always exactly match the end bit of the last data block.

If the requested block size for data transfers to cards is less than 4, 16, or 32 bytes for 1-bit, 4-bit, or 8-bit data transfer modes, respectively, the data-transmit state machine terminates the data transfer when all data transfers. The internally-generated stop command is loaded in the command path. The data path ignores any subsequent data received from the card.

If the MSI_BYTCNT =0 (the block size must be greater than 0), it is an open-ended block transfer. For this type of data transfer, the data-receive state machine continues the block-read data transfer until the host software issues a stop or abort command.

Data Transfer Commands

Data transfer commands transfer data between the memory card and the MSI. To send a data command, the MSI needs a command argument, total data size, and block size.

Prior to a data transfer command, software must confirm that the card is not busy and is in a transfer state. This confirmation uses the CMD13 and CMD7 commands, respectively. For the data transfer commands, it is important that the same bus width that is programmed in the card is set in the MSI_CTYPE register.

The MSI generates an interrupt for different conditions during data transfer, which are reflected in the MSI_ISTAT register.

NOTE: The DCRC, SBEBCI, EBE, and SBEBCI conditions indicate that the received data could have errors. If there was a response timeout, then no data transfer occurred. For more information, see the Register Descriptions section.

Transmission and Reception with Internal DMAC (IDMAC)

The general sequence of events for transmit and receive is as follows.

1. Program the required programming in the bus mode register (MSI_BUSMODE). If the MSI_CTL.INTDMAC bit is disabled during the middle of an IDMAC transfer, it has no effect. It only takes effect for a new data transfer command. Issuing a software reset immediately terminates the transfer. It is recommended that the software issue a reset to the DMA interface by setting the MSI_CTL.DMARST bit and then issuing an IDMAC software reset using the MSI_CTL.CTLRST bit. Program the fixed burst bit (MSI_BUSMODE.FB) appropriately for system performance.
2. When a descriptor unavailable interrupt is asserted, the software must form the descriptor, appropriately set its own bit and then write to poll the demand register for the IDMAC to re fetch the descriptor.
3. It is always appropriate for the application to enable abnormal interrupts since any errors related to the transfer are reported to the application.

MSI Event Control

The following sections describe MSI events.

Dedicated Interrupt Pins

Interrupt lines are defined for only eSDIO devices, which are connected to the controller interrupt line in MSI. These interrupt lines can operate even when the card clock is switched off and can be used only during an asynchronous interrupt period.

MSI Status and Error Signals

The following provides information on MSI errors.

Error Detection

The MSI has an error detection mechanism which operates for any of the following situations.

Response

- Response timeout – Response expected with the response start bit is not received within programmed number of clocks in timeout register.
- Response CRC error – Response is expected and check response CRC requested; response CRC7 does not match with the internally-generated CRC7.
- Response error – Response transmission bit is not 0, command index does not match with the command index of the send command, or response end bit is not 1.

Data Transmit

- No CRC status – During a write data transfer, if the CRC status start bit is not received two clocks after the end bit of the data block is sent out, the datapath does the following:
    - Signals no CRC status error to the BIU
    - Terminates further data transfer
    - Signals data transfer done to the BIU
- Negative CRC – If the CRC status received after the write data block is negative (that is, not 010), a data CRC error is signaled to the BIU and further data transfer is continued.
- Data starvation due to empty FIFO – If the FIFO becomes empty during a write data transmission, or if the card clock is stopped and the FIFO remains empty for data timeout clocks, then a data-starvation error is signaled to the BIU. The datapath continues to wait for data in the FIFO.

Data Receive

- Data timeout – During a read-data transfer, if the data start bit is not received before the number of clocks that are programmed in the timeout register, the datapath does the following.
    - Signals data-timeout error to the BIU
    - Terminates further data transfer
    - Signals data transfer done to BIU
- Data start bit error – During a 4-bit or 8-bit read-data transfer, when an SBE occurs, the application or driver must issue CMD12 for SD/MMC cards and CMD52 for a SDIO card to exit the error condition. After a CMD done is received, the application or driver must reset IDMAC and CIU (if required) to clear the condition. The FIFO must be reset before issuing any data transfer commands in general.
- Data CRC error – During a read-data-block transfer, if the received CRC16 does not match with the internally-generated CRC16, the datapath signals a data CRC error to the BIU and continues further data transfer.

- Data end-bit error – During a read-data transfer, if the end bit of the received data is not 1, the datapath signals an end-bit error to the BIU, terminates further data transfer, and signals to the BIU that the data transfer is done.
- Data starvation due to FIFO full – During a read data transmission and when the FIFO becomes full, the card clock is stopped. If the FIFO remains full for data timeout clocks, a data starvation error is signaled to the BIU. The datapath continues to wait for the FIFO to start to empty. (The data starvation by host timeout bit is set in the `MSI_ISTAT` register).

NOTE: In an error situation, DTO generation depends on when the error has occurred, since the descriptors are closed after the error scenarios as follows.

- If the data remains in the FIFO when the DMA detects the error scenario, then a DTO is not generated.
- If the data is completely read out before the error occurs, then a DTO is generated. CMD12 ensures DTO generation in error situations; therefore, issuing CMD12 is recommended.

Error Handling

The MSI implements error checking. Errors are reflected in the `MSI_ISTAT` register and can be communicated to the software through an interrupt, or the software can poll these bits. On power-on, interrupts are disabled, and all the interrupts are masked (bits 31:0 of the `MSI_IMSK` register are all 0). The MSI module captures the following errors:

- Response and data timeout errors. For response timeout, software can retry the command. For a data timeout error, the MSI has not received the data start bit - either for the first block or the intermediate block - within the timeout period. Software can either retry the whole data transfer again or retry from a specified block onwards. By reading the contents of the `MSI_TCBCNT` register later, the software can decide how many bytes remain to be copied.
- Response errors. Set when an error is received during response reception. In this case, the response that copied in the response registers is invalid. Software can retry the command.
- Data errors. Set when error in data reception is observed. For example, this error can occur in the data CRC, when the start bit is not found, when the end bit is not found, and so on. These errors could be set for any block-first block, intermediate block, or last block. On receipt of an error, the software can issue a STOP or ABORT command and retry the command for either whole data or partial data.
- Hardware locked error. Set when the MSI cannot load a command issued by software. When software sets the `MSI_CMD.STARTCMD` bit, the MSI tries to load the command. If the command buffer is already filled with a command, this error is generated. The software then has to reload the command.
- FIFO underrun or overrun error. If the FIFO is full and DMA tries to write data into the FIFO, then an overrun error is set. Conversely, if the FIFO is empty and the DMA tries to read data from the FIFO, an underrun error is set.

- Data starvation by host timeout. Raised when the MSI is waiting for software intervention to transfer the data to or from the FIFO, but the software does not transfer within the stipulated timeout period. Under this condition and when a read transfer is in progress, the software must read data from the FIFO and create space for further data reception. When a transmit operation is in process, the software must fill data in the FIFO to start transferring data to the card.
- CRC Error on command. If a CRC error is detected for a command.

NOTE: During a multiple-block data transfer, if a negative CRC status is received from the device, the datapath signals a data CRC error to the BIU by setting the data `MSI_ISTAT.DCRC` register. It then continues further data transmission until all the bytes are transmitted.

Fatal Bus Errors (FBE)

An FBE occurs due to an error response from SCB. This response is a system error, so the software driver must not perform any further programming of the MSI. The only recovery mechanism from such scenarios is to do one of the following.

- Issue a hard reset.
- Do a controller reset by writing to the `MSI_CTL.CTLRST` bit.

MSI Programming Model

This section provides procedures that allow programmers to use the MSI properly. These procedures include MSI initialization, single and multiple block reads and writes and many others.

MSI Programming Concepts

Using the features, operating modes, and event control for the MSI to their greatest potential requires an understanding of the following MSI-related concepts.

Software and Hardware Restrictions

Before issuing a new data transfer command, the software should ensure that the card is not busy due to any previous data transfer command. Before changing the card clock frequency, the software must ensure that there are no data or command transfers in progress.

To avoid glitches in the card clock outputs, use the following steps when changing the card clock frequency.

1. Before disabling the clocks, ensure that the card is not busy due to any previous data command. To determine this status, check for 0 in the `MSI_STAT.DBUSY` bit.
2. Update the `MSI_CLKEN` register to disable all clocks. To ensure completion of any previous command before this update, send a command to the CIU to update the clock registers by setting the `MSI_CMD.STARTCMD` bit, the `MSI_CMD.WTPRIVDATA` bits, and the `MSI_CMD.UCLKREGS` bit.

3. Set the MSI_CMD.STARTCMD bit to update the clock divider register, and send a command to the CIU in order to update the clock registers. Wait for the CIU to take the command.
4. Set the MSI_CMD.STARTCMD to update the MSI_CLKEN register to enable the required clocks and send a command to the CIU to update the clock registers. Wait for the CIU to take the command.

If the software issues a controller reset command by setting the MSI_CTL.CTLRST bit, all the CIU state machines are reset and the FIFO is not cleared. The DMA sends all remaining bytes to the host. In addition to a card-reset, if a FIFO reset is also issued (MSI_CTL.FIFORST), then:

- Any pending DMA transfer on the bus completes correctly
- DMA data reads are ignored
- Write data is unknown (x)

Additionally, if a DMA reset is also issued (MSI_CTL.DMARST), any pending DMA transfer is abruptly terminated.

If any of the previous data commands do not properly terminate, then the software must issue the FIFO reset. The reset removes any residual data, if any, in the FIFO. After asserting the FIFO reset, the program waits until this bit is cleared. One data transfer requirement between the FIFO and host is that the number of transfers must be a multiple of the FIFO data width. For example, if the FIFO data width = 32-bit and the program must write only 15 bytes to the card, the host must program the DMA to do 16-byte transfers to the card. (The program writes to the card using the MSI_BYTCNT register).

Initializing the MSI

After the power-up, the MSI is reset. The reset initializes the registers, ports, FIFO-pointers, DMA interface controls, and state-machine. After power-on reset, the software performs the following steps which are reflected in the ***Data Transmit State Machine*** figure.

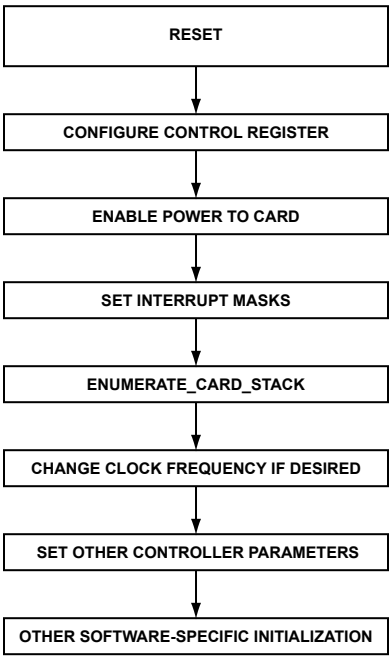

Figure 33-10: Data Transmit State Machine

1. Configure the control register (`MSI_CTL`).
2. Set masks for interrupts by clearing appropriate bits in the interrupt mask register. Set the global `MSI_CTL.INTEN` bit. It is recommended that programs write 0xFFFF_FFFF to the `MSI_ISTAT` register to clear any pending interrupts before setting the `MSI_CTL.INTEN` bit.
3. Enumerate the card stack. Each card is enumerated according to card type. For details, refer to Enumerating the Card Stack. For enumeration, restrict the clock frequency to 400 kHz in accordance with SD/MMC standards.
4. Set the card frequency using the `MSI_CLKDIV` register.
5. Set other parameters, which normally do not need changing with every command, with a typical value such as response and data timeout values. Set the values according to the SD/MMC specifications and the FIFO threshold value.

Enumerating the Card Stack

The card stack does the following:

- Enumerates the connected card
- Sets the RCA for the connected card
- Reads card-specific information
- Stores card-specific information locally

Each card is enumerated separately. The identification procedure depends on whether the card connected is SD, SDIO, or MMC.

Identifying Card Types

The MSI can have an MMC, SD, or SDIO type of card connected to it. All types of SDIO cards are supported; that is, SDIO_IO_ONLY, SDIO_MEM_ONLY, and SDIO_COMBO cards. Each card is enumerated separately. The enumeration sequence includes the following steps:

1. Check if the card is connected.
2. Clear the bits in the MSI_CTYPE register to configure the controller in 1-bit mode
3. Identify the card type; that is, SD, MMC, SDIO, or COMBO card.
    a. Send CMD5 with argument 0.
    b. Read the response register (MSI_RESP0), which gives the voltage window that the card supports-the response to CMD5 gives the voltages that the card supports.
    c. Send CMD5 again with the desired voltage window set in the argument. This CMD5 is used to set the voltage window and move the card state m/c out of the initialization state.
    d. Check the response [27] bit. If this bit = 1, this response indicates that the memory is present and the SDIO card is a COMBO card.
    e. If the card is SDIO go to Step 4. If the card is combinational logic (COMBO) or if the response is not received, continue with the following steps.
    f. Send CMD8 with the following argument:

```
Bit[31:12] = 20'h0; // Reserved bits
Bit[11:8] = 4'b0001 // VHS value
Bit[7:0] = 8'b10101010 // Preferred Check Pattern by SD2.0
```

    g. If the response is received, the card supports high capacity SD2.0; send ACMD41 with the following argument:

```
Bit[31] = 1'b0; // Reserved bits
Bit[30] = 1'b1; // High Capacity Status
Bit[29:25] = 6'h0; // Reserved bits
Bit[24] = 1'b0;
Bit[23:0] = Supported Voltage Range
```

    h. If the response is received for ACMD41, then the card is SD; otherwise the card is MMC.
    i. If the response is not received for initial CMD8, then the card does not support high capacity SD2.0. Issue CMD0 followed byMD41 with the following argument:

```
Bit[31] = 1'b0; // Reserved bits
Bit[30] = 1'b0; // High Capacity Status
```

```
Bit[29:24] = 6'h0; // Reserved bits
Bit[23:0] = Supported Voltage Range
```

j. If the response is received for ACMD41, then the card is SD. Otherwise, the card is MMC.

4. Enumerate the card according to the card type.
5. Use a card clock frequency = f_{OD} (that is, 400 kHz) and use the following enumeration command sequence:
   - SD card – Send CMD0, CMD8, ACMD41, CMD2, CMD3.
   - SDIO – Send CMD5; if the function count is valid, CMD3. For the SDIO memory section, follow the same commands as for the SD card.
   - MMC – Send CMD0, CMD1, CMD2, CMD3.

The card clock frequency can be configured after enumeration.

Programming Card Clocks

The MSI supports programming the desired operational frequency for the card. The clock for the card can also be enabled or disabled. Registers that support this feature are:

- MSI_CLKDIV: Programs clock source frequency.
- MSI_CLKEN: Enables or disables clock for the card and enables low-power mode, which automatically stops the clock to a card when the card is idle for more than 8 clocks.

The MSI loads each of these registers only when the MSI_CMD.STARTCMD bit and the MSI_CMD.UCLKREGS bit in the CMD register are set. When a command is successfully loaded, the MSI clears the MSI_CMD.STARTCMD bit. This operation happens unless the MSI already has another command in the queue, at which point it generates a Hardware Locked Error (HLE).

Software must look for the MSI_CMD.STARTCMD and the MSI_CMD.UCLKREGS bits, and set the MSI_CMD.WTPRIVDATA bit to ensure that clock parameters do not change during data transfer.

NOTE: Even though MSI_CMD.STARTCMD is set for updating clock registers, the MSI does not raise the MSI_DONE signal upon command completion.

Use the following procedure to program the clock-related registers.

1. Confirm that the card is not engaged in any transaction; if there is a transaction, wait until it finishes.
2. Stop all clocks by writing 0 to the MSI_CLKEN register. Set the MSI_CMD.STARTCMD, MSI_CMD.UCLKREGS, and the MSI_CMD.WTPRIVDATA bits. Wait until the MSI_CMD.STARTCMD bit is cleared or an HLE is set; in case of an HLE, repeat the command.
3. Program the MSI_CLKDIV register. Set the MSI_CMD.STARTCMD, MSI_CMD.UCLKREGS, and MSI_CMD.WTPRIVDATA bits. Wait until the MSI_CMD.STARTCMD is cleared or an HLE is set; in case of an HLE, repeat the command.

4. Re enable all clocks by programming the MSI_CLKEN register. Set the MSI_CMD.STARTCMD, MSI_CMD.UCLKREGS, and the MSI_CMD.WTPRIVDATA bits. Wait until the MSI_CMD.STARTCMD bit is cleared or a HLE is set; in case of a HLE, repeat the command.

Sending Non-Data Commands With or Without a Response Sequence

To send any non-data command, the software must program the MSI_CMD register and the MSI_CMDARG register with appropriate parameters. Using these two registers, the MSI forms the command and sends it to the command bus. The MSI reflects the errors in the command response through the error bits of the MSI_ISTAT register. The ***Command Register Settings for No-Data Command*** table shows the MSI_CMD bit settings.

When the MSI receives a response, either erroneous or valid, it sets the MSI_ISTAT.CMDDONE bit. A short response is copied in the MSI_RESP0 register, while a long response is copied to all four response registers. The MSI_RESP3 register bit 31 represents the MSB, and the MSI_RESP0 register bit 0 represents the LSB of a long response.

Table 33-12: Command Register Settings for No-Data Command

Parameter	Value	Comment
Default		
start_cmd	1	
use_hold_reg	1/0	Choose value based on speed mode in use; refer to “MSI Register Descriptions”
Update_clk_regs_only	0	No clock parameters update command
data_expected	0	No data command
card_number	*n*CardNo	Actual card number
cmd_index	command index	
send_initialization	0	Can be 1, but only for card reset commands, such as CMD0
stop_abort_cmd	0	Can be 1 for commands to stop data transfer, such as CMD12
response_length	0	Can be 1 for R2 (long) response
response_expect	1	Can be 0 for commands with no response; for example, CMD0, CMD4, CMD15, and so on
User selectable		
wait_prvdata_complete	0	Before sending a command on a command line, the host must wait for the completion of any data command in process, if any. (It is recommended to set this bit always, unless the current command is to query status or stop data transfer when transfer is in progress)
check_response_crc	1	If host crosschecks, CRC of response received

Use the following procedure to issue basic commands or non-data commands.

1. Program the MSI_CMD register with the appropriate command argument parameter.

2. Program the MSI_CMD register with the settings in the ***Command Register Settings for No-Data Command*** table.
3. Wait for command acceptance by host. When software loads the command into the MSI:
   - The MSI accepts the command for execution and clears the MSI_CMD.STARTCMD bit. This process occurs unless one command is in process. Then, the MSI can load and keep the second command in the buffer.
   - If the MSI is unable to load the command, then it generates an HLE (hardware-locked error). (For example, a command is already in progress, a second command is in the buffer, and a third command is attempted).
4. Check if there is an HLE.
5. Wait for command execution to complete. After receiving either a response from a card or response timeout, the MSI sets the MSI_ISTAT.CMDDONE bit. Software can either poll for this bit or respond to a generated interrupt.
6. Check if the response timeout error, response CRC error, or response error is set. This confirmation can be done either by responding to an interrupt raised by these errors or by polling bits RE, RCRC, and RTO from the MSI_ISTAT register. If no response error is received, then the response is valid. If necessary, the software can copy the response from the response registers. Software must not modify clock parameters while a command is executing.

Single-Block or Multiple-Block Read

Use the following procedure to perform a single-block or multiple-block read.

1. Write the data size in bytes in the MSI_BYTCNT register.
2. Write the block size in bytes in the MSI_BLKSIZ register.

   ADDITIONAL INFORMATION: The MSI expects data from the card in blocks of size BLKSIZ each.
3. If card has a round-trip delay of more than 0.5 card clock period, program the MSI_CDTHRCTL.RDTHR bit field. The programming ensures that the card clock does not stop in the middle of a block of data transferring from the card to the host.

   ADDITIONAL INFORMATION: For programming guidelines, refer to the Card Read Threshold section. If the card read threshold feature is not enabled for such cards, then the program must ensure that the Rx FIFO does not become full during a read data transfer. The Rx FIFO must drain out at a rate faster than the rate that data is pushed into the FIFO.
4. Program the MSI_CMDARG register with the data address of the beginning of a data read. Program the MSI_CMD register with the parameters listed in the ***Command Register Settings for Single-Block or Multiple-Block Read*** table.

ADDITIONAL INFORMATION: For SD and MMC cards, use CMD17 for a single-block read and CMD18 for a multiple-block read. For SDIO cards, use CMD53 for both single-block and multiple-block transfers.

After writing to the MSI_CMD register, the MSI starts executing the command. When the command is sent to the bus, the CMDONE interrupt is generated.

5. Software looks for data error interrupts on bits 7, 9, 13, and 15 of the MSI_ISTAT register. If required, software can terminate the data transfer by sending a STOP command
6. Software or the DMA looks for a receive FIFO data request or data starvation by host timeout conditions. In both cases, the DMA reads data from the FIFO and makes space in the FIFO for receiving more data.

When all the data is received, a DTO (Data Transfer Over) interrupt is generated.

Table 33-13: Command Register Settings for Single-Block or Multiple-Block Read

Parameter	Value	Comment
Default		
start_cmd	1	
use_hold_reg	1/0	Choose value based on speed mode in use; refer to "MSI Register Descriptions"
Update_clk_regs_only	0	No clock parameters update command
card_number	*n*CardNo	Actual card number
send_initialization	0	Can be 1, but only for card reset commands, such as CMD0
stop_abort_cmd	0	Can be 1 for commands to stop data transfer, such as CMD12
send_auto_stop	0 or 1	See Auto-Stop
transfer_mode	0	Block transfer
read_write	0	Read from card
data_expected	1	Data command
response_length	0	Can be 1 for R2 (long) response
response_expect	1	Can be 0 for commands with no response; for example, CMD0, CMD4, CMD15, and so on
User selectable		
cmd_index	command index	Can be 1 for commands to stop data transfer, such as CMD12
wait_prvdata_complete	1	0 = sends command immediately. 1 = sends command after previous data transfer ends
check_response_crc	1	0 = MSI does not check response CRC 1 = MSI checks response CRC

Single-Block or Multiple-Block Write

Use the following procedure to perform a single-block or multiple-block write.

1. Write the data size in bytes in the MSI_BYTCNT register.
2. Write the block size in bytes in the MSI_BLKSIZ register.

   ADDITIONAL INFORMATION: The MSI sends data in blocks of size BLKSIZ each.
3. Program MSI_CMDARG register with the data address to which data is written.
4. Write data in the FIFO; it is best to start filling data the full depth of the FIFO.
5. Program the MSI_CMD register with the parameters listed the ***Command Register Settings for Single-Block or Multiple-Block Read*** table.

   ADDITIONAL INFORMATION: For SD and MMC cards, use CMD24 for a single-block write and CMD25 for a multiple-block write. For SDIO cards, use CMD53 for both single-block and multiple-block transfers. After writing to the MSI_CMD register, the MSI starts executing a command; when the command is sent to the bus, a CMDONE interrupt is generated.
6. Software looks for data error interrupts in the MSI_ISTAT.DCRC, MSI_ISTAT.DRTO, and MSI_ISTAT.EBEbits. If necessary, software can terminate the data transfer by sending the STOP command.
7. Software looks for a transmit FIFO data request or timeout conditions from data starvation by the host. In both cases, the software or the DMA writes data into the FIFO.
8. When a DTO interrupt is received, the data command is over. For an open-ended block transfer, if the byte count is 0, the software must send the STOP command. If the byte count is not 0, then on completion of a transfer of a given number of bytes, the MSI sends the STOP command, if necessary. The MSI_ISTAT.ACD bit reflects the completion of the AUTO-STOP command. A response to AUTO_STOP is stored in the MSI_RESP1 register.
9. Wait for the busy clear interrupt.

The card can drive the busy clear interrupt on the DAT line; the host controller generates the interrupt after the busy is completed.

Table 33-14: Command Register Settings for Single-Block or Multiple-Block Write

Parameter	Value	Comment
Default		
start_cmd	1	
use_hold_reg	1 or 0	Choose value based on speed mode in use; refer to “MSI Register Descriptions”
Update_clk_regs_only	0	No clock parameters update command
card_number	*n*CardNo	Actual card number
send_initialization	0	Can be 1, but only for card reset commands, such as CMD0
stop_abort_cmd	0	Can be 1 for commands to stop data transfer, such as CMD12
send_auto_stop	0 or 1	See Auto-Stop

Table 33-14: Command Register Settings for Single-Block or Multiple-Block Write (Continued)

Parameter	Value	Comment
transfer_mode	0	Block transfer
read_write	1	Write to card
data_expected	1	Data command
response_length	0	Can be 1 for R2 (long) response
response_expect	1	Can be 0 for commands with no response; for example, CMD0, CMD4, CMD15, and so on
User selectable		
cmd_index	command index	Can be 1 for commands to stop data transfer, such as CMD12
wait_prvdata_complete	1	0 – sends command immediately 1 – sends command after previous data transfer ends
check_response_crc	1	0 – MSI does not check response CRC 1 – MSI checks response CRC

Stream Reads and Writes

Stream reads and writes are like the block reads and writes described in the previous sections except for the following bits in the `MSI_CMD.XFRMODE` register:

- For reads, the transfer mode (`MSI_CMD.XFRMODE` bit) =1 and the command index =CMD20.
- For writes, the transfer mode (`MSI_CMD.XFRMODE` bit) =1 and the command index =CMD11.

In a stream transfer, if the byte count is 0, then the software must send the STOP command. If the byte count is not 0, when a given number of bytes completes a transfer, the MSI sends the STOP command

Packed Commands

In order to reduce overhead, read and write commands can be packed in groups of commands. The groups are either all read or all write. The software transfers the data for all commands in the group in one transfer on the bus.

Packed commands can be of two types:

- Packed write – CMD23 > CMD25
- Packed read – CMD23 > CMD25 >CMD23 > CMD18

The application software puts packed commands in packets. The packets are transparent to the core. For more information on packed commands, refer to the eMMC specification.

Sending Stop or Abort in Middle of Transfer

The STOP command can terminate a data transfer between a memory card and the MSI. The ABORT command can terminate an I/O data transfer for only the SDIO_IOONLY and SDIO_COMBO cards.

- Send STOP command. Can be sent on the command line while a data transfer is in-progress. This command can be sent at any time during a data transfer. For information on sending this command, refer to Sending Non-Data Commands With or Without a Response Sequence. Programs can use an extra setting for this command to set the `MSI_CMD` register bits (5–0) to CMD12 and set the `MSI_CMD.STPABORTCMD` bit to 1. If the `MSI_CMD.STPABORTCMD` bit is not set to 1, the MSI does not know that the program stopped a data transfer. Reset the `MSI_CMD.WTPRIVDATA` bit to 0 in order to make the MSI send the command at once, even though there is a data transfer in progress.
- Send ABORT command. Can be used with only an SDIO_IOONLY or SDIO_COMBO card. To abort the function that is transferring data, program the function number in ASx bits (CCCR register of card, address 0x06, bits (0–2)) using CMD52. This command is a non-data command. For information on sending this command, refer to Sending Non-Data Commands With or Without a Response Sequence.

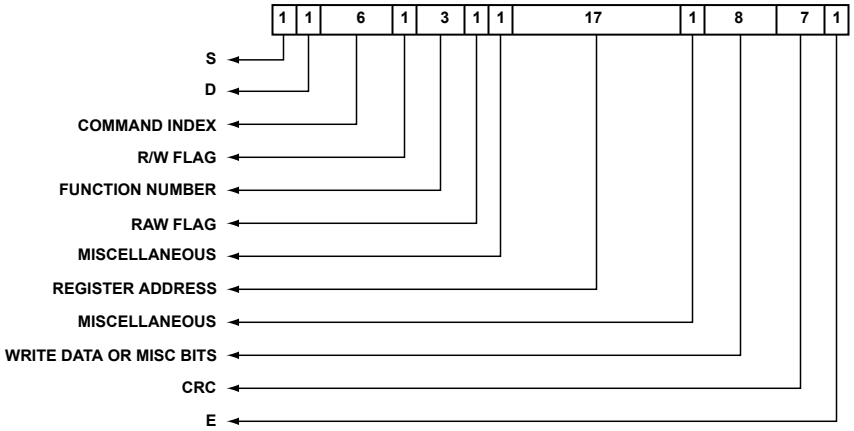

Figure 33-11: Command Format for CMD52

1. Program the `MSI_CMDARG` register with the appropriate command argument parameters listed in the ***Parameters for CMDARG Register*** table.
2. Program the `MSI_CMD` register using the command index as CMD52. Similar to the STOP command, set the `MSI_CMD.STPABORTCMD` bit =1, to inform the MSI that the program aborted the data transfer. Reset the `MSI_CMD.WTPRIVDATA` bit =0 in order to make the MSI send the command at once, even though a data transfer is in progress.
3. Wait for command transfer over.
4. Check response (R5) for errors.

Table 33-15: Parameters for CMDARG Register

MSI_CMDARG Bits	Contents	Value
31	R/W flag	1
30–28	Function number	0 = for CCCR access
27	RAW flag	1 = if needed to read after write
26	Do-not-care	N/A
25-9	Register address	0x06
8	Do-not-care	N/A
7–0	Write data	Function number to be aborted

Suspend or Resume Sequence

In an SDIO card, the data transfer between an I/O function and the MSI can be temporarily halted using the SUSPEND command. This command can be necessary to perform a high-priority data transfer with another function. When desired, the data transfer can be resume using the RESUME command.

The following functions can be implemented by programming the appropriate bits in the CCCR register of the SDIO card. To read from or write to the CCCR register, use the CMD52 command.

1. SUSPEND data transfer – Non-data command.
    a. Check if the SDIO card supports the SUSPEND or RESUME protocol; this verification can be done through the SBS bit in the CCCR @0x08 register of the card.
    b. Check if the data transfer for the required function number is in process; the function number that is currently active is reflected in bits 0–3 of the CCCR register @0x0D. If the BS (bus status) bit is 1, then only the function number given by the FSx bits is valid.
    c. To suspend the transfer, set BR (bit 2) of the CCCR register @0x0C.
    d. Poll for clear status of bits BR (bit 1) and BS (bit 0) of the CCCR @0x0C. The BS bit is 1 when the currently selected function uses the data bus; the BR (bus release) bit remains 1 until the bus release is complete. When the BR and BS bits =0, the data transfer from the selected function has been suspended.
    e. During a read-data transfer, the MSI can be waiting for the data from the card. If the data transfer is a read from a card, then the MSI must be informed after the successful completion of the SUSPEND command. The MSI then resets the data state machine and comes out of the wait state. To achieve this state, set the MSI_CTL.RDABORT bit.
    f. Wait for data completion. Get pending bytes to transfer by reading the MSI_TCBCNT register.
    g. For a write data transfer, wait for the busy clear interrupt after the Data Transfer Over (DTO) interrupt.
2. RESUME data transfer – This command is a data command.
    a. Check that the card is not in a transfer state, which confirms that the bus is free for data transfer.

b. If the card is in a disconnect state, select it using CMD7. The card status can be retrieved in response to CMD52/CMD53 commands.

c. Check that a function to be resumed is ready for data transfer. Confirm the readiness by reading the RFx flag in CCCR @0x0F. If RF =1, then the function is ready for data transfer.

d. To resume the transfer, use CMD52 to write the function number at the FSx bits (0–3) in the CCCR register @0x0D. Form the command argument for CMD52 and write it in the MSI_CMDARG register. The ***Parameters for CMDARG Register*** table lists the bit values.

e. Write the block size in the MSI_BLKSIZ register. Data transfers in units of this block size.

f. Write the byte count in the MSI_BYTCNT register. This amount is the total size of the data; that is, the remaining bytes for transfer. It is the responsibility of the software to handle the data.

g. Program the MSI_CMD register similarly to a block transfer. For details, refer to the block read and write sections.

h. When the MSI_CMD register is programmed, the command is sent and the function resumes data transfer. Read the DF flag (Resume Data Flag). If it is 1, then the function has data for the transfer and begins a data transfer as soon as the function or memory is resumed. If it is 0, then the function has no data for the transfer.

i. If the DF flag is 0, the MSI waits for data for a read. After the data timeout period, it gives a data timeout error.

Table 33-16: Parameters for CMDARG Register

CMDARG Bits	Contents	Value
31	R/W flag	1
30–28	Function number	0 = for CCCR access
27	RAW flag	1 = read after write
26	Do-not-care	N/A
25-9	Register address	0x0D
8	Do-not-care	N/A
7–0	Write data	Function number that is to be resumed

Read Wait Sequence

Read_wait is used only with SDIO cards. It can temporarily stall the data transfer either from function or memory and allow the host to send commands to any function within the SDIO device. The host can stall this transfer for as long as required. The MSI provides the facility to signal this stall transfer to the card.

1. Check if the card supports the read_wait facility; read SRW (bit 2) of the CCCR register in SDIO card. If this bit is 1, then all functions in the card support the read_wait facility. Use CMD52 to read this bit.

2. If the card supports the read_wait signal, then assert it by setting the MSI_CTL.RDWAIT bit.
3. Clear the MSI_CTL.RDWAIT bit.

Card Read Threshold

When an application must perform a single or multiple block read command, the application must program the MSI_CDTHRCTL register with the appropriate card read threshold size. It also must set the card MSI_CDTHRCTL.RDTHREN (read threshold enable) bit. This additional programming ensures that the controller sends a read command only if there is space equal to the MSI_CDTHRCTL.RDTHR (card read threshold) available in the Rx FIFO. This programming in turn ensures that the card clock is not stopped in the middle a block of data being transmitted from the card. The card read threshold can be set to the block size of the transfer. This size guarantees that there is a minimum of one block size of space in the Rx FIFO before the controller enables the card clock.

Recommended Usage Guidelines for Card Read Threshold

1. Program the MSI_CDTHRCTL register before the programming the CMD register for a data read command.
2. Do not program the MSI_CDTHRCTL register when a data transfer command is in progress.
3. Program the MSI_CDTHRCTL.RDTHR value greater than or equal to the block size of the read transfer. This programming ensures that the card clock does not stop in between a block of data.
4. If round-trip delay > 0.5 card clock, then enable the card read threshold and program the card threshold as per guideline #3. This programming guarantees that the card clock does not stop in between a block of data. The controller samples data incorrectly if the card clock stops in between a block of data if round-trip delay > 0.5 card clock.
5. MSI_CDTHRCTL.RDTHR is greater than or equal to MSI_BLKSIZ (recommended). Program the card read threshold size (MSI_CDTHRCTL.RDTHR) to at least 1 × BlockSize of the multi-block transfer. This programming guarantees that the card clock does not stop in between a block of data due to the Rx FIFO becoming full during the read transfer.
6. MSI_CDTHRCTL.RDTHR is less than MSI_BLKSIZ. If the MSI_CDTHRCTL.RDTHR bit is programmed to less than the MSI_BLKSIZ of the transfer, then the system must ensure that the receive FIFO never becomes full and overflows during the read transfer. This programming can cause the card clock from the MSI to stop. The MSI is not able to guarantee that the card clock does not stop during a read transfer.

NOTE: If the MSI_CDTHRCTL.RDTHR, MSI_FIFOTH.RXWM, and MSI_FIFOTH.DMAMSZ values are programmed incorrectly, then the card clock can stop indefinitely and no interrupts are generated from the controller.

Card Read Threshold Programming Sequence

Most cards, such as SDHC or SDXC, typically support block sizes that are specified in the card or are fixed to 512 bytes. For SDIO cards, standard capacity SD cards that support READ_BL_PARTIAL =1 and MMC cards, the block size is variable. The application can choose block size.

Use the following steps for the card read threshold feature. The steps guarantee that the card clock does not stop because of a FIFO full condition in the middle of a block of data being read from the card.

1. Choose the block size (configured using the MSI_BLKSIZ register). The block size requested by the application from the card for the read transfer card must be 32-bit aligned.
2. Enable the card read threshold feature. The card read threshold can be enabled only if the block size for the given transfer is less than the total depth of the FIFO (*BlkSiz* ≤ *FifoDepth*)

   Where: *BlkSiz* = (block size in bytes) × 8 ÷ 32; that is, the number of the block size in terms of FIFO locations

   FifoDepth = total number of FIFO locations.
3. Choose the card read threshold.
   - If *BlkSiz* ≥ ½ *FifoDepth*, configure MSI_CDTHRCTL.RDTHR such that the value ≤ *BlkSize* in bytes
   - If *BlkSiz* < ½ *FifoDepth*, configure MSI_CDTHRCTL.RDTHR such that the value = *BlkSize* in bytes
4. Choose the DMA multiple transaction size (DMAMSZ) using the MSI_FIFOTH.DMAMSZ bit field. The possible values for the DMAMSZ are 1, 4, 8, 16, 32, 64, 128, and 256 transfers. Choose the value of DMAMSZ from the transfer values so that BlkSiz is a multiple of DMAMSZ. BlkSize% (DMAMSZ) = 0. Note the following special cases:
   - When a MSIZE transfer =1 (configured using the DMA_CFG.MSIZE bit field). The DMAMSZ is equal to 1 when the block size chosen in Step 1 is not a multiple of the FIFO width (in bytes). If DMAMSZ =1 is not acceptable and a higher burst size is desired-that is, a higher DMAMSZ. Return to Step 1 and recalculate the block size.
   - Internal DMA (IDMAC). The size of the data buffer (in bytes) for each descriptor must be a multiple of DMAMSZ × 4.
5. Choose the RX watermark using the MSI_FIFOTH.RXWM bit field.

   If DMAMSZ = 1, then the RXWM = 1 or RXWM = BlkSize – 1

   Additionally, for all DMA modes the RXWM must be a multiple of the chosen DMAMSZ.

MSI Programming Model, Boot Operation

This section discusses details on programming the MSI for boot operation.

Normal Boot Operation

Normal boot operation is applicable to MMC4.3, MMC4.4, and MMC4.41 cards. It is performed in push-pull mode. The ***Normal Boot Timing*** figure shows the timing for the transmit state machine in a normal boot mode.

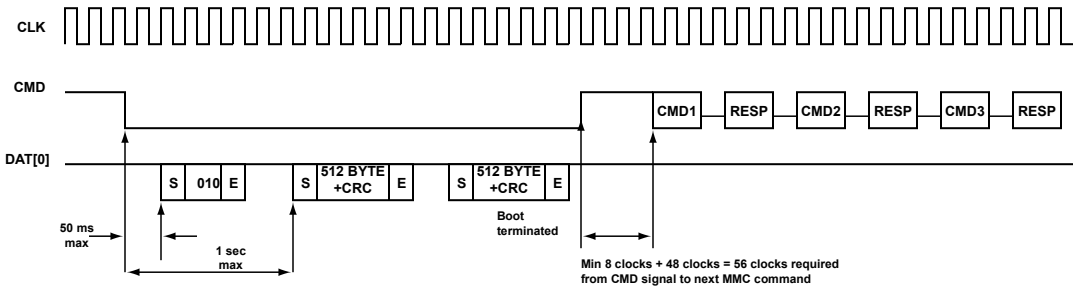

Figure 33-12: Normal Boot Timing

Normal Boot Operation; eMMC

Normal boot operation is applicable to MMC4.3, MMC4.4, and MMC4.41 cards. It must be performed in push-pull mode. See the ***Normal Boot Timing*** figure.

Following are the steps that the software must follow when working with eMMC or removable MMC 4.3 cards for boot operation. Note that the software knows that the card supports boot operation (the BOOT_PARTITION_ENABLE bit set in the card.

The software also knows the BOOT_SIZE_MULT value in the card and the data bus width to use during boot operation: cExtend CSD register byte [177] bit[0:1].

1. Set the masks for interrupts by clearing appropriate bits in the `MSI_IMSK` register.
2. Set the `MSI_CTL.INTEN` bit. Before this bit is set, programs should write 0xFFFF_FFFF to the `MSI_ISTAT` and `MSI_IDSTS` registers to clear any pending interrupts. For internal DMAC, unmask all the relevant fields in the `MSI_IDINTEN` register.
3. Set the and `MSI_CTL.INTEN` bits =1. In order to use the internal DMAC to transfer the boot data received, it must also set up the descriptors and program the `MSI_CTL.INTDMAC` bit =1.
4. Set the card frequency to 400 kHz using the clock-divider register.
5. Set Data Time Out = (10 × ((TAAC × f_{OP}) + (100 × NSAC)); this is NAC.
6. Program the `MSI_BLKSIZ` register with 0x200 (512 bytes).
7. Program the `MSI_BYTCNT` register with multiples of 128K bytes, as indicated by the BOOT_SIZE_MULT value in the card.
8. Program the Rx FIFO threshold value in bytes in the `MSI_FIFOTH` register. Typically, the threshold value can be set to half the FIFO depth.
9. Program the following fields: `MSI_CMD.STARTCMD` =1'b1, `MSI_CMD.BOOTEN` =1'b1, `MSI_CMD.XPECTBOOTACK` (for start-acknowledge pattern ACK from the card) and `MSI_CMD.DXPECT` =1'b1.
10. If `MSI_CMD.XPECTBOOTACK` =1'b1, the software driver must start a timer after step 9; the terminal value is 50 ms.

- Before this timer elapses, the BAR interrupt should be received from the MSI. If this action does not occur, program the CMD register as follows:

  `MSI_CMD.STARTCMD` =1'b1

  `MSI_CMD.BOOTDIS` =1'b1

  All other fields = 0

- The MSI generates a Command Done (CD) interrupt after de-asserting the CMD line of the card. In IDMAC:

  Descriptor is closed

  `MSI_IDSTS.CES` =1, indicating BAR timeout

  `MSI_IDSTS.RI` =0

- If the BAR interrupt is received, the software should clear this interrupt by writing a 1 to it. The software should then start another timer with a terminal value of 1 – 0.05 = 0.95 seconds. Before this timer elapses, the BDS interrupt should be received from the MSI. If this action does not occur, the software driver must program the CMD register as follows:

  `MSI_CMD.STARTCMD` =1'b1

  `MSI_CMD.BOOTDIS` =1'b1

  All other fields =0

- The MSI generates a CD interrupt after de-asserting the CMD line of the card. In IDMAC:

  Descriptor is closed

  `MSI_IDSTS.CES` =1, indicating BDS timeout

  `MSI_IDSTS.RI` =0

- If the BDS interrupt is received, it indicates that the boot data is being received from the card. The IDMAC engine starts transferring the data from the FIFO to the system memory as soon as the programmed RX_WMark level is attained. At the end of a successful boot data transfer from the card, the following interrupts are generated:

  Command Done (CD) with the `MSI_ISTAT.CMDDONE` bit

  Data Transfer Over (DTO) with the `MSI_ISTAT.DTO` bit

  Receive Interrupt (RI) with the `MSI_IDSTS.RI` bit

- If an error occurs in Boot Ack pattern (010) or an end bit error occurs

  Controller automatically aborts boot by pulling CMD line high

  Controller generates CD interrupt

Controller does not generate BAR interrupt

Application aborts boot transfer

- In IDMAC:

  If the software creates more descriptors than the boot data received, the MSI does not close the extra descriptors.

  If the software creates less descriptors than the boot data received, the MSI generates a Descriptor Unavailable (DU) interrupt and does not transfer any further data to system memory.

- If between data block transfers NAC is violated, DRTO (Data Read Timeout) is asserted. If there are errors associated with Start or End bits, SBE/EBE interrupts are also generated.

11. If `MSI_CMD.XPECTBOOTACK` =1'b0, the software should start a timer after the step 9 where the terminal value is 1 second.

    - Before this timer elapses, a BDS interrupt should be received from the MSI. If the interrupt is not received, the software must program their CMD register with the following fields:

      `MSI_CMD.STARTCMD` =1'b1

      `MSI_CMD.BOOTDIS` =1'b1

      All other fields =0

    ADDITIONAL INFORMATION: MSI generates a CD interrupt after de-asserting the CMD line of the card. In IDMAC mode, the descriptor is closed and the `MSI_IDSTS.CES` bit =1, indicating a BDS timeout.

    - If a BDS interrupt is received, it indicates that the boot data is being received from the card. At the end of a successful boot data transfer from card, the following interrupts are generated.

      Command Done (CD) with the `MSI_ISTAT.CMDDONE` bit

      Data Transfer Over (DTO) with the `MSI_ISTAT.DTO` bit

      Receive Interrupt (RI) with the `MSI_IDSTS.RI` bit

Normal Boot Operation; Removable MMC4.3, MMC4.4, and MMC4.41 Cards

Removable MMC4.3, MMC4.4, and MMC4.41 cards are different than eMMC in that the software is not aware whether these cards support the boot mode of operation when plugged in. Thus the software must:

- Enumerate these cards as it would enumerate MMC4.0/4.1/4.2 cards for the first time
- Know the card characteristics
- Decide whether to perform a boot operation or not

Use the following procedure when booting with these card types,

1. Enumerate the card.

2. Read the EXT_CSD register of the card and examine the following fields:
   - BOOT_PARTITION_ENABLE
   - BOOT_SIZE_MULT
   - BOOT_INFO
3. If necessary, the controller can manipulate the boot information in the card.
4. If the controller must perform a boot operation at the next power-up cycle, it can manipulate the EXT_CSD register contents by using a SWITCH (CMD6) command.
5. From this point, use the same steps as in Normal Boot Operation; eMMC.

Alternate Boot Operation; eMMC

The alternate boot operation differs from the normal boot operation in that CMD0 is used to boot the card rather than holding down the CMD-line of the card. The alternate boot operation can occur only if bit 0 in the extended CSD byte[228] (BOOT_INFO) is set to 1.

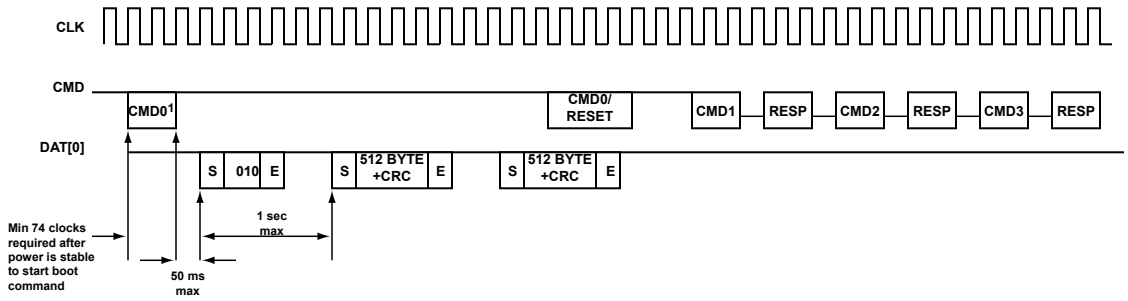

Figure 33-13: Alternate Boot Timing

Use the following procedure when working with eMMC or removable MMC 4.3 cards for the alternative boot operation. The software is aware that the card supports the Alternate Boot operation—the BOOT_INFO bit is set in the card. Also, the software is aware of the BOOT_SIZE_MULT value in the card and the data bus width to use during the boot operation-extend CSD register byte[177] bit[0:1].

1. Set the masks for interrupts by clearing appropriate bits in the Interrupt Mask register.
2. Set the `MSI_CTL.INTEN` bit =1.

   ADDITIONAL INFORMATION: Set the `MSI_CTL.INTEN` bit. Before this bit is set, programs should write 0xFFFF_FFFF to the `MSI_ISTAT` and `MSI_IDSTS` registers to clear any pending interrupts. Software also must unmask all the relevant fields in `MSI_IDINTEN` register.
3. Configure the following bits in the control register.
   a. `MSI_CTL.INTEN` = 1'b1
   b. Other fields are 1'b0

c. For IDMAC, set up the descriptors.

d. Program the MSI_CTL.INTDMAC bit to 1

4. Set the card frequency to 400 kHz using the clock-divider register

   ADDITIONAL INFORMATION: Wait for a time that ensures that at least 74 card clock cycles have occurred on the card interface.

5. Set Data Time Out = (10 × ((TAAC × f_{OP}) + (100 × NSAC)); this is NAC

6. Program the MSI_BLKSIZ register with 0x200 (512 bytes).

7. Program the MSI_BYTCNT register with multiples of 128K bytes, as indicated by the BOOT_SIZE_MULT value in the card

8. Program the Rx FIFO threshold value in bytes in the MSI_FIFOTH register. Typically, the threshold value is set to half the FIFO depth; that is, MSI_FIFOTH.RXWM = (FIFO_DEPTH/2) – 1.

9. Program CMDARG = 0xFFFFFFFA.

10. Program the following fields.

    a. MSI_CMD.STARTCMD = 1'b1

    b. MSI_CMD.BOOTMODE= 1'b1

    c. MSI_CMD.XPECTBOOTACK depending on whether a start-acknowledge pattern is expected from the card.

    d. MSI_CMD.DXPECT = 1'b1

    e. MSI_CMD.INDX = 0

    f. Remainder of MSI_CMD register fields = 1'b0

    ADDITIONAL INFORMATION: Wait for the Command Done (CD) interrupt.

11. If MSI_CMD.XPECTBOOTACK = 1'b1 in step 10, the software driver must start a timer with a terminal value of 50 ms.

    a. Before this timer elapses, the BAR interrupt should be received from the MSI. If this action does not occur, the software must infer that the start-pattern has not been received and must discontinue the boot process and start with normal enumeration. In IDMAC:

       - Descriptor is closed
       - MSI_IDSTS.CES =1, indicating BAR timeout
       - MSI_IDSTS.RI =0

    b. If the BAR interrupt is received, the software driver should clear this interrupt by writing a 1 to it. The software driver then starts another timer with a terminal value of 1 - 0.05 = 0.95 seconds. Before this timer elapses, the BDS interrupt should be received from the MSI. If this action does not occur, the software driver discontinues the boot process and start with normal enumeration.

- Descriptor is closed
- MSI_IDSTS.CES =1, indicating BDS timeout
- MSI_IDSTS.RI =0

c. If the BDS interrupt is received, it indicates that the boot data is being received from the card. The IDMAC engine starts transferring the data from the FIFO to the system memory as soon as the programmed MSI_FIFOTH.RXWM level is hit.

d. It is the responsibility of the software driver to terminate the boot operation by programming the MSI to send a CMD0 by programming the registers MSI_CMDARG =0 and the command register bits MSI_CMD.STARTCMD =1, MSI_CMD.INDX =0, all_other_fields = 0.

e. At the end of a successful boot data transfer from the card, the following status bits are set:

- Command Done (CD) with the MSI_ISTAT.CMDDONE bit
- Data Transfer Over (DTO) with the MSI_ISTAT.DTO bit
- Receive Interrupt (RI) with the MSI_IDSTS.RI bit

f. If an error occurs in Boot Ack pattern (010) or an end bit error occurs:

- Controller does not generate BAR interrupt
- Controller detects boot data start and generates BDS interrupt
- Controller continues to receive boot data
- Application must abort boot after receiving BDS interrupt

g. In IDMAC:

- If the software driver creates more descriptors than the boot data received, the extra descriptors are not closed by the MSI.
- If the software driver creates less descriptors than the boot data received, the MSI generates a Descriptor Unavailable (DU) interrupt and does not transfer any further data to system memory.

h. If between data block transfers NAC is violated, DRTO (Data Read Timeout) is asserted. Apart from this, if there are errors associated with Start/End bits, SBE/EBE interrupts are also generated.

12. If MSI_CMD.XPECTBOOTACK =1'b0 in Step 10, the software should start a timer after Step 10with a terminal value of 1 second.

a. Before this timer elapses, the BDS interrupt should be received from the MSI. If this does not occur, the software driver should discontinue the boot process and start with normal enumeration. In IDMAC:

- Descriptor is closed
- MSI_IDSTS.CES =1, indicating BDS timeout
- MSI_IDSTS.RI =0

b. If the BDS interrupt is received, it indicates that the boot data is being received from the card. The IDMAC engine starts transferring the data from the FIFO to the system memory as soon as the programmed MSI_FIFOTH.RXWM level is hit.

c. It is the responsibility of the software to terminate the boot operation. Software programs the MSI to send a CMD0 through the MSI_CMDARG register =0 and command register bits MSI_CMD.STARTCMD =1, MSI_CMD.INDX =0, and all_other_fields =0.

d. At the end of a successful boot data transfer from card, the following interrupts are generated.

- Command Done (CD) with the MSI_ISTAT.CMDDONE bit
- Data Transfer Over (DTO) with the MSI_ISTAT.DTO bit
- Receive Interrupt (RI) with the MSI_IDSTS.RI bit

e. In IDMAC

- If the software driver creates more descriptors than the boot data received, the MSI does not close the extra descriptors.
- If the software driver creates less descriptors than the boot data received, the MSI generates a Descriptor Unavailable (DU) interrupt and does not transfer any further data to system memory.

Alternate Boot Operation; Removable MMC4.3 Card

Removable MMC4.3 cards are different than eMMC in that software is not aware whether these cards support the boot mode of operation. The software must:

- Enumerate these cards as it would enumerate MMC4.0/4.1/4.2 cards for the first time
- Know the card characteristics
- Decide whether to perform a boot operation

Use the following procedure when working with removable MMC 4.3 cards for the Alternative Boot operation.

1. Enumerate the card.
2. Read the EXT_CSD register of the card and examine the following fields:
   a. BOOT_PARTITION_ENABLE
   b. BOOT_SIZE_MULT
   c. BOOT_INFO
3. If necessary, the controller can also manipulate the boot information in the card.
4. If the host controller must perform a boot operation at the next power-up cycle, it can manipulate the EXT_CSD register contents by using a SWITCH (CMD6) command.
5. From this point, use the same steps as in Alternate Boot Operation; eMMC.

ADSP-BF70x MSI Register Descriptions

Mobile Storage Interface (MSI) contains the following registers.

Table 33-17: ADSP-BF70x MSI Register List

Name	Description
MSI_BLKSIZ	Block Size Register
MSI_BUFADDR	Current Buffer Descriptor Address Register
MSI_BUSMODE	Bus Mode Register
MSI_BYTCNT	Byte Count Register
MSI_CDETECT	Card Detect Register
MSI_CDTHRCTL	Card Threshold Control Register
MSI_CLKDIV	Clock Divider Register
MSI_CLKEN	Clock Enable Register
MSI_CMD	Command Register
MSI_CMDARG	Command Argument Register
MSI_CTL	Control Register
MSI_CTYPE	Card Type Register
MSI_DBADDR	Descriptor List Base Address Register
MSI_DEBNCE	Debounce Count Register
MSI_DSCADDR	Current Host Descriptor Address Register
MSI_ENSHIFT	Enable Phase Shift Register
MSI_FIFOTH	FIFO Threshold Watermark Register
MSI_IDINTEN	Internal DMA Interrupt Enable Register
MSI_IDSTS	Internal DMA Status Register
MSI_IMSK	Interrupt Mask Register
MSI_ISTAT	Raw Interrupt Status Register
MSI_MSKISTAT	Masked Interrupt Status Register
MSI_PLDMND	Poll Demand Register
MSI_RESP0	Response Register 0
MSI_RESP1	Response Register 1
MSI_RESP2	Response Register 2
MSI_RESP3	Response Register 3
MSI_STAT	Status Register
MSI_TBBCNT	Transferred Host to BIU-FIFO Byte Count Register

Table 33-17: ADSP-BF70x MSI Register List (Continued)

Name	Description
MSI_TCBCNT	Transferred CIU Card Byte Count Register
MSI_TMOUT	Timeout Register
MSI_UHS_EXT	Ultra High Speed Register Extension

Block Size Register

The MSI_BLKSIZ register provides bits that configure data block sizes. Sizes supported are 1 to 65,535 bytes.

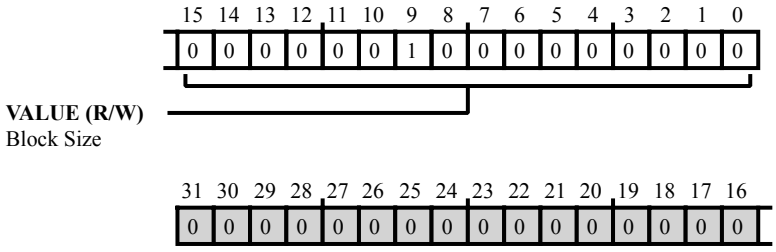

Figure 33-14: MSI_BLKSIZ Register Diagram

Table 33-18: MSI_BLKSIZ Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Block Size. The MSI_BLKSIZ.VALUE bit field configures data block size in bytes. Sizes supported are 1 to 65,535 bytes.

Current Buffer Descriptor Address Register

The MSI_BUFADDR register points to the data buffer address of the current descriptor read by the IDMAC.

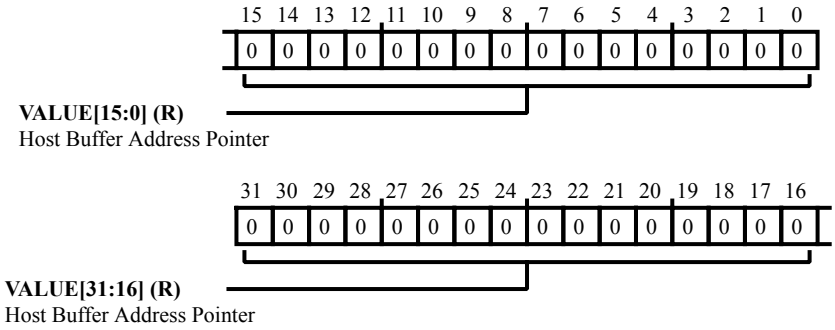

Figure 33-15: MSI_BUFADDR Register Diagram

Table 33-19: MSI_BUFADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Host Buffer Address Pointer. The MSI_BUFADDR.VALUE bit field points to the data buffer address of the current descriptor read by the IDMAC.

Bus Mode Register

The MSI_BUSMODE register provides bits that control the burst mode, descriptor skip length, and IDMAC. This register also provides a software reset bit.

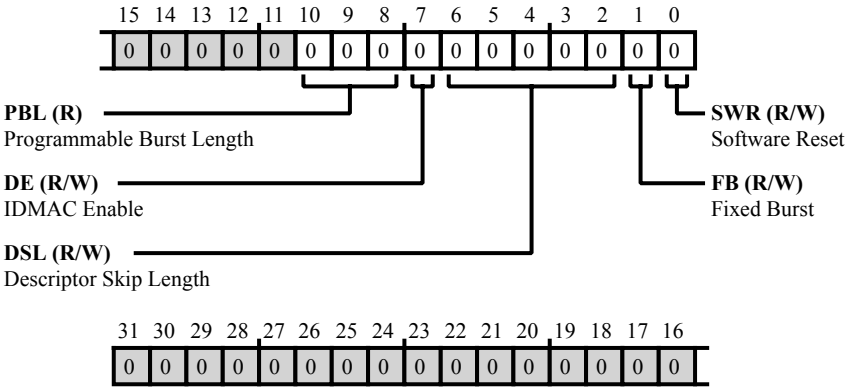

Figure 33-16: MSI_BUSMODE Register Diagram

Table 33-20: MSI_BUSMODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
10:8 (R/NW)	PBL	Programmable Burst Length. The MSI_BUSMODE.PBL bit field indicate the maximum number of beats to be performed in one IDMAC transaction. The IDMAC always attempts to burst as specified in PBL each time it starts a burst transfer on the host bus. This value is the mirror of the MSI_FIFOTH.DMAMSZ bits. In order to change this value, write the required value to MSI_FIFOTH.DMAMSZ bit field. The units for transfer are 32-bit.	
		0	1 transfer
		1	4 transfers
		2	8 transfers
		3	16 transfers
		4	32 transfers
		5	64 transfers
		6	128 transfers
		7	256 transfers
7 (R/W)	DE	IDMAC Enable. Setting the MSI_BUSMODE.DE bit enables the internal DMA interface.	
		0	Disable internal DMA
		1	Enable internal DMA
6:2	DSL	Descriptor Skip Length.	

Table 33-20: MSI_BUSMODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
(R/W)		The MSI_BUSMODE.DSL bit field specifies the number of words to skip between two unchained descriptors. This is applicable only for dual buffer structures.	
1 (R/W)	FB	Fixed Burst. The MSI_BUSMODE.FB bit controls whether the peripheral bus master interface performs fixed burst transfers or not. When set, the peripheral bus uses only SINGLE, INCR4, INCR8 or INCR16 during the start of normal burst transfers. When reset, the peripheral bus uses SINGLE and INCR burst transfer operations.	
		0	Use SINGLE and INCR
		1	Use SINGLE, INCR4, INCR8 or INCR16
0 (R/W)	SWR	Software Reset. When the MSI_BUSMODE.SWR bit is set, the DMA controller resets all its internal registers. The MSI_BUSMODE.SWR bit is automatically cleared after 1 clock cycle.	
		0	No reset
		1	Reset internal registers

Byte Count Register

The MSI_BYTCNT register provides bits that configure the byte count to be transferred.

In SDIO mode, if a single transfer is greater than 4 bytes and non-DWORD-aligned, the transfer should be broken where only the last transfer is non-DWORD-aligned and less than 4 bytes. For example, if a transfer of 129 bytes must occur, then the driver should start at least two transfers; one with 128 bytes and the other with 1 byte.

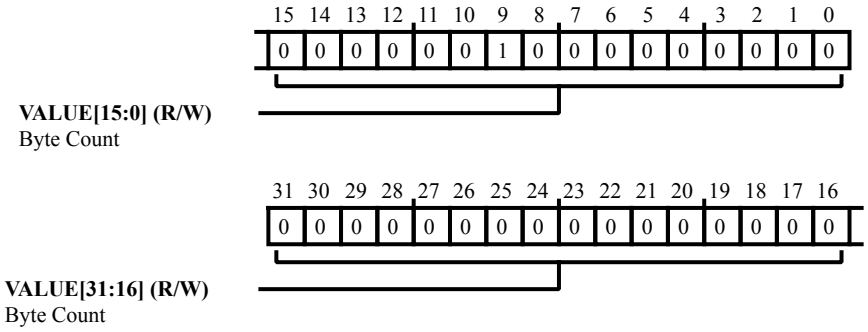

Figure 33-17: MSI_BYTCNT Register Diagram

Table 33-21: MSI_BYTCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Byte Count. The MSI_BYTCNT.VALUE bit field provides bits that configure the byte count to be transferred.

Card Detect Register

The MSI_CDETECT register configures the value on card_detect_n input ports (1 bit per card).

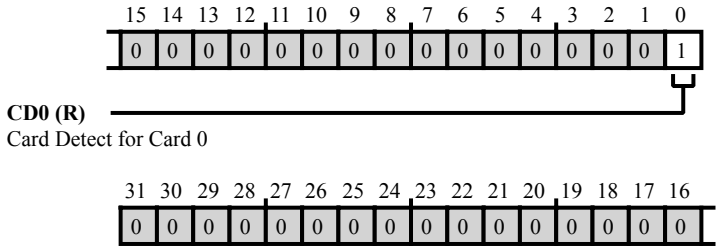

Figure 33-18: MSI_CDETECT Register Diagram

Table 33-22: MSI_CDETECT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/NW)	CD0	Card Detect for Card 0. The MSI_CDETECT.CD0 bit sets the value on card_detect_n input ports.

Card Threshold Control Register

The MSI_CDTHRCTL register sets the card read threshold size and enables card read threshold. This register also has a Busy Clear interrupt generation bit.

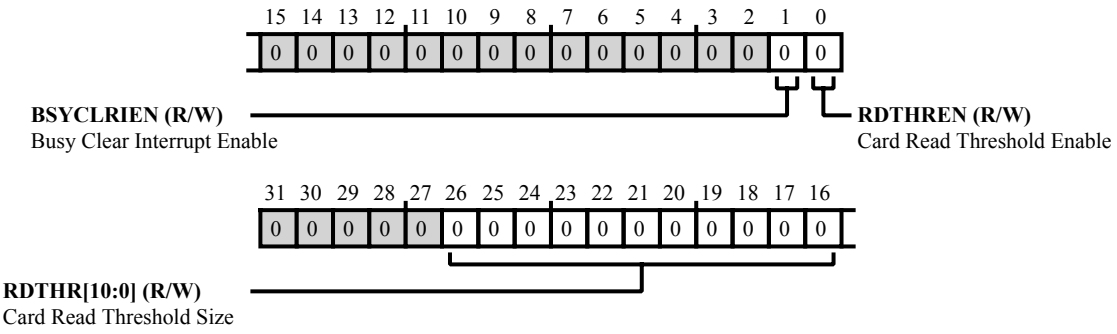

Figure 33-19: MSI_CDTHRCTL Register Diagram

Table 33-23: MSI_CDTHRCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26:16 (R/W)	RDTHR	Card Read Threshold Size. The MSI_CDTHRCTL.RDTHR bit configures the card read threshold size. For information on using this feature, see the "Card Read Threshold" section in the MSI chapter.	
1 (R/W)	BSYCLRIEN	Busy Clear Interrupt Enable. The MSI_CDTHRCTL.BSYCLRIEN bit indicates the completion of a busy driven by the card after a write data transfer.	
		0	Busy clear interrupt disabled
		1	Busy clear interrupt enabled
0 (R/W)	RDTHREN	Card Read Threshold Enable. The MSI_CDTHRCTL.RDTHREN bit enables the card read threshold size feature.	
		0	Card read threshold disabled
		1	Card read threshold enabled

Clock Divider Register

The MSI_CLKDIV register provides clock divider bit fields. The bit field value is: clock division = 2 x n.

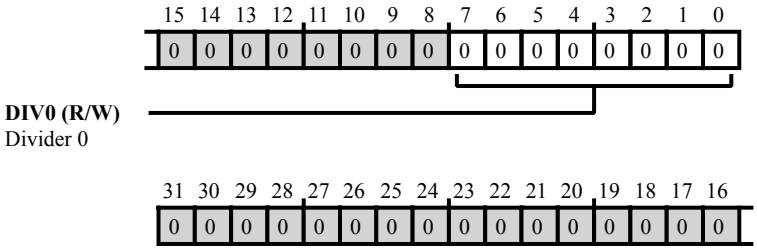

Figure 33-20: MSI_CLKDIV Register Diagram

Table 33-24: MSI_CLKDIV Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/W)	DIV0	Divider 0. The MSI_CLKDIV.DIV0 bit field provides clock division. The value is 2 x n. For example, a value of 0 means divide by 2 x 0 = 0 (no division, bypass), a value of 1 means divide by 2 x 1 = 2, and a value of 0xFF means divide by 2 x 255 = 510, and so on.

Clock Enable Register

The MSI_CLKEN register enables clock control. This register also provides low-power control for the card clock.

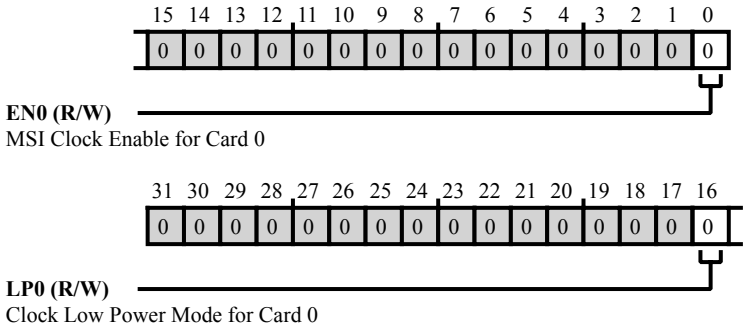

Figure 33-21: MSI_CLKEN Register Diagram

Table 33-25: MSI_CLKEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	LP0	Clock Low Power Mode for Card 0. Setting the MSI_CLKEN.LP0 bit puts the card clock into low-power mode; it stops the clock when the card is in IDLE (it should be normally set to only MMC and SD memory cards. For SDIO cards, if interrupts must be detected, the clock should not be stopped). Clearing the MSI_CLKEN.LP0 bit puts the cards into non-low power mode.
0 (R/W)	EN0	MSI Clock Enable for Card 0. The MSI_CLKEN.EN0 bit is the clock-enable control for the card clock.

Command Register

The MSI_CMD register provides bits that configure various command parameters such as boot modes, data transfer modes, and command response settings.

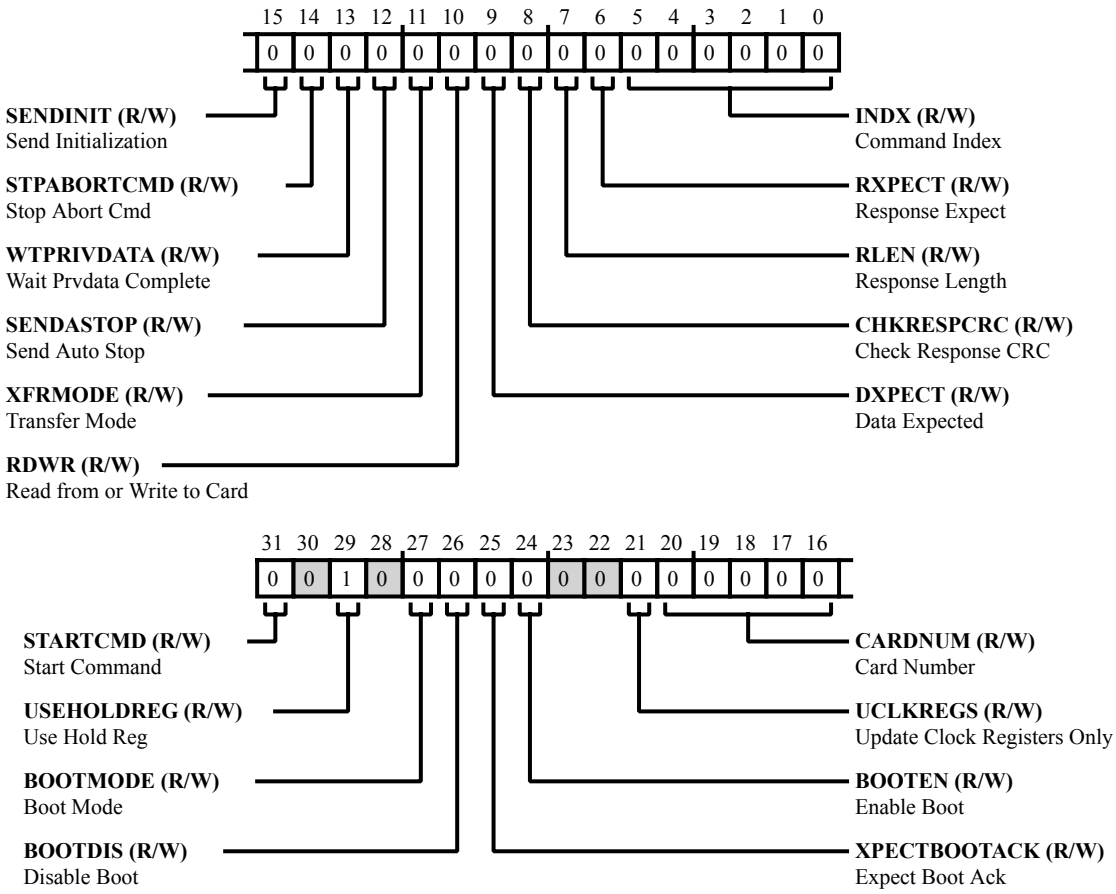

Figure 33-22: MSI_CMD Register Diagram

Table 33-26: MSI_CMD Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/W)	STARTCMD	Start Command. Once the command is taken by CIU, the MSI_CMD.STARTCMD bit is cleared. When this bit is set, the host should not attempt to write to any command registers. If a write is attempted, a hardware lock error is set in the raw interrupt register. Once the command is sent and a response is received from the SD_MMC cards, the Command Done bit is set in the raw interrupt register.

Table 33-26: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	USEHOLDREG	Use Hold Reg. When the MSI_CMD.USEHOLDREG bit is set, CMD and DATA are sent to the card through the HOLD register. When the MSI_CMD.USEHOLDREG bit is cleared, CMD and DATA are sent to the card bypassing the HOLD register.	
		0	CMD and DATA sent to card bypassing the HOLD register
		1	CMD and DATA sent to card through the HOLD register
27 (R/W)	BOOTMODE	Boot Mode. When the MSI_CMD.BOOTMODE bit is set, the MSI has an alternate boot operation. When the MSI_CMD.BOOTMODE bit is cleared, the MSI has a mandatory boot operation.	
		0	Mandatory Boot
		1	Alternate Boot
26 (R/W)	BOOTDIS	Disable Boot. When the MSI_CMD.BOOTDIS bit is set with the MSI_CMD.STARTCMD bit, the CIU terminates the boot operation. Do NOT set MSI_CMD.BOOTDIS and MSI_CMD.BOOTEN together.	
		0	No action
		1	Terminate boot
25 (R/W)	XPECTBOOTACK	Expect Boot Ack. When the MSI_CMD.XPECTBOOTACK bit is set with with the MSI_CMD.BOOTEN bit, the CIU expects a boot acknowledge start pattern of 0-1-0 from the selected card.	
		0	No ACK expected
		1	ACK expected
24 (R/W)	BOOTEN	Enable Boot. The MSI_CMD.BOOTEN bit should be set only for mandatory boot mode. When software sets this bit along with the MSI_CMD.STARTCMD bit, CIU starts the boot sequence for the corresponding card by asserting the CMD line low. Do NOT set the MSI_CMD.BOOTEN bit and the MSI_CMD.BOOTDIS bit together.	

Table 33-26: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
21 (R/W)	UCLKREGS	Update Clock Registers Only. When the MSI_CMD.UCLKREGS bit is set, do not send commands. Update the clock register value into card clock domain. When cleared, the normal command sequence is used. The following register values are transferred into card clock domain: CLKDIV, and CLKENA. Changes card clocks (change frequency, truncate off or on, and set low-frequency mode); provided in order to change clock frequency or stop clock without having to send command to cards. During the normal command sequence, when update_clock_registers_only = 0, the following control registers are transferred from BIU to CIU: CMD, CMDARG, TMOUT, CTYPE, BLKSIZ, BYTCNT. CIU uses new register values for new command sequence to card(s). When the MSI_CMD.UCLKREGS bit is set, there are no Command Done interrupts because no command is sent to SD_MMC cards.	
		0	Normal command sequence
		1	Do not send commands, just update clock register value
20:16 (R/W)	CARDNUM	Card Number. The MSI_CMD.CARDNUM bit represents the physical slot number of the card being accessed.	
15 (R/W)	SENDINIT	Send Initialization. When the MSI_CMD.SENDINIT bit is set, send an initialization sequence before sending this command. When the MSI_CMD.SENDINIT bit is cleared, do not send an initialization sequence (80 clocks of 1) before sending this command. After power-on, 80 clocks must be sent to the card for initialization before sending any commands to the card. The bit should be set while sending the first command to the card, so that the controller will initialize clocks before sending a command to the card. This bit should not be set for either of the boot modes (alternate or mandatory).	
		0	Do not send initialization sequence
		1	Send initialization sequence

Table 33-26: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	STPABORTCMD	Stop Abort Cmd. When the MSI_CMD.STPABORTCMD bit is set, the stop or abort command is intended to stop the current data transfer in progress. When an open-ended or predefined data transfer is in progress, and the host issues a stop or abort command to stop the data transfer, the MSI_CMD.STPABORTCMD bit should be set, so that the command/data state-machines of CIU can return correctly to an idle state. This is also applicable for boot mode transfers. To abort boot mode, this bit should be set along with CMD[26] = disable_boot. When cleared, neither stop nor abort command to stop current data transfer in progress. If abort is sent to function-number currently selected or not in data-transfer mode, then the bit should be set to 0.	
		0	Neither stop nor abort command
		1	Stop or abort command
13 (R/W)	WTPRIVDATA	Wait Prvdata Complete. When the MSI_CMD.WTPRIVDATA bit is set, wait for the previous data transfer to complete before sending a command. When the MSI_CMD.WTPRIVDATA bit is cleared, send the command at once, even if the previous data transfer has not completed. The MSI_CMD.WTPRIVDATA = 0 option is typically used to query the status of the card during data transfer or to stop the current data transfer; the card number should be the same as in the previous command.	
		0	Send command at once
		1	Wait for previous data transfer completion
12 (R/W)	SENDASTOP	Send Auto Stop. When the MSI_CMD.SENDASTOP bit is set, MSI sends the stop command to SD_MMC cards at the end of the data transfer. Refer to the Auto Stop section in the chapter to determine: • When the MSI_CMD.SENDASTOP bit should be set, since some data transfers do not need explicit stop commands • When software should explicitly send a stop command during open-ended transfers Additionally, when resuming suspended memory access of SD-Combo card, the MSI_CMD.SENDASTOP bit should be set correctly to send a stop command. The bit does not need to be set if no data is expected from the card.	
		0	No stop command sent at end of data transfer
		1	Send stop command at end of data transfer

Table 33-26: MSI_CMD Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
11 (R/W)	XFRMODE	Transfer Mode. When the MSI_CMD.XFRMODE bit is set, stream data transfer command. If no data expected, do-not-care.	
		0	Block data transfer command
		1	Stream data transfer command
10 (R/W)	RDWR	Read from or Write to Card. When the MSI_CMD.RDWR bit is set, write to the card. When the MSI_CMD.RDWR bit is cleared, read from the card.	
		0	Read from card
		1	Write to card
9 (R/W)	DXPECT	Data Expected. When the MSI_CMD.DXPECT bit is set, data transfer is expected. When the MSI_CMD.DXPECT bit is cleared, no data transfer is expected.	
		0	No data transfer expected
		1	Data transfer expected
8 (R/W)	CHKRESPCRC	Check Response CRC. When the MSI_CMD.CHKRESPCRC bit is set, check the response CRC. When cleared, do not check the response. Some of the command responses do not return valid CRC bits. Software should disable CRC checks for those commands in order to disable CRC checking by controller.	
		0	Do not check response
		1	Check response
7 (R/W)	RLEN	Response Length. When the MSI_CMD.RLEN bit is set, a long response is expected from the card. When cleared, a short response is expected.	
		0	Short response expected
		1	Long response expected
6 (R/W)	RXPECT	Response Expect. When the MSI_CMD.RXPECT bit is set, a response is expected from the card. When cleared, no response is expected.	
		0	No response expected
		1	Response expected
5:0 (R/W)	INDX	Command Index.	

Command Argument Register

The MSI_CMDARG register provides bits that specify the command argument to be passed to the card.

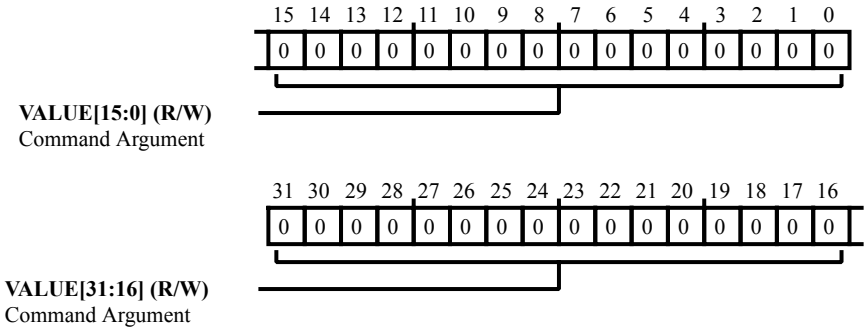

Figure 33-23: MSI_CMDARG Register Diagram

Table 33-27: MSI_CMDARG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Command Argument. The MSI_CMDARG.VALUE bit field specifies the command argument to be passed to the card.

Control Register

The MSI_CTL register controls the various settings used by the MSI module.

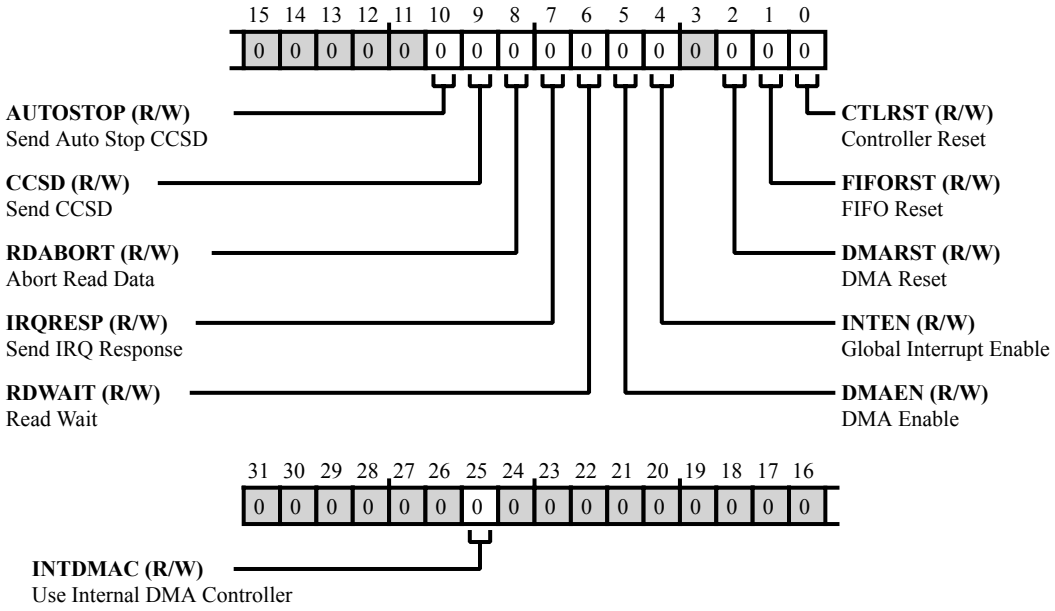

Figure 33-24: MSI_CTL Register Diagram

Table 33-28: MSI_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	INTDMAC	Use Internal DMA Controller. The MSI_CTL.INTDMAC bit is present only for the internal DMAC configuration and determines whether the host or DMA is used for data transfers.	
		0	Host performs transfers
		1	DMA performs transfers
10 (R/W)	AUTOSTOP	Send Auto Stop CCSD. The MSI_CTL.AUTOSTOP bit sends an internally-generated STOP command.	
		0	Clear bit if MSI does not reset the bit
		1	Send internally generated STOP command
9 (R/W)	CCSD	Send CCSD. When set, the MSI_CTL.CCSD bit sends a command completion signal disable.	
		0	Clear bit if the MSI does not reset the bit
		1	Send Command Completion Signal Disable

Table 33-28: MSI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
8 (R/W)	RDABORT	Abort Read Data. Setting the `MSI_CTL.RDABORT` bit after the suspend command is issued during a read-transfer operation, software polls the card to find out when the suspend happened. Once the suspend occurs, software sets the bit to reset data state-machine, which is waiting for next block of data. The bit automatically clears once the data state machine resets to idle. It is used in the SDIO card suspend sequence.	
		0	No change
		1	Reset data state machine
7 (R/W)	IRQRESP	Send IRQ Response. Setting the `MSI_CTL.IRQRESP` bit sends an auto IRQ response. It is cleared once the response is sent. To wait for MMC card interrupts, the host issues CMD40, and the MSI waits for an interrupt response from the MMC card(s). In the meantime, if the host wants the MSI to exit waiting for interrupt state, it can set the `MSI_CTL.IRQRESP` bit, at which time the MSI command state-machine sends the CMD40 response on the bus and returns to the idle state.	
		0	No change
		1	Send auto IRQ response
6 (R/W)	RDWAIT	Read Wait. Setting the `MSI_CTL.RDWAIT` bit allows sending read-wait to SDIO cards.	
		0	Clear read wait
		1	Assert read wait
5 (R/W)	DMAEN	DMA Enable. Setting the `MSI_CTL.DMAEN` bit enables DMA transfer mode.	
		0	Disable
		1	Enable
4 (R/W)	INTEN	Global Interrupt Enable. Setting the `MSI_CTL.INTEN` bit enables global interrupts. The int port is 1 only when this bit is 1 and one or more unmasked interrupts are set.	
		0	Disable
		1	Enable

Table 33-28: MSI_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	DMARST	DMA Reset. Setting the `MSI_CTL.DMARST` bit resets internal DMA interface control logic. To reset the DMA interface, firmware should set this bit to 1. This bit is auto-cleared after two peripheral clocks.	
		0	No change
		1	Reset
1 (R/W)	FIFORST	FIFO Reset. Setting the `MSI_CTL.FIFORST` bit provides a reset to data FIFO to reset FIFO pointers. To reset FIFO, firmware should set bit to 1. This bit is auto-cleared after completion of reset operation.	
		0	No change
		1	Reset
0 (R/W)	CTLRST	Controller Reset. Setting the `MSI_CTL.CTLRST` bit resets the MSI. To reset the controller, firmware should set the bit to 1. This bit is auto-cleared after two peripheral bus and two cclk_in clock cycles. This resets the: • BIU/CIU interface • CIU and state machines • `MSI_CTL.RDABORT`, `MSI_CTL.IRQRESP`, and `MSI_CTL.RDWAIT` bits • `MSI_CMD.STARTCMD` bit It does not affect any registers or DMA interface, or FIFO or host interrupts.	
		0	No change
		1	Reset

Card Type Register

The MSI_CTYPE register provides bits that configure card widths.

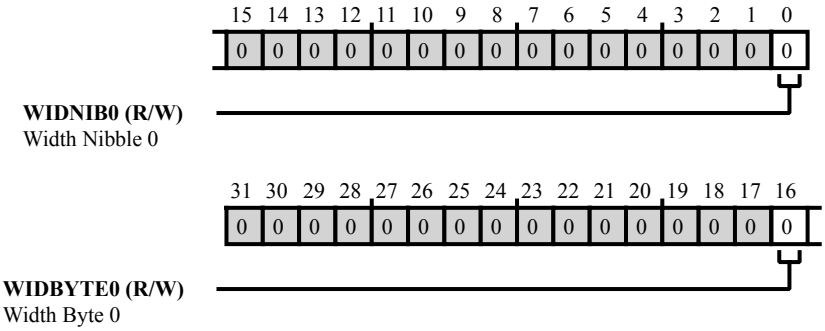

Figure 33-25: MSI_CTYPE Register Diagram

Table 33-29: MSI_CTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
16 (R/W)	WIDBYTE0	Width Byte 0. The MSI_CTYPE.WIDBYTE0 bit enables 8-bit mode for card 0.
0 (R/W)	WIDNIB0	Width Nibble 0. The MSI_CTYPE.WIDNIB0 bit enables 4-bit mode for card 0.

Descriptor List Base Address Register

The MSI_DBADDR register contains the base address of the first descriptor. The LSB bits [1:0] bits are ignored and taken as all-zero by the IDMAC internally and these LSB bits are read-only.

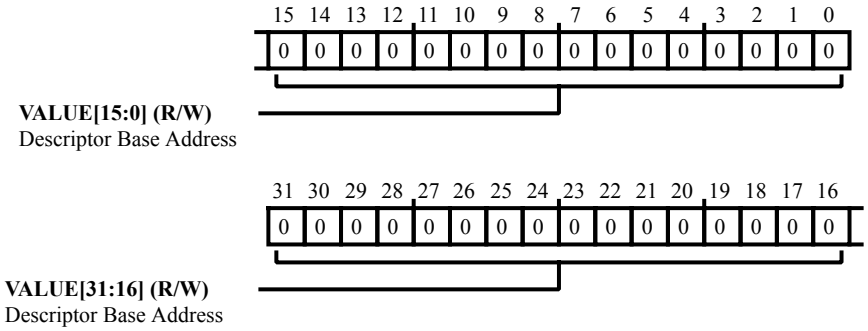

Figure 33-26: MSI_DBADDR Register Diagram

Table 33-30: MSI_DBADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Descriptor Base Address. The MSI_DBADDR.VALUE bit field contains the base address of the first descriptor.

Debounce Count Register

The MSI_DEBNCE register provides the number of host clocks (clk) used by debounce filter logic; typical debounce time is 5-25 ms.

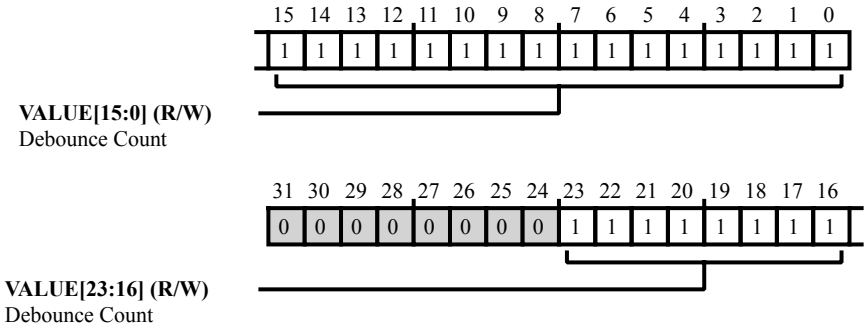

Figure 33-27: MSI_DEBNCE Register Diagram

Table 33-31: MSI_DEBNCE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
23:0 (R/W)	VALUE	Debounce Count. The MSI_DEBNCE.VALUE bit field provides the number of host clocks (clk) used by debounce filter logic.

Current Host Descriptor Address Register

The MSI_DSCADDR register points to the start address of the current descriptor read by the IDMAC.

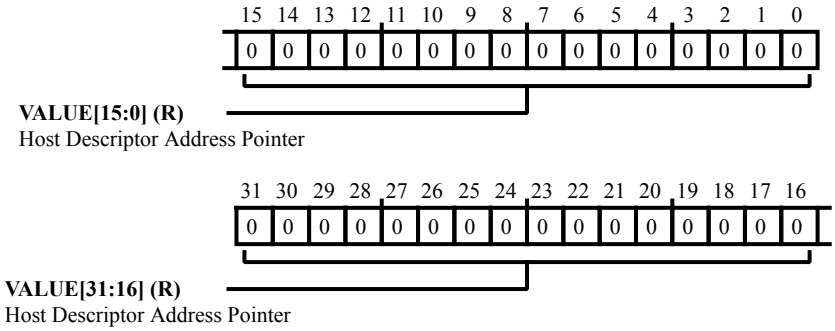

Figure 33-28: MSI_DSCADDR Register Diagram

Table 33-32: MSI_DSCADDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Host Descriptor Address Pointer. The MSI_DSCADDR.VALUE bit field points to the start address of the current descriptor read by the IDMAC.

Enable Phase Shift Register

The MSI_ENSHIFT register provides control for the amount of phase shift provided on the default enables in the design. Two bits are assigned for each card/slot.

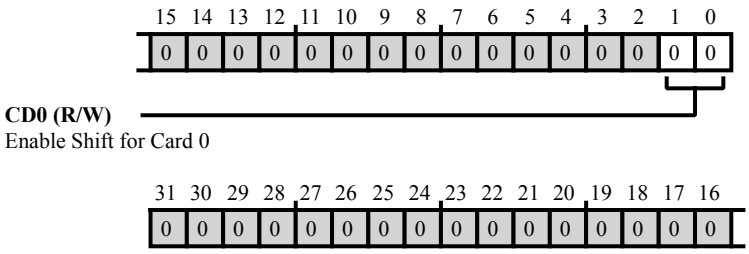

Figure 33-29: MSI_ENSHIFT Register Diagram

Table 33-33: MSI_ENSHIFT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
1:0 (R/W)	CD0	Enable Shift for Card 0. The MSI_ENSHIFT.CD0 bit field provides control for the amount of phase shift provided on the default enables in the design.

FIFO Threshold Watermark Register

The MSI_FIFOTH register provides bits that manage the FIFO transactions.

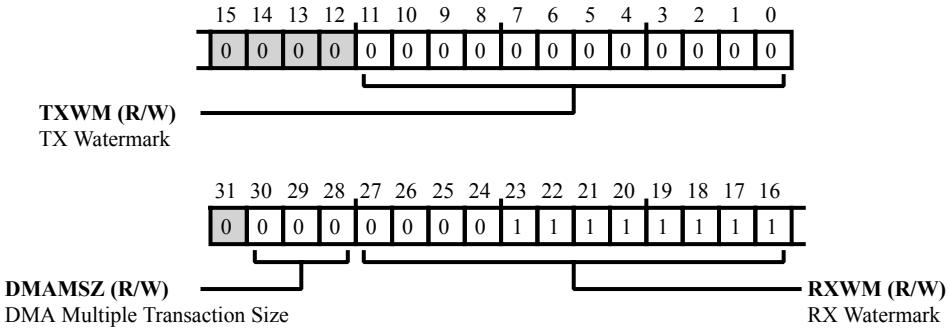

Figure 33-30: MSI_FIFOTH Register Diagram

Table 33-34: MSI_FIFOTH Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
30:28 (R/W)	DMAMSZ	DMA Multiple Transaction Size. The MSI_FIFOTH.DMAMSZ bit field configures the burst size of a multiple transaction. The units for transfer are 32-bit.	
		0	1 transfer
		1	4 transfers
		2	8 transfers
		3	16 transfers
		4	32 transfers
		5	64 transfers
		6	128 transfers
		7	256 transfers
27:16 (R/W)	RXWM	RX Watermark. The MSI_FIFOTH.RXWM bit field sets the FIFO threshold watermark level when receiving data to card. When the FIFO data count is greater than this number, a DMA/FIFO request is raised.	
11:0 (R/W)	TXWM	TX Watermark. The MSI_FIFOTH.TXWM bit field sets the FIFO threshold watermark level when transmitting data to card. When the FIFO data count is less than or equal to this number, a DMA/FIFO request is raised.	

Internal DMA Interrupt Enable Register

The MSI_IDINTEN register provides bits for setting various interrupts in DMA mode.

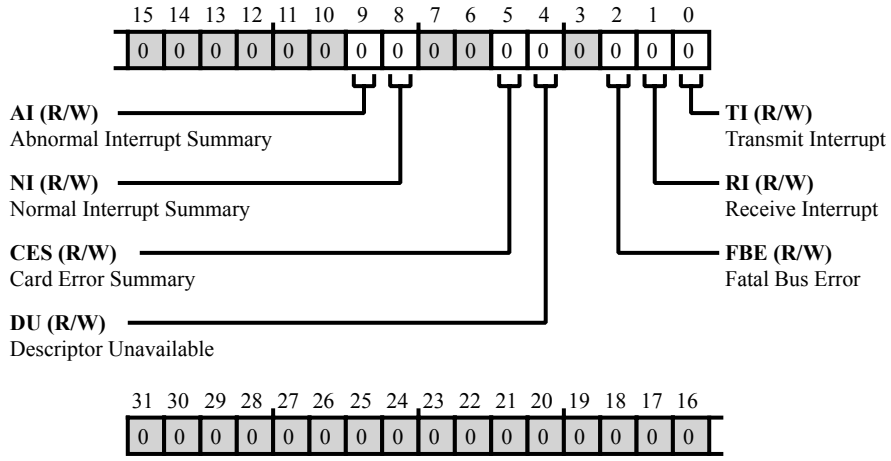

Figure 33-31: MSI_IDINTEN Register Diagram

Table 33-35: MSI_IDINTEN Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/W)	AI	Abnormal Interrupt Summary. Setting the MSI_IDINTEN.AI bit enables the abnormal interrupt summary. See the bit descriptions in the MSI_IDSTS register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
8 (R/W)	NI	Normal Interrupt Summary. Setting the MSI_IDINTEN.NI bit enables the normal interrupt summary. See the bit descriptions in the MSI_IDSTS register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
5 (R/W)	CES	Card Error Summary. Setting the MSI_IDINTEN.CES bit enables the card error summary interrupt. See the bit descriptions in the MSI_IDSTS register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled

Table 33-35: MSI_IDINTEN Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DU	Descriptor Unavailable. Setting the `MSI_IDINTEN.DU` bit enables the descriptor unavailable interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
2 (R/W)	FBE	Fatal Bus Error. Setting the `MSI_IDINTEN.FBE` bit enables the FBE interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
1 (R/W)	RI	Receive Interrupt. Setting the `MSI_IDINTEN.RI` bit enables the receive interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled
0 (R/W)	TI	Transmit Interrupt. Setting the `MSI_IDINTEN.TI` bit enables the transmit interrupt. See the bit descriptions in the `MSI_IDSTS` register for interrupt descriptions.	
		0	Interrupt disabled
		1	Interrupt enabled

Internal DMA Status Register

The MSI_IDSTS register provides DMA status information. This register is updated only when the DMA is active.

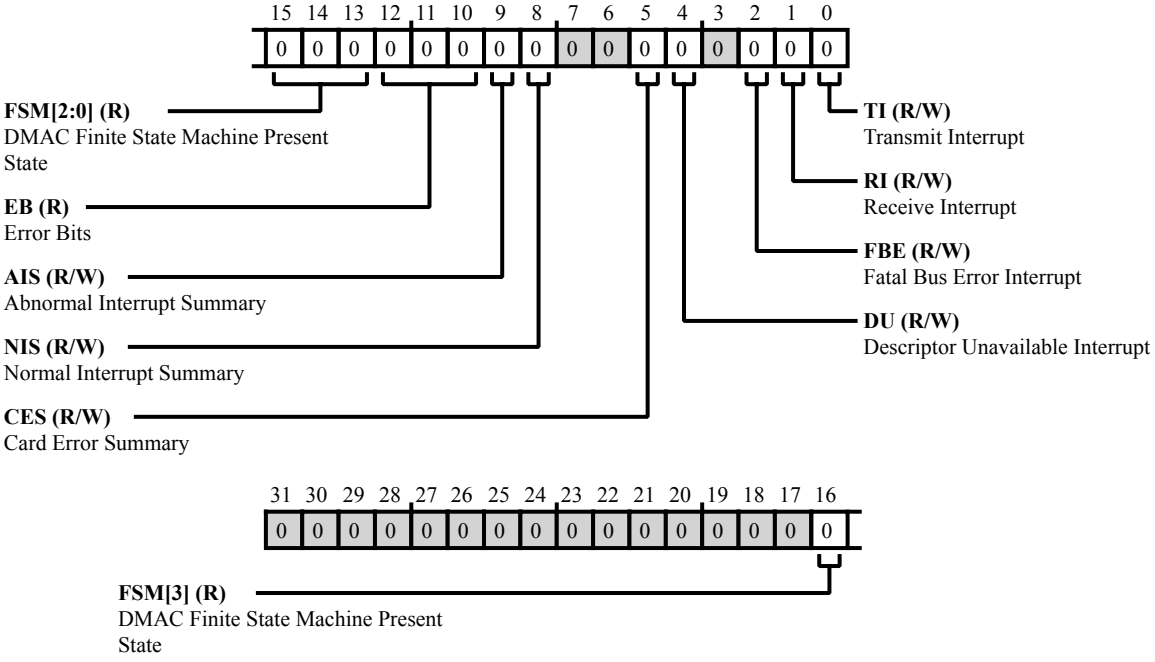

Figure 33-32: MSI_IDSTS Register Diagram

Table 33-36: MSI_IDSTS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:13 (R/NW)	FSM	DMAC Finite State Machine Present State. The MSI_IDSTS.FSM bit field indicates the present state of the finite state machine IDMAC (DMA controller).	
		0	DMA Idle
		1	DMA suspend
		2	DESC_RD
		3	DESC_CHK
		4	DMA_RD_REQ_WAIT
		5	DMA_WR_REQ_WAIT
		6	DMA_RD
		7	DMA_WR
		8	DESC_CLOSE

Table 33-36: MSI_IDSTS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
12:10 (R/NW)	EB	Error Bits. The MSI_IDSTS.EB bit field indicates the type of error that caused a bus error. It is valid only with MSI_IDSTS.FBE set. This field does not generate an interrupt. The MSI_IDSTS.EB bit field is read-only.	
		0	3'b001 - Host abort received during transmission
		1	3'b010 - Host abort received during reception
9 (R/W)	AIS	Abnormal Interrupt Summary. The MSI_IDSTS.AIS bit field is a logical OR of the following: IDSTS[2] - Fatal Bus Interrupt IDSTS[4] - DU bit Interrupt Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes MSI_IDSTS.AIS to be set is cleared. Writing a 1 clears this bit.	
8 (R/W)	NIS	Normal Interrupt Summary. The MSI_IDSTS.NIS bit field is a logical OR of the following: IDSTS[0] - Transmit Interrupt IDSTS[1] - Receive Interrupt Only unmasked bits affect this bit. This is a sticky bit and must be cleared each time a corresponding bit that causes MSI_IDSTS.NIS to be set is cleared. Writing a 1 clears this bit.	
5 (R/W)	CES	Card Error Summary. The MSI_IDSTS.CES bit indicates the logical OR of the following bits: • EBE - End Bit Error • RTO - Response Timeout/Boot Ack Timeout • RCRC - Response CRC • SBE - Start Bit Error • DRTO - Data Read Timeout/BDS timeout • DCRC - Data CRC for Receive • RE - Response Error Writing a 1 clears this bit.	
		0	No error occurred
		1	Error occurred

Table 33-36: MSI_IDSTS Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	DU	Descriptor Unavailable Interrupt. The MSI_IDSTS.DU bit indicates a descriptor unavailable error. If software creates less descriptors than the boot data received, the MSI generates this interrupt and does not transfer any further data to system memory.	
		0	No error occurred
		1	Error occurred
2 (R/W)	FBE	Fatal Bus Error Interrupt. The MSI_IDSTS.FBE bit indicates a fatal bus error error. An FBE occurs due to an error response from the SCB. Because this is a system error, the software driver should not perform any further programming of the MSI. The only recovery mechanism from such an error is to issue a hard reset or to perform a controller reset by writing to the MSI_CTL.CTLRST bit.	
		0	No error occurred
		1	Error occurred
1 (R/W)	RI	Receive Interrupt. The MSI_IDSTS.RI bit indicates that data reception is finished for a descriptor.	
		0	No event occurred
		1	Event occurred
0 (R/W)	TI	Transmit Interrupt. The MSI_IDSTS.TI bit indicates that data transmission is finished for a descriptor.	
		0	No event occurred
		1	Event occurred

Interrupt Mask Register

The MSI_IMSK register provides bits that allow the masking of unwanted interrupts.

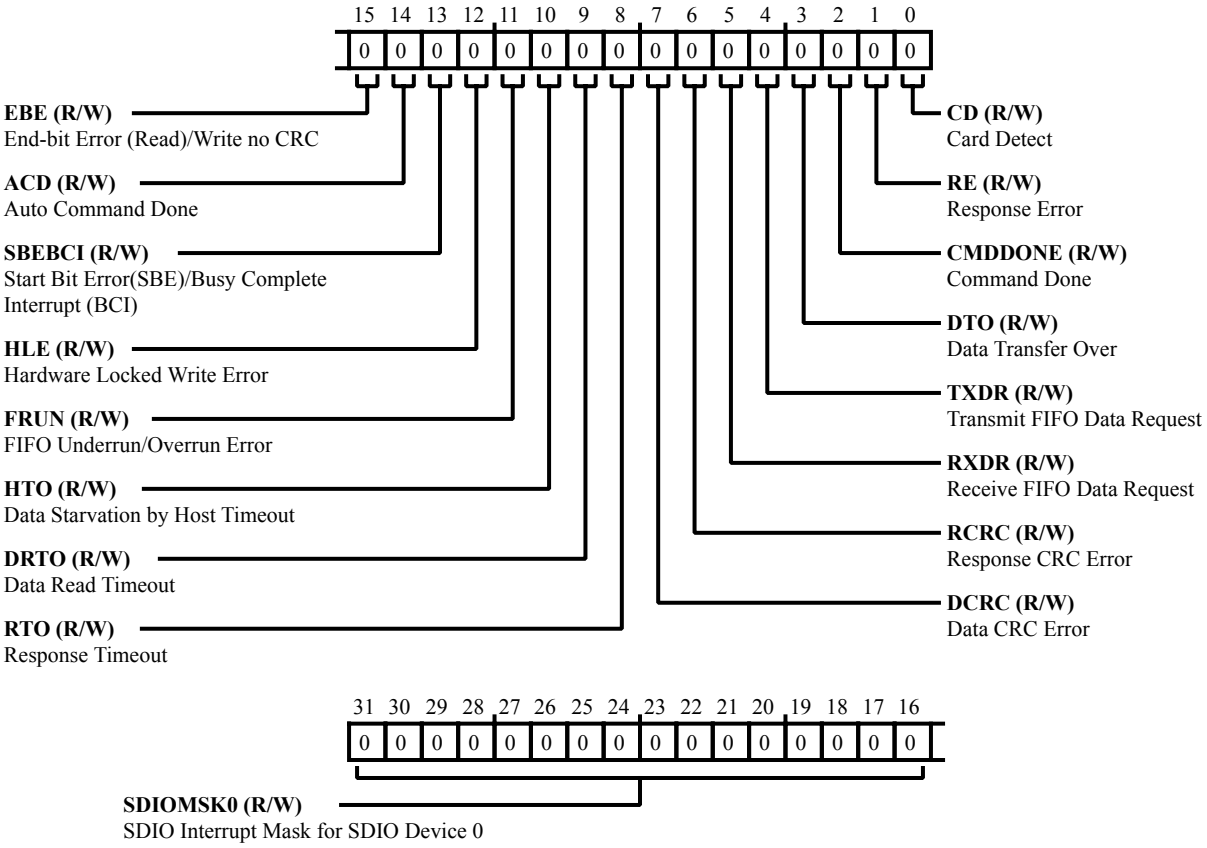

Figure 33-33: MSI_IMSK Register Diagram

Table 33-37: MSI_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:16 (R/W)	SDIOMSK0	SDIO Interrupt Mask for SDIO Device 0. The MSI_IMSK.SDIOMSK0 bits mask SDIO interrupts, one bit for one card. When masked, SDIO interrupt detection for that card is disabled. A 0 masks an interrupt, and 1 enables an interrupt.	
15 (R/W)	EBE	End-bit Error (Read)/Write no CRC. The MSI_IMSK.EBE bit masks the end bit read/write no CRC error.	
		0	Masked
		1	Enabled

Table 33-37: MSI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W)	ACD	Auto Command Done. The MSI_IMSK.ACD bit masks the auto command done error.	
		0	Masked
		1	Enabled
13 (R/W)	SBEBCI	Start Bit Error(SBE)/Busy Complete Interrupt (BCI). The MSI_IMSK.SBEBCI bit masks the start bit/busy complete error.	
		0	Masked
		1	Enabled
12 (R/W)	HLE	Hardware Locked Write Error. The MSI_IMSK.HLE bit masks the hardware locked write error.	
		0	Masked
		1	Enabled
11 (R/W)	FRUN	FIFO Underrun/Overrun Error. The MSI_IMSK.FRUN bit masks the FIFO overrun or underrun error.	
		0	Masked
		1	Enabled
10 (R/W)	HTO	Data Starvation by Host Timeout. The MSI_IMSK.HTO bit masks the host timeout error.	
		0	Masked
		1	Enabled
9 (R/W)	DRTO	Data Read Timeout. The MSI_IMSK.DRTO bit masks the data read timeout error.	
		0	Masked
		1	Enabled
8 (R/W)	RTO	Response Timeout. The MSI_IMSK.RTO bit masks the response timeout error.	
		0	Masked
		1	Enabled

Table 33-37: MSI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W)	DCRC	Data CRC Error. The MSI_IMSK.DCRC bit masks the data CRC error where the received data CRC does not match with locally-generated CRC in CIU. The error can also occur if the write CRC status is incorrectly sampled by the host.	
		0	Masked
		1	Enabled
6 (R/W)	RCRC	Response CRC Error. The MSI_IMSK.RCRC bit masks the response CRC error which is set when the response CRC does not match with the locally-generated CRC in the CIU.	
		0	Masked
		1	Enabled
5 (R/W)	RXDR	Receive FIFO Data Request. The MSI_IMSK.RXDR bit masks the receive FIFO data request interrupt which is set during write operation to card when FIFO level reaches less than or equal to transmit-threshold level.	
		0	Masked
		1	Enabled
4 (R/W)	TXDR	Transmit FIFO Data Request. The MSI_IMSK.TXDR bit masks the transmit FIFO data request interrupt which is set during write operation to card when FIFO level reaches less than or equal to transmit-threshold level.	
		0	Masked
		1	Enabled
3 (R/W)	DTO	Data Transfer Over. The MSI_IMSK.DTO bit masks the data transfer over interrupt which indicates the data transfer completed. Though on detection of errors such as the start bit error, the data CRC error, and so on, DTO may or may not be set; the application must issue CMD12, which ensures that DTO is set.	
		0	Masked
		1	Enabled

Table 33-37: MSI_IMSK Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
2 (R/W)	CMDDONE	Command Done. The `MSI_IMSK.CMDDONE` bit masks the command done interrupt where a command is sent to a card and received a response from the card, even if a response error or a CRC error occurs.	
		0	Masked
		1	Enabled
1 (R/W)	RE	Response Error. The `MSI_IMSK.RE` masks a response error. This is an error in received response set if one of following occurs: transmission bit != 0, command index mismatch, End-bit != 1.	
		0	Masked
		1	Enabled
0 (R/W)	CD	Card Detect. The `MSI_IMSK.CD` bit masks the card detect interrupt. On power-on, the controller should read in the card_detect port and store the value in the memory. Upon receiving a card-detect interrupt, it should again read the card_detect port and XOR with the previous card-detect status to find out which card has interrupted.	
		0	Masked
		1	Enabled

Raw Interrupt Status Register

The MSI_ISTAT register provides bits that clear interrupts. Conditions 6 through 9 indicate that the received data may have errors. If there was a response timeout, then no data transfer occurred.

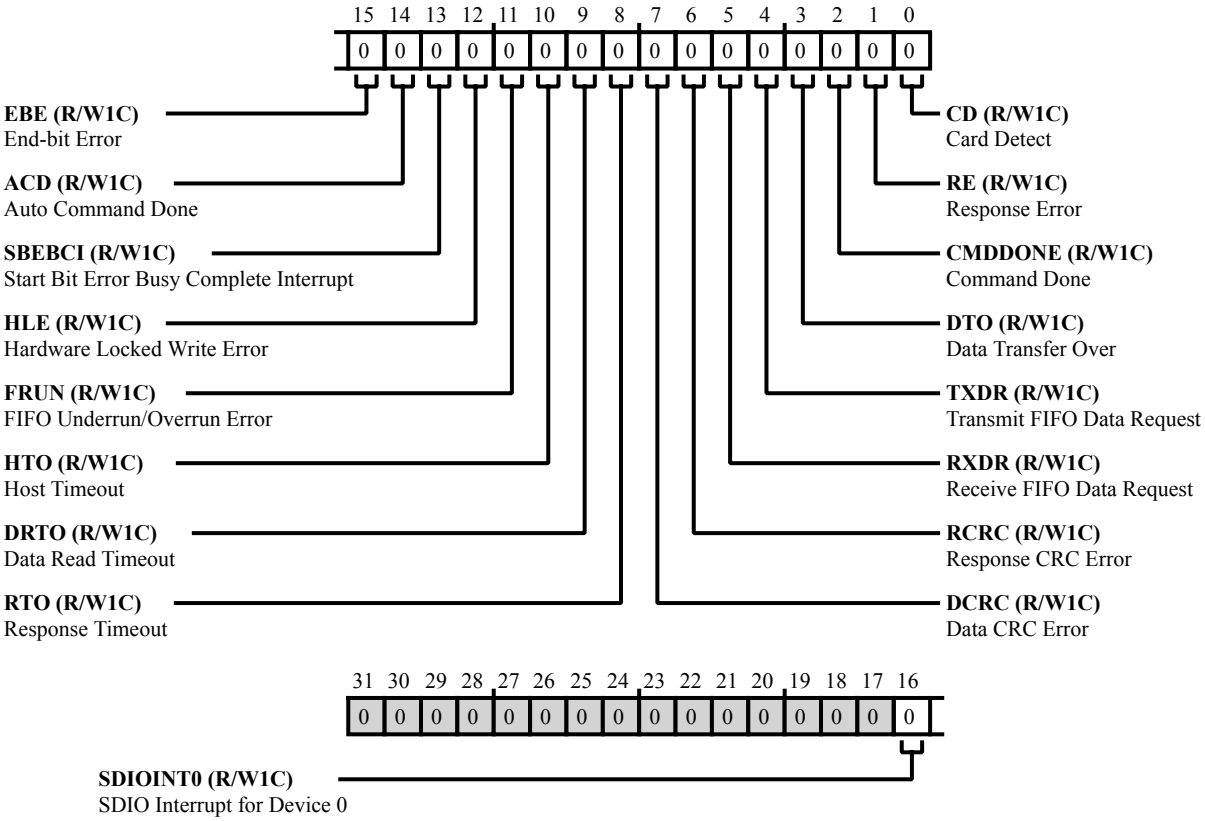

Figure 33-34: MSI_ISTAT Register Diagram

Table 33-38: MSI_ISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W1C)	SDIOINT0	SDIO Interrupt for Device 0. The MSI_ISTAT.SDIOINT0 bit indicates that an interrupt occurred on an SDIO card.	
15 (R/W1C)	EBE	End-bit Error. The MSI_ISTAT.EBE bit indicates that the start bit of the CRC status was not received by two clocks after the end of the data block. A CRC error is indicated by the card.	
		0	No error
		1	Error occurred

Table 33-38: MSI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
14 (R/W1C)	ACD	Auto Command Done.	
		0	No error
		1	Error occurred
13 (R/W1C)	SBEBCI	Start Bit Error Busy Complete Interrupt. The `MSI_ISTAT.SBEBCI` bit indicates the completion of a busy signal driven by the card after a write data transfer.	
		0	No error
		1	Error occurred
12 (R/W1C)	HLE	Hardware Locked Write Error.	
		0	No error
		1	Error occurred
11 (R/W1C)	FRUN	FIFO Underrun/Overrun Error.	
		0	No error
		1	Error occurred
10 (R/W1C)	HTO	Host Timeout. The `MSI_ISTAT.HTO` bit indicates that the FIFO is empty during transmission or is full during reception. Unless software/DMA writes data for an empty condition or reads data for a full condition, the MSI cannot continue with data transfer. The clock to the card is stopped.	
		0	No error
		1	Error occurred
9 (R/W1C)	DRTO	Data Read Timeout. The `MSI_ISTAT.DRTO` bit indicates that the card has not sent data within the time-out period.	
		0	No error
		1	Error occurred
8 (R/W1C)	RTO	Response Timeout.	
		0	No error
		1	Error occurred

Table 33-38: MSI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
7 (R/W1C)	DCRC	Data CRC Error. The MSI_ISTAT.DCRC bit indicates that a data CRC error where the received data CRC does not match with the locally-generated CRC in the CIU. This error can also occur if the write CRC status is incorrectly sampled by the host.	
		0	No error
		1	Error occurred
6 (R/W1C)	RCRC	Response CRC Error. The MSI_ISTAT.RCRC bit indicates that the response CRC error which is caused when the response CRC does not match with the locally-generated CRC in the CIU.	
		0	No error
		1	Error occurred
5 (R/W1C)	RXDR	Receive FIFO Data Request. The MSI_ISTAT.RXDR bit indicates that the FIFO threshold for receiving data was reached and software/DMA is expected to read data from the FIFO.	
		0	No error
		1	Error occurred
4 (R/W1C)	TXDR	Transmit FIFO Data Request. The MSI_ISTAT.TXDR bit indicates that the FIFO threshold for transmitting data was reached and is less than or equal to transmit-threshold level. Software/DMA is expected to write data, if available, in the FIFO. This interrupt is set during a write operation to the card.	
		0	No error
		1	Error occurred
3 (R/W1C)	DTO	Data Transfer Over. The MSI_ISTAT.DTO bit indicates the data transfer completed. If there is a response timeout error, then the MSI does not attempt any data transfer and this bit is never set.	
		0	No error
		1	Error occurred
2 (R/W1C)	CMDDONE	Command Done. The MSI_ISTAT.CMDDONE bit indicates that a command that was sent to a card received a response from the card, even a response error or CRC error occurred.	
		0	No error
		1	Error occurred

Table 33-38: MSI_ISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/W1C)	RE	Response Error. The `MSI_ISTAT.RE` bit indicates that an error was received during response reception. In this case, the response that copied in the response registers is invalid. Software can retry the command.	
		0	No error
		1	Error occurred
0 (R/W1C)	CD	Card Detect. The `MSI_ISTAT.CD` bit indicates a card detect interrupt.	
		0	No error
		1	Error occurred

Masked Interrupt Status Register

The MSI_MSKISTAT register indicates the status for the bits which are unmasked in the MSI_IMSK register. In other words, MSI_MSKISTAT = MSI_ISTAT and MSI_IMSK.

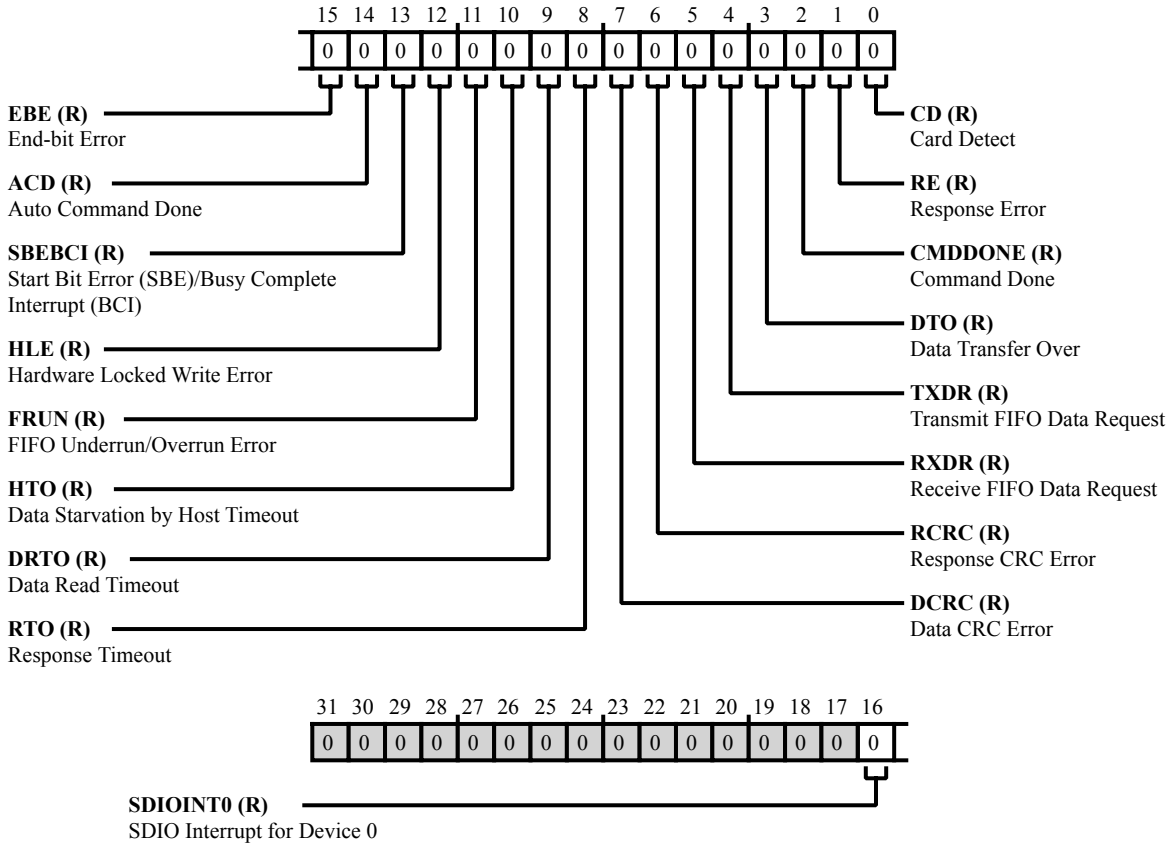

Figure 33-35: MSI_MSKISTAT Register Diagram

Table 33-39: MSI_MSKISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/NW)	SDIOINT0	SDIO Interrupt for Device 0. The MSI_MSKISTAT.SDIOINT0 bit indicates an interrupt occurred on the SDIO interrupt for device 0 masked interrupt.	
		0	No interrupt
		1	Interrupt occurred

Table 33-39: MSI_MSKISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15 (R/NW)	EBE	End-bit Error. The `MSI_MSKISTAT.EBE` bit indicates an interrupt occurred on the end-bit error (read)/write no CRC masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
14 (R/NW)	ACD	Auto Command Done. The `MSI_MSKISTAT.ACD` bit indicates an interrupt occurred on the auto command done masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
13 (R/NW)	SBEBCI	Start Bit Error (SBE)/Busy Complete Interrupt (BCI). The `MSI_MSKISTAT.SBEBCI` bit indicates an interrupt occurred on the start bit error/busy complete masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
12 (R/NW)	HLE	Hardware Locked Write Error. The `MSI_MSKISTAT.HLE` bit indicates an interrupt occurred on the hardware locked write error masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
11 (R/NW)	FRUN	FIFO Underrun/Overrun Error. The `MSI_MSKISTAT.FRUN` bit indicates an interrupt occurred on the FIFO underrun/overrun error masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
10 (R/NW)	HTO	Data Starvation by Host Timeout. The `MSI_MSKISTAT.HTO` bit indicates an interrupt occurred on the data starvation by host timeout masked interrupt.	
		0	No interrupt
		1	Interrupt occurred

Table 33-39: MSI_MSKISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
9 (R/NW)	DRTO	Data Read Timeout. The `MSI_MSKISTAT.DRTO` bit indicates an interrupt occurred on the data read time-out masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
8 (R/NW)	RTO	Response Timeout. The `MSI_MSKISTAT.RTO` bit indicates an interrupt occurred on the response timeout masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
7 (R/NW)	DCRC	Data CRC Error. The `MSI_MSKISTAT.DCRC` bit indicates an interrupt occurred on the data CRC error request masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
6 (R/NW)	RCRC	Response CRC Error. The `MSI_MSKISTAT.RCRC` bit indicates an interrupt occurred on the response CRC error data request masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
5 (R/NW)	RXDR	Receive FIFO Data Request. The `MSI_MSKISTAT.RXDR` bit indicates an interrupt occurred on the receive FIFO data request masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
4 (R/NW)	TXDR	Transmit FIFO Data Request. The `MSI_MSKISTAT.TXDR` bit indicates an interrupt occurred on the transmit FIFO data request masked interrupt.	
		0	No interrupt
		1	Interrupt occurred

Table 33-39: MSI_MSKISTAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	DTO	Data Transfer Over. The `MSI_MSKISTAT.DTO` bit indicates an interrupt occurred on the data transfer over masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
2 (R/NW)	CMDDONE	Command Done. The `MSI_MSKISTAT.CMDDONE` bit indicates an interrupt occurred on the command done masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
1 (R/NW)	RE	Response Error. The `MSI_MSKISTAT.RE` bit indicates an interrupt occurred on the response error masked interrupt.	
		0	No interrupt
		1	Interrupt occurred
0 (R/NW)	CD	Card Detect. The `MSI_MSKISTAT.CD` bit indicates an interrupt occurred on the card detect masked interrupt.	
		0	No interrupt
		1	Interrupt occurred

Poll Demand Register

If the OWN bit of a descriptor is not set, the FSM goes into the suspend state. The host needs to write any value into the MSI_PLDMND register for the IDMAC FSM to resume normal descriptor fetch operation. This is a write-only register.

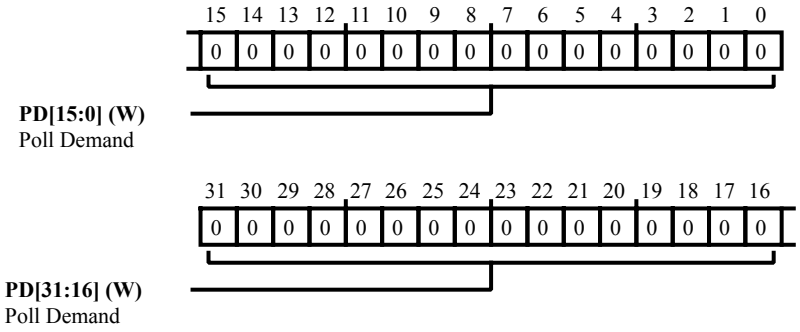

Figure 33-36: MSI_PLDMND Register Diagram

Table 33-40: MSI_PLDMND Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	PD	Poll Demand. The MSI_PLDMND.PD bit field needs to be written so that the IDMAC FSM can resume normal descriptor fetch operation.

Response Register 0

The MSI_RESP0 register represents bits[31:0] of a long response.

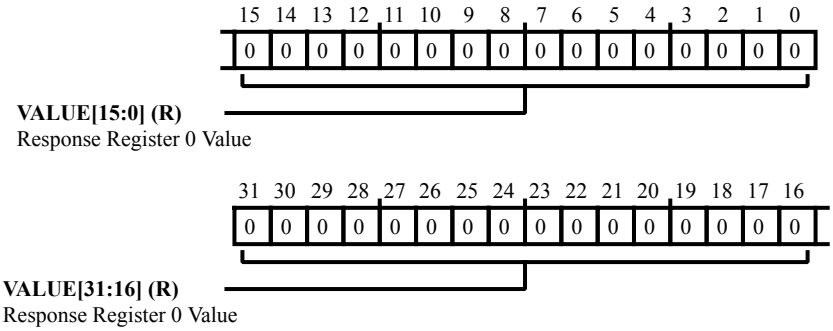

Figure 33-37: MSI_RESP0 Register Diagram

Table 33-41: MSI_RESP0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Response Register 0 Value. The MSI_RESP0.VALUE bit field represents bits[31:0] of a long response.

Response Register 1

The MSI_RESP1 register represents bits[63:32] of a long response.

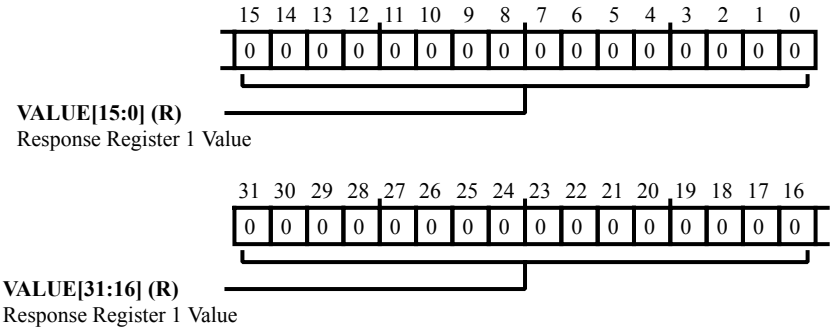

Figure 33-38: MSI_RESP1 Register Diagram

Table 33-42: MSI_RESP1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Response Register 1 Value. The MSI_RESP1.VALUE bit field represents bits[63:32] of a long response.

Response Register 2

The MSI_RESP2 register represents bits[95:64] of a long response.

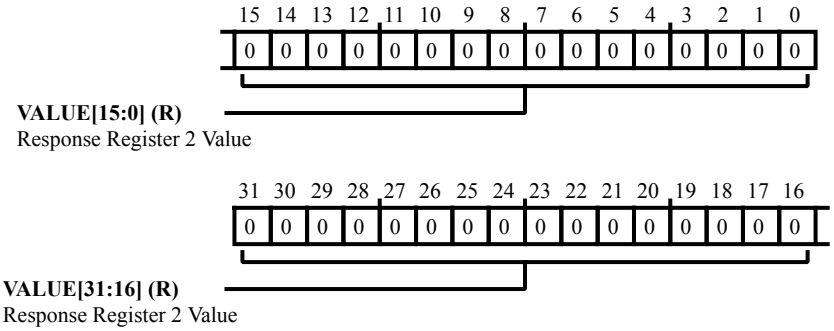

Figure 33-39: MSI_RESP2 Register Diagram

Table 33-43: MSI_RESP2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Response Register 2 Value. The MSI_RESP2.VALUE bit field represents bits[95:64] of a long response.

Response Register 3

The MSI_RESP3 register represents bits[127:96] of a long response.

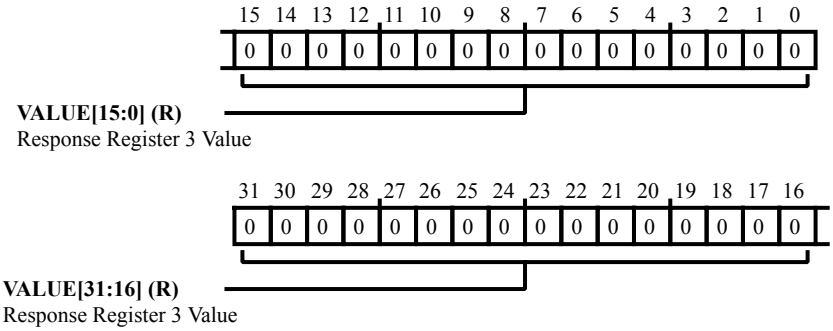

Figure 33-40: MSI_RESP3 Register Diagram

Table 33-44: MSI_RESP3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Response Register 3 Value. The MSI_RESP3.VALUE bit field represents bits[127:96] of a long response.

Status Register

The MSI_STAT register provides MSI DMA and data transfer status and information.

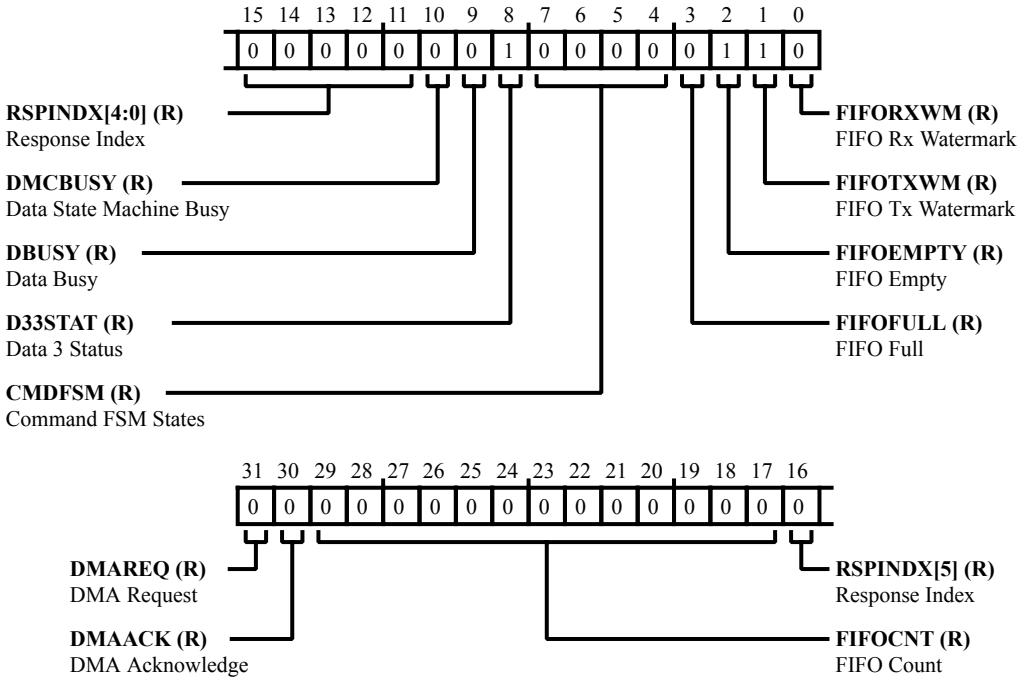

Figure 33-41: MSI_STAT Register Diagram

Table 33-45: MSI_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31 (R/NW)	DMAREQ	DMA Request. The MSI_STAT.DMAREQ bit indicates the DMA request signal state.
30 (R/NW)	DMAACK	DMA Acknowledge. The MSI_STAT.DMAACK bit indicates the DMA acknowledge signal state.
29:17 (R/NW)	FIFOCNT	FIFO Count. The MSI_STAT.FIFOCNT bit field indicates the number of filled locations in the FIFO.
16:11 (R/NW)	RSPINDX	Response Index. The MSI_STAT.RSPINDX bit field indicates the index of the previous response, including any auto-stop sent by core.

Table 33-45: MSI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/NW)	DMCBUSY	Data State Machine Busy. The `MSI_STAT.DMCBUSY` bit indicates that the data transmit or receive state-machine is busy.	
		0	DMC idle
		1	DMC busy
9 (R/NW)	DBUSY	Data Busy. The `MSI_STAT.DBUSY` bit indicates the inverted version of the raw selected card_data[0].	
		0	Card data not busy
		1	Card data busy
8 (R/NW)	D33STAT	Data 3 Status.	
		0	Card not present
		1	Card present
7:4 (R/NW)	CMDFSM	Command FSM States. The `MSI_STAT.CMDFSM` bit field indicates the state machine status of CIU (card interface unit) or in general command FSM (not IDMA).	
		0	Idle
		1	Send init sequence
		2	Tx cmd start bit
		3	Tx cmd tx bit
		4	Tx cmd index + arg
		5	Tx cmd CRC7
		6	Tx cmd end bit
		7	Rx resp start bit
		8	Rx resp IRQ response
		9	Rx resp tx bit
		10	Rx resp cmd idx
		11	Rx resp data
		12	Rx resp CRC7
		13	Rx resp end bit
		14	Cmd path wait NCC
		15	Wait; CMD-to-response turnaround

Table 33-45: MSI_STAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
3 (R/NW)	FIFOFULL	FIFO Full. The `MSI_STAT.FIFOFULL` bit indicates that the FIFO is full.	
2 (R/NW)	FIFOEMPTY	FIFO Empty. The `MSI_STAT.FIFOEMPTY` bit indicates that the FIFO is empty.	
1 (R/NW)	FIFOTXWM	FIFO Tx Watermark. The `MSI_STAT.FIFOTXWM` bit indicates that the FIFO reached the transmit watermark level and is not qualified with data transfer.	
		0	Did not reach watermark level
		1	Reached watermark level
0 (R/NW)	FIFORXWM	FIFO Rx Watermark. The `MSI_STAT.FIFORXWM` bit indicates that the FIFO reached the receive watermark level and is not qualified with data transfer.	
		0	Did not reach watermark level
		1	Reached watermark level

Transferred Host to BIU-FIFO Byte Count Register

The MSI_TBBCNT register provides the number of bytes transferred between host/DMA memory and BIU FIFO. The register should be accessed in full to avoid read-coherency problems.

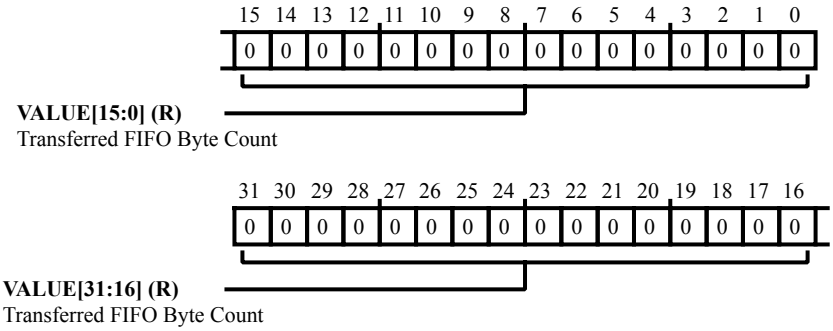

Figure 33-42: MSI_TBBCNT Register Diagram

Table 33-46: MSI_TBBCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Transferred FIFO Byte Count. The MSI_TBBCNT.VALUE bit field provides the number of bytes transferred between the host/DMA memory and the BIU FIFO.

Transferred CIU Card Byte Count Register

The MSI_TCBCNT register provides the number of bytes transferred by the CIU unit to a card. This register should be accessed in full to avoid read-coherency problems. Both the MSI_TCBCNT and MSI_TBBCNT registers share the same coherency register.

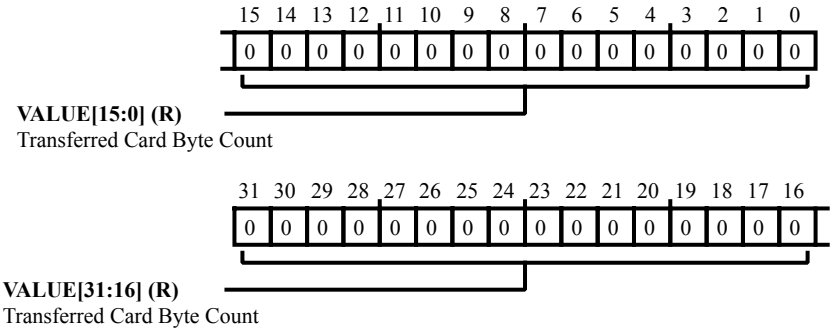

Figure 33-43: MSI_TCBCNT Register Diagram

Table 33-47: MSI_TCBCNT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	Transferred Card Byte Count. The MSI_TCBCNT.VALUE bit field provides the number of bytes transferred by the CIU unit to a card.

Timeout Register

The MSI_TMOUT register provides bits that configure card data read and response timeout values.

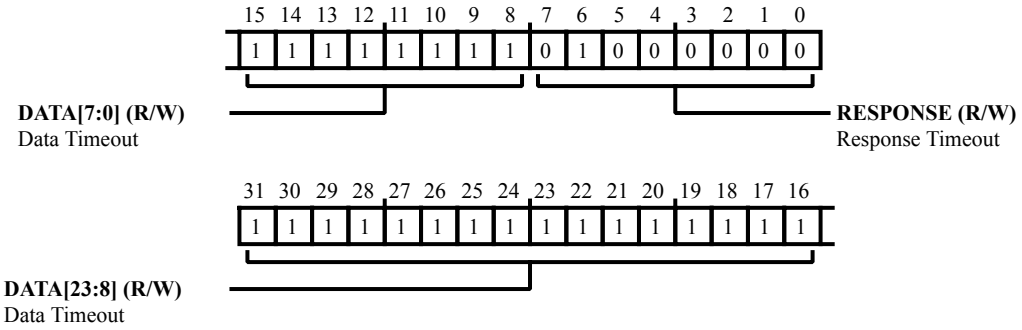

Figure 33-44: MSI_TMOUT Register Diagram

Table 33-48: MSI_TMOUT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:8 (R/W)	DATA	Data Timeout. The MSI_TMOUT.DATA bit field configures the value for the card data read timeout; the same value is also used for the data starvation by host timeout. The timeout counter is started only after the card clock is stopped. The value is in number of card output clocks cclk_out of a selected card.
7:0 (R/W)	RESPONSE	Response Timeout. The MSI_TMOUT.RESPONSE bit field configures the response timeout in number of card output clocks.

Ultra High Speed Register Extension

The MSI_UHS_EXT register provides control bits for the input clock.

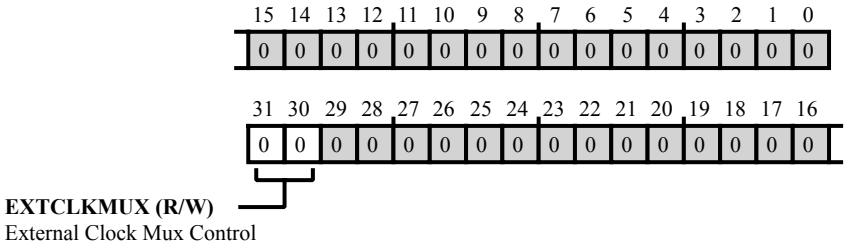

Figure 33-45: MSI_UHS_EXT Register Diagram

Table 33-49: MSI_UHS_EXT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:30 (R/W)	EXTCLKMUX	External Clock Mux Control. The MSI_UHS_EXT.EXTCLKMUX bit field selects the source of the MSI input clock.	
		0	SCLK0/2. Selects SCLK0 divided by 2.
		1	SCLK0. Selects SCLK0
		2	SCLK1/3. Selects SCLK1 divided by 3
		3	Reserved

34 Housekeeping ADC (HADC)

The Housekeeping ADC is a 12-bit (with 10-bit accuracy), successive approximation ADC. It operates from single supply and features throughput rates up to 1 MSPS. The HADC can be used for the collection of housekeeping parameters like voltages, temperatures in the system or for any general-purpose use as well.

HADC Features

The HADC supports following features.

- 12-bit ADC core with built-in sample and hold
- ENOB = 10 bit
- 4 input channels
- Throughput rates up to 1M SPS
- Single Ended Operation
- External Reference 2.5 V to 3.3 V
- Analog Input 0 V to 3.3 V.
- Selectable ADC clock frequency through a pre-scaler
- Conversion type adaptable to each application: allows single or continuous conversion with option of auto-scan
- Auto sequencing capability provides up to 4 *auto-conversions* in a single session. Each conversion can be programmed to select any of the available input channels
- Four data registers (individually addressable) to store conversion values

HADC Functional Description

The HADC provides the analog to digital conversion capability for general-purpose housekeeping tasks, such as voltage and temperature monitoring. The core of HADC is a 12-bit SAR ADC, providing multiple analog input channels.

The HADC has the following functionality.

Fixed and continuous conversion modes

ADC converts the input channel sequence for a fixed number of times or continuously converts an input channel sequence.

Auto scanning

All the input channels can be sampled in a sequential manner.

Channel sequence programming

The sequence of a channel can be selected by programming the Channel Mask register. If the bit corresponding to the channel is programmed to zero, that channel is included in the auto-scan chain.

ADSP-BF70x HADC Register List

The Housekeeping ADC (HADC) provides a general purpose, multi-channel successive approximation A-to-D converter. A set of registers governs HADC operations. For more information on HADC functionality, see the HADC register descriptions.

Table 34-1: ADSP-BF70x HADC Register List

Name	Description
HADC_CHAN_MSK	Channel Mask Register
HADC_CTL	Control Register
HADC_DATA[nn]	Channel Data Registers
HADC_IMSK	Interrupt Mask Register
HADC_STAT	Status Register

ADSP-BF70x HADC Interrupt List

Table 34-2: ADSP-BF70x HADC Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
60	HADC0_EVT	HADC0 Event	Edge	

ADSP-BF70x HADC Trigger List

Table 34-3: ADSP-BF70x HADC Trigger List Masters

Trigger ID	Name	Description	Sensitivity
44	HADC0_EOC	HADC0 End of Conversion Trigger	Edge

Table 34-4: ADSP-BF70x HADC Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
None			

HADC Definitions

The following definitions are helpful when using the HADC module.

Auto-scan

Auto-scan is a feature which allows the multiple channels to be scanned and converted in sequence one after the other.

HADC Wakeup Time

It is the time required by the module after coming out of power down before it can start converting.

Signal-to-Noise and Distortion Ratio (SINAD)

The measured ratio of signal-to-noise and distortion at the output of the ADC. The signal is the rms amplitude of the fundamental. Noise is the sum of all non-fundamental signals up to half the sampling frequency ($f_S/2$), excluding dc. The ratio depends on the number of quantization levels in the digitization process; the more levels, the smaller the quantization noise. The theoretical signal-to-noise and distortion ratio for an ideal N-bit converter with a sine wave input is given by:

Signal-to-(Noise + Distortion) = (6.02 N + 1.76) dB

Total Harmonic Distortion (THD)

The ratio of the rms sum to the harmonics to the fundamental.

Peak Harmonic or Spurious Noise

The ratio of the rms value of the next largest component in the ADC output spectrum (up to $f_S/2$ and excluding dc) to the rms value of the fundamental. Typically, the value of this specification is determined by the largest harmonic in the spectrum, but for ADCs where the harmonics are buried in the noise floor, it is a noise peak.

Integral Nonlinearity

The maximum deviation from a straight line passing through the endpoints of the ADC transfer function. The endpoints are zero scale, a point 1 LSB below the first code transition, and full scale, a point 1 LSB above the last code transition.

Differential Nonlinearity

The difference between the measured and the ideal 1 LSB change between any two adjacent codes in the ADC.

Offset Error

The deviation of the first code transition (00...000) to (00...001) from the ideal—that is, GND1 + 1 LSB.

Offset Error Match

The difference in offset error between any two channels.

Gain Error

The deviation of the last code transition (111...110) to (111...111) from the ideal (that is, REFIN – 1 LSB) after the offset error has been adjusted out.

Gain Error Matching

The difference in gain error between any two channels.

Power Supply Rejection Ratio (PSRR)

PSRR is defined as the ratio of the power in the ADC output at full-scale frequency, f, to the power of a 100 mV p-p sine wave applied to the ADC VDD supply of frequency, f_S. The frequency of the input varies from 5 kHz to 25 MHz. PSRR (dB) = 10 log(Pf/Pf_S) where: Pf is the power at frequency, f, in the ADC output. Pf_S is the power at frequency, f_S, in the ADC output.

HADC Block Diagram

The ***HADC Block Diagram*** figure shows the functional blocks within the HADC and the interface to the processor core and the peripherals. The HADC supports four channels with no provision of external multiplexer.

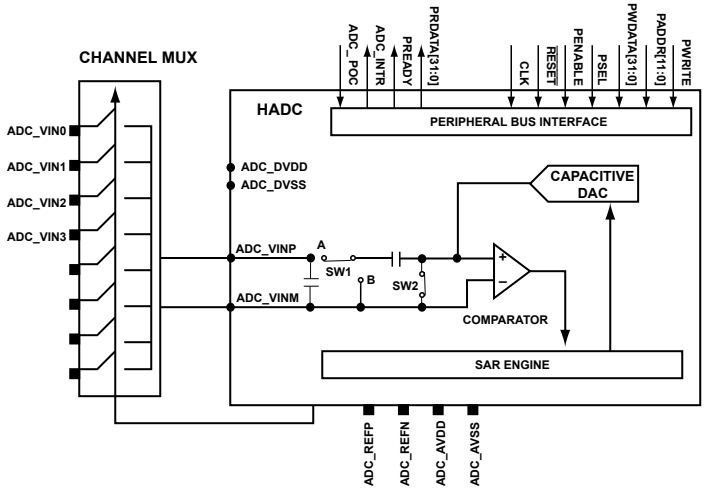

Figure 34-1: HADC Block Diagram

HADC Signal Descriptions

The ***HADC Signal Descriptions*** table provides descriptions of the signals used by the HADC.

Table 34-5: HADC Signal Descriptions

Signal Name	Signal Description
AVDD	ADC I/O supply
AVSS	I/O ground for analog blocks
VREFP	External reference for ADC
VREFN	Ground reference for ADC
VINn	Analog input at channel n

HADC Architectural Concepts

The HADC is based on a 12-bit SAR ADC that provides a simple register-based access model to obtain the results of conversion. The digital front end of the HADC provides a set of registers to configure the mode of operation, sampling frequency, and input channel selection control. The ADC supports multiple input analog channels which can be individually selected or deselected for conversion. The results of each analog channel are stored in a register. The core can access the register to read the conversion results once the conversion is complete. The HADC also provides the interrupts on completion of each channel conversion to avoid polling by the core. The following sections provide more details about the architecture of the HADC.

Converter Operation

The housekeeping ADC is a 12-bit successive approximation ADC based around a capacitive DAC. The ***ADC Acquisition Phase*** figure and the ***ADC Conversion Phase*** figure show simplified schematics of the ADC. The ADC is comprised of control logic, SAR, and a capacitive DAC. The components are used to add and subtract fixed

amounts of charge from the sampling capacitor to bring the comparator back into a balanced condition. The ***ADC Conversion Phase*** figure shows the ADC during its acquisition phase. SW2 is closed and SW1 is in Position A. The comparator is held in a balanced condition and the sampling capacitor acquires the signal on the selected V_{IN} channel.

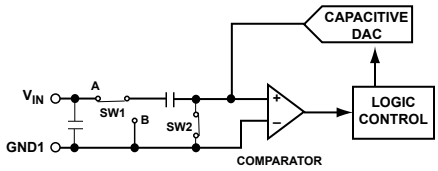

Figure 34-2: ADC Acquisition Phase

When the ADC starts a conversion (see the ***ADC Conversion Phase*** figure), SW2 opens and SW1 moves to Position B, causing the comparator to become unbalanced. The control logic and the capacitive DAC are used to add and subtract fixed amounts of charge to bring the comparator back into a balanced condition. When the comparator is rebalanced, the conversion is complete. The control logic generates the ADC output code.

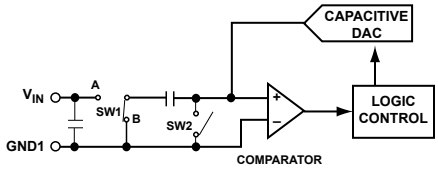

Figure 34-3: ADC Conversion Phase

Auto-Scan

The HADC features auto-scan mode where all the input channels can be sampled in a sequential manner. The number of channels enabled in auto-scan mode can be selected by programming the HADC_CHAN_MSK register. If the bit corresponding to the channel is set high, that particular channel is masked, and is not included in the auto-scan chain. In this way programs can sample all, none, or a selected set of channels by writing a high or a low for the individual channel. Auto sequencing allows the system to convert the same channel multiple times, allowing programs to perform oversampling algorithms.

For example, if the HADC_CHAN_MSK register bits [3:0] are set to 1101, then channel 0, channel 2 and channel 3 are not included in the auto-scan chain. Whether the conversion is a single or fixed number or continuous depends on the status of HADC_CTL.CONT bit. If this bit is low, the HADC_CTL.FIXEDCNV bits determine the number of sequence conversions.

The maximum number of fixed sequence conversions is four and they are enabled by default. The program must configure the HADC_CHAN_MSK register to enable any desired channel.

Channel Sequence Programming

The sequence of a channel can be selected by programming the HADC_CHAN_MSK register. If the bit corresponding to the channel is programmed to zero, that channel is included in the auto-scan chain. If the program must get the

conversion results from a particular channel, then the bit corresponding to that channel should be zero. The auto-scan section has more details.

ADC Transfer Function

The output coding is straight binary for the analog input channel conversion. The designed code transitions occur at successive LSB values (that is, 1 LSB, 2 LSBs, and so forth). The LSB size is $V_{REF}/4096$ for the HADC. The *ADC Transfer Function* figure shows the ideal transfer characteristic for straight binary coding.

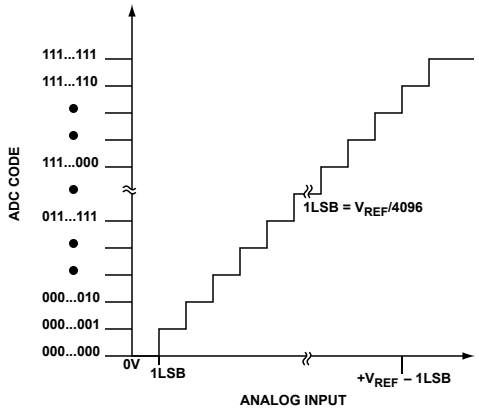

Figure 34-4: ADC Transfer Function

Results

The HADC takes 20 cycles of f_{SAMPLE} for one channel conversion. (The value of the `HADC_CTL.FDIV` bit field determines f_{SAMPLE}). The time taken to complete one sequence depends on the number of channels in the auto-scan chain. There is a latency of 1 cycle from the time the channel is selected internally to sample to the time the data is ready. After the end of each channel conversion, the data is written into the corresponding data register. An interrupt is generated (if the interrupt mask is not enabled) to signal that the data is ready for that particular channel.

HADC Operating Modes

The HADC has two modes of operation described in the following sections.

Fixed Conversion Mode

In this mode, the ADC converts the input channel sequence for a fixed number of times. The frequency is configured in the `HADC_CTL.FIXEDCNV` bit field. To use this mode, clear the `HADC_CTL.CONT` bit.

Continuous Conversion Mode

In this mode ADC continuously converts an input channel sequence, as long as `HADC_CTL.STARTCNV` bit is held high. To use this mode, set the `HADC_CTL.CONT` bit.

HADC Event Control

The HADC generates different events depending on the state of the ADC and the status of channel conversions. It can generate an event for each of the following conditions:

- When ADC is ready for conversion
- At the end of sequence conversion
- At the end of each individual channel conversion

Each of these events can generate an interrupt. To generate an interrupt on any desired event, clear the respective bit in the HADC_IMSK register.

HADC Programming Model

Following sections provide some guidelines for HADC programming.

Powering Up the HADC

To power-up the HADC, program the following bits in the HADC_CTL register.

- Deassert the HADC_CTL.PD bit (HADC power down)
- Set the HADC_CTL.NRST bit (Reset)
- Set the HADC_CTL.ENLS bit (Enable level shifters)

After deasserting HADC_CTL.PD, the HADC requires a finite wake-up time (t_{WAKEUP}) before it can start converting. The HADC requires only two f_{SAMPLE} clocks from the assertion of the HADC_CTL.NRST bit before the module is ready to convert. (HADC_CTL.PD is low). Poll the HADC_STAT.RDY bit. A 1 on this bit indicates that the HADC is ready to convert data.

Enabling the HADC

Setting the HADC_CTL.STARTCNV bit enables the HADC. When this bit is kept high, the HADC can work in either continuous or fixed conversion mode. After the HADC_CTL.STARTCNV bit is set =1, the HADC_CHAN_MSK can still be re-programmed, but the new sequence only comes into effect after the current sequence conversion is complete.

ADSP-BF70x HADC Register Descriptions

Housekeeping ADC (HADC) contains the following registers.

Table 34-6: ADSP-BF70x HADC Register List

Name	Description
HADC_CHAN_MSK	Channel Mask Register

Table 34-6: ADSP-BF70x HADC Register List (Continued)

Name	Description
HADC_CTL	Control Register
HADC_DATA[nn]	Channel Data Registers
HADC_IMSK	Interrupt Mask Register
HADC_STAT	Status Register

Channel Mask Register

The HADC_CHAN_MSK register provides bits that mask each channel.The first MSB corresponds to channel 3, the second MSB to channel 2 and so on. If the mask is set for a particular channel, that channel is not converted. By default, channels 0-3 are not masked.

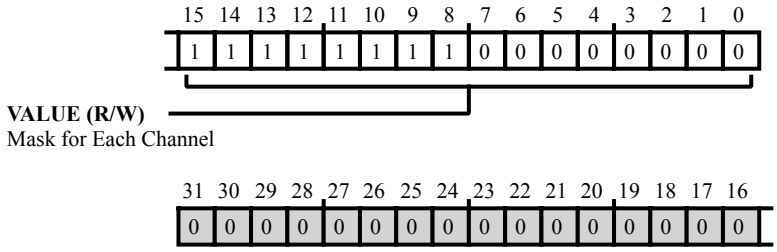

Figure 34-5: HADC_CHAN_MSK Register Diagram

Table 34-7: HADC_CHAN_MSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Mask for Each Channel. The HADC_CHAN_MSK.VALUE bit field is the mask bit for each channel. The first MSB corresponds to channel 3, the second MSB to channel 2 and so on. If the mask is set for a particular channel, that channel is not converted. By default, channels 0-3 are not masked.

Control Register

The HADC_CTL register contains control bits that configure various module settings start or reset the module.

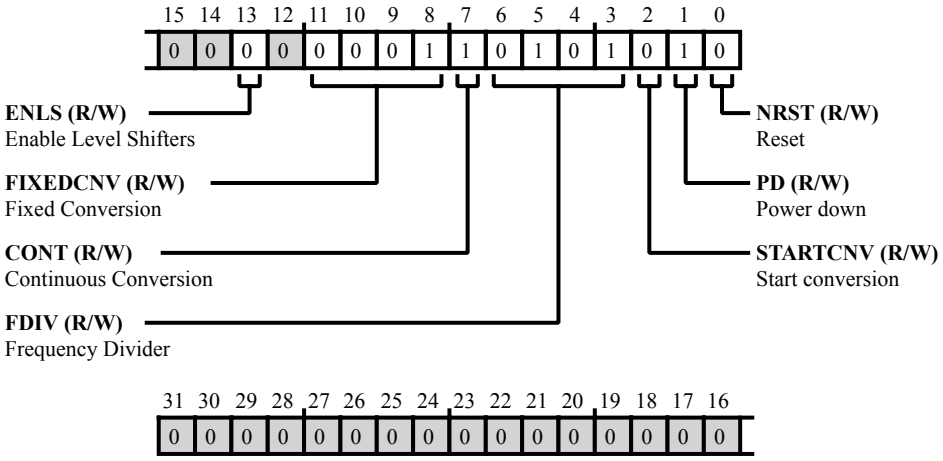

Figure 34-6: HADC_CTL Register Diagram

Table 34-8: HADC_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
13 (R/W)	ENLS	Enable Level Shifters. Setting the HADC_CTL.ENLS bit enables the level shifters, allowing the HADC analog side (which works in the VDD_EXT domain) to work with the digital core and interface is (in the VDD_INT domain).	
		0	Disable down level shifters
		1	Enable down level shifters
11:8 (R/W)	FIXEDCNV	Fixed Conversion. The HADC_CTL.FIXEDCNV bit configures the number of conversions = FIXEDCNV. This value determines how many times a sequence is converted when the HADC is in fixed conversion mode. This only applies when the HADC_CTL.CONT bit =0. Before changing the HADC_CTL.FIXEDCNV bit, clear the HADC_CTL.NRST bit (=0).	
7 (R/W)	CONT	Continuous Conversion. When the HADC_CTL.CONT bit =0, the ADC converts a sequence for a fixed number of times. This number is configured using the HADC_CTL.FIXEDCNV bit field. When the HADC_CTL.CONT bit =1, the ADC continuously converts a given sequence, provided the HADC_CTL.STARTCNV is held high.	
		0	ADC converts sequence for fixed number of times
		1	ADC continuously converts given sequence
6:3	FDIV	Frequency Divider.	

Table 34-8: HADC_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
(R/W)		The HADC_CTL.FDIV bit field configures $f_{SAMPLE}=f_{CLK}=/(FDIV+1)$. Select f_{CLK} and HADC_CTL.FDIV values so that f_{SAMPLE} is in the range of 50 kHz to 22.5 MHz. The minimum value for HADC_CTL.FDIV is 1. Before changing the HADC_CTL.FDIV bits, clear the HADC_CTL.NRST bit.	
2 (R/W)	STARTCNV	Start conversion. The HADC_CTL.STARTCNV bit needs to be set for the ADC to start converting data. If the ADC is running in non continuous mode, it is reset by hardware after the desired number of conversions is completed.	
		0	No action
		1	Start converting
1 (R/W)	PD	Power down. The HADC_CTL.PD bit powers down the analog circuitry of the ADC. After this bit returns to 0 a finite power-up time is required before the ADC can start converting data.	
		0	No action
		1	Power down the analog circuitry of the ADC
0 (R/W)	NRST	Reset. The HADC_CTL.NRST bit resets the ADC.	
		0	Reset the ADC
		1	No action

Channel Data Registers

The HADC_DATA[nn] registers N ranges from 0-3. Each corresponding to an ADC channel. ADC_DATA_0 corresponds to channel 0, ADC_DATA_1 to channel 1 and so on.

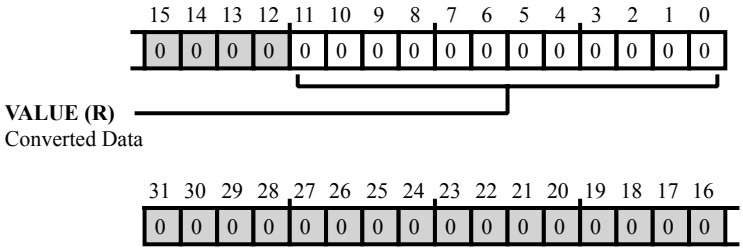

Figure 34-7: HADC_DATA[nn] Register Diagram

Table 34-9: HADC_DATA[nn] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/NW)	VALUE	Converted Data. The HADC_DATA[nn].VALUE bit field contains the digital code for the sampled analog value. Each channel has its own data register.

Interrupt Mask Register

The HADC_IMSK register masks (disables) or unmasks (enables) the interrupts as programmed. The reset value of the HADC_IMSK register is 0x00000000, masking these interrupts after reset.

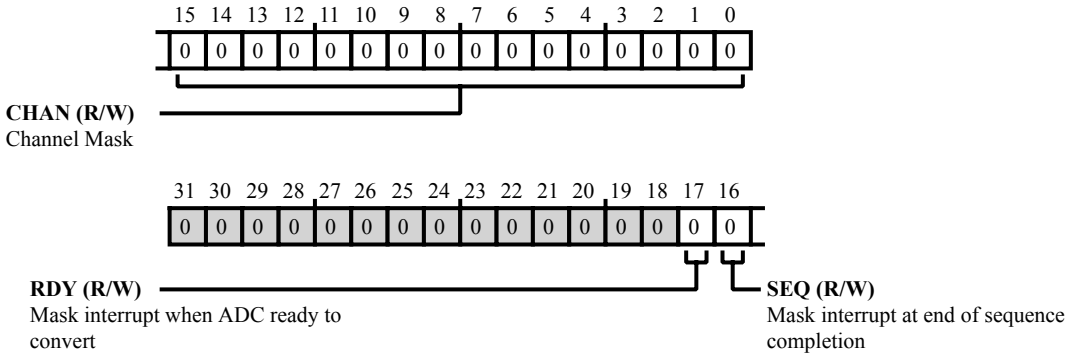

Figure 34-8: HADC_IMSK Register Diagram

Table 34-10: HADC_IMSK Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
17 (R/W)	RDY	Mask interrupt when ADC ready to convert. The HADC_IMSK.RDY bit masks the interrupt generated when ADC is ready to convert.	
16 (R/W)	SEQ	Mask interrupt at end of sequence completion. The HADC_IMSK.SEQ bit masks the interrupt which is generated at the end of sequence completion.	
		0	Interrupt is unmasked
		1	Interrupt is masked
15:0 (R/W)	CHAN	Channel Mask. The HADC_IMSK.CHAN bit field provides the interrupt mask bit for each channel. N ranges from 0-3. The MSB corresponds to channel 3, the second MSB to channel 2 and so on. If the interrupt mask for a particular channel is high, no interrupt is generated when that channel finishes the end of data conversion.	

Status Register

The HADC_STAT register contains bits that provide status information on the HADC module.

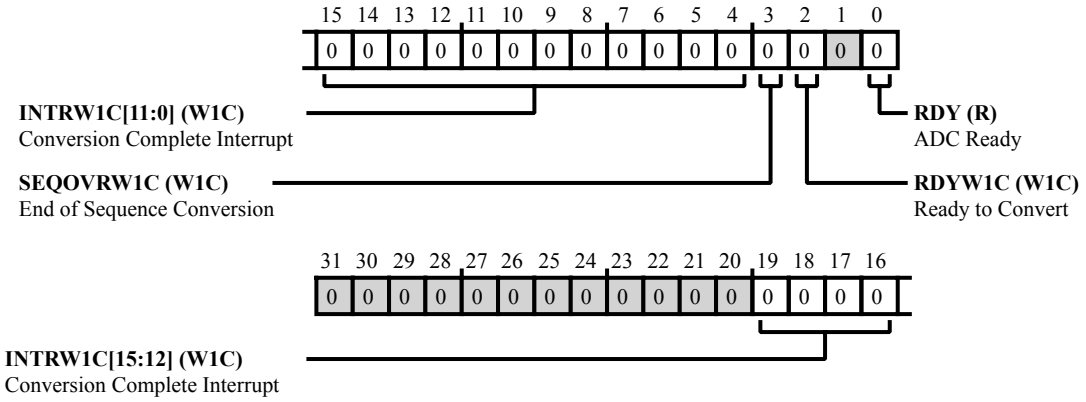

Figure 34-9: HADC_STAT Register Diagram

Table 34-11: HADC_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
19:4 (RX/W1C)	INTRW1C	Conversion Complete Interrupt. The HADC_STAT.INTRW1C bit field indicates when the corresponding ADC channel completes conversion. N ranges from 0-3 and the MSB corresponds to channel 3, the second MSB to channel 2 and so on. These bits are sticky and are W1C.
3 (RX/W1C)	SEQOVRW1C	End of Sequence Conversion. The HADC_STAT.SEQOVRW1C bit indicates the end of a sequence conversion and is a sticky status bit which is W1C.
2 (RX/W1C)	RDYW1C	Ready to Convert. The HADC_STAT.RDYW1C bit is the stick version of the HADC_STAT.RDY bit.
0 (R/NW)	RDY	ADC Ready. The HADC_STAT.RDY bit is set (=1) when the ADC is ready to convert data.

35 Reset Control Unit (RCU)

Reset is the initial state of the whole processor (or one of the cores). It is the 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. Exiting a full system reset starts with Core-0 only being ready to boot.

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).

RCU Features

The RCU module supports the following features:

- Hardware reset through the $\overline{\texttt{SYS\_HWRST}}$ pin
- Software system reset through RCU registers
- Hardware system reset through:
  - TRU module
  - SEC module
- Core reset through RCU registers

RCU Functional Description

This section provides information on the function of RCU module.

Hardware reset using SYS_HWRST pin

Asserting the $\overline{\texttt{SYS\_HWRST}}$ pin resets all functional units, except the RTC (if present)

Hardware reset through RCU

RCU can perform a full system reset which can be initiated through hardware blocks like the SEC, the TRU, and the oscillator watchdog

Software reset using RCU registers

Setting RCU_STAT.SWRST issues a software reset for all system units except the RTC module and RCU_BCODE, RCU_CRCTL and RCU_STAT registers

Core reset RCU registers

Core can be individually reset by software, or by setting the CRn bit in the RCU_CRCTL register

ADSP-BF70x 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 35-1: ADSP-BF70x 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_REVID	Revision ID Register
RCU_SIDIS	System Interface Disable Register
RCU_SISTAT	System Interface Status Register
RCU_STAT	Status Register
RCU_SVECT0	Software Vector Register 0
RCU_SVECT1	Software Vector Register 1
RCU_SVECT_LCK	SVECT Lock Register

ADSP-BF70x RCU Trigger List

Table 35-2: ADSP-BF70x RCU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
None			

Table 35-3: ADSP-BF70x 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
46	RCU0_C0RST0	RCU0 Core reset	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 the debug interface are set to their default states. History is lost.

System Reset (by target)

All functional units except the RCU 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 $\overline{\text{SYS\_HWRST}}$ input signal is asserted active (pulled low).

System Reset (by source)

Software can trigger the reset by writing to the RCU_CONTROL register or by another functional unit such as the TRU or any of the generic reset inputs.

For processors supporting hibernate mode, wake up from hibernate through the DPM is an extra reset source.

Core n Only Reset (by source)

For CORERESETEN =0, triggered by software. For CORERESETEN =1, triggered by software or system.

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 n when the core is reset.

The ***RCU Reset Sources*** table defines how reset sources affect the different functional units.

Table 35-4: RCU Reset Sources

Reset Source	Reset Type	Affected Functional Units
$\overline{\texttt{SYS\_HWRST}}$ pin assertion	Hardware Reset	All functional units, except RTC (if present)
Hibernate wake up event (wake up triggered reset)	System Reset	All functional units, except: • RTC (if present), • RCU_STAT, and • the units on the VDDEXT power domain
SYSCLK clock domain system reset	System Reset	All functional units, except: • RTC (if present), • RCU_CRCTL.CRn, • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
RCU_CTL.SYSRST bit set (software triggered reset)	System Reset	All functional units, except: • RTC (if present), • RCU_CRCTL.CRn, • RCU_STAT, • RCU_BCODE, and • the units on the VDDEXT power domain
RCU_CRCTL.CR[n] bit set (software triggered reset)	Core Only Reset	Core n only, for ($15 \geq n \geq 0$)

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.

NOTE: In processors with multiple cores, another core can act as a system master.

A core can be reset by software, either by setting any of the `RCU_CRCTL.CR[n]` bits ($15 \geq n \geq 0$). 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_SIDIS.SI[n]` bit to disable the interfaces. This stops DMA accesses to the core's L1 memory and accesses to the MMRs.
4. Test the `RCU_SISTAT.SI[n]` bit to detect that accesses to the core are disabled and all the pending transactions are complete.
5. Set the `RCU_CRCTL.CR[n]` bit to reset the core.
6. Poll the `RCU_CRSTAT.CR[n]` bit until the core is in reset.
7. Once the core is in reset, clear the `RCU_SIDIS.SI[n]` bit to reenable the core interfaces.
8. Clear the `RCU_CRCTL.CR[n]` bit to take the core out of reset.
9. Poll the `RCU_CRSTAT.CR[n]` bit until the core is out of reset.

Using a Core to Reset Itself

Core n can reset itself when the following sequence is followed. This functionality helps ensure that all the system transactions to or from core n complete.

1. Program MDMA0 with descriptor-based DMA to de-assert the disable request of the RCU for core n in RCU0_SIDIS.SIn.
2. Program the TRU such that the core 0 system interface disable acknowledge (C0_SI_DIS_ACK) is assigned the master trigger for MDMA0_SRC, MDMA0_DST, and RCU0_CnRST0 (core n reset) slave triggers.
3. Execute code on core n to program the disable request in the RCU (RCU0_SIDIS.SIn).
4. Go to IDLE or JUMP (PC, 0) loop.

As a result of this disable request, core n completes any outstanding transactions on its interfaces (MMR bus, MEM bus, and input DMA bus) and asserts the core disable ACK. This signal connects to the C0_SI_DIS_ACK master trigger of the TRU, and results in core n reset slave trigger being asserted. This assertion causes core n to be reset through the RCU. Simultaneously, MDMA0 clears the disable request of the RCU.

Having been reset, the core begins executing from the default address.

ADSP-BF70x Specific Information

The following RCU information is specific to ADSP-BF70x processors. When applying RCU feature to ADSP-BF70x systems, be aware that:

- The ADSP-BF70x processor is a single-core processor (core 0 only).
- The ADSP-BF70x processor does have an RTC (real-time clock).
- The SYSCLK domain sources for system reset come from the TRU (two slave triggers) and SEC.
- The SYSCLK domain sources for core reset come from the TRU.

ADSP-BF70x RCU Register Descriptions

Reset Control Unit (RCU) contains the following registers.

Table 35-5: ADSP-BF70x 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_REVID	Revision ID Register
RCU_SIDIS	System Interface Disable Register
RCU_SISTAT	System Interface Status Register
RCU_STAT	Status Register
RCU_SVECT0	Software Vector Register 0
RCU_SVECT1	Software Vector Register 1
RCU_SVECT_LCK	SVECT Lock 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 RESET or after hibernate.

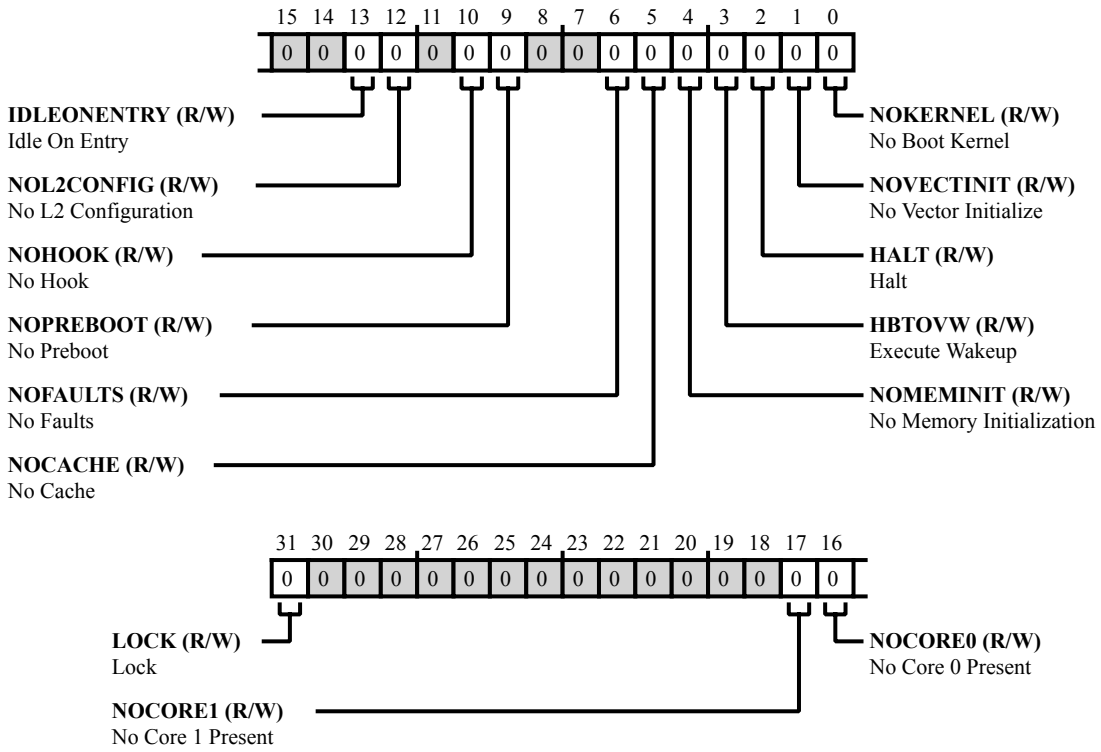

Figure 35-1: RCU_BCODE Register Diagram

Table 35-6: 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
17 (R/W)	NOCORE1	No Core 1 Present. The RCU_BCODE.NOCORE1 bit indicates the presence of core 1.	
		0	Core does not exist
		1	Core exists

Table 35-6: RCU_BCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
16 (R/W)	NOCORE0	No Core 0 Present. The RCU_BCODE.NOCORE0 bit indicates the presence of core 0.	
		0	Core does not exist
		1	Core exists
13 (R/W)	IDLEONENTRY	Idle On Entry. The RCU_BCODE.IDLEONENTRY bit configures the RCU to enter the idle state at startup.	
		0	Do not enter idle state
		1	Enter idle state
12 (R/W)	NOL2CONFIG	No L2 Configuration. The RCU_BCODE.NOL2CONFIG bit configures the RCU to not perform the L2 memory configuration.	
		0	Configure L2 memory
		1	Do not configure L2 memory
10 (R/W)	NOHOOK	No Hook. The RCU_BCODE.NOHOOK bit configures the RCU to not perform the hook routine.	
		0	Perform hook routine
		1	Do not perform hook routine
9 (R/W)	NOPREBOOT	No Preboot. The RCU_BCODE.NOPREBOOT bit configures the RCU to not perform the customer preboot routine.	
		0	Perform preboot
		1	Do not perform preboot
6 (R/W)	NOFAULTS	No Faults. The RCU_BCODE.NOFAULTS bit configures the RCU to not perform fault initialization.	
		0	Perform fault initialization
		1	Do not perform fault initialization
5 (R/W)	NOCACHE	No Cache. The RCU_BCODE.NOCACHE bit configures the RCU to not perform a cache initialization and to not enable the cache.	
		0	Enable and initialize cache
		1	Do not initialize or enable cache

Table 35-6: RCU_BCODE Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
4 (R/W)	NOMEMINIT	No Memory Initialization. The RCU_BCODE.NOMEMINIT bit configures the RCU to not perform a memory initialization.	
		0	Perform memory initialization
		1	Do not perform memory initialization
3 (R/W)	HBTOVW	Execute Wakeup. The RCU_BCODE.HBTOVW bit configures the RCU to execute a wakeup.	
		0	Do not wakeup
		1	Execute wakeup
2 (R/W)	HALT	Halt. The RCU_BCODE.HALT bit configures the RCU to execute the no boot routine.	
		0	Do not execute routine
		1	Execute routine
1 (R/W)	NOVECTINIT	No Vector Initialize. The RCU_BCODE.NOVECTINIT bit configures the RCU to not vector to the application.	
		0	Vector
		1	Do not vector
0 (R/W)	NOKERNEL	No Boot Kernel. The RCU_BCODE.NOKERNEL bit configures the RCU to not execute the boot kernel.	
		0	Execute boot kernel
		1	Do not execute boot kernel

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. Only bits enabled by the CORE_DISABLE_MASK can be written. Bits disabled by the CORE_DISABLE_MASK are read as "1" and cannot be modified by software.

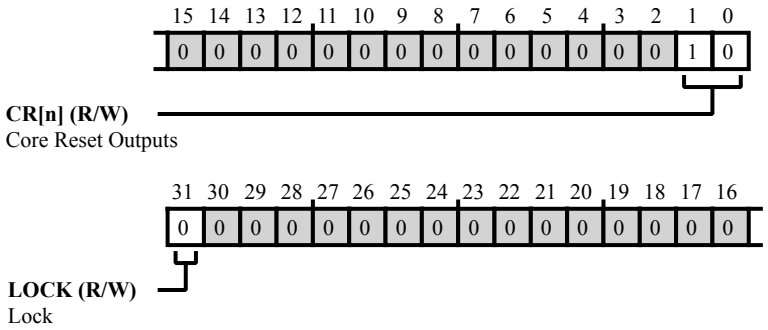

Figure 35-2: RCU_CRCTL Register Diagram

Table 35-7: 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
1:0 (R/W)	CR[n]	Core Reset Outputs. The RCU_CRCTL.CR[n] bits control CRES[1:0] core reset signals. RCU_CRES[n] can be individually controlled. It is reset to its 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[1:0] Deasserted
		3	RCU_CRES[1:0] 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.

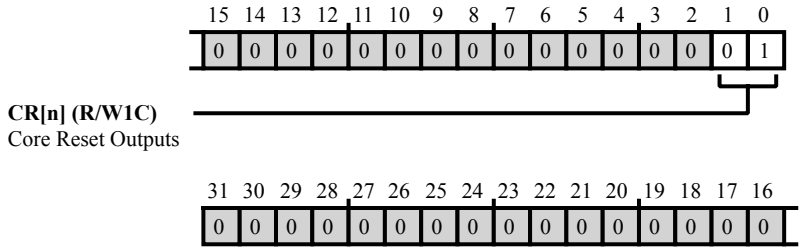

Figure 35-3: RCU_CRSTAT Register Diagram

Table 35-8: RCU_CRSTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1:0 (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].
		3	RCU_CRES[1:0] 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.

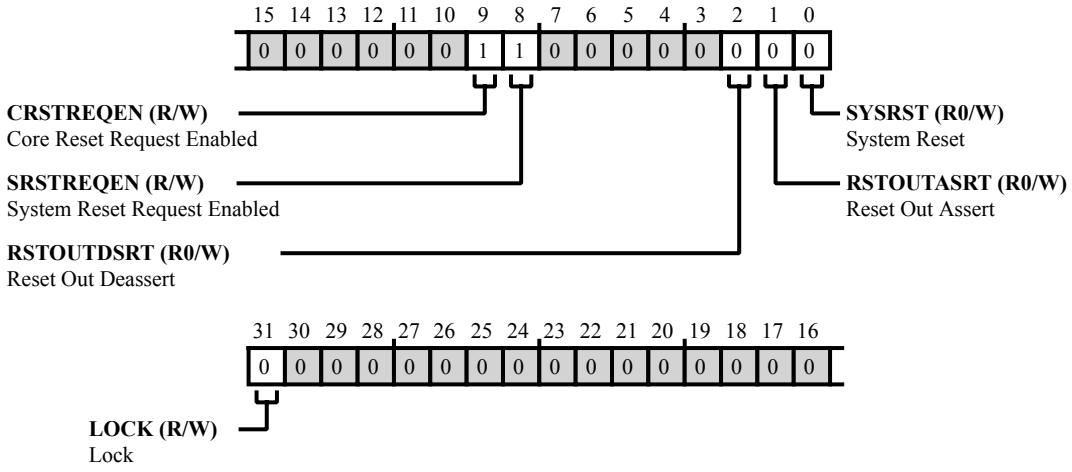

Figure 35-4: RCU_CTL Register Diagram

Table 35-9: 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
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
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

Table 35-9: RCU_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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.

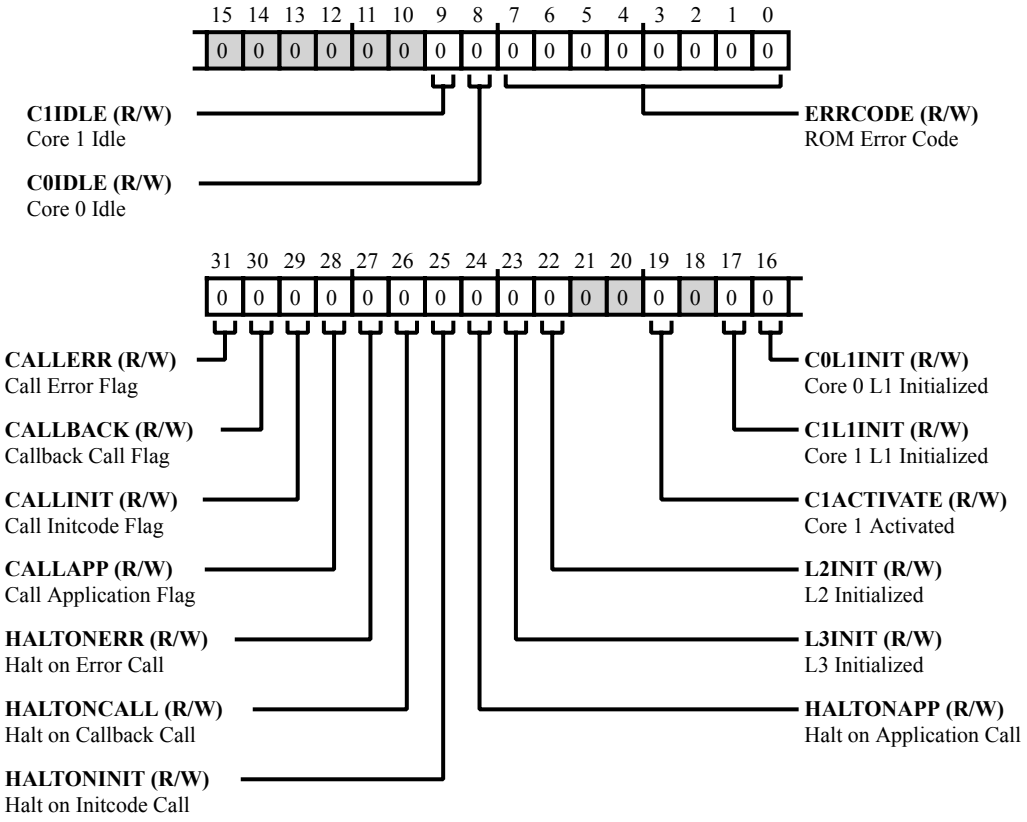

Figure 35-5: RCU_MSG Register Diagram

Table 35-10: RCU_MSG Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31 (R/W)	CALLERR	Call Error Flag. The RCU_MSG.CALLERR bit indicates that a flag has been set by the boot code prior to an error call.	
		0	Flag not set
		1	Flag set

Table 35-10: RCU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
30 (R/W)	CALLBACK	Callback Call Flag. The RCU_MSG.CALLBACK bit indicates that a flag has been set by the boot code prior to a callback call.	
		0	Flag not set
		1	Flag set
29 (R/W)	CALLINIT	Call Initcode Flag. The RCU_MSG.CALLINIT bit indicates that a flag has been set by the boot code prior to an initcode call.	
		0	Flag not set
		1	Flag set
28 (R/W)	CALLAPP	Call Application Flag. The RCU_MSG.CALLAPP bit indicates that a flag has been set by the boot code prior to an application call.	
		0	Flag not set
		1	Flag set
27 (R/W)	HALTONERR	Halt on Error Call. The RCU_MSG.HALTONERR bit generates an emulation exception prior to an error call.	
		0	Do not generate exception
		1	Generate exception
26 (R/W)	HALTONCALL	Halt on Callback Call. The RCU_MSG.HALTONCALL bit generates an emulation exception prior to a callback call.	
		0	Do not generate exception
		1	Generate exception
25 (R/W)	HALTONINIT	Halt on Initcode Call. The RCU_MSG.HALTONINIT bit generates an emulation exception prior to an init-code call.	
		0	Do not generate exception
		1	Generate exception

Table 35-10: RCU_MSG Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
24 (R/W)	HALTONAPP	Halt on Application Call. The RCU_MSG.HALTONAPP bit generates an emulation exception prior to an application call.	
		0	Do not generate exception
		1	Generate exception
23 (R/W)	L3INIT	L3 Initialized. The RCU_MSG.L3INIT bit indicates that the L3 resource is initialized.	
		0	Resource not initialized
		1	Resource initialized
22 (R/W)	L2INIT	L2 Initialized. The RCU_MSG.L2INIT bit indicates that the L2 resource is initialized.	
		0	Resource not initialized
		1	Resource initialized
19 (R/W)	C1ACTIVATE	Core 1 Activated. The RCU_MSG.C1ACTIVATE bit is used by tools for activation of Core 1.	
17 (R/W)	C1L1INIT	Core 1 L1 Initialized. The RCU_MSG.C1L1INIT bit indicates that the core 1 L1 resource is initialized.	
		0	Resource not initialized
		1	Resource initialized
16 (R/W)	C0L1INIT	Core 0 L1 Initialized. The RCU_MSG.C0L1INIT bit indicates that the core 0 L1 resource is initialized.	
		0	Resource not initialized
		1	Resource initialized
9 (R/W)	C1IDLE	Core 1 Idle. The RCU_MSG.C1IDLE bit indicates that core 1 is in a safe idle state in ROM.	
8 (R/W)	C0IDLE	Core 0 Idle. The RCU_MSG.C0IDLE bit indicates that core 0 is in a safe idle state in ROM.	
7:0 (R/W)	ERRCODE	ROM Error Code. The RCU_MSG.ERRCODE bit indicates the error code of the ROM. It is valid only when in the error handler.	

Message Clear Bits Register

The RCU_MSG_CLR register is used to clear bits in RCU_MSG register. Reading this register returns 0x00000000.

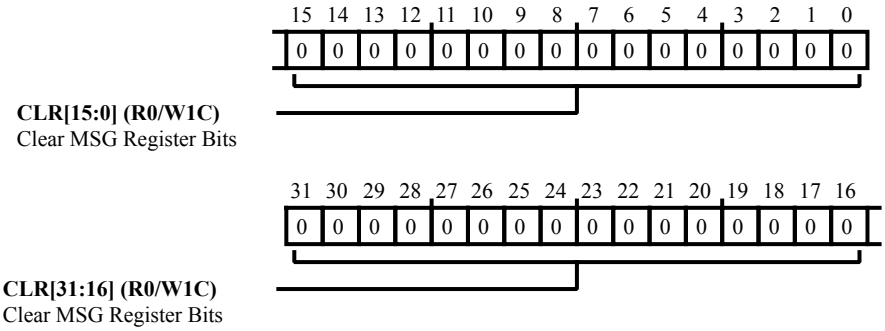

Figure 35-6: RCU_MSG_CLR Register Diagram

Table 35-11: 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.

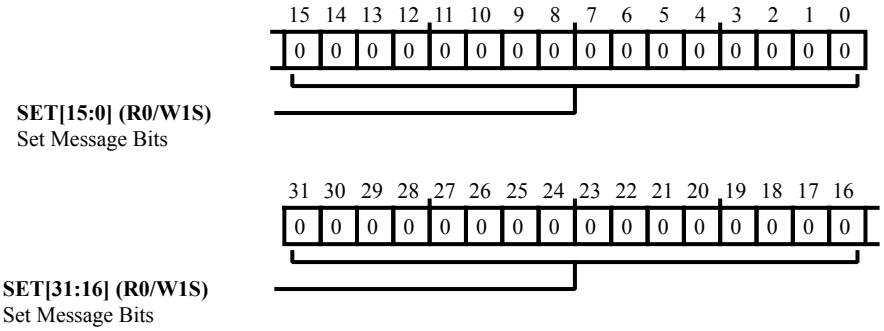

Figure 35-7: RCU_MSG_SET Register Diagram

Table 35-12: 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.

Revision ID Register

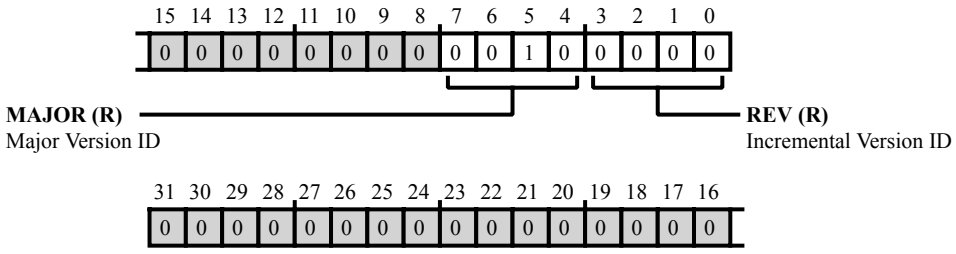

Figure 35-8: RCU_REVID Register Diagram

Table 35-13: RCU_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.

System Interface Disable Register

The RCU system interface disable register (RCU_SIDIS) lets the RCU assert a system interface disable request to functional units in the processor. This register is set to its default values by a hard reset or any system reset event. For information on mapping between RCU_SIDIS bits and functional units, see the RCU functional description.

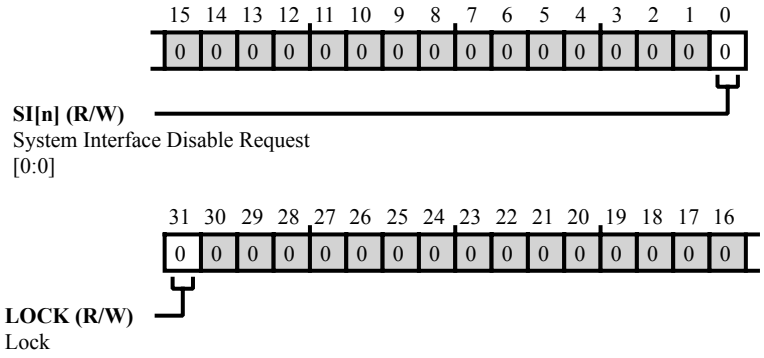

Figure 35-9: RCU_SIDIS Register Diagram

Table 35-14: RCU_SIDIS 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_SIDIS.LOCK bit is set, the RCU_SIDIS register is read only (locked).	
		0	Unlock
		1	Lock
0 (R/W)	SI[n]	System Interface Disable Request [0:0]. Each RCU_SIDIS.SI[n] bit corresponds to a functional unit in the processor that supports the system interface disable request-acknowledge protocol.	
		0	RCU_SI_DISABLE_REQ[0:0] deasserted
		1	RCU_SI_DISABLE_REQ[0:0] asserted

System Interface Status Register

The RCU system interface status register (RCU_SISTAT) indicates whether a functional unit has or has not acknowledged an RCU unit disable request.

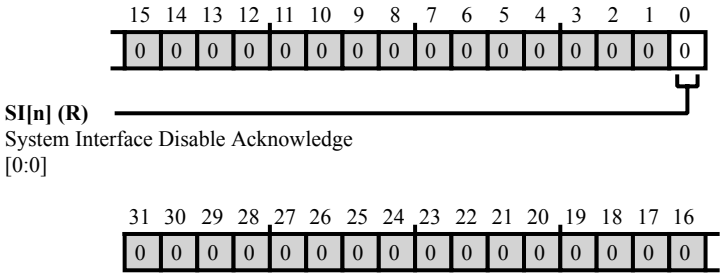

Figure 35-10: RCU_SISTAT Register Diagram

Table 35-15: RCU_SISTAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
0 (R/NW)	SI[n]	System Interface Disable Acknowledge [0:0]. The RCU_SISTAT.SI[n] bit indicates whether a functional unit has or has not acknowledged an RCU unit disable request.	
		0	No Acknowledge
		1	SI_DISABLE_ACK[0:0] asserted

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.

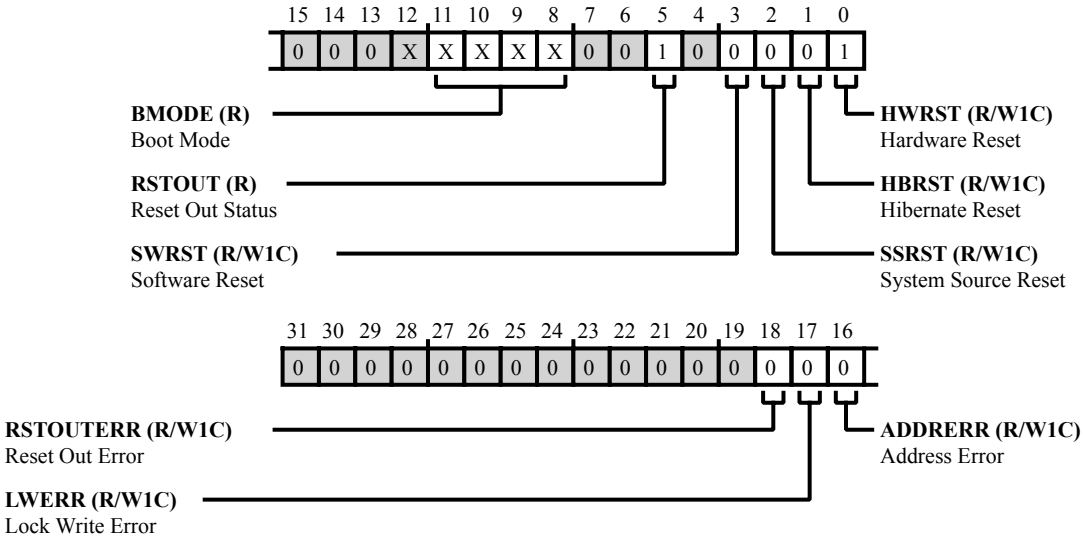

Figure 35-11: RCU_STAT Register Diagram

Table 35-16: 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 35-16: 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
1 (R/W1C)	HBRST	Hibernate Reset. The RCU_STAT.HBRST bit indicates that a hibernate reset has occurred since the last time a hardware reset occurred or since the RCU_STAT.HWRST 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

Software Vector Register 0

The RCU_SVECT0 register contains the default location of the first instruction to execute after a reset.

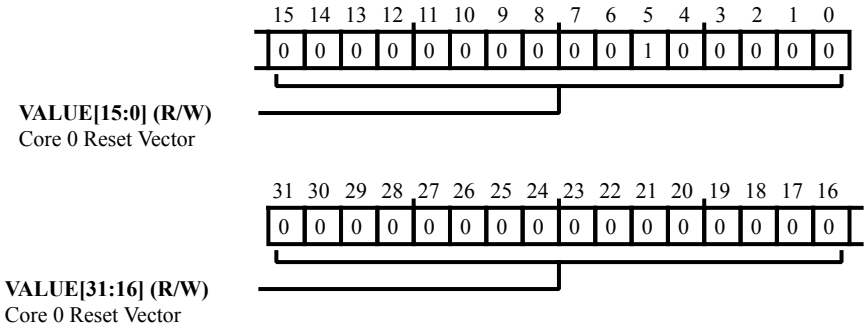

Figure 35-12: RCU_SVECT0 Register Diagram

Table 35-17: RCU_SVECT0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Core 0 Reset Vector. The RCU_SVECT0.VALUE bit field contains the default location of the first instruction to execute after a reset.

Software Vector Register 1

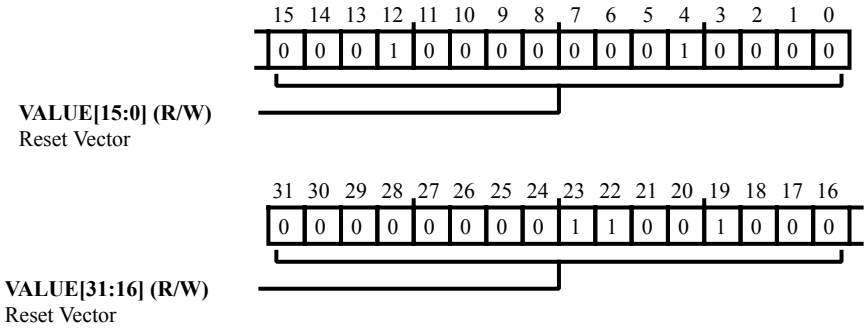

Figure 35-13: RCU_SVECT1 Register Diagram

Table 35-18: RCU_SVECT1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Reset Vector.

SVECT Lock Register

The RCU software vector lock register (RCU_SVECT_LCK) provides a register lock and software vector n enable bits for each processor core on the product. This register is set to its default values by a hard reset or any system reset event.

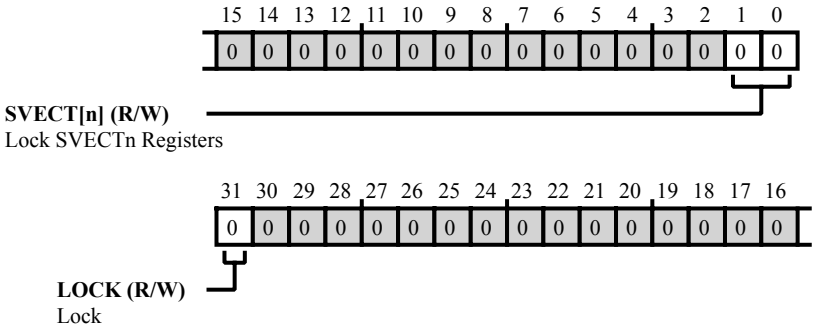

Figure 35-14: RCU_SVECT_LCK Register Diagram

Table 35-19: RCU_SVECT_LCK 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_SVECT_LCK.LOCK bit is set, the RCU_SVECT_LCK register is read only (locked).	
		0	Unlock
		1	Lock
1:0 (R/W)	SVECT[n]	Lock SVECTn Registers. If the global lock bit is set (SPU_CTL.GLCK bit =1) and the RCU_SVECT_LCK.SVECT[n] bit is set, the RCU_SVECT0 and RCU_SVECT1 registers are read only (locked).	

36 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 reset state. This section gives an in-depth description of these events and how to integrate an application effectively.

On reset, the processor begins fetching instruction from an internal ROM. The boot code contained within the ROM is designed to facilitate loading an application. The boot code can automatically initialize certain peripherals for communication based on a chosen boot mode, then load an application. For more information on what boot modes are available, see the Boot Modes section. The boot code can efficiently load an entire application, code, and data, into appropriate locations after the development tools repackage the application into a boot stream.

A boot stream is an application or data that the boot-loader tool splits into blocks. A 16-byte header in each block provides instruction to the boot code for processing the associated data. The processor can perform several boot functions, depending on the flags set in the header. For more details on what options are available and a description of the stream format, refer to the Boot Loader Stream section.

The boot ROM provides a mechanism through available non-volatile programmable memory (OTP on this processor) to customize different aspects of the boot process. These customizations include: overriding default boot-peripheral instance, overriding default peripheral-timing parameters and disabling boot modes.

Many of the utilities of the boot code are also available to the application. These utilities include features such as copying memory, comparing memory, or loading another boot stream at run-time. The APIs may be used to help ensure that application code is more compatible with future products. For more details on available APIs, see the API Reference section.

In addition to APIs, the boot code provides the ability to define a custom boot mode. This capability helps when support is not available for a desired boot mode. It allows second stage boot loaders for non supported boot-peripherals to leverage a significant amount of the existing boot ROM functionality.

SRAM Requirements

The boot process reserves 8KB of L2 ECC Protected SRAM. This topic describes how the reserved memory region is utilized during boot.

The boot process requires SRAM resources for stack usage and for storage of various data items that require read write access during boot such as the descriptors used for secure boot operations. 8KB of L2 ECC protected SRAM is reserved for this purpose. The table below describes the various items stored in this memory region.

Table 36-1: Boot Process SRAM Requirements

Address	Size (Bytes)	Item	Description
0x08000000	3072	Stack for the boot process	The primary booting cores stack. Any processor core that is mastering a boot operation should locate the stack in this region in order to preserve security in secure boot operations.
0x08000C00	1024	Reserved	
0x08001000	4	Pointer to the `ADI_ROM_BOOT_CONFIG` object	Pointer to the boot configuration structure that is located on the stack. This location can be used to find the location of the boot structure on the stack for debug purposes.
0x08001004	12	Reserved	
0x08001010	1024	Internal Intermediate Buffer 0	The first of two internal buffers used for intermediate storage of boot content when using indirect and page mode accesses and for secure boot operations. Two buffers are used to allow SHA-224 and AES-128 operations to be performed on one buffer while simultaneously loading the other buffer.
0x08001410	1024	Internal Intermediate Buffer 1	The second of two internal buffers used for intermediate storage of boot content when using indirect and page mode accesses and for secure boot operations. Two buffers are used to allow SHA-224 and AES-128 operations to be performed on one buffer while simultaneously loading the other buffer.
0x08001810	16	`ADI_ROM_BOOT_HEADER` object	Storage location for all the block headers of the boot stream.
0x08001820	1750	Storage for Secure Boot related descriptors	Contains a number of buffers for the various descriptors used by the Cryptographic Accelerators as well as providing storage for the secure header of a secure boot stream.
0x08001EF6	0x08001FFFF	Reserved	

NOTE: In order to preserve the security of the product the 8KB region described here is not a bootable region of memory. If the boot process determines that a block of data in the boot stream is targeted towards this memory region the boot process will terminate and enter either the default error handler or if applicable a user defined error handler if installed. This reserved memory region is free for use after the boot process completes. In order to preserve security when using the boot API to boot a secure boot stream, the stack

used during the execution of the boot API must be located to the default location in this reserved 8KB region of memory.

Preboot Operations

Preboot is responsible for configuration of all system resources prior to executing the required boot operation.

The steps performed by the preboot process are described here in the order of execution. Numerous stages of the preboot process are conditional based upon the content of RCU_BCODE.

NOTE: Upon completion of a power on reset, hard reset and software triggered system reset events the processor is initially running by default in PLL Full-on mode with default CGU settings. The user may provision custom CGU settings in the OTP to improve boot performance.

Start-up Sequence

Describes the initial start-up sequence of all cores in the processor.

Upon completion of a power-on reset, hardware reset or system reset event, only a single core is released from reset and is responsible for managing the boot process. The following sections describe in detail the sequence of events that occur for each of the cores on the various product derivatives.

Soft Vector Processing

Soft vector processing provides a flexible core execution vectoring process supported across different core types.

The first operation of the boot software whenever a hard reset, software triggered system reset or core reset occurs is to start execution from the reset vector in the boot ROM located at 0x04000000. The reset vector contains software to implement a flexible software vectoring scheme for reset operations.

The soft vectoring implementation results in a core vectoring to the address stored in the cores corresponding RCU_SVECTn register. These registers are reset to default values upon a hardware or system reset event however for core only resets via RCU_CRCTL, a custom address may be loaded prior to executing the core reset event allowing for a core only reset to start execution from any user defined address. Refer to the RCU chapter to understand what reset sources result in the default reset values being applied to the RCU_SVECTn registers.

During the boot process the contents of the RCU_SVECT[n] register may change multiple times. Typically it gets set to a default application entry point during the preboot phase of the boot process. It may then be further updated depending upon the contents of the boot stream itself.

During debug sessions, debug tools may be required to manipulate the RCU_SVECT[n] registers and perform various reset sequences in order to get the processor into a valid and consistent state.

For the ADSP-BF70x family of processors, if the contents of RCU_SVECT0 are equal to the default reset value then the boot code will be executed, else the processor will vector to the address stored in the RCU_SVECT0 register.

Table 36-2: RCU_SVECTn Default Reset Values

Core ID	Corresponding RCU_SVECTn Register	Default Reset Value
0	RCU_SVECT0	0x04000000

Servicing Reset Interrupts

The reset interrupt has top priority and the processor services a reset event like other interrupts. Only emulation events have higher priority. The processor comes out of reset in supervisor mode having access to all system resources. The boot kernel can be seen as part of the reset service routine as it runs at top interrupt priority level.

The reset interrupt has top priority and the processor services a reset event like other interrupts. Only emulation events have higher priority. The processor comes out of reset in supervisor mode having access to all system resources. The boot kernel can be seen as part of the reset service routine as it runs at top interrupt priority level.

Even when the boot process has finished and the boot kernel passes control to the user application, the processor is still in the reset interrupt. To enter the user mode, the reset service routine must initialize the RETI register and terminate by an RTI instruction.

NOTE: As the boot kernel is running at the reset interrupt priority, $\overline{\text{NMI}}$ events, hardware errors, and exceptions are not served at boot time. As soon as the reset service routine returns, the processor can service the events that occurred during the boot sequence. It is recommended that programs install $\overline{\text{NMI}}$, hardware error and exception handlers before leaving the reset service routine.

Core Startup

Describes the initial operations performed by the core immediately after being released from reset and soft vector processing has completed.

Upon initial release from the reset event the soft vectoring process results in the core executing the boot process as long as the RCU_SVECT0 register contained the default reset value during the soft vector processing. The core performs the following initial operations for setting up the boot process:

- Disable DCLK via DMC_PHY_CTL4.CLKDIS
- Save RCU_STAT to the core registers
- Disable ICPLBs and DCPLBs
- Execute Wakeup Optimization commands
- Configure L2 Controller
- Initialize the ECC protected L2 memory
- Setup the C runtime environment
- Call the preboot function passing the previously read RCU_STAT contents as an argument

NOTE: The boot kernel is running at the reset interrupt priority. The loaded application is expected to enter the user mode, by initializing initialize the RETI register and terminate the reset interrupt by executing an RTI instruction.

Wakeup Optimizations

Wakeup optimizations allow for various clock domains to be shut down to help minimize power in wakeup events until those domains are required.

The wakeup optimizations are only executed in the event from a wakeup from hibernate and the wakeup actions are enabled as dictated by the ADI_ROM_SYSCTRL::ulWUA_Flags parameter that is stored in DPM_RESTORE0.

The following operations are supported at this stage of the processing:

- Restoration of RCU_BCODE from DPM_RESTORE4 register if ROM_WUA_BCODE flag is set.
- Setting of CGU_SCBF_DIS.SCLK0BF if ROM_WUA_SCLK0BF0DIS is set.
- Setting of CGU_SCBF_DIS.SCLK1BF if ROM_WUA_SCLK0BF1DIS is set.
- Setting of CGU_SCBF_DIS.DCLKBF if ROM_WUA_DCLKBF0DIS is set.
- Setting of CGU_SCBF_DIS.DCLKBF if ROM_WUA_OUTCLKBF0DIS is set.

CAUTION: Disabling of some of the above mentioned clocks may have a direct impact on boot. Users need to ensure that they do not disable a clock that has a resource present that is required for completion of the boot process to occur otherwise a boot failure will result.

L2 Controller Configuration

The L2 controller is configured to ensure all L2 memory banks have ECC enabled and that the L2 memory scrub feature is disabled.

Table 36-3: L2 Controller Configuration

Register	Value	Description
L2CTL_CTL	0x00000000	Ensure all L2 memory banks have ECC enabled
L2CTL_ACTL_C0	0x00000000	Ensure core can access all banks
L2CTL_ACTL_SYS	0x00000000	Ensure DMA can access all banks
L2CTL_SCTL	Current contents with L2CTL_SCTL.SEN cleared	Ensure L2 Memory Scrub feature is disabled

NOTE: The L2 Controller configuration is enabled by default and may be optionally bypassed by setting RCU_BCODE.NOL2CONFIG.

L2 Memory Initialization

The L2 memory subsytem is ECC protected by default. The L2 memory must therefore be intiallized to ensure that no read from the L2 memory generates an ECC error. All L2 memory banks are initialized via L2CTL_INIT. The boot process waits for each memory bank to signal initialization completion via L2CTL_ISTAT. Upon completion of the L2 memory initialization RCU_MSG.L2INITis set.

NOTE: The L2 memory initialization process is enabled by default and may be optionally bypassed by setting RCU_BCODE.NOMEMINIT.

Run-Time Environment Initialization

Prior to using any software that requires a stack the processor sets up a basic C run-time environment in compliance with the run-time model supported by the processor core by setting up a stack and frame pointer and initializing various core registers to expected values.

The cores SP and USP registers are initially set to the top of the stack frame 0x08000C00. The address of a error handler is then pushed onto the stack twice. Should the boot process ever unwind the stack back to this point the error handler would be called.

Table 36-4: Core Register Configuration for C Run-Time Setup

Register	Value	Description
RETI, RETX, RETN, RETE	0	Zero all return registers
FP	0x08000BF8	Setup a new frame pointer
SP	0x08000BEC	Add space for incoming parameters for function called from this point onwards
LC0, LC1	0	Initialize loop counters to zero to disable hardware loops
L0, L1, L2, L3	0	Set DAG Length registers to 0 to avoid looping
B0, B1, B2, B3	0	Set DAG Base registers to 0 to avoid looping

Idle On Entry

The Idle On Entry implementation allows a means for a debugger to perform a system of the processor and halt the boot code before continuing with any further preboot operations.

When this feature is enabled the processor will execute a IDLE instruction and then continue once an event such as an emulator exception is serviced.

NOTE: Idle On Entry processing is disabled by default and may be optionally enabled by setting RCU_BCODE.IDLEONENTRY prior to performing a system reset operation.

Fault Configuration

Describes the initial fault sources that are enabled allowing the processor to signal a fault to the system.

The following faults are enabled via the SEC. Please note that only the faults are enabled, the boot process does not install any SEC interrupts.

Table 36-5: Initial Faults installed during Preboot

SEC Fault ID	SEC Fault Name	Description
1	INTR_SEC0_ERR	SEC Error
3	INTR_WDOG0_EXP	WDOG Expire
7	INTR_C0_HW_ERR	Core 0 Hardware Error
6	INTR_C0_DBL_FAULT	Core 0 Double Fault
71	INTR_OTPC0_ERR	OTPC Error
73	INTR_SMPU0_ERR	SMPUError
74	INTR_SMPU1_ERR	SMPUError
89	INTR_SOFT3_INT	Software Driven Interrupt 3. This is raised in the boot rom error handler should it be entered.
64	INTR_CRC0_ERR	CRC Error
66	INTR_CRC1_ERR	CRC Error
83	INTR_CGU0_ERR	CGU Error
26	INTR_DMAC_ERR	DMA Channel Error

The SEC is configured to have a fault delay of 0x100 via SEC_FDLY.COUNT and SEC_FSRDLY.COUNT. This allows for a delay to be implemented before assertion of the fault should a customer error handler be installed and any SEC interrupts enabled and handled for more advanced second stage boot scenarios.

The $\overline{\text{SYS_FAULT}}$pins are configured via the SEC to support both incoming and out going faults by enabling both SEC_FCTL.FIEN and SEC_FCTL.FOEN. This allows the boot process to capture an incoming fault if asserted by the external system upon handover to the user application after boot.

NOTE: The installation of the fault sources is enabled by default and may be optionally bypassed via RCU_BCODE.NOFAULTS. Faults are also disabled when the NOBOOT boot mode is enabled.

SPU Configuration

The SPU is configured differently depending upon the detected security state of the device. The first operation clears any existing security violations that may be indicated via SPU_STAT.

For an open device, a device that has not had the locked bit set in OTP. The boot process makes all peripherals ignore security signals by setting SPU_SECURECTL.SSECCLR.

All master endpoints are then set to generate secure transactions. This is performed by setting SPU_SECUREP[n].MSEC for all SPU_SECUREP[n] entries.

If the security state of the processor is locked then none of the peripherals are configured to ignore security signals and only the following masters are configured to generate secure transactions.

Table 36-6: Locked Processor SPU Secure Masters During Boot

SPU Endpoint ID	Master Name
53	MDMA1 Source DMA Channel
54	MDMA1 Destination DMA Channel
55	MDMA2 Source DMA Channel
56	MDMA2 Destination DMA channel
57	CRC0
58	CRC1
65	PKTE

SMPU Configuration

The SMPU is used to restrict access to various memory regions in the processor. The configuration applied during boot differs depending on the locked state of the processor.

By default the SMPU instances only allow secure read and write transactions. The tables below describe the configurations for the different security states.

Table 36-7: SMPU Configuration

SMPU Instance	SMPU Instance	Open SMPU_SECURECTL Value	Locked SMPU_SECURECTL Value
L2 DMA	0	SMPU_SECURECTL.WNSEN | SMPU_SECURECTL.RNSEN	0x00000000 (Default Reset Value)
DDR	1	SMPU_SECURECTL.WNSEN | SMPU_SECURECTL.RNSEN	0x00000000 (Default Reset Value)

Speculative reads are also disabled by setting SMPU_CTL.RSDIS.

Secure Debug Key Processing

In the event the processor is locked, a secure debug key is required to be submitted via the debug tools and matched with a key on the processor. A 128-bit Secure Debug Key must be provisioned by user prior to locking the device.

The secure debug key is read from the OTP memory and then written to the corresponding register in the TAPC. After the key has been written the TAPC_SDBGKEY_CTL.VALID bit is set then allowing for a key compare operation to be performed once the debug tools then submit their key.

It is important that the debug tools wait for the boot software to load the key then set the TAPC_SDBGKEY_CTL.VALID before submitting the key for comparison.

The 128-bit Secure Debug Key is loaded as follows from the storage area in OTP.

Table 36-8: Secure Debug Key Load Procedure

Secure Debug Key[127:0]	**Register**
Secure Debug Key[31:0]	TAPC_SDBGKEY0
Secure Debug Key[63:32]	TAPC_SDBGKEY1
Secure Debug Key[95:94]	TAPC_SDBGKEY2
Secure Debug Key[127:96]	TAPC_SDBGKEY3

CAUTION: A key of 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF provisioned in OTP will result in the boot code bypassing the key load operation entirely. If debug access is then ever required the key must be loaded to the TAPC by user software. If the processor fails to boot perhaps due to corrupted firmware then the user will have no debug access. The only way to gain access would be to load an authenticated boot image that can then load the required keys prior to attempting to connect with a debugger.

CGU Restoration from the DPM on Wakeup Events

Wakeup from hibernate events can optionally enable the boot code to reinitialize the CGU from context saved in the DPM_RESTORE[n] registers.

In the event of a wakeup from hibernate event, adi_rom_sysControl() routine is called to retrieve the contents of the DPM_RESTORE[n] registers and store them to internal SRAM on the stack.

If wakeup actions are found to be enabled and the wakeup actions indicate instructions are provided to write the CGU registers, adi_rom_sysControl() is called a second time to allow for writes to the CGU to take effect and initialize the clocks accordingly.

NOTE: If the CGU is configured correctly and no errors occur the restoration of the CGU settings from OTP is bypassed.

CGU Configuration

Reconfigures the internal clocks on the processor for improved boot performance.

The boot process can optionally configure the CGU in order to improve boot performance. The settings to be applied to the CGU are located within the ADI_ROM_OTP_BOOT_CGU_INFO structure that has storage allocated in the OTP as part of the ADI_ROM_OTP_BOOT_INFO structure.

Typically, CGU configuration would be performed through the use of an Init Block in the boot stream. This provides greatest flexibility. In situations where boot time must be kept to a minimum, provide settings in the OTP that can be applied at this stage of preboot as opposed to during the boot process itself. When a processor is locked, the boot process does not support an Init Block in the boot stream. For a locked processor users must use the OTP in order to reconfigure the clocks without adopting a multi-stage boot strategy.

When the processor is initially released from reset, the CGU is configured for PLL Full-on mode. In order to improve boot performance the boot software may reconfigure the CGUfor optimal performance if the user has supplied settings in OTP. Once the CGU is reconfigured, the remainder of the boot process completes at user defined

clock settings. This feature is especially important for secure boot environments where the initialization codes in the boot stream are not supported.

Table 36-9: ADI_ROM_OTP_BOOT_CGU_INFO Members

Type	Name	Description
uint32_t	ctl_WEN:1 (bitfield)	Enable write to the CGU_CTL register
uint32_t	div_WEN:1 (bitfield)	Enable write to the CGU_DIV register
uint32_t	reserved0:1 (bitfield)	Reserved
uint32_t	div_DSEL:5 (bitfield)	CGU_DIV.DSEL value
uint32_t	div_CSEL:5 (bitfield)	CGU_DIV.CSEL value
uint32_t	div_S0SEL:3 (bitfield)	CGU_DIV.S0SEL value
uint32_t	div_SYSSEL:5 (bitfield)	CGU_DIV.SYSSEL value
uint32_t	div_S1SEL:3 (bitfield)	CGU_DIV.S1SEL value
uint32_t	div_OSEL:7 (bitfield)	CGU_DIV.OSEL value
uint32_t	ctl_DF:1 (bitfield)	CGU_CTL.DF value
uint32_t	ctl_MSEL:7 (bitfield)	CGU_CTL.MSEL value
uint32_t	auto_disable:1 (bitfield)	disable polling on auto-alignment of clocks, NOT RECOMMENDED!
uint32_t	clkoutsel_USBCLKSEL:6 (bitfield)	value
uint32_t	clkoutsel_CLKOUTSEL:5 (bitfield)	CGU_CLKOUTSEL.CLKOUTSEL value
uint32_t	clkoutsel_WEN:1 (bitfield)	Enable write to the CGU_CLKOUTSEL register
uint32_t	reserved1:12 (bitfield)	Reserved

If a CGU_STAT.WDIVERR, CGU_STAT.WDFMSERR, CGU_STAT.LWERR or CGU_STAT.ADDRERR is either present upon entry to the configuration routine or upon completion of the configuration then the default error handler is called and the boot process terminates.

NOTE: The configuration of the CGU will be bypassed if RCU_BCODE.NOPREBOOT is set when this part of the boot process is reached.

Default Application Entry Points

The core sets default application entry points for all cores in the processor. This step is performed in the event a boot stream does not contain blocks that specify the application entry point.

The table below defines the default application entry points for each core in the processor.

Table 36-10: RCU_SVECTn Default Application Entry Points

Core ID	Corresponding RCU_SVECTn Register	Application Entry Point
0	RCU_SVECT0	0x11A00000

The values defined here get overwritten by First Block located within the boot stream.

NOTE: The initialization of the RCU_SVECT0 register is enabled by default and may be optionally bypassed by setting RCU_BCODE.NOVECTINIT.

L1 Memory Initialization

The processor initializes all parity and ECC protected memories allowing for subsequent read operations to be performed without generation of an ECC or parity error.

The table below describes the methods used to initialize the various parity and ECC supported memories on the processor.

Table 36-11: L1 Memory Initialization

Resource To Fill Memory	Memory Type	Address	Count	Fill Value	Flag Set Upon Completion
MDMA1 and CRC0	L1 Data Bank A	0x11800000	0x8000 (Bytes)	0x00000000	RCU_MSG.C1L1INIT
MDMA2 and CRC1	L1 Data Bank B	0x11900000	0x8000 (Bytes)	0x00000000	
MDMA1 and CRC0	L1 Instruction SRAM/ Cache	0x11A00000	0x10000 (Bytes)	0x00000000	
MDMA2 and CRC1	L1 Data Bank C	0x11B00000	0x2000 (Bytes)	0x00000000	

NOTE: The Memory Initialization process is enabled by default and may be optionally bypassed by setting RCU_BCODE.NOMEMINIT.

Memory Faults Configuration

Describes the memory fault sources that are enabled allowing the processor to signal a fault to the system in the even of a memory error.

The following faults are enabled via the SEC. Please note that only the faults are enabled, the boot process does not install any SEC interrupts.

Table 36-12: Memory Faults installed during Preboot

SEC Fault ID	SEC Fault Name	Description
4	INTR_L2CTL0_ECC_ERR	ECC Error
8	INTR_C0_NMI_L1_PARITY_ERR	NMI or L1 Memory Parity Error

The SEC is configured to have a fault delay of 0x100 via SEC_FDLY.COUNT and SEC_FSRDLY.COUNT. This allows for a delay to be implemented before assertion of the fault should a customer error handler be installed and any SEC interrupts enabled and handled for more advanced second stage boot scenarios.

The $\overline{\text{SYS_FAULT}}$pins are configured via the SEC to support both incoming and out going faults by enabling both SEC_FCTL.FIEN and SEC_FCTL.FOEN. This allows the boot process to capture an incoming fault if asserted by the external system upon handover to the user application after boot.

NOTE: The installation of the fault sources is enabled by default and may be optionally bypassed via RCU_BCODE.NOFAULTS. Faults are also disabled when the NOBOOT boot mode is enabled.

Cache Configuration

The instruction cache configuration setup by the boot process is dependent upon the security state of the processor. The data cache is not enabled during boot however the boot software does initialize some initial DCPLB_ADDR[n] and DCPLB_DATA[n] registers.

The tables below describe the instruction and data cache configurations for open and locked devices. The difference with the instruction cache configuration for an open and locked processor is there are no CPLBs configured to allow for external code execution on the locked device.

Table 36-13: Open Processor Instruction Cache Configuration

ICPLB_ADDR[n]/ICPLB_DATA[n] Instance	ICPLB_ADDR[n] Value	ICPLB_DATA[n] Value
0	0x11A00000	0x03000300
1	0x04000000	0x03100200
2	0x04004000	0x03100200
3	0x04008000	0x03100200
4	0x0400C000	0x03000200
5	0x08002000	0x03000500
6	0x80000000	0x03000900
7	0x70000000	0x03000900
8	0x40000000	0x03000300
9	0x40010000	0x03000300

Table 36-14: Locked Processor Instruction Cache Configuration

ICPLB_ADDR[n]/ICPLB_DATA[n] Instance	ICPLB_ADDR[n] Value	ICPLB_DATA[n] Value
0	0x11A00000	0x03000300
1	0x04000000	0x03100200
2	0x04004000	0x03100200
3	0x04008000	0x03100200
4	0x0400C000	0x03000200
5	0x08002000	0x03000500

Table 36-15: Data Cache Configuration for Open and Locked Processor

DCPLB_ADDR[n]/DCPLB_DATA[n] Instance	DCPLB_ADDR[n] Value	DCPLB_DATA[n] Value
0	0x04000000	0x83000800
1	0x08000000	0x93700100
2	0x08001000	0x93000500
3	0x11800000	0x93000200
4	0x11804000	0x93000200
5	0x11900000	0x93000200
6	0x11904000	0x93000200
7	0x11B00000	0x93000200
8	0x80000000	0x93000900
9	0x70000000	0x93000900
10	0x40000000	0x93000800
11	0x40010000	0x93000800

The instruction cache and instruction parity support is then enabled by writing 0x00000206 to the `L1IM_ICTL` register. The data cache `L1DM_DCTL` register is left at the default reset value.

NOTE: The cache configuration is enabled by default and may be optionally bypassed by setting `RCU_BCODE.NOCACHE`. The is also a field in OTP, `ADI_ROM_OTP_BOOT_CFG::cacheDis` allowing for the bypassing of this configuration for power on and system reset events. The `ADI_ROM_OTP_BOOT_CFG::cacheDisInv` f8ield in OTP is provided as an override for this cache disable functionality. It should be noted that as the L1 instruction parity is also enabled at this stage bypassing this configuration will also bypass the enabling of L1 instruction parity support.

NO-BOOT Processing

No-Boot mode is executed on supported devices when selected by the SYS_BMODE[n] pins. The boot mode is intended as a recovery boot mode or for debug purposes. The core simply executes in an endless loop in the boot ROM thus terminating further execution of the boot process.

This boot mode is primarily intended for debug sessions when no boot source may be configured. It allows for a debugger to safely connect to the device and take control assuming the debugger has been granted access rights as defined by the processor security implementation.

NOTE: NO-BOOT processing is usually only entered as a result of the boot mode pin sampling resulting in execution of the No-Boot boot mode. This processing can also be optionally enabled by setting RCU_BCODE.HALT. The setting of RCU_BCODE.HALTcan be especially useful for debug sessions to force the execution of the NO-BOOT mode regardless of the SYS_BMODE[n] state allowing a user application to be loaded via the debug tools without fear of the image being corrupted as a result of attempting to boot through another source.

SYS_RESOUTb Processing

The $\overline{\text{SYS\_RESOUT}}$ pin can be used to signal to the external system that the processor is now in a state where it is ready boot

In order to signal to the external system that the processor in now in a configured state and ready to start the boot process, the boot software de-asserts the $\overline{\text{SYS\_RESOUT}}$ pin via RCU_CTL.RSTOUTDSRT.

DMC Restoration from the DPM on Wakeup Events

Wakeup from hibernate events can optionally enable the boot code to reinitialize the DMC from context saved in the DPM_RESTORE[n] registers.

In the event of a wakeup from hibernate event, adi_rom_sysControl() routine is called to retrieve the contents of the DPM_RESTORE[n] registers and store them to internal SRAM on the stack.

If wakeup actions are found to be enabled and the wakeup actions indicate instructions are provided to write the DMC registers, adi_rom_sysControl() is called a second time to allow for writes to the DMC to take effect and bring external memories out of self-refresh mode.

NOTE: If the DMC is configured correctly and no errors occur the restoration of the DMC settings from OTP is bypassed. Users must also be aware that this routine only supports bringing devices out of self-refresh mode. If external memory devices are not in self-refresh mode then the DMC settings must either be configured from OTP or with the use of initcodes in the boot stream.

DMC Configuration

Configures the DMC to allow booting directly to external DDR memory. Primarily intended to support booting to external memory when the processor is locked.

In order to boot to external DDR memories the DMC must be configured. Typically DMC configuration would be done with the support of

Typically, DMC configuration would be performed through the use of an Init Block in the boot stream. This provides greatest flexibility. When a processor is locked however, the boot process does not support an Init Block in the boot stream. For a locked processor users must use the OTP in order to configure the DMC without adopting a multi-stage boot strategy.

The table below provides details of the OTP region that is used to store theDMC configuration to be applied. In addition to the settings described in the table below `ADI_ROM_OTP_BOOT_CFG::dmcEn` must be set in order for the settings to be applied.

There is an additional single bit located in OTP, `ADI_ROM_OTP_BOOT_CFG::dmcInv`, allowing users to invalidate the DMC settings stored in OTP.

NOTE: Once `ADI_ROM_OTP_BOOT_CFG::dmcInv` has been set in OTP there is no means to configure the DMC during the preboot phase.

Table 36-16: ADI_ROM_OTP_DMC_CONFIG Members

Type	Name	Description
`uint32_t`	`reserved0:10` (bitfield)	Reserved
`uint32_t`	`ulDDR_DLLCTL:12` (bitfield)	Contents of `DMC_DLLCTL [11:0]`
`uint32_t`	`ulDDR_EMR2:8` (bitfield)	Contents of `DMC_EMR2 [7:0]`
`uint32_t`	`reserved1:2` (bitfield)	Reserved
`uint32_t`	`ulDDR_CFGCTL`	Packed content of `DMC_CTL` Register, `DMC_CFG` registers.
`uint32_t`	`ulDDR_MREMR1`	Packed content of `DMC_EMR1` Register, `DMC_MR` registers.
`uint32_t`	`ulDDR_TR0`	Content of `DMC_TR0`
`uint32_t`	`ulDDR_TR1`	Content of `DMC_TR1`
`uint32_t`	`ulDDR_TR2`	Content of `DMC_TR2`
`uint32_t`	`ulDDR_PHYCTL0`	Content of `DMC_PHY_CTL0`
`uint32_t`	`ulDDR_PHYCTL145`	Packed content of `DMC_PHY_CTL1` , `DMC_PHY_CTL4` and `DMC_PHY_CTL5` registers.
`uint32_t`	`ulDDR_PHYCTL2`	Content of `DMC_PHY_CTL2`
`uint32_t`	`ulDDR_PHYCTL3`	Content of `DMC_PHY_CTL3`
`uint32_t`	`ulDDR_CAL_PADCTL0_PHY_STAT3_0`	Packed content of `DMC_CAL_PADCTL0` , `DMC_PHY_STAT0` , and `DMC_PHY_STAT3` registers.
`uint32_t`	`ulDDR_CAL_PADCTL2`	Content of `DMC_CAL_PADCTL2`

NOTE: The configuration of the DMC will be bypassed if `RCU_BCODE.NOPREBOOT` is set when this part of the boot process is reached.

Memory Boot

Memory Boot allows for execution from and external memory source such as external DRAM in the event of a wakeup action bypassing the regular boot process allowing for fast bring-up of the processor for wakeup events.

In the event of a wakeup event if the settings in the `ADI_ROM_SYSCTRL::ulWUA_Flags` parameter located in `DPM_RESTORE0` indicate a memory boot is being requested the boot code vectors to the address stored in the `ADI_ROM_SYSCTRL::ulWUA_BootAddr` parameter residing in `DPM_RESTORE1`.

Bypassing the Boot Process

It is possible to bypass the actual booting process and execute from the address stored in the cores soft vector register. This can be useful when working in emulation sessions as it provides a mechanism to be able to execute directly from an accessible memory that already contains executable code.

The boot process can be bypassed by setting `RCU_BCODE.NOKERNEL` and the core will instead vector to the address stored in the cores corresponding `RCU_SVECT[n]` register instead of calling the required boot mode.

This feature would often be used along with other features such as disabling of memory initialization.

Boot Mode Disable

Specific boot modes can be permanently disabled via OTP resulting in a boot error should the disabled boot mode be enabled via the `SYS_BMODE[n]` pins.

A byte of storage is provided in the OTP allowing for disabling of up to 8 boot modes. The boot mode disable field can be programmed using the `adi_rom_otp_pgm()` routine. The `otp_data::bootModeDisable` member is used in the program operation to disable the various boot modes.

Table 36-17: Boot Mode Disable

`otp_data::bootModeDisable` Bit Position	**Corresponding Boot Mode**
0	SPI Master Boot Mode
1	SPI Slave Boot Mode
2	UART Slave Boot Mode
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

Boot Command Customization

Boot Command Customization allows for the permanent customization of a particular boot mode. For example it is possible to change the peripheral instance that is used for the boot operation.

Storage is provide in OTP for a `command` item for each of the supported boot modes. The storage is provided in the `ADI_ROM_OTP_BOOT_CMD_INFO` member of `ADI_ROM_OTP_BOOT_INFO`. Refer to the corresponding boot modes boot command description for details on the supported command options.

Table 36-18: ADI_ROM_OTP_BOOT_CMD_INFO Members

Type	Name	Description
`uint32_t`	`spiMasterBootCmd`	SPI Master boot command
`uint32_t`	`spiSlaveBootCmd`	SPI Slave boot command
`uint32_t`	`uartBootCmd`	Uart Slave boot command

NOTE: Before programming the boot command to the OTP. It is advisable to thoroughly evaluate the boot command using the `adi_rom_Boot()` API to ensure the boot command provides the desired functionality. Once the command is programmed to OTP there is no means to revert to the original default settings.

Boot Mode Specific SPU Configuration

Prior to performing the actual boot process, the processors SPU resources specific to the boot mode selected are configured. This is performed in the preboot phase as opposed to within the boot mode itself when calling the boot API as it isolates the security functionality of the processor allowing it to be handled specifically by a separate process.

The following additional SPU resources are configured as secure masters according to the boot mode selected.

Table 36-19: Boot Mode Specific SPU Configuration

Boot Mode	SPU Endpoint ID	Master Name
SPI Master Boot (Memory Mapped Mode)	51, 52	MDMA0 Source DMA Channel, MDMA0 Destination DMA Channel
SPI Master Boot (Peripheral Mode)	44, 42, 40	SPI2 Receive DMA, SPI1 Receive DMA, SPI0 Receive DMA
SPI Slave Boot	44, 42, 40	SPI2 Receive DMA, SPI1 Receive DMA, SPI0 Receive DMA
UART Slave Boot	48, 46	UART 0 Receive DMA, UART1 Receive DMA

NOTE: Note that for a given boot mode in the table not all the SPU resources are configured. Only a single peripheral instance is enabled according the peripheral instance selected for boot. For example if the boot command for the boot mode indicates boot from UART0 only the UART0 Receive DMA is configured, the other UART Receive DMAs are not configured for secure access.

Executing the Boot Mode

The boot mode is called using the `adi_rom_Boot()` routine resulting in the fetching and processing of the boot stream from the configured boot source.

The table below provides default parameters passed to each of the supported boot modes. For details on the API usage please refer to `adi_rom_Boot()`.

Table 36-20: Default Boot ROM API Parameters

Boot Mode	pAddress	flags	blockCount	pHook	command
No Boot	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x00000000
SPI Master Boot	0x40000000	0x00040000	0x00000000	Points to an empty routine in ROM	0x00010207
SPI Slave Boot	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x00000212
UART Slave Boot	0x00000000	0x00000000	0x00000000	Points to an empty routine in ROM	0x00000013

The hook function installed via `pHook`on this product performs no additional configuration.

Boot Modes

The boot implementation provides built-in support for booting from various peripherals. The ***Booting Modes*** table describes the supported boot modes.

In *slave* boot modes, the processor functions as a slave to any host device. In these modes, the host device usually controls the processor `SYS_HWRST` input. Typically, the host applies the reset sequence and waits until the processor is ready to boot, depending on the peripheral in use, and transmits the boot stream data to the processor. Handshake signals are most likely used to signal to the host that the processor is ready to accept more data.

In a *master* boot modes, the processor controls the peripheral and requests data via the peripheral as and when required.

Individual boot modes can be disabled. For more information about disabling boot modes, see Boot ROM OTP Customizations.

Table 36-21: Booting Modes

SYS_BMODE[1:0]	Boot Source	Description
00	No Boot	The processor does not boot. Rather the boot kernel executes some of the preboot operations then enters and endless `IDLE` state.
01	SPI Master Boot	Boot from integrated Flash memory through the SPI2 peripheral configured for memory mapped mode.
10	SPI Slave Boot	Boot through the SPI2 peripheral configured as a slave.
11	UART Boot	Boots through UART0 configured as a slave receiver.

No-Boot Mode

No-Boot mode is intended for device-recovery purposes caused by incorrect programming of the boot source memory allowing for target connection through an emulator. Emulation tools can also leverage the No-Boot functionality allowing for debug sessions to run the preboot software prior to loading an application while preventing the boot process from continuing and clobbering data loaded by the emulator.

This boot mode results in a number of preboot operations being performed before then placing the core into a safe endless loop located in the boot ROM. For a complete list of operations performed when No-Boot is selected please refer to Preboot Operations. The core terminates at the NO-BOOT Processing stage of the preboot process.

SPI Master Boot Mode

The SPI master boot routine provides support for booting the processor from SPI flash memories.

The SPI boot mode utilizes a device auto-detection feature that is enabled by default allowing for the boot stream itself to instruct updates to the SPI configuration and the read command used allowing for more efficient transactions.

Boot From External SPI Flash Devices

The SPI boot mode supports booting from 24-bit addressable flash devices. The boot mode uses the MDMA channels by default and configures the SPI flash for memory mapped functionality. Peripheral DMA mode is also supported when calling the boot mode via `adi_rom_Boot()`. Support for 32-bit addressable flash devices can be achieved by disabling the device auto-detection and supplying the required configuration via the `command` parameter.

When auto-device detection is enabled, the SPI memory is initially read using the standard 0x03 SPI read command with a reduced clocking frequency for maximum compatibility. The first nibble of the boot stream is then used to reconfigure the SPI interface and possible the SPI flash. Refer to SPI Device Detection Routine.

NOTE: Support for automatic device detection via the first nibble of the boot stream is not supported when booting secure boot streams. Instead when signing the boot image an attribute can be set in the image header that specifies the configuration to use.

For booting, the SPI memory is connected as shown in the ***SPI Memory Connections*** figure.

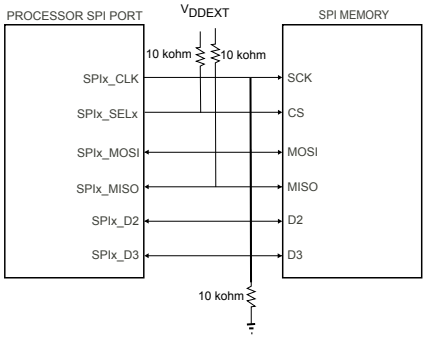

Figure 36-1: SPI Memory Connections

The pull-up resistor on the slave select signal ensures that the memory is deselected when the pin is in a high-impedance mode such as during reset.

Initialization codes are allowed to manipulate ADI_ROM_BOOT_CONFIG::dBootCommand to extend the boot mechanism to a second SPI memory connected to another slave select pin. Updating the field that specifies the slave select signal for use allows the boot process to manage larger boot streams than are able to fit in a single SPI device.

NOTE: If modifying the slave select signal used during the boot process, configure the pin multiplexing to enable the correct functionality for the pin. Once the boot process has proceeded past the configuration function and the boot process has actually started, the boot kernel will not perform any further pin multiplexing operations.

For SPI master boot peripheral mode, the SPE, MSTR, and SZ bits are set in the SPIx_CTL register. The TIMOD=2 bits enable the receive DMA mode. The CPOL and CPHA bits are set by default, resulting in SPI mode 3. The boot kernel does not allow SPIx hardware to control the SPI_SEL[n] pin. Instead, software toggles this pin.

SPI Device Detection Routine

Since the boot mode supports booting from various SPI memories, the boot kernel automatically detects what type of memory is connected. To determine whether the SPI memory device requires an 8, 16, 24, or 32-bit addressing scheme, the boot kernel performs a device detection sequence prior to booting. The SPI_MISO signal requires a pull-up resistor. The routine relies on the fact that memories do not drive their data outputs unless the right number of address bytes are received.

Initially, the boot kernel transmits the read command on the SPI_MOSI line. Once the command has been sent, the boot kernel proceeds to transmit a single address byte and waits until the receive FIFO indicates that the buffer is no longer empty. The first received byte is discarded. The boot code then proceeds to issue another address byte while simultaneously receiving a byte. The process continues until a non-0xFF or 0x00 byte is received or until the full 4 address bytes is sent without any valid data being returned.

The receiving of a non-0x00 or 0xFF byte tells the boot code whether the memory device requires 8, 16, 24, 32 address bits. The lower nibble of the received byte is then used to further customize the boot mode. This nibble is referred to as the BCODE. The boot code applies settings to the SPI peripheral according to the ***SPI Master Boot BCODE Descriptions***.

If the received value equals 0x00 or 0xFF, it is assumed that the memory device has not driven its data output thus, another zero byte is transmitted and the received data is tested again.

If the value still equals 0xFF, device detection continues. Device detection aborts immediately when a byte different than 0xFF is received. The boot process continues with normal boot operation and it re-issues a command to re-read from address 0. Two read sequences load the first block header. Separate read sequences load further block headers and block payload fields.

The ***SPI Device Detection Principle*** figure illustrates how individual devices behave.

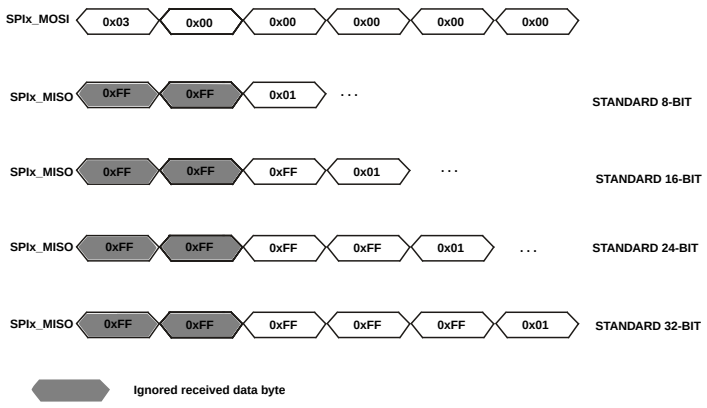

Figure 36-2: SPI Device Detection Principle

Table 36-22: SPI Master Boot BCODE Descriptions

BCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0x0						SCLK0/32	Unused
0x1	STANDARD	0x03	0	1	1	SCLK0/32	Legacy single-bit SPI mode
0x2	STANDARD	0x03	0	1	1	SCLK0/4	Legacy single-bit SPI mode
0x3	FAST READ	0x0B	1	1	1	SCLK0/2	Single bit with dummy address byte
0x4	FAST READ	0x0B	1	1	1	SCLK0/2	Single bit with dummy address byte. `SPI_CTL.FMODE` is enabled for full cycle access.
0x5	STANDARD	0x03	0	1	1	SCLK0/3	Legacy single bit. `SPI_CTL.FMODE` is enabled for full cycle access.
0x6	FAST READ	0x0B	1	2	1	SCLK0/1	Single bit with dummy byte. `SPI_CTL.FMODE` is enabled for full cycle access.
0x7	RAPID-S	0x1B	2	2	2	SCLK0/1	Single bit with dummy bytes. `SPI_CTL.FMODE` is enabled for full cycle access.
0x8	DOR	0x3B	1	2	1	SCLK0/2	Dual bit data. `SPI_CTL.FMODE` is enabled for full cycle access.
0x9	DIOR	0xBB	1	2	2	SCLK0/2	Dual data and address. `SPI_CTL.FMODE` is enabled for full cycle access.

Table 36-22: SPI Master Boot BCODE Descriptions (Continued)

BCODE	Mode	Command	Dummy Bytes	Data Lines	Address Lines	SPI Clock	Purpose
0xA	QOR READ (Quad Mode Method 1)	0x6B	1	4	1	SCLK0/2	Quad bit data mode using quad enable method 1 with `SPI_CTL.FMODE` is enabled for full cycle access.
0xB	QIOR READ (Quad Mode Method 1)	0xEB	3	4	4	SCLK0/2	Quad data and address using quad enable method 1 with `SPI_CTL.FMODE` is enabled for full cycle access.
0xC	QOR READ (Quad Mode Method 2)	0x6B	1	4	1	SCLK0/2	Quad data using quad mode enable method 2. `SPI_CTL.FMODE` is enabled for full cycle access
0xD	QIOR READ (Quad Mode Method 2)	0xEB	3	4	4	SCLK0/2	Quad data and address using quad mode enable method 2. `SPI_CTL.FMODE` is enabled for full cycle access
0xE	QIOR READ (Quad Mode Method 3)	0xEB	3	4	4	SCLK0/2	Quad data and address using quad mode enable method 3. `SPI_CTL.FMODE` is enabled for full cycle access
0xF							Unused

NOTE: For all the above configurations the addressing scheme is also set to a fixed 3-bytes for 24-bit addressable flash support only. The SPI mode byte issued for all the SPI Master peripheral based configurations is 0x00. The mode byte is the first byte transmitted after the address cycles and is used to control the continuous read mode functionality in which the next read operation is not required to issue a command cycle. Continuous read mode is not supported during the boot process.

Supported Quad Mode Enable Methods

The boot rom supports the following methods for enabling quad mode on the SPI flash device

NOTE: Please note that the procedures listed here write to a non-volatile register in the SPI flash. There are typically delays implemented with writes to such registers and these need to be accounted for in the boot time. If the flash supports a non-volatile setting using a different procedure from those supported here then it may be more beneficial to boot initially in dual mode and use an initcode to enable quad mode.

Quad Mode Method 1.

1. Issue the Read Status Register command (0x05) and read in the value of the status register.
2. Issue the Read Configuration Register command (0x3F) and read in the configurations register value.

3. Issue the Write Enable command (0x06).
4. Issue the Read Status Register command (0x05) to verify the write latch status is set.
5. Set bit 7 of the read configuration register value to enable quad mode.
6. Issue the Write Register command (0x3E) and write the value of the configuration register.
7. Issue the Read Status Register command (0x05) until the device is ready and write latch is cleared.
8. Issue the Read Configuration Register command (0x3F) to verify quad mode is enabled.

Quad Mode Method 2.

1. Issue the Read Status Register command (0x05) and read in the value of the status register.
2. Issue the Read Configuration Register command (0x35) and read in the value of the configuration register.
3. Issue the Write Enable command (0x06).
4. Issue the Read Status Register command (0x05) to verify the write latch status is set.
5. Set bit 1 of the read configuration register value to enable quad mode.
6. Issue the Write Register command (0x01) and write the value of the status register and the updated value of the configuration register.
7. Issue the Read Status Register command (0x05) until the device is ready.

Quad Mode Method 3.

1. Issue the Read Status Register command (0x05) and read in the value of the status register.
2. Issue the Write Enable command (0x06).
3. Issue the Read Status Register command (0x05) to verify the write latch status is set.
4. Set bit 6 of the read status register value to enable quad mode.
5. Issue the Write Status command (0x01) and write the value of the status register back to enable quad mode.
6. Issue the Read Status Register command (0x05) until the device is ready.

Run-time API

The following table provides descriptions of the `adi_rom_Boot()` `command` parameter.

Table 36-23: SPI Master Boot `command`Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration
31:28	ROM_BCMD_SPIM_SPEED	SPI clock divider to be used. The value written to the SPI Peripherals clock divider register.

Table 36-23: SPI Master Boot `command`Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
26:25	ROM_BCMD_SPIM_IOPROT	SPI I/O protocol. If multiple I/O pins are required, then I/O mode is enabled on the flash device. The number of data bits required for the data transfer determines the I/O mode enabled. This field is only applied if NOAUTO is set to 1. Otherwise, the protocol for enabling the additional pins is defined through the BCODE value supplied.	
		0x0	No protocol required to implement multiple I/O
		0x1	Enable quad functionality using protocol 1
		0x2	Enable quad functionality using protocol 2
		0x3	Enable quad functionality using protocol 3
24:22	ROM_BCMD_SPIM_DUMMY	Number of dummy bytes to be issued. Specifies the number of dummy bytes to be issued after the address bytes are issued for the required read command.	
		0x0	Do not issue dummy bytes
		0x1	Issue 1 dummy byte
		0x2	Issue 2 dummy bytes
		0x3	Issue 3 dummy bytes
		0x4	Issue 4 dummy bytes
		0x5	Issue 5 dummy bytes
		0x6	Issue 6 dummy bytes
		0x7	Issue 7 dummy bytes
21:20	ROM_BCMD_SPIM_ADDR	Number of address bytes to be issued. Specifies the number of address bytes that are required to be issued to the SPI flash for the required read command.	
		0x0	Issue 1 address byte
		0x1	Issue 2 address bytes
		0x2	Issue 3 address bytes
		0x3	Issue 4 address bytes
19:16	ROM_BCMD_SPIM_BCODE	Boot mode-specific code. Specifies the boot mode-specific code that can further customize and control the boot process.	
14:12	ROM_BCMD_SPIM_SSEL	SPI Slave select signal. Specifies the SPI slave select signal to be used to enable the SPI Flash. Not all slave selects are available for each SPI port. Refer to the product data sheet for further details.	
		0x0	SEL1

Table 36-23: SPI Master Boot `command`Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0x1	SEL2
		0x2	SEL3
		0x3	SEL4
		0x4	SEL5
		0x5	SEL6
		0x6	SEL7
		0x7	Reserved
11:8	ROM_BCMD_ DEVENUM	Device enumeration. Specifies the SPI device to use.	
		0x0	SPI0
		0x1	SPI1
		0x2	SPI2
		0x3 - 0xF	Reserved
6	ROM_BCMD_ NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_ NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_ HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_ DEVICE	Boot source device. Specifies the device to boot from.	
		0x2	SPI
		0x7	Memory Mapped SPI (For use with SPI2 only)

NOTE: All bits in the above table that are not defined must be set to zero. Features supported may be limited depending on peripheral instance.

SPI Slave Boot Mode

When using SPI slave mode boot, the processor consumes boot data from an external SPI host device. This mode supports single, dual, and quad-bit modes. The boot kernel always starts in single bit mode and can be changed

using the appropriate command. The following figures show the hardware configuration for the modes. As in all slave boot modes, the host device controls the SYS_HWRST input of the processor.

NOTE: Secure Boot Stream Padding For slave boot modes, the host must always send data in multiples of 1024 bytes. This requirement is due to the sizing of internal buffers used for DMA.

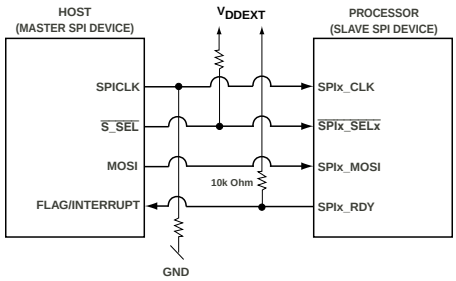

Figure 36-3: Connection Between Host (SPI Master) and Processor (SPI Slave)

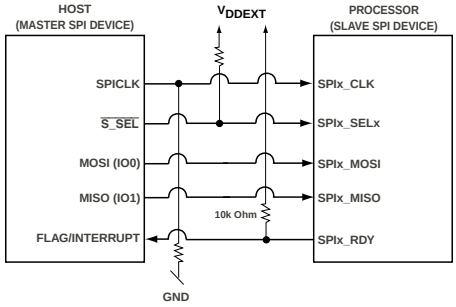

Figure 36-4: Connection Between Host (SPI Master) and Processor (SPI Slave) DIOM

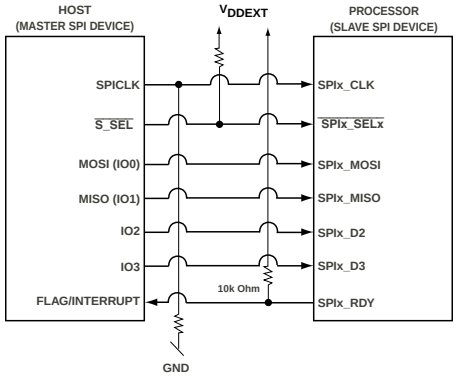

Figure 36-5: Connection Between Host (SPI Master) and Processor (SPI Slave) QSPI

The host drives the SPI clock and is responsible for timing. The host must provide an active-low chip select signal that connects to the SPIx_SS input of the processor. It can toggle with each byte transferred or remain low during the entire procedure. 8-bit data is expected and 16-bit mode is not supported.

In SPI slave boot mode, the boot kernel sets the SPI_CTL.CPHA bit and clears the SPI_CTL.CPOL bit in the SPI_CTL register. Therefore the SPI_MISO pin is latched on the falling edge of the SPI_MOSI pin.

The SPI slave processor detects the correct boot mode from the host SPI device by reading the first byte sent, defined as SPICMD. The ***SPICMD Descriptions*** table describes the available codes. These additional bytes must be sent prior to transmitting the data to configure the SPI device accordingly.

The table describes two cases, one in which the host starts in single bit mode, and one in which the host starts in a mode other than single bit.

Table 36-24: SPICMD Descriptions

SPICMD	Description
If host Starts in Single bit Mode	
0x3	Keep single-bit mode
0x7	Switch to dual-bit mode
0xB	Switch to quad-bit mode
If host device starts in DIOM or QSPI	
0xAA,0xBF	Switch to dual-bit mode
0xEE,0xEE,0xFE,0xFF	Switch to quad-bit mode

In SPI slave boot mode, SPIx_RDY functionality is critical. The SPIx_RDY output is used for back pressure and requires a pulling resistor. The boot code requires the SPIx_RDY signal function as active-low. The host is only permitted to transfer data when SPIx_RDY is in the active state. This functionality allows the processor to hold off the host while the processor is in reset or executing the pre-boot and processor initialization sequences. The SPI is configured to de-assert SPIx_RDY when the receive FIFO is filled to 75% or more. The ***SPI Program Flow on the Host Side*** figure illustrates the required program flow on the host side.

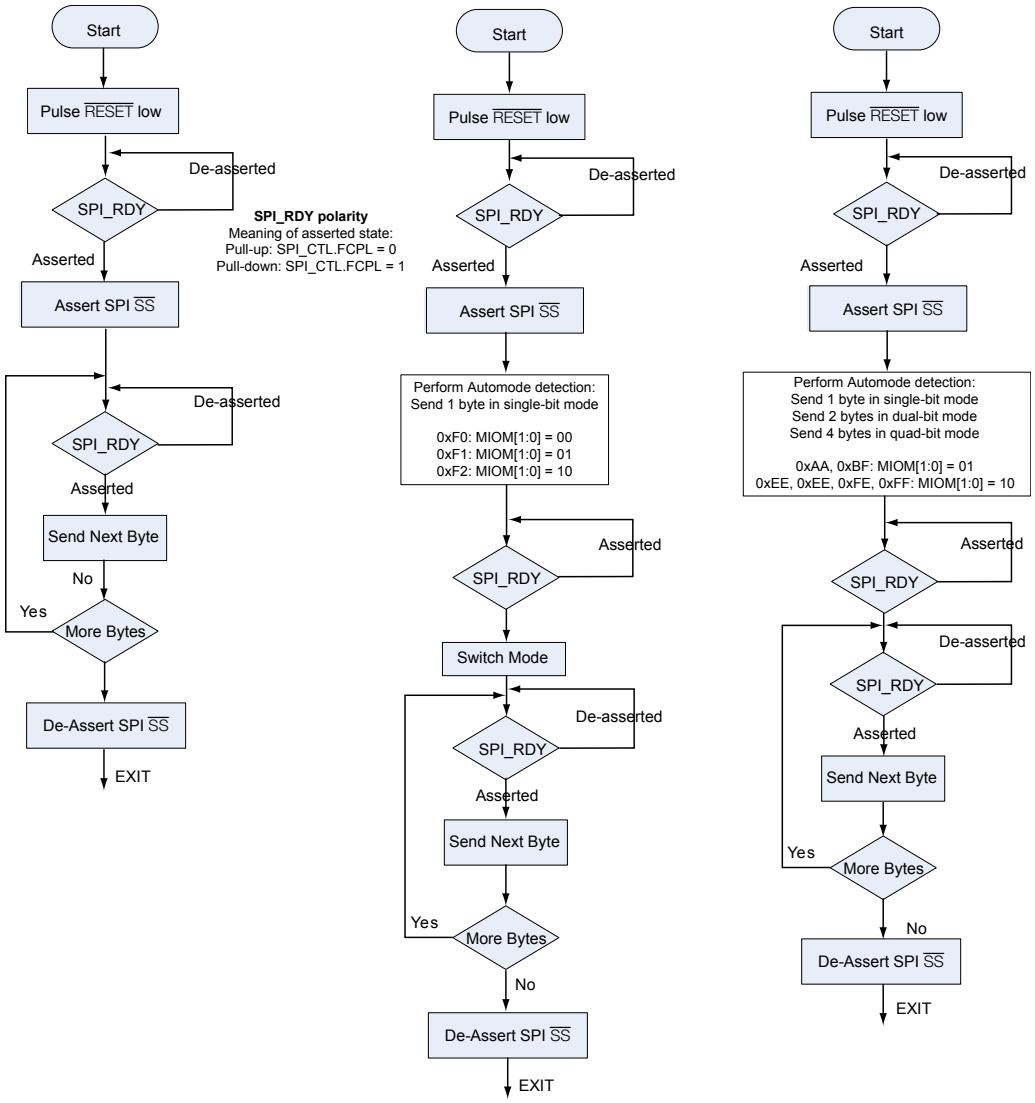

Figure 36-6: SPI Program Flow on the Host Side

Run-time API

The SPI Slave Boot mode can be called through the Boot Routine API function at run-time. Initiating a boot through the run-time API allows for extra customization such as disabling automatic device configuration or specifying a different SPI device other than the default.

When `ROM_BCMD_NOCFG` flag is specified, it is necessary to program pin multiplexing and other SPI configuration as required, while keeping the `SPI_CTL.EN` bit cleared.

The `ROM_BCMD_NOAUTO` flag can suppress auto mode detection. In that case, the desired configuration must be passed through the `ROM_BCMD_SPIS_BCODE` bit field, even if the `ROM_BCMD_NOCFG` flag is set.

The following table provides descriptions of the `adi_rom_Boot()` `command` parameter.

Table 36-25: SPI Slave Boot `command` Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration	
19:16	ROM_BCMD_SPIS_BCODE	Boot Mode Specific BCODE. Specifies the boot mode-specific code that can further customize and control the boot process.	
		00xxb	Single bit SPI bus
		01xxb	Dual SPI bus
		10xxb	Quad SPI bus
		11xxb	Reserved
11:8	ROM_BCMD_DEVENUM	Device enumeration. Specifies the SPI device to use.	
		0x0	SPI0
		0x1	SPI1
		0x2	SPI2
		0x3 - 0xF	Reserved
6	ROM_BCMD_NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_DEVICE	Boot source device. Specifies the device to boot from.	
		0x2	SPI

NOTE: All bits in the above table that are not defined must be set to zero. Features supported may be limited depending on peripheral instance.

UART Slave Boot Mode

When using UART slave mode boot, the processor receives boot data from a UART host device connected to the UART interface. The device connected to UART0 is initially detected using an autobaud detection sequence. After finishing the UART slave boot process, all control and status registers of the used resources are restored.

Further customization, such as disabling autobaud detection, and changing the device, use the boot routine API.

During the boot operation, the host device usually relies on the RTS output of the UART device. At boot time, the processor does not evaluate RTS signals driven by host. Since the RTS is in a high impedance state when the processor is in reset, or while executing a pre-boot, an external pull-up resistor to VDDEXT is recommended. The ***Connection Between Host and Processor*** figure shows the interconnection required for booting. The figure does not show physical line drivers and level shifters that are typically required to meet the individual UART-compatible standards.

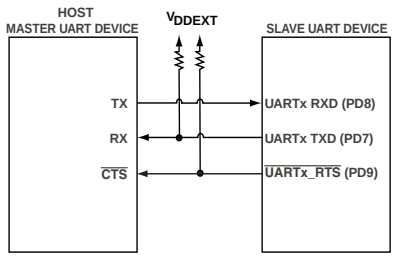

Figure 36-7: Connection Between Host and Processor

When the UART is enabled, the RTS goes immediately low, encouraging the host to send the first boot stream data as shown in the ***Host Relying on RTS*** figure. For half-duplex UART connections, the host must avoid this action. The host must wait until it has received the 4 bytes from the slave processor before sending any data.

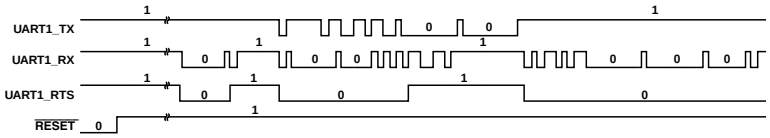

Figure 36-8: Host Relying on RTS

When the boot kernel is processing fill or Initcode blocks, it can require extra processing time and must delay the host from sending more data. This request is signaled using the RTS output.

The ***Host Relying on RTS*** figure shows RTS timing when an extended Initcode routine executes. Since code execution is distracting from the data loading, the host device must be prevented from sending more data. The timing of the RTS depends on the state of the RFRT bit in the UART control register (UART_CTL). This bit is cleared during UART slave boot mode when RTS is de-asserted, the UART receive FIFO contains 4 or more data words, and another start bit is detected.

NOTE: Secure Boot Stream Padding For slave boot modes, the host must always send data in multiples of 1024 bytes. This requirement is due to the sizing of internal buffers used for DMA.

Autobaud Detection

The kernel supports autobaud detection using the '@' character as data. The host is expected to have its clock set to a rate supported in the UART.

To determine the bit rate when performing autobaud detection, use the following steps:

1. The boot kernel expects an '@' character (0x40, eight bits data, one start bit, one stop bit, no parity bit) on the UART RXD input.
2. The EDBO and UART_CLK register is cleared.
3. The boot kernel acknowledges, and the host then downloads the boot stream. The acknowledgment consists of 4 bytes: 0xBF, UART_CLK [15:8], UART_CLK [7:0], 0x00.
4. The host is requested to not send further bytes until it has received the complete acknowledge string.
5. Once the 0x00 byte is received, the host can send the entire boot stream.

The host knows the total byte count of the boot stream, but it is not required to know the content of the boot stream.

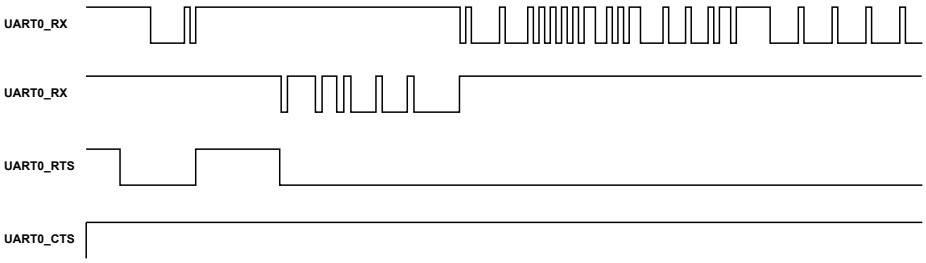

Figure 36-9: UART Autobaud Detection Waveform

The ***UART Autobaud Detection Waveform*** figure provides timing information for UART booting. After the bit rate is known, the UART is enabled and the kernel transmits the 4 acknowledge bytes.

Run-time API

The UART slave boot mode can be called through the boot routine API function at run time. The run-time API allows for more customization. Both autobaud detection and device configuration can be disabled, and a device other than the default UART0 can be specified.

If ROM_BCMD_NOCFG flag is specified, it is the programs responsibility to configure pin multiplexing as required.

Autobaud detection can be suppressed using the ROM_BCMD_NOAUTO flag. In this case, the desired configuration can be passed through the ROM_BCMD_UART_CLK bit field. If the ROM_BCMD_UART_CLK bit field is zero, UART_CLK is evaluated. If a value of 0xFFFF was present, the default error routine of the boot kernel is called and the booting process is aborted. Otherwise, the value in UART_CLK remains untouched.

The following table provides descriptions of the adi_rom_Boot() command parameter.

Table 36-26: UART Slave Boot command Bit Descriptions

Bit No. (Access)	Bit Name	Description/Enumeration
31:16	ROM_BCMD_ UART_CLK	UART Clock Divider. When set to zero this field is ignored.

Table 36-26: UART Slave Boot `command` Bit Descriptions (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
15	ROM_BCMD_ UART_EDBO	UART Clock Divider Mode When set enables EDBO functionality.	
11:8	ROM_BCMD_ DEVENUM	Device enumeration. Specifies the UART device to use.	
		0x0	UART0
		0x1	UART1
		0x2	UART2
		0x3 - 0xF	Reserved
6	ROM_BCMD_ NOAUTO	Automatic device detection disable. When set disables automatic device detection and uses the setting provided in the other fields of this register to configure the boot mode.	
5	ROM_BCMD_ NOCFG	Device configuration disable. When set, this bit disables device configuration. Device configuration includes reconfiguration of the peripherals MMR registers and device pin muxing.	
4	ROM_BCMD_ HOST	Host boot mode enable. When set, this bit indicates that SPI slave boot is to be performed. Otherwise, use the master boot mode.	
3:0	ROM_BCMD_ DEVICE	Boot source device. Specifies the device to boot from.	
		0x3	UART

NOTE: All bits in the above table that are not defined must be set to zero. Features supported may be limited depending on peripheral instance.

Boot Loader Stream

A loader stream is a set of formatted blocks containing instructions for the boot kernel, as well as the application and data for loading to the chip. This section details the different aspects of the stream, its blocks, some common use cases, and the ROM functionality.

Each block begins with a block header which contains attributes of the block as well as flags to control its processing by the boot ROM. On power-up or reset, the processor begins executing the on-chip boot ROM. The boot stream is either read from memory or received from a peripheral, depending on the boot mode specified. Each block in the boot stream instructs the boot kernel to perform an action, most commonly to load data to a specified location. For a complete list of actions, refer to Block Types . Some common actions include:

- running code that initializes a peripheral

- processing data then loading it to a location

The elfloader utility parses the application data, and creates a set of blocks representing the segmented application. When processing an actual image that must be loaded to a single contiguous memory block, this representation can contain as little as a single header block. The output of the standalone utility is stored in a loader file (.ldr). The loader file contains the boot stream and becomes available to hardware by:

- programming or burning it into non-volatile external memory, or
- sending it through a peripheral such as the UART during boot time

A loader stream must always begin with a first block and end with a final block. The loader file contains the boot stream and becomes available to hardware by:

- programming or burning it into non-volatile external memory, or
- sending it through a peripheral during boot time

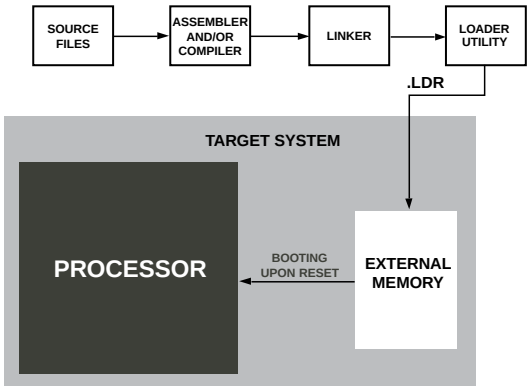

Figure 36-10: Project Flow

The ***Booting Process*** figure shows the parallel or serial boot stream contained in a flash memory device. In host boot scenarios, the non-volatile memory usually connects to the host processor rather than directly to the processor. After reset, the on-chip boot kernel reads and parses the headers and the loader stream is processed block-by-block. Finally, payload data is copied to destination addresses, either in on-chip L1 and L2 memory, or off-chip to SDRAM or SRAM.

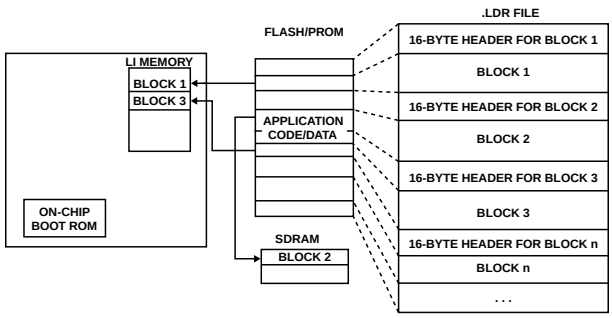

Figure 36-11: Booting Process

In some cases (for example, secure boot or when the BFLAG_INDIRECT flag for any block is set), the boot kernel uses another memory block for intermediate data storage. In order to preserve the security of the device processors will not allow these storage regions to be initialized with boot data. The boot stream is loaded to the intermediate storage then processed by the kernel and loaded to the final destination. Thge final destination cannot be the intermediate storage location otherwise the boot process will terminate in error.

Block Header

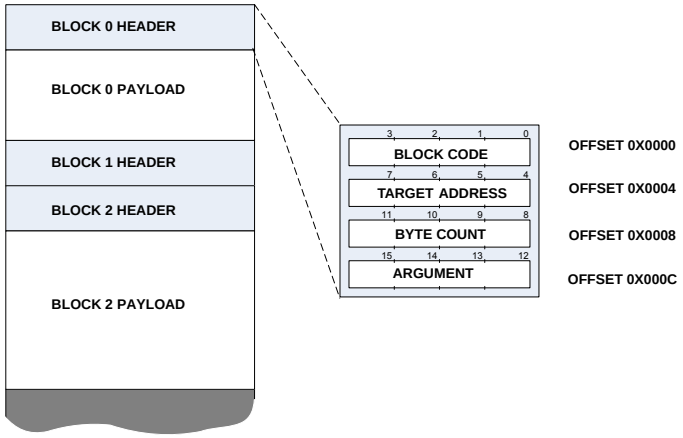

Figure 36-12: Loader Stream Block Structure

A boot stream consists of multiple boot blocks as shown in the figure. A 16-byte block header begins every block. The 16 bytes are functionally grouped into four 32-bit words:

- the block code
- the target address
- the byte count
- the argument field

This section describes the fields in general. The uses can vary depending on the particular block type and boot mode. Refer to the block type descriptions and boot modes for further information.

Block Code

Table 36-27: Block Code flags

Bit	Name	Description
0–3	BCODE	Specific to boot modes (see Boot Modes)
4	BFLAG_SAVE	Intended to allow for a user application to mark blocks for saving the memory of this block to off-chip memory in case of power failure or a hibernate request. The on-chip boot kernel does not use this flag.
5	BFLAG_AUX	The on-chip boot kernel does not use this flag. It is intended to allow for special block types to be nested within another block as required by special-purpose second-stage loaders.

Table 36-27: Block Code flags (Continued)

Bit	Name	Description
6	reserved	
8	BFLAG_FILL	Fill the target location with a specified 32-bit value.
9	BFLAG_QUICKBOOT	Does not process block for a quick boot (warm boot). Refer to the Wakeup Functionality section.
10	BFLAG_CALLBACK	Calls the previously registered callback function.
11	BFLAG_INIT	Calls function at target address. If the block contains a payload, the payload is loaded prior to the call.
12	BFLAG_IGNORE	Block payload is ignored.
13	BFLAG_INDIRECT	Boots the payload to the intermediate storage location.
14	BFLAG_FIRST	Indicates the block to be the beginning of a new application
15	BFLAG_FINAL	Indicates the last block of a loader stream. Booting will complete after processing the block. This flag does not denote the end of an application in a Multi-Application Boot Streams boot stream.
16–23	HDRCHK	A simple 8-bit XOR checksum of the other 24 bits in the boot block header.
24–31	HDRSIGN	0xAD

TARGET_ADDRESS

The TARGET_ADDRESS holds the applicable address for the block, (where the code or data is loaded). However, the interpretation of the field differs depending on what specific flags are set in the block code. Refer to the documentation for each block type for details.

The following attributes must be true:

- The target address must be divisible by 4, as the boot kernel uses 32-bit DMA for certain operations.
- The target address must point to valid on-chip or off-chip memory locations.
- When booting through peripherals that do not support DMA capability, the BFLAG_INDIRECT flag must be set if the target address points to L1 instruction memory. For performance reasons, this operation is also recommended when booting to off-chip memory.

BYTE_COUNT

The byte count must be divisible by 4, and can also be zero. This 32-bit field generally holds the size of the block. In some cases, it has a different use (such as when BFLAG_FILL is set). See the block types section for information on the variations.

ARGUMENT

The 32-bit field is a user-variable for most block types. The `Initcode` or the `callback` routine can access this value and use it for optional instructions.

The different block types use the `ARGUMENT` field in various ways. See the block type descriptions for further information.

Block Types

A loader stream is a set of linked blocks and each block is responsible for performing a certain function dependent on the block type. The flags in the block header define a block type. Operations include functions such as loading data, filling a memory region with data, and instructing the kernel to stop processing. This section describes each block type and its usage within a boot loader stream.

Normal Block

The primary function of a block is to load data into a specified location of memory. A normal block instructs the boot kernel to load the data contained in its payload to the location specified in the `TARGET_ADDRESS` field. The `BYTE_COUNT` defines the size of the payload. Once the correct amount of data has been loaded, the kernel moves on to process the next block in the stream.

Table 36-28: Flags

Flag	Required Value	Init
`TARGET_ADDRESS`	Y	Address where payload is loaded (must be valid)
`BYTE_COUNT`	Y	Size of block in bytes

First Block

A first block indicates the start of a boot stream and is always needed at the beginning of the boot stream. When a loader stream contains Multi-Application Boot Streams , a first block occurring within the loader stream indicates the beginning of a new application.

When the kernel processes the first block in a loader stream, the boot kernel uses the `TARGET_ADDRESS` to determine the application entry location. For more details, refer to Boot Termination and Application Execution .

NOTE: A first block cannot be combined with a fill block.

Table 36-29: Flags

Flag	Required Value	Init
`BFLAG_FIRST`	Y	1

Table 36-29: Flags (Continued)

Flag	Required Value	Init
ARGUMENT	Y	Offset to the next application, or first address following loader stream. Commonly referred to as the NEXTDXE field.
TARGET_ADDRESS	Y	When the block is the first block in a loader stream, also defines the start address for the application. If the block is not the first in a loader stream, use the target address as in normal operation.

Final Block

The final block marks the last block in a boot stream. After processing a final block, the boot kernel jumps to the start address of the application. For more information on the definition of the start address, refer to Boot Termination and Application Execution .

There is further customization to the kernel behavior available. For example, the kernel can be instructed to return from the boot routine rather than jump to the application using initialization codes or the `adi_rom_Boot()` API.

Before the boot kernel passes program control to the application, it does some housekeeping. Most of the registers in use are set to their default state. However, some register values can differ depending on the boot mode. See Boot Modes for more information.

Table 36-30: Flags

Flag	Required Value	Init
BFLAG_FINAL	Y	1

Indirect Block

An indirect block is first loaded to a storage location before being copied to the destination. The following situations motivate this functionality:

- Some boot modes do not use DMA from the boot peripheral. The core is not always able to access some memory locations directly or efficiently. An intermediate load to a different location improves overall efficiency.
- In some booting scenarios, the data in the payload must be operated-on or analyzed before it is fully loaded (such as decryption or checksum calculation). By using an intermediate location, such scenarios are simplified and can be more efficient when loading to off-chip memories (see Callback Block).

In some cases, a boot block does not fit into temporary storage memory. Having a larger buffer can improve boot performance. If an entire block cannot fit into the buffer, it is processed in pieces. Initialization code or callback functions can alter the temporary buffer region, including its location and size, by modifying `ADI_ROM_BOOT_CONFIG::pTempBuffer` and `ADI_ROM_BOOT_CONFIG::dTempByteCount` variables in the `ADI_ROM_BOOT_CONFIG` structure.

Table 36-31: Flags

Flag	Required Value	Init
BFLAG_INDIRECT	Y	1
BFLAG_CALLBACK	N	Defines a callback function to operate on intermediate data. These 2 flags are often used together.

Ignore Block

An ignore block is a block that is (in most cases) ignored by the loader stream. Ignore blocks are useful when it is not possible to pass information in another block header. For example, if the first block contains data such as a firmware revision rather than application code, then BFLAG_IGNORE can be set with the correct application start address. This step ensures that the loader stream uses the correct start address. Since this block has no other function, identify it as an ignore block. Then, the kernel does not attempt to process any payload.

Ignore blocks result in the clearing of the following flags, disabling the corresponding blocks from being processed if set along with BFLAG_IGNORE:

- BFLAG_INIT
- BFLAG_CALLBACK
- BFLAG_FINAL
- BFLAG_AUX

NOTE: When BFLAG_IGNORE is set along with BFLAG_FIRST, only the payload associated with the first block is ignored. The application entry point retrieved from the first block is always processed.

Table 36-32: Flags

Flag	Required Value	Init
BFLAG_IGNORE	Y	1
BYTE_COUNT	Y	Size of block to ignore, can be zero

Fill Block

A fill block instructs the boot kernel to perform a 32-bit memory fill of the memory region. Fill blocks help minimize the size of a boot stream when an application contains large arrays of data that need to be initialized upon startup. Zero fill is the most common fill block type however any 32-bit fill value is supported.

Table 36-33: Flags

Flag	Required Value	Init
BFLAG_FILL	Y	1
TARGET_ADDRESS	Y	Address where payload is loaded (must be valid)
BYTE_COUNT	Y	Size of block in bytes (must be multiple of 4)
ARGUMENT	Y	32-bit Fill Value

Init Block

An initialization block instructs the boot kernel to do a function call to the target address after the entire block has loaded. The function called is referred to as the *initialization code (Initcode) routine*. Please refer to the API reference `initcode()` for full details.

If the Initcode routine has been previously loaded, the block can declare a zero-size and have no payload.

Initcode routines can be used to speed up and customize booting mechanisms exposed by the boot kernel. Traditionally, an Initcode routine is used to setup system PLL, bit rates, wait states, and the external memory controllers. If executed early in the boot process, the boot time can be significantly reduced.

Initcode routines are required to follow the C language calling conventions. The expected C prototype is:

```
void initcode(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

NOTE: When programming in assembly, use a return from subroutine instruction for returns.

The structure provided to the Initcode routine by the boot kernel contains various information about the block being processed. It includes header information, locations of temporary block data (for indirect blocks), target address, and byte count. See `ADI_ROM_BOOT_CONFIG` for a full list and details on the provided data.

In the simplest case, an Initcode routine consists of only a single block in which the `BFLAG_INIT` flag is set. For larger routines, a sequence of blocks can incrementally load the routine, and only the last block sets the `BFLAG_INIT` flag. In the latter case, the last block has no payload attached, and simply instructs the boot kernel to issue a call to subroutine instruction.

An Initcode routine can be overwritten by a successive block when it is no longer needed. Otherwise, the routine can be called at multiple points during the boot process, and even remain in memory after the application completes booting.

NOTE: The following list provides requirements for Initcode that is written in C or C++.

- Ensure that the Initcode routine does not contain calls to the run-time libraries
- Do not assume that parts of the run-time environment, such as the heap, are fully functional
- Ensure that all run-time components are loaded and initialized before the routine executes

The loader utility and tool set provide mechanisms to aid in implementing initialization codes and organizing them properly within the boot loader stream. A special project type is provided to allow the creation of Initcode routines as separate projects. Options are available to assign particular pieces of the application to be Initcode routines. For details and more information on the utility, see to the *Loader and Utilities manual*.

Table 36-34: Flags

Flag	Required Value	Init
BFLAG_INIT	Y	1
TARGET_ADDRESS	Y	Location to load payload data. Call to subroutine issued to the same location.
ARGUMENT	N	Can be used to supply block specific arguments
BYTE_COUNT	Y	Size of payload, can be zero

NOTE: Init blocks result in execution of software not located in the boot rom during the boot process. In the case of a secure boot scenrio initcode routines are not supported as the secure boot authentication process is performed at the end of the boot process and execution of any user software prior to the authentication process would violate the secure boot requirements.

Callback Block

A callback block instructs the boot kernel to call a pre-registered function upon completion of loading the payload of the block. The purpose of a callback routine is to apply standard processing to the block payload. The callback routine is registered through an Initcode routine prior to loading a block using the routine. Typically, callback routines contain checksum, decryption, decompression or hash algorithms. Please refer to `callback()` API reference.

To register a callback, create an Init Block whose Initcode modifies `ADI_ROM_BOOT_CONFIG::pCallBackFunction` with the address of the callback function to execute. A callback function must be registered prior to processing a callback block.

Since callback routines require access to the payload data of the boot blocks, load the block data before processing. Often an Indirect Block block is used in combination with a callback block. Indirect blocks in combination with callback blocks can allow for post processing of the loaded data before it is then transferred to the final destination.

Callback routines are expected to meet the C language calling conventions. The prototype is as follows:

```
ROM_BOOT_RESULT callback(
                          ADI_ROM_BOOT_CONFIG * pBootConfig,
                          ADI_ROM_BOOT_BUFFER * pBuffer,
                          uint32_t nFlags
                         )
```

The `pBootConfig` argument contains a pointer to the `ADI_ROM_BOOT_CONFIG` information, and `pBuffer` provides access to the address and size of the block (can vary when using indirect). The `nFlags` parameter is specifically used when `BFLAG_INDIRECT` is also used. `CBFLAG_DIRECT` flag indicates that the `BFLAG_INDIRECT` bit is not

active and so that the program only calls the callback routine once per block. When the CBFLAG_DIRECT is set, CBFLAG_FIRST and CBFLAG_FINAL are also set.

NOTE: Callback blocks result in execution of software not located in the boot rom during the boot process. In the case of a secure boot scenario callback routines are not supported as the secure boot authentication process is performed at the end of the boot process and execution of any user software prior to the authentication process would violate the secure boot requirements.

Callback Block Used in Conjunction with Indirect Block

When a block using a callback routine is also loaded indirectly, there are slight behavior differences. The procedure for loading is:

1. Load data into the temporary buffer defined by ADI_ROM_BOOT_CONFIG::pTempBuffer.
2. Issue a call to ADI_ROM_BOOT_CONFIG::pCallBackFunction.
3. After the callback routine returns, if the return value is zero, the memory DMA copies data to the destination.

When a block does not fit entirely into the temporary buffer, loading is performed similar to indirect blocks. The software calls the callback function after each chunk is loaded into the temporary storage. The nFlags parameter gives information on the specific iteration.

When a block does not fit entirely into the temporary storage area, the nFlags tells the callback routine whether it is invoked for the first time (CBFLAG_FIRST) or called the last time (CBFLAG_FINAL) for a specific block.

When the software invokes DMA to copy the data, it relies on the supplied pBuffer data, not the ADI_ROM_BOOT_CONFIG::pTempBuffer and ADI_ROM_BOOT_CONFIG::dTempByteCount members of the boot structure. The callback routine can control the source of the memory DMA by altering the content of the pBuffer structure. This alteration can be necessary when the callback routine performs data manipulation such as decompression.

When the software uses an indirect block, the return value of the callback routine determines whether the DMA transfer occurs. If the value is non-zero, then the transfer does not occur.

Table 36-35: Flags

Flag	Required Value	Init
BFLAG_CALLBACK	Y	1

NOTE: Callback blocks result in execution of software not located in the boot rom during the boot process. In the case of a secure boot scenrio callback routines are not supported as the secure boot authentication process is performed at the end of the boot process and execution of any user software prior to the authentication process would violate the secure boot requirements.

Quick Boot Block

There are some booting scenarios in which not all memories are required to be reinitialized during the boot process and as a result the boot kernel supports conditional processing of boot blocks in specific circumstances. For example, if a processor supports wakeup from hibernate, off-chip SRAM or DRAM may not need to be loaded if it is powered while the processor is in a hibernate state. Dynamic RAM is also not always impacted if it was put into a self-refresh mode before the processor powered down.

The processing of a quick boot block is determined by the state of the BFLAG_WAKEUP, BFLAG_QUICKBOOT and BFLAG_IGNORE flags.

Table 36-36: Quick Boot Block Processing

BFLAG_WAKEUP	BFLAG_QUICKBOOT	BFLAG_IGNORE	Block Processed
0	0	0	Yes
0	0	1	No
0	1	0	Yes
0	1	1	No
1	0	0	Yes
1	0	1	No
1	1	0	No
1	1	1	Yes

The BFLAG_WAKEUP flag is applied to the entire boot process when calling the boot kernel. Refer to adi_rom_Boot() for further details on the global flags available that can be applied to the boot process.

When the boot process is triggered via a hard reset or a software triggered system reset event the boot kernel is always called with BFLAG_WAKEUP cleared. Processors not supporting the wakeup from hibernate feature would generally not have this quickboot feature available for typical boot events. However, when using adi_rom_Boot() or adi_rom_BootKernel() from the user application to boot the processor, the quickboot feature and selective processing of boot blocks is fully available regardless if the processor supports wakeup from hibernate.

NOTE: When BFLAG_WAKEUP is set and a block also has BFLAG_QUICKBOOT set, BFLAG_IGNORE flag is toggled within the boot kernel. This event occurs before the processing of an Ignore Block and as such also has an influence on the processing of blocks that are disabled as a result of BFLAG_IGNORE being set.

NOTE: If an initcode contains an implementation that requires some operations to be performed for wakeup type events, do not set BFLAG_QUICKBOOT in the initblock, disabling it completely. Instead, ensure BFLAG_QUICKBOOT is clear and perform any conditional processing based on the state of the BFLAG_WAKEUP flag.

Table 36-37: Flags

Flag	Required Value	Init
BFLAG_QUICKBOOT	Y	1

Save Block

A save block is intended to mark blocks in a boot stream that are to be saved to off-chip memory. The on-chip boot kernel does not use this flag. A user application can process the boot steam to find address of regions of memory that are to be saved off to external memory. Upon a reboot the user application may then restore the previously saved contents. It provides a means of doing a context restore after a reboot.

Table 36-38: Flags

Flag	Required Value	Init
BFLAG_SAVE	Y	1

Single-Block Boot Streams

The simplest boot stream consists of a single block header and one contiguous block of instructions and possibly data. When the appropriate flags are set in the block header, the kernel loads the block to the target address, terminates the boot process, and begins executing from the entry address of the application.

The ***Initial Header for Single-Block Stream*** table shows an example of a single-block boot stream header with settings that can be loaded using any boot mode. The BFLAG_FIRST and BFLAG_FINAL flags are both set at the same time. The desired location and size of the application determines the target address and byte count.

Table 36-39: Initial Header for Single-Block Stream

Field	Description of Value
BLOCK_CODE	0xAD000000|XORSUM|BFLAG_FINAL|BFLAG_FIRST
TARGET_ADDRESS	Start address of block
BYTE_COUNT	Number of bytes in the block
ARGUMENT	Functions as next-application pointer in multi-application boot streams.

Direct Code Execution

Applications can avoid long booting times and execute code from flash or SDRAM memory with minimal processing by the boot kernel. This feature is called direct code execution.

An initial boot block header is required for the processor to fetch and execute program code from the boot device as early as possible. The block uses safety mechanisms of the block to avoid unpredictable processor behavior when

boot memory is not yet programmed with valid data. Safety mechanisms include the header signature and the byte-wise XOR checksum. Rather than blindly executing code, the boot kernel first executes the pre-boot routine for system customization. It then loads the first block header and checks it for consistency. If the block header is corrupt, the boot kernel calls the error handler and does not start code execution.

If the initial block header check is good, the boot kernel interrogates the block flags. If the block has the `BFLAG_FINAL` flag set, the boot kernel terminates and executes the sequence as described in the Boot Termination and Application Execution section. To cause the boot kernel to customize the starting address in advance, the first block must also have the `BFLAG_FIRST` flag set. The target address field is then saved as the application start address.

Table 36-40: Example Direct Code Execution Header

Field	Value	Comments
BLOCK_CODE	0xAD7BD006	0xAD000000|XORSUM|`BFLAG_FINAL`|`BFLAG_FIRST`|`BFLAG_IGNORE`|BCODE
TARGET_ADDRESS	0x20000020	Start address of application code. Provided as an example this would be dependent upon the start address required for a given product.
BYTE_COUNT	0x00000010	Ignores 16 bytes to provide space for control data such as version code and build data. This field is optional and can be zero. The payload is ignored due to the `BFLAG_IGNORE` flag being set.
ARGUMENT	0x00000010	Functions as next-application pointer in multi-application boot streams.

Multi-Application Boot Streams

This section describes loader streams that contain multiple applications.

A boot stream is typically generated from an application file. It is therefore common to refer to loader stream with more than one application multi-application booting. A loader utility often accepts multiple application files as input parameters and generates a contiguous boot image. The subsequent applications are appended to the first.

The loader utility must also update the argument field of all `BFLAG_FIRST` blocks. The argument field of a `BFLAG_FINAL` block is called the *next-application* pointer.

The next-application pointer of the first application boot stream points relatively to the start address of the second application boot stream. A multi-application boot image can be seen as a linked list of boot streams. The next-application pointer of the last application boot stream points relatively to the next free address. The ***Multi-application Boot Stream Example*** figure illustrates an example.

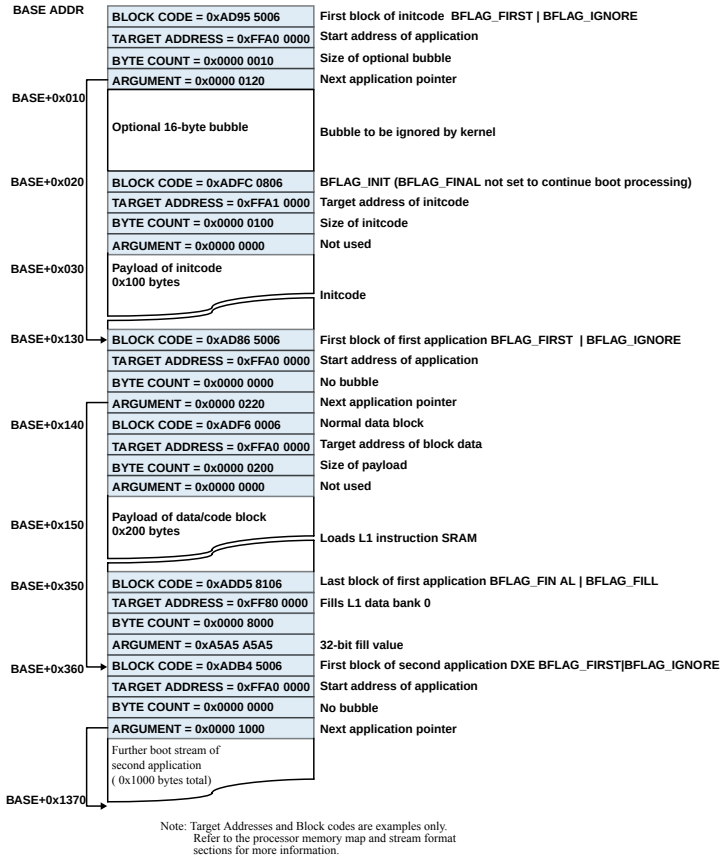

Figure 36-13: Multi-application Boot Stream Example

The ***Multi-application Direct Code Execution Example*** figure shows a linked list of initial block headers. The blocks instruct the boot kernel to terminate immediately and to start code execution at the address provided by the target address field of the individual blocks. There is nothing in the boot code that prevents multi-application streams from mixing regular boot streams and direct code execution blocks.

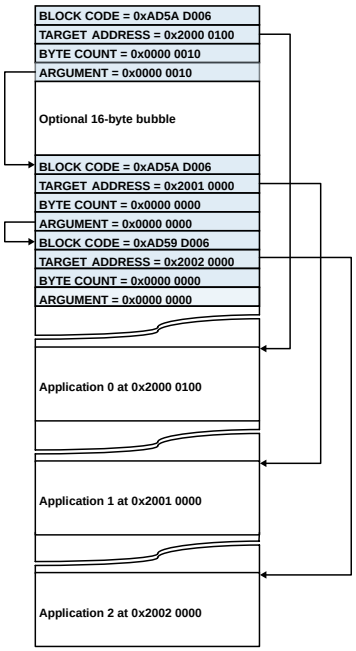

Figure 36-14: Multi-application Direct Code Execution Example

CRC32 Protection

This section describes the CRC32 protection provisions.

The boot kernel provides mechanisms to allow verification of each blocks payload using a 32-bit CRC. The boot rom contains a function in the rom that can be called as an initcode to register the CRC callback and initialize the CRC peripheral with a user specified polynomial. The utilities provided by the supporting toolchain for the processor allow for generation of CRC32 protected loader streams.

To enable this feature an Init Block must be located in the boot stream with a `TARGET_ADDRESS` that points to the `adi_rom_Crc32Init()` function in the ROM. The `ARGUMENT` field contains the CRC32 checksum polynomial to be used to initialize the CRC lookup table. Once this CRC initcode function in the ROM has been executed CRC verification is enabled for all subsequent blocks except:

- Ignore
- First

NOTE: Due to the fact that the enabling of the CRC functionality is dependent upon the use of an Initblock. The feature is not supported in secure boot situations.

Secure Boot

The secure boot process provides a means of integrating security in the processor boot sequence. A chain of trust is established within the system by ensuring the integrity and authenticity of the boot image. Confidentially support is also supported.

Secure boot increases protection against malicious, unsecured accesses to critical and confidential resources of the processor. The boot stream application code and data must be digitally signed in order to build up a chain of trust in the system. This allows the processor to distinguish between authentic and trusted code from non-authentic and untrusted code.

Secure boot also provides confidentiality support. The digitally signed boot image can be optionally encrypted as well. When loading an encrypted image, the ROM decrypts while loading, then authenticates, before any application code is executed.

Secure boot is an optional feature of the processor and is disabled by default. The feature is enabled using the OTP lock API, and secure boot cannot be disabled after it has been enabled. When security is enabled, developers are not dependent upon on Analog Devices to provision the devices, sign code or provide security certificates. The required tools for signing and encrypting the boot images are provided with the development tools for the processor.

Integrity and Authenticity Protection

Integrity protection is based on the secure hash SHA-2 224 bit algorithm. Authenticity protection is based on the ECDSA algorithm.

ECDSA uses public key cryptography consisting of two keys, a private key and a public key. The public key is stored in OTP memory on the processor so that the secure boot process can verify the authenticity of the signed boot image. Only parties in possession of the private key are able to sign the images.

Confidentiality Protection

Confidentiality protection uses the AES algorithm. Two variants are supported, wrapped and unwrapped.

The wrapped variant utilizes a 128-bit Key Encryption Key (KEK) stored on the processor to decrypt the 128-bit AES decryption key embedded in the secure header. The unwrapped variant stores the AES description key on the processor and utilizes it to decrypt the entire image.

The privacy of the key stored on the device (whether AES or KEK) is paramount to the security of the system. Disclosure of this key compromises security of the entire system.

Anti-Cloning Protection

Anti-cloning protection is based on the confidentiality protection. If each processor uses a unique private key for the confidentiality protection, then cloning between these devices can be prevented. The boot image is incompatible with devices using a different private key for the decryption.

Anti-Rollback Protection

The secure boot process supports anti-rollback protection through a 32-bit counter in the OTP memory. A value of `0x00000000` in the OTP results in anti-rollback being disabled by default. If anti-rollback protection is required, then the user may set the Rollback ID when signing the boot image. Upon successful authentication of the boot image, the secure boot software then updates the counter in the OTP. The software updates the counter if the rollback ID in the boot image is greater than the value currently stored in the OTP counter.

The rollback ID stored in the secure boot image header is integrity-protected preventing altering of the rollback ID.

NOTE: In order to initially enable anti-rollback protection for secure boot operations, a non-zero must be written to the 32-bit counter in the OTP memory. As long as this register filed remains at the default value of zero anti-rollback protections will not be enabled regardless of the rollback ID located in a secure boot stream.

CAUTION: As the rollback ID is implemented in the OTP module, there are a number of restrictions for its use. It is therefore recommended to only use the OTP boot program ROM API to set the counter. Refer to the OTP counters section for information detailing the implementation strategy.

Terminology

ECDSA

Elliptical Curve Digital Signature Algorithm

BLp

Boot Loader plain text, Plaintext Format

BLx

Boot Loader without key, Keyless Format

BLw

Boot Loader wrapped, Wrapped Format

BLe

Boot Loader encrypted, Encrypted Format

SBLS

Secure Boot Loader Stream

SBH

Secure Boot Header

SBCR

Secure Boot Confidentiality Root

AES

Advanced Encryption Standard

Secure Boot Image Signing

All boot images must be digitally signed to create secure boot images. The boot image is processed by the security utilities included with the development tools to sign and optionally encrypt the boot image. The security utilities operate with key-pairs consisting of a private and a public key. The private key is used for signing the images, and the public key is used to validate an image being loaded into the processor.

CAUTION: The private key generated from the singing utility, used for signing images, is never required by the processor for successful secure boot. The private key is only ever required by the signing utility and should be made available only within the system responsible for the image signing process.

The image signing utility provides the following functionality:

- Signing and encrypting of images
- Generation of ECDSA key pairs
- Generation of random encryption keys
- Extraction of the public key from an ECDSA key pair
- Setting Secure Boot Image Attributes

For more information on the use of the signing utility, refer to the Loader and Utilities manual.

Secure Boot Image Types

This section provides an overview of the different image types supported, as well as supporting information on how to use them.

Plaintext Format (BLp)

Provides integrity plus authentication protection of the boot image. The boot image is produced using a 224-bit Elliptical Curve Digital Signature Algorithm (ECDSA) private key. To authenticate the image, program the corresponding public key into the OTP `public_key` field using the OTP boot program API.

SBH	Boot Loader Stream

Wrapped Format (BLw)

Provides the highest level of protection: integrity plus authentication, confidentiality, and anti-cloning protection. The image contains an ECDSA wrapped image encryption key (denoted by [K]) within the secure header. The image data is encrypted with the wrapped key, preventing cloning. An extra key is required to unwrap the header, program this key into OTP `pvt_128key` field using the OTP boot program API.

SBH [K]	Encrypted Boot Loader Stream

Keyless Format (BLx)

Similar to the BLw format except that the image does not contain the key at all. This format provides anti-cloning protection only if the secure key is unique per device. Program the decryption key for the data into OTP `pvt_128key` field using the OTP boot program API.

SBH	Encrypted Boot Loader Stream

Secure Boot Image Format

Secure Boot images provide authenticity and integrity protection during the boot process. A secure boot image is comprised of a secure boot header and an optionally encrypted loader stream.

Signed images consist of the following sections to comprise a complete secure boot image:

- Secure Boot Header
- Image Attributes
- Image Section

The Figure 36-15 Secure Boot Image figure shows that the image attributes are encapsulated within the secure boot header. The image attributes are actually integrity protected along with the image section. The image section contains a standard Boot Loader Stream . Some block types are not allowed as described in Unsupported Boot Stream Blocks.

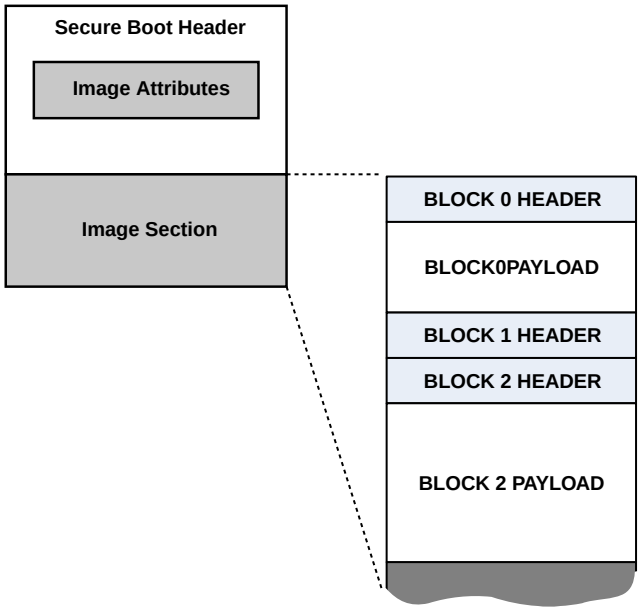

Figure 36-15: Secure Boot Image

Secure Boot Header

Table 36-41: Secure Boot Header

Bytes	Name	Description	Values		
			Keyless Format (BLx)	**Wrapped Format (BLw)**	**Plaintext Format (BLp)**
3:0	Type	Format and version of the image. Upper 24 bits is the image format and lower 8 bits is the image version	0x424c7801	0x424c7701	0x424c7001
67:4	Signature	The ECDSA signature of the image	Two 256-bit numbers		
91:68	Key	Confidentiality (only applicable for certain formats)	Reserved	192-bit AES-WRAP data holding a 128-bit AES key	Reserved
107:92	IV	Initialization Vector (only applicable for certain formats)	Reserved	16-byte IV generated during signing process	
111:108	Length	The length of the image section in bytes	Maximum supported byte count 0x10000000 bytes		
175:112	Attributes	Image attributes	Support for up to 8 image attributes		
179:176	Reserved	Reserved	Reserved		

Overview of Secure Boot Processing

The ***Secure Boot Processing*** figure illustrates how the block is processed regarding secure features. The figure does not detail all the block header type handling when processing the 16-byte block headers of the image section. For details on the various block types and their functionality, refer to Block Types . Some image types are decrypted. [Decrypt] indicates that the data is decrypted when applicable to the Secure Boot Image Types at that particular stage.

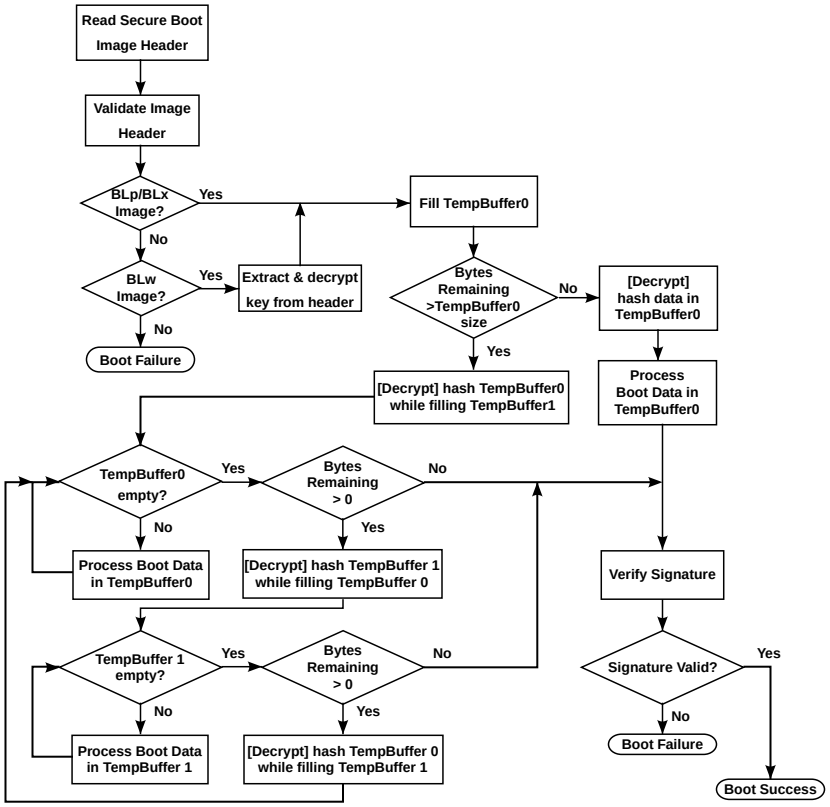

Figure 36-16: Secure Boot Processing

Unsupported Boot Stream Blocks

To ensure the security of the processor, the following block types are not supported in a secure boot image. If the boot kernel finds one of these block types, the boot process terminates.

- Init Block

  Init blocks require a call to user application code prior to the authentication of the boot image, and, therefore, cannot be supported. If customizations or optimizations are necessary to improve the load performance, use a second stage loader style implementation. The first application will contain only the custom code. Issue a call using `adi_rom_Boot()` to boot using the desired device.

- Callback Block

  Callback blocks require a call to a user-defined address prior to the authentication of the boot image, and therefore cannot be supported.

NOTE: Secure boot streams use double buffer Page Mode to optimize the boot process. This functionality allows for the performance of decrypt and hash operations on received data while new data is fetched from the boot source. This host in slave boot mode must ensure that more data is sent after the boot stream to ensure that the temp buffer is filled completely. The size of the secure boot stream minus the size of the

secure boot header must be a multiple of the size of the temp buffer. The temp buffer default size is 1024 bytes.

Secure Boot Image Attributes

Secure boot image attributes form part of the secure boot header . The attributes provide more information about the content of the secure boot image.

All image attributes are integrity protected using the same algorithm as the image section. When the image authentication process completes and the image is successfully authenticated, the image attributes are known to be trustworthy.

Attributes are specified as type value pairs with both the type and value being a 32-bit value. The boot code supports the following image attributes.

Table 36-42: Secure Boot Image Attributes

ID	Name	Description	Values	
0x00000000	Unused	Unused attribute	Value must be 0x00000000	
0x00000001	Version	Version of the attribute format	Value must be 0x00000000	
0x00000002	Rollback ID	Current value of the rollback counter	0x00000000	Rollback disabled
			0x00000001 - 0x0000001F	Current firmware revision. Must be greater than or equal to the value retrieved from OTP Refer to Anti-Rollback Protection and the OTP chapter for details on the counter implementation.
0x80000001	NoRestore	Controls whether the boot process restored registers back to default values and clears sensitive security related information from the stack and dedicated structures.	0x1	Do not restore registers and clear security information from the stack and dedicated structures
			other	Restore registers and clear security sensitive information from the stack and dedicated structures
0x80000002	BCODE	Used by the boot mode drivers that support auto-detection to configure the device from a range of pre-configured settings	Values between 0x00000000 and 0x0000000F supported. Please refer to the boot mode specific documentation for details on a boot modes supported BCODE.	

Secure Debug Access

The TAPCcontroller provides a means of restricting access to secure resources of the processor. Secure access through the debug port is protected through a 128-bit security key that must match a key that has been loaded into OTP for access.

To access a locked processor, the TAPC must allow access to the part. The TAPC only allows access to the part if it is provided with a matching key to the data loaded into its `TAPC_USERKEYn` registers.

With the processor in a locked state, on initial boot the boot ROM reads the 128-bit `secure_emu_key` from the OTP memory and programs the key into the `TAPC_SDBGKEY0`, `TAPC_SDBGKEY1`, `TAPC_SDBGKEY2`, and `TAPC_SDBGKEY3` registers before then setting the `TAPC_USERKEY_CTL.USERKEY_VALID` bit. Then, the TAPC is able to access a matching outside key to allow access.

CAUTION: A key of 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF provisioned in OTP will result in the boot code bypassing the key load operation entirely. If debug access is then ever required the key must be loaded to the TAPC by user software. If the processor fails to boot perhaps due to corrupted firmware then the user will have no debug access. The only way to gain access would be to load an authenticated boot image that can then load the required keys prior to attempting to connect with a debugger.

The key is set in OTP using the OTP boot program API to program the `secure_emu_key` , this key is read and loaded by the ROM in the following sequence:

1. Bits 31:0 of the key in OTP are stored to `TAPC_SDBGKEY0` bits 31:0
2. Bits 63:32 of the key in OTP are stored to `TAPC_SDBGKEY1` bits 31:0
3. Bits 95:64 of the key in OTP are stored to `TAPC_SDBGKEY2` bits 31:0
4. Bits 127:96 of the key in OTP are stored to `TAPC_SDBGKEY3` bits 31:0

Once the ROM has loaded the user key, a test key can be provided to the TAPC through JTAG. Refer to the Emulator manual for details for providing the key.

A key failure indication can be detected through the `TAPC_SDBGKEY_STAT` register. The boot code does not check the key status, nor does it enable any associated interrupts to signal key failure. The boot code continues to boot upon a key failure in a secure manner. The key failure status remains intact so that the application loaded can check for a failed challenge on the debug port.

The boot code can be configured to bypass the loading of the key during the boot sequence by setting the value of the `otp_data::secure_emu_key` in the OTP to all ones. In this case, the only way to gain access to the secure resources through the debug port is to load an alternate key using the application. The alternative key must always reside in a secure region of memory. Or, if sent remotely, it should be transmitted over a secure connection.

Errors and Failures

Any errors encounters while processing a secure boot image results in the ROM jumping to the Error Handler . This includes decryption failures, authentication failures, and configuration errors.

As the boot process doe no checking for a matched secure debug key, should an incorrect key be supplied during boot, no boot failure will occur and the processor will continue to boot as normal. A user supplying an incorrect key will not be able to gain access to any secure resources of the processor.

Boot ROM Programming Model

This section describes the programming model for booting the processor. The programming model includes booting functions, API calls, and data structures.

Boot Mode Driver API

The kernel provides a mechanism to provide a customization of supported boot modes or for implementation of completely new boot modes as second stage boot loaders. This allows users to customize booting while still taking advantage of the rest of the booting framework. A custom boot mode could provide support for a peripheral that is not supported for boot by the ROM, or it could support one of the same peripherals but with a different configuration.

All the same security features can be supported when using a custom boot mode.

A full boot mode, as perceived by the boot implementation, is a collection of five functions.

- Register - installs the driver functions listed below so they can be accessed by the boot process
- Initialization - initialize the boot source
- Configuration - configure the boot source
- Load - read from the boot source
- Cleanup - called after booting

Of these later four functions the boot kernel is only ever aware and has a requirement to support the Load function. It is this function that is responsible for the fetching of the boot stream from the boot peripheral. The other functions are used prior to executing the kernel or for cleaning up after the kernel has completed processing the boot stream.

To install a custom boot mode:

- Create a first stage boot application to define a Load function
- Use the `adi_rom_BootKernel()` API to call the boot kernel once the boot peripheral and pinmuxing has configured. Ensure all the fields of `ADI_ROM_BOOT_CONFIG` are configured accordingly prior to performing the call.

The boot mode can use the `pModeData` member of `ADI_ROM_BOOT_CONFIG` to preserve and access shared data across the different function calls if required.

All functions have the following prototype:

```
void apiFunction(
ADI_ROM_BOOT_CONFIG* pBootStruct);
```

Load Function

The load function is required to read data from the source into the specified destination, according to the parameters given through the configuration struct parameter. The structure provides all of the required information read from the block header, or specified by the kernel to read the block header. The load function often makes use of the supplied DMA APIs in order to simplify the load function implementation.

As the kernel processes the stream, it calls the load function to request data. Initially, the request is for the header, then the kernel requests according to the block flags it parses. The load function must only read from the device, and write where requested.

Relevant fields within the `ADI_ROM_BOOT_CONFIG` object for the load function can be (not limited to): `uwDataWidth`, `pSource`, `dByteCount`, `pDestination`, `loadType`.

Custom load functions must meet the following requirements.

- Protect against `dByteCount` values of zero
- Use multiple DMA units if `dByteCount` is greater than 65536 and the peripheral does not support byte count transfers greater than 65536
- The `pSource` and `pDestination` pointers must be properly updated after loading.

In slave boot modes, the boot kernel uses the address of the `dArgument` field in the `pHeader` block as the destination for the required dummy DMAs when payload data is consumed from `ROM_BFLAG_IGNORE` blocks. If the load function requires access to the `ARGUMENT` word of the block, it should be read early in the function.

Initialization/ Configuration Function

The initialization and configuration functions are called in sequence when calling a boot operation using an already supported boot peripheral via the `adi_rom_Boot()` API. These functions are used to configure the boot peripheral prior to calling the boot kernel. Both functions are called in sequence separated only by a call to a user-defined hook function. This hook function is useful when using built-in boot modes to further customize their functionality. The initialization and configuration functions are responsible for applying any required settings to any devices in use. For example, pin multiplexing may need to be applied, and any data or pointers that are used by the load function must be initialized. The specific actions depend on the device and functionality used.

Cleanup Function

The cleanup function is called after the entire boot stream is read, and the kernel has completed its boot mode-specific function. This is only performed when using the `adi_rom_Boot()` API. Resetting of any status registers, or device parameters is done to prepare the environment for the execution of the newly loaded application.

Wakeup Functionality

When the boot code detects that the chip is waking up from hibernate, it can take some special actions. These actions may include configuration of the clocks and memory controllers and bypassing the regular boot process and executing code from an external memory.

The wake-up sequence executes when the RCU_BCODE.HBTOVW bit is set during the boot process. The boot software sets this bit when RCU_STAT.HBRST is set, indicating the part is coming out of hibernate. Some wake-up functionality can also be used in system reset scenarios where the RCU_BCODE register is preserved.

The DPM_RESTORE0 register controls specific wake-up behavior. Refer to ADI_ROM_SYSCTRL::ulWUA_Flags for full details on the wakeup action flags that describe the various actions that can be performed upon a wakeup event by the boot code.

Do not assume that the host was aware of the processor being in the hibernate state, especially when considering slave boot modes. The processor enters hibernate on its own decision. There is no direct path for the host to control this action.

The wake-up functionality utilizes adi_rom_sysControl() to manage the DPM registers for saving context prior to entering hibernate and for restoring context upon wakeup. The DPM registers are assigned according to the assignments described in ADI_ROM_SYSCTRL and allow for restoring primarily the context of the CGU and DMC infrastructure components.

CGU Initialization after Wake up

CGU configuration is lost when the processor enters the hibernate state, and default settings are applied when the device wakes up. Using the DPM_RESTORE[n] registers optimal CGU configuration can be restored. To use this functionality, load the DPM_RESTORE[n] registers using adi_rom_sysControl() and set the ROM_WUA_CGU and ROM_WUA_EN bits of ADI_ROM_SYSCTRL::ulWUA_Flags .

Settings are applied when the boot process applies configured the CGU during the preboot phase of the boot process.

NOTE: Enabling CGU configuration from the wakeup actions will prevent CGU configuration from values stored in OTP from taking effect.

DMC Controller Initialization after Wake up

In the hibernate state, power is cut off to some portions of the processor, while power is limited to others, allowing certain information to be saved. For specifics, refer to the DPM chapter. When the device exits the hibernate state, the DMC may be reinitialized.

NOTE: The DMC restoration functionality of the wakeup actions can only be used to bring external memory out of self-refresh mode. If external memory is not placed into self-refresh mode prior to entering hibernate then wakeup actions for the DMC must not be used.

In order for the DMC to be reinitialized and the memory brought out of self-refresh mode, programs must save the current DMC configuration to the DPM_RESTORE[n] registers prior to the processor entering the hibernate state .

The adi_rom_sysControl() routine provides a means for programs to save the context prior to entering hibernate in a manner that is compatible with the boot code for restoration.

NOTE: When saving the context of DMC, please be aware that there are some register fields in the DMC that always read as zero such asDMC_CTL.INIT and DMC_CTL.DLLCAL. Users should ensure these bits are set accordingly after calling adi_rom_sysControl() either in the ADI_ROM_SYSCTRL object in memory prior to saving to the DPM or within the DPM registers directly.

Memory Boot

Normal boot loading functionality can be bypassed allowing for execution from a given external memory instead. To use this functionality, configure the appropriate ADI_ROM_SYSCTRL::ulWUA_Flags fields with the ROM_WUA_MEMBOOT and ROM_WUA_EN bits set. Other initial configurations can still be applied. Prior to calling the boot kernel the boot code checks the ROM_WUA_MEMBOOT flag and if set will then vector to the address stored in ADI_ROM_SYSCTRL::ulWUA_BootAddr. The purpose of this functionality is to allow for code execution from memory interfaced to the DMC that is brought out of self refresh after wakeup allowing for a quick bring-up process in wakeup events.

Conditional Processing of Boot Blocks

During any wake-up event, the requested boot mode is called with ROM_BFLAG_WAKEUP flag set in the flags parameter of adi_rom_Boot() resulting in optional conditional processing of boot blocks for wakeup events as described in Quick Boot Block .

Error Handler

This section describes the default error handler for the ROM including information on how to customize the error handling.

The default error handler eventually puts the core into an idle state. This functionality can be overridden by using an Init Block (see Block Types) to modify the error function point in the ADI_ROM_BOOT_CONFIG structure. The error handler has access to the entire boot info structure and receives the instruction address that triggered the error.

When a part is locked, and the boot type has not disabled secure boot, only the default error handler is called.

The expected prototype is:

```
void ErrorFunction(
ADI_ROM_BOOT_CONFIG* pBootStruct, void *pFailingAddress);
```

The error handler saves the failing address to the ADI_ROM_BOOT_CONFIG structure then raises the INTR_SOFT3 fault signaling a fault condition to the system before then entering an endless loop in the boot rom.

NOTE: When using the adi_rom_Boot() function to perform a boot action. Users may need to manually configure the INTR_SOFT3 fault signaling depending on previous application software executed. Calling the boot process via adi_rom_Boot() does not result in all the SEC and Fault configuration being reset and installed as described in Preboot Operations.

Page Mode

For the benefit of page oriented boot source devices, and to improve boot performance for secure boot operations, the boot kernel provides support for page operations. Page mode optimizes memory reads for block organized devices by always reading a page, rather than reading data on demand. Two 1024 byte buffers are used in page mode.

Two buffers are used allowing the contents of one buffer to be processed by the boot kernel while DMA is used to load the next data into the second buffer.

The load to the active buffer uses a blocking DMA, forcing the process to pause until the DMA is complete. Loading to the non-active buffer uses non-blocking DMA allowing the active buffer to be processing while loading the new data in parallel.

Page mode can be enabled when calling a boot mode via the `adi_rom_Boot()`. Refer to the API documentation for the various supported by this API. Additionally users can set the flag via `ADI_ROM_BOOT_CONFIG::dFlags` when using hook functions

NOTE: Due to security requirements it is not recommended to customize the page mode settings from the default installed by the boot process.

Boot Hook Function

The boot software allows installation of callback hooks through the use of the `adi_rom_Boot()` APIs hook function parameter. By using this feature, it is possible to alter the state of the processor, at different stages of the boot process and customize the boot structures to alter the behavior of the boot process.

The hook function must adhere to the following prototype:

```
int32_t
hookFunction(
ADI_ROM_BOOT_CONFIG* pBootconfig,
ROM_HOOK_CALL_CAUSE cause);
```

By modifying settings in the `ADI_ROM_BOOT_CONFIG` structure, many alterations of the boot process can be achieved. Much of the same functionality that is available in an Init Block can be provided through the hook function, with even more flexibility for customization. The hook function is called once after executing the boot modes Init routine then once again after executing the boot modes Config routine. A flag passed to the hook function allows software to determine at which point the call took place to allow for conditional processing to occur at different stages of the setup phase.

The hook function must return a zero value in order for normal booting to continue. A non-zero return value will cause the ROM to skip over loading of any data and immediately transfer control according to Boot Termination and Application Execution .

When the hook function is called, a parameter is passed indicating why the hook function was called. Refer to `ROM_HOOK_CALL_CAUSE` for further details.

enum ROM_HOOK_CALL_CAUSE

Enumeration Type Declaration: `ROM_HOOK_CALL_CAUSE`

Passed to a user hook routine to indicate the reason of the call.

When calling a boot mode via adi_rom_Boot, the user may provide an optional hook routine as a callback. This hook routine is called by the boot software firstly after the execution of the boot modes initialization routine then again after execution of the boot modes configuration routine. This parameter allows the users routine to identify at which point the call was made allowing the user to perform different actions for each call.

Table 36-43: ROM_HOOK_CALL_CAUSE Members

Enumerator	Description
`ROM_HOOK_CALL_INIT_COMPLETE`	Call was as a result of completion of the boot modes initialization function
`ROM_HOOK_CALL_CONFIG_COMPLETE`	Call was as a result of the completion of the boot modes configuration function

ROM_HOOK_CALL_INIT_COMPLETE

Call was as a result of completion of the boot modes initialization function

ROM_HOOK_CALL_CONFIG_COMPLETE

Call was as a result of the completion of the boot modes configuration function

Boot Return Feature

The `adi_rom_Boot()` API provides a feature to bypass calling of the loaded application upon boot completion, and to simply return to the routine that made the call instead. This can be useful when using the . The boot software returns the next address after the last loaded application block in the boot source when this feature is enabled..

To enable this feature, set the `ROM_BFLAG_RETURN` flag in the `flags` argument when calling the API.

Boot Termination and Application Execution

When the boot kernel completes the processing of the boot stream, a sequence of events is required to then pass control to the loaded application.

When the boot process is complete, the boot code must pass control to the loaded application. Typically, the first block of a boot stream, which is marked with the `BFLAG_FIRST` flag, contains the address of the application. This data is loaded into the `RCU_SVECT0` register. Refer to the first block section of Block Types for more information.

When boot stream processing has either been completed or otherwise bypassed and the kernel reaches the end of its execution, the `RCU_SVECT0` register is read. The data is assumed to be the address of the main application. The software then calls that address. The main application must be configured to meet the calling conventions of the C run-time environment.

The No Return feature can suppress a call to the application.

Boot ROM OTP Customizations

The boot ROM provides a mechanism through available non-volatile programmable memory (OTP on this processor) to customize different aspects of the boot process. These customizations include: overriding default boot-peripheral instance, overriding default peripheral-timing parameters and disabling boot modes.

Data in OTP memory controls all ROM customization. The `ADI_ROM_OTP_BOOT_INFO` data structure accounts for most of the options.

CGU Initializations

Refer to CGU Configuration

Boot Command Customization

Refer to Boot Command Customization

DMC Configuration

Refer to DMC Configuration

Secure Boot Customization

All the public and private keys can be invalidated using the various key invalidation fields provided in the `ADI_ROM_OTP_BOOT_INFO` structure. This configuration is useful when a new key is required. The boot ROM always uses the lowest valid key enumeration. If key0 is valid, then it is used, if key0 is invalid and key1 is valid, then key1 is used. Refer to Secure Boot for details on the secure boot functionality.

Disabling Boot Modes

Refer to Boot Mode Disable

API Reference

The APIs defined in this section are exposed for general use.

adi_rom_Boot()

Provides access to boot an application at run-time through a supported peripheral.

API Details

```
void * adi_rom_Boot(
                      void * pAddress,
                      uint32_t flags,
                      int32_t blockCount,
                      ROM_BOOT_HOOK_FUNC * pHook,
                      uint32_t command
                     )
```

pAddress

Pointer to source address of the boot stream.

flags

contains the global flags to be applied to the entire boot process

blockCount

Number of block to be booted. Zero results in processing until a final block is reached.

pHook

Pointer to user implemented hook function for enabling callbacks during the registering of the boot mode with the boot kernel

command

The boot command defining the boot mode to use, the peripheral instance to boot from as well as some boot mode specific configuration

Returns

The 32-bit address of the next address in the boot source to be processed

Function Description

This function may be used for any kind of second-stage boot for an already supported boot mode. It provides options to boot from any peripheral enumeration and in the case of SPI Master boot using any SPI slave select signal.

Boot modes may support an auto-detection mechanism to identify the type of connected device and the function provides options to bypass such auto-detection and use custom configuration options. Options are also provided to bypass peripheral configuration such as pinmux settings or peripheral configuration if existing peripheral already configured are more appropriate to allow communication with the boot source.

These features are all provided via the command parameter which is specific for each particular boot mode.

The source address of the boot stream is required for master boot modes that require an address to be issued in order to request data from the boot source. Slave boot modes are under full control of the host and use a handshake mechanism to indicate that the processor is ready to receive data. For boot modes such as UART Slave and SPI Slave this parameter is of little value in regards to the boot process itself however it can prove useful in debug to see how far through the boot stream the boot process got in the event of a boot failure.

NOTE: The processor supports both SPI Memory Mapped boot as well as Peripheral Based SPI Boot. When the boot mode is called to boot from the memory mapped boot mode via the command argument the address must coincide with the processors memory mapped SPI address space as defined by the processors internal memory map. When using the peripheral based boot mode the absolute address of the boot stream in flash must be used.

Flags passed via the flags argument are global flags and the functionality gets applied throughout the entire boot process. These must not be confused with the boot block specific flags which are part of the boot stream and indicate how a particular block in the boot stream is processed. Internally the boot kernel will take the global flags supplied via this function call and combine them with a boot blocks local flags to determine all the operations to be performed on a given block. After processing the boot block the local flags get cleared ready to be populated from the next boot block and the global flags remain.

The global flags supported by the product are:

Bit Position	Flag Name	Description
18	ROM_BFLAG_HOOK	Calls the user supplied hook function after execution of bootmode init and config routines
19	ROM_BLAG_PAGEMODE	Enables page mode processing where blocks of data are fetched and processed from internal memory
20	ROM_BFLAG_NOFIRS-THEADER	Set this if calling the boot mode and the first block header has already been fetched and present in the block header storage location
21	ROM_BFLAG_HEADER	Not intended to be set by the user, set by the boot code each time it fetched a block header
22	Reserved	Reserved
23	Reserved	Reserved
24	Reserved	Reserved
25	ROM_BFLAG_PERIPHERAL	Boot mode is a peripheral boot mode as opposed to a memory boot mode
26	ROM_BFLAG_SLAVE	Boot mode is a slave boot mode. This results in different handling of ignore blocks by the kernel
27	ROM_BFLAG_WAKEUP	Set this to enable conditional processing of boot blocks intended for wakeup events but not exclusively
28	ROM_BFLAG_NEXTDXE	This flag when set will result in the navigating of the boot stream via the first blocks and their corresponding NextDxe argument. If blockCount is 0 then the routine will return the next free address in the boot source at the end of the list of DXEs. A value of greater than zero is used to return the address of a particular image within the list of DXEs.
29	ROM_BFLAG_RETURN	Return the application after calling the API instead of running the new application
30	Reserved	Reserved
31	ROM_BFLAG_NORESTORE	Do not execute the boot peripherals cleanup routine to restore register contents

The blockCount argument specifies the number of blocks to be processed before terminating the boot process. The default would normally be 0x00000000. A value of 0x00000000 instructs the boot software to continue processing

a boot stream until the ROM_BFLAG_FINAL flag is set. Should users wish to load only a specified number of blocks they can instruct the boot kernel to do so via this parameter.

When the block count is used in combination with the ROM_BFLAG_NEXTDXE flag then the block count is re purposed as a next application count. The boot kernel will navigate the first blocks of multiple boot streams similar to a linked list and upon reaching the requested application count will return the pointer to this application in the boot source. This allows users to use the boot kernel to find a specific application when multiple application boot stream are stored contiguously in the boot source.

The pHook argument is a function pointer to a hook routine. When set along with the ROM_BFLAG_HOOK global flag the boot mode will call the hook routine after calling the boot modes init and config functions in the boot modes driver allowing customization and altering of the configuration performed by the boot software.

This can be used to enabled new features not supported by the boot software and allows for installing of a user defined load function or error handler as an example.

The command argument describes the boot peripheral to boot from, the peripheral instance and contains additional boot mode specific configuration information and flags specific to the boot mode

adi_rom_BootKernel()

Calls the boot kernel allowing for implementation of custom boot modes.

API Details

```
void * adi_rom_BootKernel(ADI_ROM_BOOT_CONFIG * pBoot)
```

pBoot

Pointer to the ADI_ROM_BOOT_CONFIG boot structure containing the complete context of the boot configuration

Returns

Pointer containing the address of the byte immediately following the end of the boot stream.

Function Description

The boot kernel performs the core processing of the boot stream. The boot kernel calls a load function to load in data from the peripheral to the required destination. The boot kernel itself has no concept of what the boot peripheral is or how that peripheral is configured. The kernel calls the registered load function and the load function must then analyze the boot structure and provide the requested amount of data to the required destination.

The load function called by the kernel is provided via the ADI_ROM_BOOT_REGISTRY::pLoadFunction member of ADI_ROM_BOOT_CONFIG::bootRegistry.

The boot kernel basically works in a cycle of fetching a boot stream block header then a payload if one exists. The boot kernel takes care of the size of the data being requested and the destination address.

The load function that is registered with the kernel is required to update the ADI_ROM_BOOT_CONFIG::pSource member. Keeping this control under the load function as opposed to the boot kernel itself allows load functions to better control where the next block of data is fetched in the event the boot stream is fragmented or split into different areas of the boot source.

This function would typically be used to implement a second stage boot loader for a peripheral in which there is no driver support in the boot rom. The user is responsible for initializing the peripheral prior and the complete ADI_ROM_BOOT_CONFIG object prior to calling this function. Upon return form the function the user application is then responsible for performing a vector to the newly loaded application.

NOTE: Users must ensure that a new application being loaded does not clobber the load function and the part of the software responsible for making the core jump to the start of the newly loaded application.

adi_rom_Crc32Init()

The CRC32 Initcode function in the boot rom that is called to enable CRC32 support of boot stream payloads.

API Details

```
ROM_BOOT_RESULT adi_rom_Crc32Init(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

pBootConfig

Pointer to the ADI_ROM_BOOT_CONFIG object containing the complete boot configuration

Returns

Returns the following results

- ROM_BOOT_RESULT::ROM_BOOT_CRC_INITCODE_ERR when pBootConfig or pBootConfig->pHeader are zero
- ROM_BOOT_RESULT::ROM_BOOT_SUCCESS when the callback is registered and lookup table initialized

Function Description

The boot process supports CRC32 protection of all boot block payloads. In order to enable this feature a global callback must be registered with the boot process via ADI_ROM_BOOT_CONFIG::pCrcFunction and the CRC peripherals look up table initialized from the users polynomial.

In order to enable the CRC functionality a init block header in which the ROM_BFLAG_INIT flag is set must be included in the boot stream. The _ADI_ROM_BOOT_HEADER::pTargetAddress field must be set to the address of this function and the users polynomial is provided via the blocks _ADI_ROM_BOOT_HEADER::dArgument member.

When the boot kernel processes the init block described the boot kernel calls this function in the boot rom registering the callback with the kernel and performing the look up table initialization.

The CRC functionality is enabled on `MDMA channel 1` interfaced to the `CRC0` peripheral instance

adi_rom_Crc32Poly()

Initializes the CRC peripheral for use with the user supplied polynomial.

API Details

```
ROM_BOOT_RESULT adi_rom_Crc32Poly(
                                    uint32_t CrcPoly,
                                    ROM_BOOT_MDMA_REGS const *const pDma
                                  )
```

CrcPoly

The CRC polynomail to be used for the intialization of the CRC lookup table.

pDma

Pointer to the `ROM_BOOT_MDMA_REGS` objects that provides access to the DMA channels MMRs and associated CRC peripheral

Function Description

In order to prepare the CRC peripheral for use, the CRC lookup table must be intialized for the CRC polynomial of choice. Users may perform this taks via this function.

adi_rom_GetAddress()

Used to find the location of various look-up tables and data objects used during the boot process.

API Details

```
int32_t adi_rom_GetAddress(ROM_GETADDR_VALUE value)
```

value

The `ROM_GETADDR_VALUE` enumeration specifying the object to retrieve the address of in the ROM memory

Returns

The byte address of the object in memory

Function Description

The function returns the address of the object specified by the enumerator provided as an argument to the function. Using this function cane make software more code compatible with future products and silicon revisions.

Table 36-44: ROM_GETADDR_VALUE Members

Enumerator	Description
ROM_GETADDR_CONSTANTS	ROM_CONSTANTS_TYPE object containing ROM version information
ROM_GETADDR_BMODE	ROM_BOOTMODES_TYPE object containing default boot mode parameters when calling adi_rom_Boot() from preboot
ROM_GETADDR_MDMAREGS	ROM_BOOT_MDMA object containing all the MDMA configuration details for use by the MDMA API routines
ROM_GETADDR_SPILUT	First ROM_SPI_LUTENTRY object in the list of 16 containing all the configuration details for SPI Master boot defined by each BCODE value during autodetection
ROM_GETADDR_ECDSA_DOMAIN	NIST- P-224 Parameter Curves
ROM_GETADDR_ICPLB_DATA	First 32-bit item of an array of 10 containing the ICPLB_DATA[n] items for cache configuration in an open device
ROM_GETADDR_ICPLB_ADDR	First 32-bit item of an array of 10 containing the ICPLB_ADDR[n] items for cache configuration in an open device
ROM_GETADDR_ICPLB_DATA_SECURE	First 32-bit item of an array of 6 containing the ICPLB_DATA[n] items for cache configuration in a locked device
ROM_GETADDR_ICPLB_ADDR_SECURE	First 32-bit item of an array of 6 containing the ICPLB_ADDR[n] items for cache configuration in a locked device
ROM_GETADDR_DCPLB_DATA	First 32-bit item of an array of 12 containing the DCPLB_DATA[n] items for cache configuration in an open device
ROM_GETADDR_DCLB_ADDR	First 32-bit item of an array of 12 containing the DCPLB_DATA[n] items for cache configuration in an open device

adi_rom_MemCompare()

Verifies that a block of data is filled with a user supplied 32-bit value.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCompare(
                                    ROM_DMA_MDMA_CONFIG * pDmaCfg,
                                    ROM_BOOT_MDMA_REGS const *const pDma
                                   )
```

pDmaCfg

Pointer to the ROM_DMA_MDMA_CONFIG object containing the MDMA configuration

pDma

Pointer to the ROM_BOOT_MDMA_REGS objects that provides access to the DMA channels MMRs and associated CRC peripheral

Function Description

The CRC peripheral used in compare mode and a source MDMA channel is used to read data from a buffer and supply each 32-bit value to the CRC. The CRC peripheral checks the incoming 32-bit value matches the 32-bit value to compare against.

adi_rom_MemCopy()

Performs a Memory to Memory DMA (MDMA) operation using a source and destination pair of DMA channels.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCopy(
                                    ROM_DMA_MDMA_CONFIG * pDmaCfg,
                                    ROM_BOOT_MDMA_REGS const *const pDma
                                   )
```

pDmaCfg

Pointer to the `ROM_DMA_PDMA_CONFIG` object containing the peripheral DMA configuration

pDma

Pointer to the `ROM_BOOT_MDMA_REGS` objects that provides access to the DMA channels MMRs and associated CRC peripheral

Returns

Returns the following results

- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` for a successful operation or when byte count is 0 as no operation to be performed
- `ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE` if a configuration error was detected in the DMA channel prior to configuring the channels for the new operation
- `ROM_BOOT_RESULT::ROM_BOOT_MDMA_SRC_ERR` if a configuration error occurred in the source MDMA channel
- `ROM_BOOT_RESULT::ROM_BOOT_MDMA_DST_ERR` if a configuration error occurred in the destination MDMA channel

Function Description

The memory copy routine performs transfers of blocks of data from one memory location to another. The routine takes a basic descriptor providing configuration details of the operation to be performed via the `ROM_DMA_MDMA_CONFIG` object passed as the first argument. The second argument is a descriptor that provides access the DMA channels MMR registers and associated CRC peripheral. When called from the higher level `adi_rom_MemDma()` routine this object is retrieved from the ROM.

NOTE: Users are expected to make use the adi_rom_MemDma() routine for all MDMA operations, there is little additional optional configuration that is supported by using this routine.

adi_rom_MemCrc()

Performs CRC32 verification of a block of block of data by reading the contents and comparing with an expected result.

API Details

```
ROM_BOOT_RESULT adi_rom_MemCrc(
                                    ROM_DMA_MDMA_CONFIG * pDmaCfg,
                                    ROM_BOOT_MDMA_REGS const *const pDma
                                   )
```

pDmaCfg

Pointer to the ROM_DMA_MDMA_CONFIG object containing the MDMA configuration

pDma

Pointer to the ROM_BOOT_MDMA_REGS objects that provides access to the DMA channels MMRs and associated CRC peripheral

Returns

Returns the following results

- ROM_BOOT_RESULT::ROM_BOOT_SUCCESS for a successful operation or when byte count is 0 as no operation to be performed
- ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE if a configuration error was detected in the DMA channel prior to configuring the channels for the new operation
- ROM_BOOT_RESULT::ROM_BOOT_MDMA_SRC_ERR if a configuration error occurred in the source MDMA channel
- ROM_BOOT_RESULT::ROM_BOOT_MDMA_DST_ERR if a configuration error occurred in the destination MDMA channel

Function Description

The routine uses an MDMA channel pairs source DMA channel and the CRC peripheral to calculate a CRC32 result of a data block using a previously supplied polynomial.

The polynomial should be supplied through the adi_rom_Crc32Poly() routine in order to ensure consistent CRC peripheral configuration for the look up table initialization that uses the polynomial and the

NOTE: Users are expected to make use the adi_rom_MemDma() routine for all MDMA operations, there is little additional optional configuration that is supported by using this routine.

adi_rom_MemDma()

Provides access to all the MDMA operations supported by the boot ROM implementation.

API Details

```
ROM_BOOT_RESULT adi_rom_MemDma(ROM_DMA_MDMA_CONFIG * pDmaCfg)
```

pDmaCfg

Pointer to the ROM_DMA_MDMA_CONFIG object containing the MDMA configuration

Returns

Returns the following results

- ROM_BOOT_RESULT::ROM_BOOT_SUCCESS for a successful operation or when byte count is 0 as no operation to be performed
- ROM_BOOT_RESULT::ROM_BOOT_MDMA_ID_ERR if an MDMA channel ID is provided that is not supported
- ROM_BOOT_RESULT::ROM_BOOT_CRC_SUPPORTED_ERR if a CRC operation was attempted on an MDMA channel not supporting CRC
- ROM_BOOT_RESULT::ROM_BOOT_MDMA_OPERATION_ERR if the operation to be performed is not supported
- ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE if a configuration error was detected in the DMA channel prior to configuring the channels for the new operation
- ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE if a configuration error was detected in the DMA channel and the operation requested involves only a single DMA channel
- ROM_BOOT_RESULT::ROM_BOOT_MDMA_SRC_ERR if a configuration error occurred in the source MDMA channel
- ROM_BOOT_RESULT::ROM_BOOT_MDMA_DST_ERR if a configuration error occurred in the destination MDMA channel
- ROM_BOOT_RESULT::ROM_BOOT_CRC_COUNT_ERR if a CRC operation is being requested and the byte count is not a multiple of 4
- ROM_BOOT_RESULT::ROM_BOOT_CRC_FAILURE if CRC32 verification fails
- ROM_BOOT_RESULT::ROM_BOOT_CRC_FAILURE if 32-bit memory compare fails

Function Description

The MDMA operations supported are:

Table 36-45: ROM_DMA_MDMA_OPERATION Members

Enumerator	Description
ROM_DMA_MEM_COPY	Standard MDMA transfer from a source to a destination
ROM_DMA_MEM_CRC	Performs a CRC32 MDMA read operation and compares the result with an expected result
ROM_DMA_MEM_FILL	Uses the CRC peripheral to perform a fill operation with a 32-bit value
ROM_DMA_MEM_COMPARE	Uses the CRC peripheral to compare data with a constant 32-bit value
ROM_DMA_CRC_LUT_INIT	Initializes the CRC LUT from the supplied CRC Polynomial

NOTE: While the MDMA and CRC peripherals support on the fly CRC32 calculations during the transfer of data from one location to another, the MDMA functionality of the boot ROM software does not support this. For CRC calculations data is instead read back from its final destination and verified.

While this function is the main entry point for all the MDMA functionality supported by the boot ROM software, the individual functions that are called for each operation type are also exposed via the public API.

NOTE: The recommendation is to use this function for all operations. The lower level functions allow for some basic reconfiguration of default parameters however such modifications should not be required in most cases where these basic MDMA operations are required.

The boot ROM has an MDMA configuration data structure included that is used to specify the overall MDMA configuration of the processor. It provides details on the DMA channel ID associated with each MDMA channels source and destination DMA channels as well as information relating to CRC support and the CRC peripheral instance that is to be used for a given MDMA channel. Please refer to ROM_BOOT_MDMA and ROM_BOOT_MDMA_REGS for full details of the content stored.

The configuration provided as the only argument to the function is provided below:

Table 36-46: ROM_DMA_MDMA_CONFIG Members

Type	Name	Description
ROM_DMA_MDMA_OPERATION	eOperation	Type of operation to perform
ROM_DMA_MDMA_ID	eId	MDMA Channel ID
void *	pSource	Source Pointer
void *	pDestination	Destination Pointer
uint32_t	ByteCount	Byte Count

Table 36-46: ROM_DMA_MDMA_CONFIG Members (Continued)

Type	Name	Description
ROM_DMA_DONE_DETECT_METHOD	eDoneDetect	DMA Done Detection Method
uint32_t	CrcCtl	CRC_CTL value when CRC operations are required
uint32_t	FillVal	Fill value for memory fill operations
uint32_t	CrcPoly	CRC Polynomial for CRC operations
uint32_t	CrcCompare	Value used for CRC compare operations or for a CRC32 result compare

For a basic MDMA transfer from source to destination the user needs to configure:

- The ROM_DMA_MDMA_CONFIG::eOperation type as ROM_DMA_MDMA_OPERATION::ROM_DMA_MEM_COPY
- The MDMA channel to use via ROM_DMA_MDMA_CONFIG::eId for example ROM_DMA_MDMA_ID::ROM_DMA_MDMA1
- Set the address of the source data via ROM_DMA_MDMA_CONFIG::pSource
- Set the destination address of the source data via ROM_DMA_MDMA_CONFIG::pDestination
- Set the byte count via ROM_DMA_MDMA_CONFIG::ByteCount
- Set ROM_DMA_MDMA_CONFIG::eDoneDetect to ROM_DMA_DONE_DETECT_METHOD::ROM_DMA_DONE_POLL_IRQDONE in order to poll for DMA completion

NOTE: The implementation of the MDMA operations does not support interrupt driven data transfers the routines were implemented for the intentions of polling on the DMA status for use during the boot process. Also there are restrictions in regards to the boot stream in regards to byte counts and source and destination address alignment all being a multiple of 4 bytes and as such, compliance to these restrictions must be adhered to when using these MDMA routines.

When wishing to use the CRC32 functionality of the MDMA routines, the user must first of all initialize the CRC lookup table from the user supplied polynomial. This operation can be performed by setting ROM_DMA_MDMA_CONFIG::eOperation type as ROM_DMA_MDMA_OPERATION::ROM_DMA_CRC_LUT_INIT. If an MDMA channel is specified that does not support CRC functionality an error result is returned.

For further details of the individual operations supported please refer to the following API references:

- adi_rom_MemCopy()
- adi_rom_MemCrc()
- adi_rom_MemFill()

- adi_rom_MemCompare()
- adi_rom_Crc32Poly()

adi_rom_MemFill()

Fills a block of memory with a 32-bit user supplied value.

API Details

```
ROM_BOOT_RESULT adi_rom_MemFill(
                                    ROM_DMA_MDMA_CONFIG * pDmaCfg,
                                    ROM_BOOT_MDMA_REGS const *const pDma
                                   )
```

pDmaCfg

Pointer to the ROM_DMA_MDMA_CONFIG object containing the MDMA configuration

pDma

Pointer to the ROM_BOOT_MDMA_REGS objects that provides access to the DMA channels MMRs and associated CRC peripheral

Returns

Returns the following results

- ROM_BOOT_RESULT::ROM_BOOT_SUCCESS for a successful operation or when byte count is 0 as no operation to be performed
- ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE if a configuration error was detected in the DMA channel prior to configuring the channels for the new operation
- ROM_BOOT_RESULT::ROM_BOOT_MDMA_SRC_ERR if a configuration error occurred in the source MDMA channel
- ROM_BOOT_RESULT::ROM_BOOT_MDMA_DST_ERR if a configuration error occurred in the destination MDMA channel

Function Description

The CRC peripheralis configured for fill mode and the destination MDMA channel is configured to fill a block of memory with a fixed 32-bit value.

adi_rom_PeriphDma()

Provides access to any peripherals dedicated DMA channel for receive operations only.

API Details

```
ROM_BOOT_RESULT adi_rom_PeriphDma(ROM_DMA_PDMA_CONFIG * pDmaCfg)
```

pDmaCfg

Pointer to the ROM_DMA_PDMA_CONFIG object containing the peripheral DMA configuration

Returns

Returns the following results

- ROM_BOOT_RESULT::ROM_BOOT_SUCCESS for a successful operation or when byte count is 0 as no operation to be performed
- ROM_BOOT_RESULT::ROM_BOOT_DMA_ACTIVE if the DMA channel is currently running
- ROM_BOOT_RESULT::ROM_BOOT_DMA_FAILURE if a configuration error was detected in the DMA channel after starting the DMA operation

Function Description

The peripheral DMA routine is used by the load routines of boot peripherals that have dedicated DMA channels and do not support MDMA channel pairs. Examples are the SPI when not configured for memory mapped mode and UART peripherals.

In the boot implementation this routine is called from the peripheral load function to request data from the boot source. The routine supports both polling on DMA completion and non-blocking operation to allow for immediate return after starting the DMA operation and continuing with further processing.

NOTE: The function only supports read operations from the peripheral to memory. Transmit operations from memory to peripheral are not supported

adi_rom_dpmMgmt()

Provides access to functionality for saving and restoring CGU and DMC context to or from memory and the DPM_RESTORE[n] registers.

API Details

```
void adi_rom_dpmMgmt(
                      uint32_t actionFlags,
                      ADI_ROM_SYSCTRL * pSyscontrol,
                      void * reserved
                     )
```

actionFlags

Flags specifying the operations to be performed

pSyscontrol

Pointer to the ADI_ROM_SYSCTRL object in memory

reserved

Reserved for future use and thus should always be NULL

Function Description

The DPM_RESTORE[n] registers are either read or written depending on the supplied actionFlags. If a write to the DPM_RESTORE[n] registers is required the user can specify whether to write the contents stored in the ADI_ROM_SYSCTRL object in memory or to write the context directly from the CGU and DMC peripheral themselves.

When reading the context from the DPM_RESTORE[n] the content is always read into the ADI_ROM_SYSCTRL object in memory and then further operations are required in order to restore that context to the CGU or DMC .

Context save to DPM_RESTORE[n] is initiated by setting ROM_SYSCTRL_WUA_EN and ROM_SYSCTRL_WUA_DPMWRITE flags.

Context restore from DPM_RESTORE[n] to the ADI_ROM_SYSCTRL object is initiated by setting ROM_SYSCTRL_WUA_EN and clearing ROM_SYSCTRL_WUA_DPMWRITE.

The operations to be performed on a call to the function are defined by the actionFlags.

* Bit	Flag Name	Description
[7:0]	Reserved	Reserved
8	ROM_SYSCTRL_DMC_PADRESTORE	When set enables saving or restoring of the pad impedance settings for the DMC else a new calibration initialization is performed, refer to ADI_ROM_SYSCTRL
[15:9]	Reserved	Reserved
16	ROM_SYSCTRL_WUA_EN	Must be set to either save or restore any context to or from the DPM_RESTORE[n] registers
17	ROM_SYSCTRL_WUA_DPMWRITE	Must be set to save context to the DPM_RESTORE[n] registers from the ADI_ROM_SYSCTRL object or cleared to restore context from the DPM_RESTORE[n] registers to the ADI_ROM_SYSCTRL object
18	ROM_SYSCTRL_WUA_CGU	When set enables saving of the CGU items of the ADI_ROM_SYSCTRL object to the DPM_RESTORE[n] registers if a DPM write action was enabled
19	ROM_SYSCTRL_WUA_DMC	When set enables saving of the DMC items of the ADI_ROM_SYSCTRL object to the DPM_RESTORE[n] registers if a DPM write action was enabled
20	ROM_SYSCTRL_WUA_DMC_DLLEN	When set enables an extended delay for DMC DLL calibration

* Bit	Flag Name	Description
21	ROM_SYSCTRL_WUA_BCODE	When set will result in the RCU_BCODE item of ADI_ROM_SYSCTRL being saved to the DPM_RESTORE[n] registers if a write action was enabled
[23:22]	Reserved	Reserved
24	ROM_SYSCTRL_WUA_OVERRIDE	When set instead of writing the various fields of the ADI_ROM_SYSCTRL object to the DPM_RESTORE[n] registers the peripheral registers will be read and stored directly
[31:25]	Reserved	Reserved

NOTE: Some registers may implement read always as zero bits and therefore it is recommended to save the context to the DPM from the structure is in memory as opposed to reading directly from the peripherals and storing. This allows users to set any additional bits that may require to be set but read always as zero.

adi_rom_otp_cfg()

Configures the OTPC to enable read and program operations to be be performed.

API Details

```
bool adi_rom_otp_cfg(void)
```

Function Description

Users may call this routine to ensure the OTPC is configured correctly for read and write access.

NOTE: The preboot process configured the OTPC for use and as such there should be no direct requirement to call this function when using the OTP.

adi_rom_otp_get()

Reads the field from OTP as defined by the supplied OTPCMD.

API Details

```
bool adi_rom_otp_get(
                      OTPCMD cmd,
                      uint32_t data[]
                     )
```

cmd

The OTPCMD enumeration describing the OTP content to be read

data[]

Pointer to storage area to store the read OTP contents

Function Description

Users can read the various fields of the OTP via this routine. The supplied `OTPCMD` object is used to specify the object to be read.

Table 36-47: OTPCMD Members

Enumerator	Description
`otpcmd_reserved0`	Reserved
`otpcmd_huk`	Hardware Unique Key
`otpcmd_gp0`	General Purpose Block 0
`otpcmd_pvt_128key0`	Customer Private Key 0 128bits
`otpcmd_pvt_128key1`	Customer Private Key 1 128bits
`otpcmd_pvt_128key2`	Customer Private Key 2 128bits
`otpcmd_pvt_128key3`	Customer Private Key 3 128bits
`otpcmd_ek`	Endorsement Key
`otpcmd_secure_emu_key`	Secure Emulation Key
`otpcmd_public_key0`	Customer Public Key0
`otpcmd_public_key1`	Customer Public Key1
`otpcmd_boot_info`	Customer Programmable Boot Information
`otpcmd_otpTiming`	OTP Read timing override
`otpcmd_antiroll_nv_cntr`	AntiRollback NV Counter
`otpcmd_gp1`	General Purpose Block 1
`otpcmd_bootModeDisable`	Boot Mode Disable Bits
`otpcmd_preboot_ddr_cfg`	User Preboot DMC configuration
`otpcmd_stageID`	StageID
`otpcmd_reserved1`	Reserved

adi_rom_otp_lock()

Locks the processor, enabling all security features.

API Details

```
bool adi_rom_otp_lock(void)
```

Function Description

This function si used to lock the processor securing the device from unauthorized access. Once called users must supply a secure debug key in order to gain access to the device with debug tools and the part may only be booted using a secure boot stream.

WARNING: Users must ensure that the OTP secure boot fields are all programmed. Secure boot can be verified prior to locking the processor. Users should also provision a secure debug key.

adi_rom_otp_pgm()

Programs the `OTP` Memory with the contents of the `otp_data` object.

API Details

```
bool adi_rom_otp_pgm(otp_data * data)
```

data

Pointer to the::otp_data object containing the complete OTP contents to program

Function Description

The OTP memory is only programmed with values that are not 0. Any items that are 0 are ignored. Users are expected to use this function for all OTP program operations.

adi_rom_sysControl()

Provides access to functionality for saving and restoring CGU and DMC context to facilitate configuration for wakeup events.

API Details

```
ROM_BOOT_RESULT adi_rom_sysControl(
                                    uint32_t actionFlags,
                                    ADI_ROM_SYSCTRL * pSysCtrl,
                                    void * reserved
                                   )
```

actionFlags

Flags specifying the operations to be performed

pSysCtrl

Pointer to the `ADI_ROM_SYSCTRL` object in memory

reserved

Reserved for future use and thus should always be a NULL

Returns

Returns the following results:

- `ROM_BOOT_RESULT::ROM_BOOT_FAILURE` if a write to the CGU failed
- `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` for all other operations

Function Description

The function is intended only for use regarding wakeup events, and not for general CGU or DMC configuration during application run-time. Using this function various settings can be saved and restored in order to be used on a wakeup from hibernate event. All context to be saved is stored in the `DPM_RESTORE[n]` registers.

The same function is used by the boot implementation to handle wakeup events ensuring consistent DPM register organization.

The function provides a normalized method to control configuration of the CGU and the DMC . The following types of operations are provided.

- CGU context save prior to entering hibernate
- DMC context save prior to entering hibernate
- CGU context restore upon wakeup from hibernate
- DMC context restore upon wakeup from hibernate

The sequence of operations supported by this function during boot allows for interfaced memory that was placed into self refresh before hibernate to be brought out of self refresh upon wakeup. The function does not place the actual memory into self refresh mode. Users are required to manually place the the memory into self refresh mode prior to calling the function in order to save the correct context.

NOTE: The function does not provide support for initial configuration of the DMC after a power up sequence. Users should also be aware that when saving the context directly from a peripherals MMR registers that there may be bits that are set but read always as zero. Users need to ensure they set such bits manually in the `ADI_ROM_SYSCTRL` object prior to saving the content to the `DPM_RESTORE[n]` registers.

The routine is implemented such that each call can only save context or restore context. When saving context, control is provided to allow for CGU and DMC settings to be saved to a structure in memory or saved directly into the DPM .

The operations to be performed on a call to the function are defined by the actionFlags.

* Bit	Flag Name	Description
0	ROM_SYSCTRL_CGU_READ	When set will result in the various CGU fields being read from the peripheral into the supplied ADI_ROM_SYSCTRL object in preparation of being written to the DPM_RESTORE[n] registers
1	ROM_SYSCTRL_CGU_WRITE	When set will result in the various CGU fields from the supplied ADI_ROM_SYSCTRL object being written to configure the CGU
[3:2]	Reserved	Reserved
4	ROM_SYSCTRL_CGU_CTL	When set enables saving or restoring of the CGU_CTL register depending on whether a read or write was requested
5	ROM_SYSCTRL_CGU_STAT	When set enables saving of the CGU_STAT to the supplied ADI_ROM_SYSCTRL object
6	ROM_SYSCTRL_CGU_DIV	When set enables saving or restoring of the CGU_DIV register depending on whether a read or write was requested
7	ROM_SYSCTRL_CGU_ CLKOUTSEL	When set enables saving or restoring of the CGU_CLKOUTSEL register depending on whether a read or write was requested
8	ROM_SYSCTRL_DMC_ PADRESTORE	When set enables saving or restoring of the pad impedance settings for the DMC else a new calibration initialization is performed, refer to ADI_ROM_SYSCTRL
[10:9]	Reserved	Reserved
11	ROM_SYSCTRL_CGU_AUTO_ DIS	When set will disable automatic handling of CGU_DIV.UPDT and CGU_DIV.ALGN in the CGU driver. It is not recommended to set this flag.
12	ROM_SYSCTRL_DMC_READ	When set will result in the various DMC fields being read from the peripheral into the supplied ADI_ROM_SYSCTRL object in preparation of being written to the DPM_RESTORE[n] registers
13	ROM_SYSCTRL_DMC_WRITE	When set will result in the various DMC fields from the supplied ADI_ROM_SYSCTRL object being written to configure the DMC
[15:14]	Reserved	Reserved
16	ROM_SYSCTRL_WUA_EN	Must be set to either save or restore any context to or from the DPM_RESTORE[n] registers
17	ROM_SYSCTRL_WUA_ DPMWRITE	Must be set to save context to the DPM_RESTORE[n] registers from the ADI_ROM_SYSCTRL object or cleared to restore context from the DPM_RESTORE[n] registers to the ADI_ROM_SYSCTRL object
18	ROM_SYSCTRL_WUA_CGU	When set enables saving of the CGU items of the ADI_ROM_SYSCTRL object to the DPM_RESTORE[n] registers if a DPM write action was enabled
19	ROM_SYSCTRL_WUA_DMC	When set enables saving of the DMC items of the ADI_ROM_SYSCTRL object to the DPM_RESTORE[n] registers if a DPM write action was enabled
20	ROM_SYSCTRL_WUA_DMC_ DLLEN	When set enables an extended delay for DMC DLL calibration

* Bit	Flag Name	Description
21	ROM_SYSCTRL_WUA_BCODE	When set will result in the RCU_BCODE item of ADI_ROM_SYSCTRL being saved to the DPM_RESTORE[n] registers if a write action was enabled
[23:22]	Reserved	Reserved
24	ROM_SYSCTRL_WUA_OVERRIDE	When set instead of writing the various fields of the ADI_ROM_SYSCTRL object to the DPM_RESTORE[n] registers the peripheral registers will be read and stored directly
[31:25]	Reserved	Reserved

The CGU configuration capacity when restored during the preboot phase of booting is highlighted in the following table

CGU_CTL Frequency Change	CGU_DIV Frequency Change	CGU_DIV.UPDT	CGU_DIV.ALGN	Register Written
No	No	NA	NA	None
No	Yes	1	1	CGU_DIV
Yes	No	NA	NA	CGU_CTL
Yes	Yes	0	0	CGU_DIV then CGU_CTL

When CGU_STAT indicates that the PLL is disabled then CGU_DIV.ALGN is never set.

The CGU and DMC context save and restore operations from the DPM_RESTORE[n] registers are enabled with the setting of the ROM_SYSCTRL_WUA_EN flag in the actionFlags parameter. This results in a call to adi_rom_dpmMgmt() for initial management of content in the DPM_RESTORE[n] registers prior to performing CGU or DMC configuration if requested.

In the boot rom when the processor wakes up from hibernate and is ready to perform the bulk of the wakeup actions it first of all calls this routine with only the ROM_SYSCTRL_WUA_EN flag set. This results in the contents of the DPM_RESTORE[n] registers being read and populated into a local ADI_ROM_SYSCTRL object on the stack.

ADI_ROM_SYSCTRL::ulWUA_Flags is then analyzed to determine what wakeup actions were required. If restoration of CGU and or DMC is requested then the function is called again setting the various write flags for the items requested in the actionFlags argument.

callback()

Callback function for implementing custom callbacks to previously loaded code during boot.

API Details

```
ROM_BOOT_RESULT callback(
                            ADI_ROM_BOOT_CONFIG * pBootConfig,
```

```
                                ADI_ROM_BOOT_BUFFER * pBuffer,
                                uint32_t nFlags
                               )
```

pBootConfig

Pointer to the ADI_ROM_BOOT_CONFIG object containing the complete context of the boot procedure

pBuffer

Pointer to the ADI_ROM_BOOT_BUFFER object containing details of the payload associated with the callback

nFlags

The callback flags as set by the boot kernel

Function Description

A single callback function may be registered with the boot kernel via ADI_ROM_BOOT_CONFIG::pCallBackFunction. This functions is then called whenever a block is processed with the ROM_BFLAG_CALLBACK flag set. Only a single callback function can be registered for the complete boot process.

Callbacks may typically be used alongside indirect blocks. This would be used if there was a requirement for post processing of the received boot data before sending to the final destination. An example of this would be if compression was applied to block payloads. The compressed payload would be loaded indirectly to the intermediate buffer where it would be decompressed by the callback. The callback can modify the source address and byte count for the final MDMA transfer of the decompressed payload via the supplied pBuffer parameter such that when the callback returns the boot kernel then handles the final transfer of the uncompressed data to the destination.

When dealing with indirect blocks, there are restrictions on the amount of data that can be loaded depending on the size of the intermediate buffer. For this reason the nFlags parameter is used to indicate the status of the callback when handling larger blocks of indirect data. The table below defines the supported flags:

Bit Position	Flag Name	Description
0	ROM_CBFLAG_DIRECT	When set indicates the call was from the processing of a block header with the ROM_BFLAG_CALLBACK flag set
1	ROM_CBLAG_PAGESTART	Indicates the callback was a result of a fetch of a page of data to the intermediate buffers
2	ROM_CBFLAG_FIRST	Set if the first fetch of payload data
3	ROM_CBFLAG_FINAL	Set if the final fetch of payload data
31:4	Reserved	Reserved

When a callback block header is received by the boot kernel a call to the callback is performed with the ROM_CBFLAG_DIRECT flag set. If the ROM_BFLAG_INDIRECT flag or the ROM_BFLAG_PAGEMODE flags are set indicating the use of indirect or page mode the ROM_CBFLAG_FIRST and ROM_CBFLAG_FINAL flags are cleared. If

the transfer is a direct transfer straight to the final destination and not via the intermediate buffers then the ROM_CBFLAG_FIRST and ROM_CBFLAG_FINAL flags are also set.

This allows software to identify a callback call based on the processing of a block header with the ROM_BFLAG_CALLBACK flag set.

In addition to callbacks being performed on processing of the block header they are also called when processing payloads indirectly or when page mode is enabled. When the callback is called as a result of processing the payload data via the intermediate buffers ROM_CBFLAG_DIRECT is cleared. If the callback is being called as a result of fetching the first block of data in the payload the ROM_CBFLAG_FIRST flag is set. If the complete block of data fits in the intermediate buffer ROM_BFLAG_FINAL is also set. If the payload does not fit completely in the intermediate buffers multiple fetches must take place and thus multiple callbacks generated. If no flags are set it indicates a callback on a payload transfer and it is neither the first nor the last block of data in the payload, so there is still further data in the payload to be fetched. If only ROM_CBFLAG_FINAL is set then it is the final block in a payload transfer.

The following table provides an overview of the flag states and their meaning for the processing of callbacks.

ROM_CBFLAG_DIRECT	ROM_CBFLAG_PAGESTART	ROM_CBFLAG_FIRST	ROM_CBFLAG_FINAL	Description
1	0	0	0	Callback as a result of processing a block header with indirect or pagemode enabled
1	0	1	1	Callback as a result of processing a block header with indirect and pagemode disabled
0	1	0	0	Callback as a result of fetching a page of data in pagemode
0	1	0	1	Callback as a result of fetching a page of data in pagemode and the final page in the block
0	0	1	0	Callback as a result of fetching the first part of payload in an indirect payload
0	0	0	0	Callback as a result of fetching an indirect payload, not first or last transfer in payload
0	0	0	1	Callback as a result of fetching the final part of payload in an indirect payload
0	0	1	1	Callback as a result of fetching the complete payload in an indirect payload

initcode()

Initcode function for implementing custom callbacks to previously loaded code during boot.

API Details

```
void initcode(ADI_ROM_BOOT_CONFIG * pBootConfig)
```

pBootConfig

Pointer to the ADI_ROM_BOOT_CONFIG object containing the complete context of the boot procedure

Function Description

Initcode functions can be embedded into the boot stream to allow for execution of user defined code during the boot phase. This is typically used to allow for optimal configuration of the CGU or any external memory interfaces that may be required to be initialized in order to be able to boot data to those memories.

A boot stream may have any number of initcodes present. The only requirement is that the code to be executed must be loaded prior to the BFLAG_INIT block being processed.

The initcode routine is passed the pointer to the complete boot context allowing for initcodes to provide extensive boot customization tasks is so desired.

Data Structures

The programming model for booting the processor uses the data structures defined in this section.

struct ADI_ROM_BOOT_BUFFER

Structure Type Declaration: ADI_ROM_BOOT_BUFFER

Boot Buffer.

A basic buffer type consisting of a pointer to the buffer and its size

Table 36-48: ADI_ROM_BOOT_BUFFER Members

Type	Name	Description
void *	pBuffer	Pointer to the buffer
int32_t	dByteCount	Size of the buffer

pBuffer

Pointer to the buffer

dByteCount

Size of the buffer

struct ADI_ROM_BOOT_CONFIG

Structure Type Declaration: ADI_ROM_BOOT_CONFIG

The Boot Configuration Object that contains all context for the boot process.

This structure contains the complete context for the boot process. A pointer to this object is passed through many of the routines and is presented to routines that are expected to be customized by users such as initcodes, custom initialization, configuration, load and cleanup routines. The object is passed to error handlers and callbacks giving the

end user opportunity to significantly customize and adapt the boot process to their specific needs, especially in regards to multi-stage boot loader development.

Table 36-49: ADI_ROM_BOOT_CONFIG Members

Type	Name	Description
void *	pSource	Source address from where to fetch the next boot data.
void *	pDestination	Destination address to store the fetched data.
int32_t	dByteCount	Number of bytes to fetch from the boot source.
int32_t	dFlags	Control flags related to the boot kernel processing of blocks.
uint32_t	ulBlockCount	Limit of blocks to be processed during boot.
uint32_t	ulBlockCurrent	The number of blocks currently processed by the boot kernel
void *	pNextDxe	Pointer to the next application in the boot stream or the first free location after the boot stream.
uint32_t volatile *	pControlRegister	Pointer to the boot peripherals control register.
int32_t	dControlValue	Storage for the boot peripheral main control value to enable that peripheral in a required configuration.
uint32_t volatile *	pPeripheralBase	Pointer to the boot peripherals base MMR address
uint32_t volatile *	pAuxControlRegister	Pointer to any register that may be used for auxiliary operations such as a timer control register for UART autobaud detection
uint32_t volatile *	pAuxPeripheralBase	Pointer to the base address of any peripheral used for auxiliary operations such as the TIMER block
uint32_t volatile *	pSecControlRegister	Base MMR address of the SEC SSI instance associated with the boot peripheral should they be required for advanced second stage boot loader development
ADI_DMA_TypeDef *	pDmaBaseRegister	Base MMR address of the DMA channel associated with the boot peripheral.
ROM_DMA_DONE_DETECT_METHOD	loadType	Set by the kernel to specify to the boot peripherals load function if it is requesting a blocking or non-blocking DMA
ROM_DMA_MDMA_CONFIG	MdmaCfg	An MDMA descriptor that is used by the boot kernel for internal MDMA operations.
uint16_t	uwDataWidth	The maximum data width supported by the boot peripherals DMA channel. Set to 0 for 8-bit, 1 for 16-bit and 2 for 32-bit
uint16_t	uwSrcModifyMult	The source modify multiplier used to set DMA_XMOD for source MDMA operations or peripheral DMA transmit operations
uint16_t	uwDstModifyMult	The destination modify multiplier used to set DMA_XMOD for destination MDMA operations or peripheral DMA receive operations
uint16_t	uwUserShort	Free to use by the user
int32_t	dUserLong	Free to use by the user
int32_t	dReserved0	Reserved for future use

Table 36-49: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
void *	pModeData	Pointer to the boot mode specific data structure.
int32_t	dBootCommand	The boot command value supplied during the call to the adi_rom_Boot() routine
ADI_ROM_BOOT_HEADER *	pHeader	Pointer to the boot header storage location where all boot stream block headers eventually reside for processing by the kernel
void *	pTempBuffer	Pointer to the internal intermediate buffer. Used for processing of indirect blocks
void	dReserved1	Reserved
int32_t	dTempByteCount	Size of the internal intermediate buffer in bytes
void *	pTempSource	Current source address that is being processed in the internal intermediate buffer
int32_t	dPageByteCount	The page size to be used for page mode processing. On this product page size is fixed to 1024 bytes and this member is not used for the load requests
ADI_ROM_BOOT_INTER_BUFFERS	bootBuffers	The internal intermediate buffer descriptors required when using indirect and page mode features
ROM_BOOT_REGISTRY	bootRegistry	The registry object that is used to register a boot peripherals routines with the kernel.
ROM_BOOT_ERROR_FUNC *	pErrorFunction	Pointer to the error handler to be called in the event of an error
ROM_BOOT_CALLBACK_FUNC *	pCallBackFunction	Pointer to the callback function that is called when processing boot blocks with the callback flag set
ROM_BOOT_CALLBACK_FUNC *	pCrcFunction	Pointer to the CRC function that is used to perform CRC validation of the boot stream payload data
ROM_BOOT_CALLBACK_FUNC *	pForwardFunction	Feature not supported on this product
ADI_ROM_BOOT_MODES	bootModes	Access to all boot mode specific resources
void *	pLogBuffer	Pointer to the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines
void *	pLogCurrent	The current position within the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines
int32_t	dLogByteCount	The size of the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines
ADI_ROM_OTP_BOOT_INFO *	pOtpBootInfo	Pointer to the ADI_ROM_OTP_BOOT_INFO boot information block that gets read from OTP and contains boot customization options
ADI_ROM_BOOT_KEY_TYPE	keyType	When set to a specific value allows keys not stored in OTP to be used for secure boot evaluation on an open processor.

Table 36-49: ADI_ROM_BOOT_CONFIG Members (Continued)

Type	Name	Description
ADI_ROM_BOOT_TYPE	bootType	A key to indicate if the boot type is secure or non-secure for open parts
ROM_SB_IMAGE_TYPE	secureBootImageType	The type of secure boot image
void *	pSecureHeader	Pointer to the secure boot stream header that is loaded by the boot peripheral from the boot source during the configuration phase
ADI_SBIF_ECDSA_PublicKey_t	publicKey	The public key used for secure boot image authentication
CRYPTO_DESCRIPTORS	cryptoDescriptors	The descriptor items as required for the PKTE operations
void *	Reserved2	Reserved for internal use
int32_t	secureBytesRemaining	The number of bytes remaining to be processed in the secure boot stream
uint32_t[4]	aesKey	The 128-bit AES decryption key
uint32_t[6]	aesWrapKey	The key wrapped key from the BLw secure boot image
uint32_t[4]	IV	The IV as read from the secure boot header as required to initialize the PKTE
uint8_t *	pHash	Pointer to the output destination of the SHA-224 result that is required for authentication of the secure boot stream
uint32_t	errorReturn	Storage location for the address of the instruction line following a call to the error handler

pSource

Source address from where to fetch the next boot data.

The source address must be maintained by the boot peripherals load function. The kernel does not update the source pointer automatically after requesting data. This allows for load routines to control and change the source address, especially useful for advanced second stage loaders if they have a requirement to change the source address due to a fragmented boot stream or to reset the address if expanding into a second SPI flash device.

During debug it is useful in identifying the block in the boot stream that is currently being processed.

pDestination

Destination address to store the fetched data.

Used by the boot kernel to indicate the destination address for the data to be fetched. Boot peripherals load function must transfer the data to this location before returning back to the kernel. The boot kernel updates this field depending on whether a block header or payload is being fetched. In normal mode of operation the kernel will load this field with the location of the storage area to store a block header then after processing the block header loads it with the ADI_ROM_BOOT_HEADER::pTargetAddress contents red from the fetched block header. When using page mode the destination points to the internal buffers and is then updated for transferring data to the final destination.

dByteCount

Number of bytes to fetch from the boot source.

The kernel sets this parameter to indicate to he load function the number of bytes the kernel is requesting. The kernel is responsible for adjusting the byte count for page mode based accesses. The peripherals load function must return the required number of bytes to the destination address provided.

dFlags

Control flags related to the boot kernel processing of blocks.

When calling a boot mode via the `adi_rom_Boot()` routine the `flags` supplied to that routine are used to initialize this item. These become global flags that remain set through the entire boot process. When a block header is received the lower 16-bits of the block header are OR'ed with the global flags. The boot software may clear some flags if it detects some flags are not compatible with some others and then writes the resulting flags back to this member. The boot kernel then processes the block payload as instructed by the combination of global and boot block specific flags. Upon completion of the block processing original set of global flags are restored and the process repeated.

ulBlockCount

Limit of blocks to be processed during boot.

When calling the boot process the `adi_rom_Boot()` routine can accept a block limit for the number of blocks to process before terminating the boot process. If the block count is set to zero then the boot process will continue until a final block reached indicating end of the boot stream. This member holds the user specified limit for the number of blocks to be processed and is used by the boot kernel after processing of each block and compares it against the `ADI_ROM_BOOT_CONFIG::ulBlockCurrent` value. Boot process terminates when `ADI_ROM_BOOT_CONFIG::ulBlockCurrent` equals `ADI_ROM_BOOT_CONFIG::ulBlockCount`

ulBlockCurrent

The number of blocks currently processed by the boot kernel

pNextDxe

Pointer to the next application in the boot stream or the first free location after the boot stream.

This member is initialized when processing a first block in the boot stream. The `ADI_ROM_BOOT_HEADER::dArgument` field of a first block contains the number of bytes left in the boot stream before we reach the end of that boot stream. This allows for the this pointer to be used to point to the next boot stream or to the first empty location after the boot stream. This allows for a feature when using the `adi_rom_Boot()` routine to find the address of an application in a linked list of boot streams or to find the first empty location after the boot stream.

pControlRegister

Pointer to the boot peripherals control register.

This can be used by a boot mode peripherals driver in order to gain efficient access to a control MMR register in the boot peripheral.

NOTE: This is not used in this products boot implementation but may be leveraged by developers of second stage boot loaders if required

dControlValue

Storage for the boot peripheral main control value to enable that peripheral in a required configuration.

This can be used by a boot modes peripheral driver in order to store a control value that can be used to enable the peripheral for a required configuration.

pPeripheralBase

Pointer to the boot peripherals base MMR address

pAuxControlRegister

Pointer to any register that may be used for auxiliary operations such as a timer control register for UART autobaud detection

pAuxPeripheralBase

Pointer to the base address of any peripheral used for auxiliary operations such as the TIMER block

pSecControlRegister

Base MMR address of the SEC SSI instance associated with the boot peripheral should they be required for advanced second stage boot loader development

pDmaBaseRegister

Base MMR address of the DMA channel associated with the boot peripheral.

This is used by the boot kernel to gain access to the DMA channels status when using non blocking DMA operations when page mode or secure boot is required, so it must be set when implementing custom boot loaders in order for the kernel to get access to that peripherals DMA status.

NOTE: If a custom boot peripheral does not support the standard DMA instance then the custom driver will be required to set up a DMA instance in SRAM that this location points to and the load function would need to update the status accordingly to indicate when the DMA was running and when the DMA completed.

loadType

Set by the kernel to specify to the boot peripherals load function if it is requesting a blocking or non-blocking DMA

MdmaCfg

An MDMA descriptor that is used by the boot kernel for internal MDMA operations.

The boot kernel may be required to perform internal MDMA operations outside the control of the boot peripheral driver. Such operations include processing of fill blocks CRC callbacks for CRC verification and MDMA operations from the internal intermediate buffers for indirect block and page mode processing.

NOTE: Users must not use this item or reconfigure this item when developing custom boot drivers, it is intended purely for the internal use by the boot kernel

uwDataWidth

The maximum data width supported by the boot peripherals DMA channel. Set to 0 for 8-bit, 1 for 16-bit and 2 for 32-bit

uwSrcModifyMult

The source modify multiplier used to set `DMA_XMOD` for source MDMA operations or peripheral DMA transmit operations

uwDstModifyMult

The destination modify multiplier used to set `DMA_XMOD` for destination MDMA operations or peripheral DMA receive operations

uwUserShort

Free to use by the user

dUserLong

Free to use by the user

pModeData

Pointer to the boot mode specific data structure.

Can be set by a boot peripheral driver to allow for a single point of access to the boot mode specific object containing control and configuration information specific to that single boot mode.

dBootCommand

The boot command value supplied during the call to the `adi_rom_Boot()` routine

pHeader

Pointer to the boot header storage location where all boot stream block headers eventually reside for processing by the kernel

pTempBuffer

Pointer to the internal intermediate buffer. Used for processing of indirect blocks

dTempByteCount

Size of the internal intermediate buffer in bytes

pTempSource

Current source address that is being processed in the internal intermediate buffer

dPageByteCount

The page size to be used for page mode processing. On this product page size is fixed to 1024 bytes and this member is not used for the load requests

bootBuffers

The internal intermediate buffer descriptors required when using indirect and page mode features

bootRegistry

The registry object that is used to register a boot peripherals routines with the kernel.

When using the `adi_rom_Boot()` function the boot software calls a peripherals initialization, configuration, load and cleanup routines from the pointers stored in this object. The kernel itself only makes calls to the load function for the peripheral so when using the `adi_rom_BootKernel()` function the load function that is called by the boot kernel to fetch data from the boot source must be registered via `ADI_ROM_BOOT_REGISTRY::pLoadFunction`.

pErrorFunction

Pointer to the error handler to be called in the event of an error

pCallBackFunction

Pointer to the callback function that is called when processing boot blocks with the callback flag set

pCrcFunction

Pointer to the CRC function that is used to perform CRC validation of the boot stream payload data

pForwardFunction

Feature not supported on this product

bootModes

Access to all boot mode specific resources

pLogBuffer

Pointer to the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

pLogCurrent

The current position within the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

dLogByteCount

The size of the log buffer. Logging is disabled by default on this product and thus must be configured from within initcodes or hook routines

pOtpBootInfo

Pointer to the ADI_ROM_OTP_BOOT_INFO boot information block that gets read from OTP and contains boot customization options

keyType

When set to a specific value allows keys not stored in OTP to be used for secure boot evaluation on an open processor.

By default when performing boot on an open part the decryption keys and the public key are fetched from OTP. By setting this field to ADI_ROM_BOOT_KEY_TYPE::ADI_ROM_CUSTOM_SECURITY users can disable the fetching of the keys from OTP and instead provision the keys directly in the ADI_ROM_BOOT_CONFIG::publicKey and ADI_ROM_BOOT_CONFIG::aesKey members via hook routines when using the adi_rom_Boot() function.

bootType

A key to indicate if the boot type is secure or non-secure for open parts

secureBootImageType

The type of secure boot image

pSecureHeader

Pointer to the secure boot stream header that is loaded by the boot peripheral from the boot source during the configuration phase

publicKey

The public key used for secure boot image authentication

cryptoDescriptors

The descriptor items as required for the PKTE operations

secureBytesRemaining

The number of bytes remaining to be processed in the secure boot stream

aesKey

The 128-bit AES decryption key

aesWrapKey

The key wrapped key from the BLw secure boot image

IV

The `IV` as read from the secure boot header as required to initialize the PKTE

pHash

Pointer to the output destination of the SHA-224 result that is required for authentication of the secure boot stream

errorReturn

Storage location for the address of the instruction line following a call to the error handler

struct ADI_ROM_BOOT_CUSTOM

Structure Type Declaration: `ADI_ROM_BOOT_CUSTOM`

A custom boot structure for storing information relating to a custom boot mode.

This structure is not used by the boot rom software at all but it is included in `ADI_ROM_BOOT_CONFIG::bootModes` in the event storage is required for custom boot loaders. Users can make us of this storage are to register DMA channels and define flags that may be used by a custom load routine.

Table 36-50: ADI_ROM_BOOT_CUSTOM Members

Type	Name	Description
`uint32_t`	`nFlags`	Flags related to the custom boot mode
`void *`	`pRegisters`	Used to store a pointer to the MMR registers for the peripheral
`ADI_DMA_TypeDef *`	`pRxDmaRegisters`	Pointer to the peripherals Tx DMA channel
`ADI_DMA_TypeDef *`	`pTxDmaRegisters`	Pointer to the peripherals Rx DMA channel
`ADI_SEC_Sysblock_TypeDef *`	`pSecSsi`	Pointer to the peripherals SEC SSI block
`void *`	`pModeData`	Void pointer that can be used to access additional boot mode specific resources not originally accessible via this data object
`ADI_ROM_BOOT_REGISTRY`	`registry`	For storage of the custom boot mode registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry` to register custom functions

nFlags

Flags related to the custom boot mode

pRegisters

Used to store a pointer to the MMR registers for the peripheral

pRxDmaRegisters

Pointer to the peripherals Tx DMA channel

pTxDmaRegisters

Pointer to the peripherals Rx DMA channel

pSecSsi

Pointer to the peripherals SEC SSI block

pModeData

Void pointer that can be used to access additional boot mode specific resources not originally accessible via this data object

registry

For storage of the custom boot mode registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry` to register custom functions

struct ADI_ROM_BOOT_HEADER

Structure Type Declaration: `ADI_ROM_BOOT_HEADER`

Boot Block Header.

Boot block headers control the loading process of the boot stream, For dull details on the contents of the block header and supported flags see Boot Loader Stream .

Table 36-51: ADI_ROM_BOOT_HEADER Members

Type	Name	Description
`int32_t`	`dBlockCode`	Instructs the boot kernel how to process the block.
`void *`	`pTargetAddress`	Destination Address of Payload
`int32_t`	`dByteCount`	Byte Count of the Payload
`int32_t`	`dArgument`	Argument functionality varies depending on operation

dBlockCode

Instructs the boot kernel how to process the block.

Contains a number of fields for verification of the block header and flags to indicate the type of block allowing the kernel to process the block accordingly.

pTargetAddress

Destination Address of Payload

dByteCount

Byte Count of the Payload

dArgument

Argument functionality varies depending on operation

struct ADI_ROM_BOOT_INTER_BUFFER

Structure Type Declaration: `ADI_ROM_BOOT_INTER_BUFFER`

The buffer object for the internal intermediate buffers used for indirect and page mode operations.

Table 36-52: ADI_ROM_BOOT_INTER_BUFFER Members

Type	Name	Description
`uint8_t *`	`pBuffer`	Pointer to the buffer
`uint32_t`	`size`	Size of the buffer
`uint32_t`	`pageSize`	Page size for block based devices.

pBuffer

Pointer to the buffer

size

Size of the buffer

pageSize

Page size for block based devices.

NOTE: On this product this field is not used. A fixed page size of 1024 bytes is used in this product.

struct ADI_ROM_BOOT_INTER_BUFFERS

Structure Type Declaration: `ADI_ROM_BOOT_INTER_BUFFERS`

The boot kernels internal buffer object used to access the intermediate buffers and obtain buffer status.

Table 36-53: ADI_ROM_BOOT_INTER_BUFFERS Members

Type	Name	Description
`ADI_ROM_BOOT_INTER_BUFFER[2]`	`buffer`	The two buffer descriptors
`ADI_ROM_BOOT_BUFFER_STATE`	`state`	Buffer Status Information
`void *`	`pSource`	Original source address pointer of data loaded to active buffer. Not used on this product
`ADI_DMA_TypeDef *`	`pDma`	Pointer to the DMA channel responsible for loading the buffer. Not used on this product

buffer

The two buffer descriptors

state

Buffer Status Information

pSource

Original source address pointer of data loaded to active buffer. Not used on this product

pDma

Pointer to the DMA channel responsible for loading the buffer. Not used on this product

struct ADI_ROM_BOOT_MODES

Structure Type Declaration: `ADI_ROM_BOOT_MODES`

Holds all boot mode specific configuration items.

A boot mode may have requirements for some dedicated storage. This object is used to collect together all those storage items for all the boot modes supported by the boot rom.

Table 36-54: ADI_ROM_BOOT_MODES Members

Type	Name	Description
`ADI_ROM_BOOT_SPI`	`spi`	Access to all SPI boot mode resources
`ADI_ROM_BOOT_UART`	`uart`	Access to all UART boot mode resources
`ADI_ROM_BOOT_CUSTOM`	`custom`	Access to all custom boot mode resources

spi

Access to all SPI boot mode resources

uart

Access to all UART boot mode resources

custom

Access to all custom boot mode resources

struct ADI_ROM_BOOT_REGISTRY

Structure Type Declaration: `ADI_ROM_BOOT_REGISTRY`

Boot Mode Registration.

Used to hold pointers for the boot modes initialization, configuration, load and cleanup functions. By using pointer users can customize the registered content via hook routines or install their own load functions or cleanup functions from within init codes.

When using `adi_rom_Boot()` the boot process will make a call to the initialization function and the configuration function before calling the kernel. The kernel then runs making calls to the load function. Upon reaching the end of the boot stream the cleanup function is then called.

When using the `adi_rom_BootKernel()` function only the load function is called during execution of the software in the boot rom. All the functions here must return a `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS` result in order for the boot process to continue. All functions expect a single argument that is the pointer to the boot structure object `ADI_ROM_BOOT_CONFIG`.

Table 36-55: ADI_ROM_BOOT_REGISTRY Members

Type	Name	Description
`ROM_BOOT_MODE_INIT_FUNC *`	`pInitFunction`	Pointer to the boot modes Initialization function
`ROM_BOOT_MODE_CONFIG_FUNC *`	`pConfigFunction`	Pointer to the boot modes Configuration function
`ROM_BOOT_MODE_LOAD_FUNC *`	`pLoadFunction`	Pointer to the boot modes Load function
`ROM_BOOT_MODE_CLEANUP_FUNC *`	`pCleanUpFunction`	Pointer to the boot modes Cleanup function
`void *`	`pReserved`	Reserved for future use
`int32_t`	`dReserved`	Reserved for future use

pInitFunction

Pointer to the boot modes Initialization function

pConfigFunction

Pointer to the boot modes Configuration function

pLoadFunction

Pointer to the boot modes Load function

pCleanUpFunction

Pointer to the boot modes Cleanup function

struct ADI_ROM_BOOT_SPI

Structure Type Declaration: `ADI_ROM_BOOT_SPI`

The SPI Master Boot Mode Specific Structure.

This structure contains all the boot context information that is specific to the use for the SPI master boot mode. During auto-detection information is copied from the required ROM_SPI_LUTENTRY item into this structure and used to configure the SPI peripheral for the mode of operation.

Table 36-56: ADI_ROM_BOOT_SPI Members

Type	Name	Description
uint8_t	ubReadCommand	Read command to use to read data from the SPI device
uint8_t	ubDummyBytes	Number of dummy bytes to issue after the read command
uint8_t	ubAddressBytes	Number of address bytes required to access the device
uint8_t	ubDataBits	The bus width to be used when reading the data. 0 for single bit, 1 for dual, 2 for quad
uint16_t	uwClkLower	The SPI clock divider value to be used
uint16_t	uReserved0	Reserved
uint32_t	nTxCtl	The value to be written to the SPI_TXCTL register that is used for the address transmit operations such as address cycles
uint32_t	nRxCtl	The value to be written to the SPI_RXCTL register that is used for all receive operations
uint32_t	nCmdCtl	The value to be written to the SPI_TXCTL register that is used for sending the read command to the SPI Flash
ROM_BOOT_SPIM_IO_ENABLE_FUNC *	pMIOEnFunction	Pointer to the function used to enable quad mode on the SPI flash
uint8_t	nDummy	The Dummy Byte value to be used if dummy byte transfers are required and the bus is not tri-stated
uint8_t	nFlags	Flags used for some additional SPI configuration processing.
uint16_t	uReserved2	Reserved
void *	pXIPAddress	The memory mapped SPI address to boot from
ADI_ROM_BOOT_SPI_FUNC	functions	Device specific driver routines, not used on this product
ADI_ROM_BOOT_SPI_ID	id	The electronic signature of the device, not used on this product
ADI_SPI_TypeDef *	pRegisters	Pointer to the SPI peripherals base MMR address, not used on this product
ADI_DMA_TypeDef *	pRxDmaRegisters	Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product
ADI_DMA_TypeDef *	pTxDmaRegisters	Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product
ADI_SEC_Sysblock_TypeDef *	pSecSsi	Pointer to the SEC SSI item for interrupt configuration for the peripheral, not used on this product
ADI_ROM_BOOT_REGISTRY	registry	For storage of the SPI Master boot specific registration items, not used in this product, use ADI_ROM_BOOT_CONFIG::bootRegistry

ubReadCommand

Read command to use to read data from the SPI device

ubDummyBytes

Number of dummy bytes to issue after the read command

ubAddressBytes

Number of address bytes required to access the device

ubDataBits

The bus width to be used when reading the data. 0 for single bit, 1 for dual, 2 for quad

uwClkLower

The SPI clock divider value to be used

nTxCtl

The value to be written to the `SPI_TXCTL` register that is used for the address transmit operations such as address cycles

nRxCtl

The value to be written to the `SPI_RXCTL` register that is used for all receive operations

nCmdCtl

The value to be written to the `SPI_TXCTL` register that is used for sending the read command to the SPI Flash

pMIOEnFunction

Pointer to the function used to enable quad mode on the SPI flash

nDummy

The Dummy Byte value to be used if dummy byte transfers are required and the bus is not tri-stated

nFlags

Flags used for some additional SPI configuration processing.

The flags supported are defined as follows:

Bit Position	Name	Description
0	`ROM_SPI_FLAGS_CMDSKIP_EN`	When set will result in the configuration routine enabling command skip mode where the SPI will not issue a read command for read operations
1	`ROM_SPI_FLAGS_MULTICMD_EN`	Instructs the configuration routine to enable sending of command cycles over dual or quad bit bus

pXIPAddress

The memory mapped SPI address to boot from

functions

Device specific driver routines, not used on this product

id

The electronic signature of the device, not used on this product

pRegisters

Pointer to the SPI peripherals base MMR address, not used on this product

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

pSecSsi

Pointer to the SEC `SSI` item for interrupt configuration for the peripheral, not used on this product

registry

For storage of the SPI Master boot specific registration items, not used in this product, use `ADI_ROM_BOOT_CONFIG::bootRegistry`

struct ADI_ROM_BOOT_UART

Structure Type Declaration: `ADI_ROM_BOOT_UART`

The UART Slave Boot Mode Specific Structure.

This structure contains all the boot context information that is specific to the use for the UART slave boot mode.

Table 36-57: ADI_ROM_BOOT_UART Members

Type	Name	Description
`uint32_t`	`nFlags`	Flags related to UART Boot mode
`ADI_UART_TypeDef *`	`pRegisters`	Pointer to the UART peripherals base MMR address, not used on this product
`ADI_DMA_TypeDef *`	`pRxDmaRegisters`	Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product
`ADI_DMA_TypeDef *`	`pTxDmaRegisters`	Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

Table 36-57: ADI_ROM_BOOT_UART Members (Continued)

Type	Name	Description
ADI_SEC_Sysblock_TypeDef *	pSecSsi	Pointer to the SEC SSI item for interrupt configuration for the peripheral, not used on this product
ADI_ROM_BOOT_REGISTRY	registry	For storage of the UART Slave boot specific registration items, not used in this product, use ADI_ROM_BOOT_CONFIG::bootRegistry

nFlags

Flags related to UART Boot mode

pRegisters

Pointer to the UART peripherals base MMR address, not used on this product

pRxDmaRegisters

Pointer to the DMA peripherals base MMR address,for receive operations, not used on this product

pTxDmaRegisters

Pointer to the DMA peripherals base MMR address,for transmit operations, not used on this product

pSecSsi

Pointer to the SEC SSI item for interrupt configuration for the peripheral, not used on this product

registry

For storage of the UART Slave boot specific registration items, not used in this product, use ADI_ROM_BOOT_CONFIG::bootRegistry

struct ADI_ROM_OTP_BOOT_CFG

Structure Type Declaration: ADI_ROM_OTP_BOOT_CFG

The boot configuration object for storing further boot customization objects.

This is a 160-bit structure that is allocated to one contiguous region in the OTP memory array. The functionality provided allows for individual flags to be set to enable or disable specific features of the boot process. Each flag is allocated in a separate 16-bit word so that each flag can be set at different times and the ECC information will not impact the setting of another flag.

Table 36-58: ADI_ROM_OTP_BOOT_CFG Members

Type	Name	Description
uint32_t	cacheDis:1 (bitfield)	Cache Disable.
uint32_t	reserved0:15 (bitfield)	Reserved

Table 36-58: ADI_ROM_OTP_BOOT_CFG Members (Continued)

Type	Name	Description
uint32_t	decryptOnlyEn:1 (bitfield)	Decrypt Only Enable.
uint32_t	reserved1:15 (bitfield)	Reserved
uint32_t	cacheDisInv:1 (bitfield)	Cache Disable Invalidate.
uint32_t	reserved2:15 (bitfield)	Reserved
uint32_t	decryptOnlyInv:1 (bitfield)	Decrypt Only Invalidate.
uint32_t	reserved3:15 (bitfield)	Reserved
uint32_t	pubkey0Inv:1 (bitfield)	Invalidate Public Key 0, use next public key for secure boot
uint32_t	reserved4:15 (bitfield)	Reserved
uint32_t	pubkey1Inv:1 (bitfield)	Invalidate Public Key 1, Secure boot will no longer be operational as no further public keys
uint32_t	reserved5:15 (bitfield)	Reserved
uint32_t	privkey0Inv:1 (bitfield)	Invalidate Decryption Key 0, use next Decryption key for secure boot
uint32_t	reserved6:15 (bitfield)	Reserved
uint32_t	privkey1Inv:1 (bitfield)	Invalidate Decryption Key 1, use next Decryption key for secure boot
uint32_t	reserved7:15 (bitfield)	Reserved
uint32_t	privkey2Inv:1 (bitfield)	Invalidate Decryption Key 2, use next Decryption key for secure boot
uint32_t	reserved8:15 (bitfield)	Reserved
uint32_t	privkey3Inv:1 (bitfield)	Invalidate Decryption Key 3, once invalidated part will no longer be bootable
uint32_t	reserved9:15 (bitfield)	Reserved
uint32_t	dmcEn:1 (bitfield)	Enables configuration of the DMC from values in OTP stored in the format of the ADI_ROM_OTP_DMC_CONFIG object
uint32_t	reserved10:15 (bitfield)	Reserved
uint32_t	dmcInv:1 (bitfield)	Invalidates the DMC values in the OTP resulting is bypassing of DMC configuration
uint32_t	reserved11:15 (bitfield)	Reserved

cacheDis

Cache Disable.

By default the cache is enabled by boot software during preboot. Setting this bit will prevent the boot software from enabling the cache which will result in slower boot performance but allows for the memory regions that are used for cache space to be bootable memory regions for larger application images.

decryptOnlyEn

Decrypt Only Enable.

By default secure boot images that are also encrypted are decrypted then authenticated before the application is executed by the core. Authentication is a time consuming task and in situations where boot time is critical and the security of the device is less important, bypassing the authentication stage is possible by setting this bit. This will compromise security and is not recommended for use without additional security measures being implemented to verify the integrity of the loader software.

cacheDisInv

Cache Disable Invalidate.

Invalidates the Cache Disable bit to allow the boot rom to enable the cache again during preboot

decryptOnlyInv

Decrypt Only Invalidate.

Invalidates the Decrypt Only bit to allow the boot rom to perform full decrypt and authentication of secure boot streams

pubkey0Inv

Invalidate Public Key 0, use next public key for secure boot

pubkey1Inv

Invalidate Public Key 1, Secure boot will no longer be operational as no further public keys

privkey0Inv

Invalidate Decryption Key 0, use next Decryption key for secure boot

privkey1Inv

Invalidate Decryption Key 1, use next Decryption key for secure boot

privkey2Inv

Invalidate Decryption Key 2, use next Decryption key for secure boot

privkey3Inv

Invalidate Decryption Key 3, once invalidated part will no longer be bootable

dmcEn

Enables configuration of the DMC from values in `OTP` stored in the format of the `ADI_ROM_OTP_DMC_CONFIG` object

dmcInv

Invalidates the DMC values in the `OTP` resulting is bypassing of DMC configuration

struct ADI_ROM_OTP_BOOT_CGU_INFO

Structure Type Declaration: ADI_ROM_OTP_BOOT_CGU_INFO

The CGU configuration object located in OTP for configuration of the CGU by the boot software.

This is a 64-bit structure that is allocated to one contiguous region in the OTP memory array. The functionality provided allows the boot software to configure the CGU to allow for a more efficient boot process.

Table 36-59: ADI_ROM_OTP_BOOT_CGU_INFO Members

Type	Name	Description
uint32_t	ctl_WEN:1 (bitfield)	Enable write to the CGU_CTL register
uint32_t	div_WEN:1 (bitfield)	Enable write to the CGU_DIV register
uint32_t	reserved0:1 (bitfield)	Reserved
uint32_t	div_DSEL:5 (bitfield)	CGU_DIV.DSEL value
uint32_t	div_CSEL:5 (bitfield)	CGU_DIV.CSEL value
uint32_t	div_S0SEL:3 (bitfield)	CGU_DIV.S0SEL value
uint32_t	div_SYSSEL:5 (bitfield)	CGU_DIV.SYSSEL value
uint32_t	div_S1SEL:3 (bitfield)	CGU_DIV.S1SEL value
uint32_t	div_OSEL:7 (bitfield)	CGU_DIV.OSEL value
uint32_t	ctl_DF:1 (bitfield)	CGU_CTL.DF value
uint32_t	ctl_MSEL:7 (bitfield)	CGU_CTL.MSEL value
uint32_t	auto_disable:1 (bitfield)	disable polling on auto-alignment of clocks, NOT RECOMMENDED!
uint32_t	clkoutsel_USBCLKSEL:6 (bitfield)	value
uint32_t	clkoutsel_CLKOUTSEL:5 (bitfield)	CGU_CLKOUTSEL.CLKOUTSEL value
uint32_t	clkoutsel_WEN:1 (bit-field)	Enable write to the CGU_CLKOUTSEL register
uint32_t	reserved1:12 (bitfield)	Reserved

ctl_WEN

Enable write to the CGU_CTL register

div_WEN

Enable write to the CGU_DIV register

div_DSEL

CGU_DIV.DSEL value

div_CSEL

CGU_DIV.CSEL value

div_S0SEL

CGU_DIV.S0SEL value

div_SYSSEL

CGU_DIV.SYSSEL value

div_S1SEL

CGU_DIV.S1SEL value

div_OSEL

CGU_DIV.OSEL value

ctl_DF

CGU_CTL.DF value

ctl_MSEL

CGU_CTL.MSEL value

auto_disable

disable polling on auto-alignment of clocks, NOT RECOMMENDED!

clkoutsel_USBCLKSEL

value

clkoutsel_CLKOUTSEL

CGU_CLKOUTSEL.CLKOUTSEL value

clkoutsel_WEN

Enable write to the CGU_CLKOUTSEL register

struct ADI_ROM_OTP_BOOT_CMD_INFO

Structure Type Declaration: ADI_ROM_OTP_BOOT_CMD_INFO

The boot command object for storing a customized boot command for each boot mode within the OTP memory array.

This is a 96-bit structure that is allocated to one contiguous region in the OTP memory array. The functionality provided allows the boot rom software to pass a custom boot command to a specific boot mode changing the default boot behavior on startup. This can be used for example to change the default UART instance used for a UART boot operation.

Table 36-60: ADI_ROM_OTP_BOOT_CMD_INFO Members

Type	Name	Description
uint32_t	spiMasterBootCmd	SPI Master boot command
uint32_t	spiSlaveBootCmd	SPI Slave boot command
uint32_t	uartBootCmd	Uart Slave boot command

spiMasterBootCmd

SPI Master boot command

spiSlaveBootCmd

SPI Slave boot command

uartBootCmd

Uart Slave boot command

struct ADI_ROM_OTP_BOOT_INFO

Structure Type Declaration: ADI_ROM_OTP_BOOT_INFO

The 512-bit boot info object located in OTP for boot customization.

This is a 512 bit structure that is allocated to one contiguous region in the OTP memory array. The contented in OTP is stored in the format of this structure allowing boot to read the contents directly into this object.

Users can read this object using the adi_rom_otp_get() routine supplying the OTPCMD::otpcmd_boot_info enumeration

Table 36-61: ADI_ROM_OTP_BOOT_INFO Members

Type	Name	Description
ADI_ROM_OTP_BOOT_CGU_INFO	cgu	CGU Configuration information
ADI_ROM_OTP_BOOT_CMD_INFO	bcmd	Boot Command customization for each boot mode allowing for a change of default boot peripheral instance and configuration
ADI_ROM_OTP_BOOT_CFG	bcfg	Additional boot configuration flags for key invalidation and for enabling DMC configuration
uint32_t	reserved0	Reserved
uint32_t	reserved1	Reserved
uint32_t	reserved2	Reserved
uint32_t	reserved3	Reserved
uint16_t	reserved4	Reserved

Table 36-61: ADI_ROM_OTP_BOOT_INFO Members (Continued)

Type	Name	Description
`uint16_t`	`otpReadTiming`	Allows for the boot process to reconfigure the `OTP` Read timing from the default.

cgu

CGU Configuration information

bcmd

Boot Command customization for each boot mode allowing for a change of default boot peripheral instance and configuration

bcfg

Additional boot configuration flags for key invalidation and for enabling DMC configuration

otpReadTiming

Allows for the boot process to reconfigure the `OTP` Read timing from the default.

The boot process supports overriding the default read timing parameter for the OTPC for read accesses. This is provided in the event read timing is required after reconfiguration of the CGU .

WARNING: For this product this field should never be set to anything other than 0x0000

struct ADI_ROM_OTP_DMC_CONFIG

Structure Type Declaration: `ADI_ROM_OTP_DMC_CONFIG`

The 384-bit configuration object located in `OTP` for configuration of the DMC during preboot.

If the user wishes to make use of the memories connected to the DMC peripheral during boot without the use of init codes in the boot stream then the settings can be applied to this object in the `OTP` memory. During preboot the boot software reads the object from `OTP` and will configure the peripheral accordingly.

NOTE: When the device is open it advisable to avoid the use of `OTP` and instead use initcodes for DMC initialization as the initcode method is highly customizable. If users wish to lock the device to enable secure boot then in order to boot to memories interfaced to the DMC then the configuration must be provisioned to `OTP` as initcodes are not supported in secure boot.

Table 36-62: ADI_ROM_OTP_DMC_CONFIG Members

Type	Name	Description
	`reserved0:10` (bitfield)	Reserved

Table 36-62: ADI_ROM_OTP_DMC_CONFIG Members (Continued)

Type	Name	Description
uint32_t		
uint32_t	ulDDR_DLLCTL:12 (bit-field)	Contents of DMC_DLLCTL [11:0]
uint32_t	ulDDR_EMR2:8 (bitfield)	Contents of DMC_EMR2 [7:0]
uint32_t	reserved1:2 (bitfield)	Reserved
uint32_t	ulDDR_CFGCTL	Packed content of DMC_CTL Register, DMC_CFG registers.
uint32_t	ulDDR_MREMR1	Packed content of DMC_EMR1 Register, DMC_MR registers.
uint32_t	ulDDR_TR0	Content of DMC_TR0
uint32_t	ulDDR_TR1	Content of DMC_TR1
uint32_t	ulDDR_TR2	Content of DMC_TR2
uint32_t	ulDDR_PHYCTL0	Content of DMC_PHY_CTL0
uint32_t	ulDDR_PHYCTL145	Packed content of DMC_PHY_CTL1 , DMC_PHY_CTL4 and DMC_PHY_CTL5 registers.
uint32_t	ulDDR_PHYCTL2	Content of DMC_PHY_CTL2
uint32_t	ulDDR_PHYCTL3	Content of DMC_PHY_CTL3
uint32_t	ulDDR_CAL_PADCTL0_PHY_STAT3_0	Packed content of DMC_CAL_PADCTL0 , DMC_PHY_STAT0 , and DMC_PHY_STAT3 registers.
uint32_t	ulDDR_CAL_PADCTL2	Content of DMC_CAL_PADCTL2

ulDDR_DLLCTL

Contents of DMC_DLLCTL [11:0]

ulDDR_EMR2

Contents of DMC_EMR2 [7:0]

ulDDR_CFGCTL

Packed content of DMC_CTL Register, DMC_CFG registers.

The contents are packed as follows:

ADI_ROM_OTP_DMC_CONFIG::ulDDR_CFGCTL	Register
[15:0]	DMC_CFG [15:0]
[31:16]	DMC_CTL [15:0]

ulDDR_MREMR1

Packed content of DMC_EMR1 Register, DMC_MR registers.

The contents are packed as follows:

ADI_ROM_OTP_DMC_CONFIG::ulDDR_MREMR1	Register
[15:0]	DMC_EMR1 [15:0]
[31:16]	DMC_MR [15:0]

ulDDR_TR0

Content of DMC_TR0

ulDDR_TR1

Content of DMC_TR1

ulDDR_TR2

Content of DMC_TR2

ulDDR_PHYCTL0

Content of DMC_PHY_CTL0

ulDDR_PHYCTL145

Packed content of DMC_PHY_CTL1 , DMC_PHY_CTL4 and DMC_PHY_CTL5 registers.

The contents are packed as follows:

ADI_ROM_OTP_DMC_CONFIG::ulDDR_PHYCTL145	Register
[7:0]	DMC_PHY_CTL5[7:0]
[15:8]	DMC_PHY_CTL4[7:0]
[31:16]	DMC_PHY_CTL1[31:16]

ulDDR_PHYCTL2

Content of DMC_PHY_CTL2

ulDDR_PHYCTL3

Content of DMC_PHY_CTL3

ulDDR_CAL_PADCTL0_PHY_STAT3_0

Packed content of DMC_CAL_PADCTL0 , DMC_PHY_STAT0 , and DMC_PHY_STAT3 registers.

The contents are packed as follows:

ADI_ROM_OTP_DMC_CONFIG::ulDDR_CAL_PADCTL0_PHY_STAT3_0	Register
[8:0]	DMC_PHY_STAT0[8:0]
[11:9]	DMC_PHY_STAT3[2:0]
[31:12]	DMC_CAL_PADCTL0[31:12]

ulDDR_CAL_PADCTL2

Content of DMC_CAL_PADCTL2

struct ADI_ROM_SYSCTRL

Structure Type Declaration: ADI_ROM_SYSCTRL

The object for use with the adi_rom_sysControl() routine for storage of CGU , DMC and wakeup action flags.

The configuration information for the CGU and DMC are stored in this object. Through the use of the adi_rom_sysControl() function users can request that the structure be populated automatically from the current CGU and DMC peripheral settings or users may initialize the structure directly with required values. Through an additional call to the adi_rom_sysControl() function the user can then request write a write of the structure to the DPM_RESTORE[n] registers allowing the context to be restored for wakeup from hibernate events.

The storage requirements for the DMC result in a need to compress the data to be saved and restored from some MMRs so data from different registers may be located in a single DPM_RESTORE[n] register.

Table 36-63: ADI_ROM_SYSCTRL Members

Type	Name	Description
uint32_t	ulCGU_STAT	Content of CGU_STAT
uint32_t	ulCGU_CLKOUTSEL	Content of CGU_CLKOUTSEL
uint32_t	ulWUA_Flags	Flags to enable various features during boot in wakeup from hibernate events. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE0 .
uint32_t	ulWUA_BootAddr	Memory boot address when a wakeup action requesting execution from memory is detected. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE1 .
uint32_t	ulCGU_DIV	Content of CGU_DIV . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE2 .

Table 36-63: ADI_ROM_SYSCTRL Members (Continued)

Type	Name	Description
uint32_t	ulWUA_User	User variable or use to store RCU_BCODE for restoration if required in the wakeup actions. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE4 .
uint32_t	ulDDR_EMR2DLL_CGU_CTL	Packed content of CGU_CTL Register, DMC_DLLCTL and DMC_EMR2 registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE3 .
uint32_t	ulDDR_CFGCTL	Packed content of DMC_CTL Register, DMC_CFG registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE5 .
uint32_t	ulDDR_MREMR1	Packed content of DMC_EMR1 Register, DMC_MR registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE6 .
uint32_t	ulDDR_TR0	Content of DMC_TR0 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE7 .
uint32_t	ulDDR_TR1	Content of DMC_TR1 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE8 .
uint32_t	ulDDR_TR2	Content of DMC_TR2 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE9 .
uint32_t	ulDDR_PHYCTL0	Content of DMC_PHY_CTL0 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE10 .
uint32_t	ulDDR_PHYCTL145	Packed content of DMC_PHY_CTL1 , DMC_PHY_CTL4 and DMC_PHY_CTL5 registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE11 .
uint32_t	ulDDR_PHYCTL2	Content of DMC_PHY_CTL2 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE12 .
uint32_t	ulDDR_PHYCTL3	Content of DMC_PHY_CTL3
uint32_t	ulDDR_CAL_PADCTL0_PHY_STAT3_0	Packed content of DMC_CAL_PADCTL0 , DMC_PHY_STAT0 , DMC_PHY_STAT3 and DMC_PHY_STAT4 registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE14 .
uint32_t	ulDDR_CAL_PADCTL2	Content of DMC_CAL_PADCTL2 if ROM_WUA_DMC_PADRESTORE is disabled. Content of DMC_PHY_STAT5 if ROM_WUA_DMC_PADRESTORE is enabled. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE15 .

ulCGU_STAT

Content of CGU_STAT

ulCGU_CLKOUTSEL

Content of CGU_CLKOUTSEL

ulWUA_Flags

Flags to enable various features during boot in wakeup from hibernate events. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE0 .

When the processor wakes up from hibernate the boot code is able to determine that the boot request is specifically due to a wakeup from hibernate event occurring. In such booting situations the user has the ability to perform wakeup actions during boot to customize the wakeup process. The wakeup actions to be performed are defined by the following flags

Bit	Flag Name	Description
0	ROM_WUA_EN	Must be set in order for any of the wakeup actions to be performed
1	ROM_WUA_MEMBOOT	When set enables bypassing of the boot kernel upon wakeup event and starts executing from the address stored in ADI_ROM_SYSCTRL::ulWUA_BootAddr
2	ROM_WUA_BCODE	Allows for restoration of the RCU_BCODE register upon boot startup
3	Reserved	Reserved
4	ROM_WUA_CGU	When set enable restoration of the CGU configuration
5	ROM_WUA_DMC	When set enable restoration of the DMC configuration to bring DDR Memory out of self-refresh
6	Reserved	Reserved
7	ROM_WUA_DMC_LOCK	When set applies an extended delay to allow for more headroom in the DMC DLL locking time
8	ROM_WUA_DMC_PADRESTORE	When set allows for restoration of the pad calibration
9	ROM_WUA_SCLK0BF0DIS	When set will result in the setting of CGU_SCBF_DIS.SCLK0BF at boot startup
10	ROM_WUA_SCLK1BF0DIS	When set will result in the setting of CGU_SCBF_DIS.SCLK1BF at boot startup
11	ROM_WUA_DCLKBF0DIS	When set will result in the setting of CGU_SCBF_DIS.DCLKBF at boot startup
12	ROM_WUA_OUTCLKBF0DIS	When set will result in the setting of CGU_SCBF_DIS.OUTCLKBF at boot startup
[23:13]	Reserved	Reserved

Bit	Flag Name	Description
[31:24]	ROM_WUA_CHKHDR	Must be set to the value 0xAD in order for wakeup actions to be performed

NOTE: Disabling of some of the clocks may actually result in disabling of some of the items required for a particular boot mode there the user must ensure that they do not request for a clock to be disabled that result in the booting peripheral being disabled or a component that the booting peripheral is reliant upon being disabled

ulWUA_BootAddr

Memory boot address when a wakeup action requesting execution from memory is detected. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE1 .

ulCGU_DIV

Content of CGU_DIV . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE2 .

ulWUA_User

User variable or use to store RCU_BCODE for restoration if required in the wakeup actions. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE4 .

ulDDR_EMR2DLL_CGU_CTL

Packed content of CGU_CTL Register, DMC_DLLCTL and DMC_EMR2 registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE3 .

The contents are packed as follows:

ADI_ROM_SYSCTRL::ulDDR_EMR2DLL_CGU_CTL	Register
[0]	CGU_CTL.DF
[7:1]	CGU_CTL.MSEL
[21:10]	DMC_DLLCTL [11:0]
[29:22]	DMC_EMR2 [7:0]

ulDDR_CFGCTL

Packed content of DMC_CTL Register, DMC_CFG registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE5 .

The contents are packed as follows:

ADI_ROM_SYSCTRL::ulDDR_CFGCTL	Register
[15:0]	DMC_CFG [15:0]
[31:16]	DMC_CTL [15:0]

ulDDR_MREMR1

Packed content of DMC_EMR1 Register, DMC_MR registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE6 .

The contents are packed as follows:

ADI_ROM_SYSCTRL::ulDDR_MREMR1	Register
[15:0]	DMC_EMR1 [15:0]
[31:16]	DMC_MR [15:0]

ulDDR_TR0

Content of DMC_TR0 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE7 .

ulDDR_TR1

Content of DMC_TR1 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE8 .

ulDDR_TR2

Content of DMC_TR2 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE9 .

ulDDR_PHYCTL0

Content of DMC_PHY_CTL0 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE10 .

ulDDR_PHYCTL145

Packed content of DMC_PHY_CTL1 , DMC_PHY_CTL4 and DMC_PHY_CTL5 registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE11 .

The contents are packed as follows:

ADI_ROM_SYSCTRL::ulDDR_PHYCTL145	Register
[7:0]	DMC_PHY_CTL5[7:0]
[15:8]	DMC_PHY_CTL4[7:0]
[31:16]	DMC_PHY_CTL1[31:16]

ulDDR_PHYCTL2

Content of DMC_PHY_CTL2 . When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE12 .

ulDDR_PHYCTL3

Content of DMC_PHY_CTL3

ulDDR_CAL_PADCTL0_PHY_STAT3_0

Packed content of DMC_CAL_PADCTL0 , DMC_PHY_STAT0 , DMC_PHY_STAT3 and DMC_PHY_STAT4 registers. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE14 .

The contents are packed in two different manners depending on the setting of wakeup action item ROM_WUA_DMC_PADRESTORE:

ROM_WUA_DMC_PADRESTORE	ADI_ROM_SYSCTRL::ulDDR_CAL_PADCTL0_PHY_STAT3_0	Register
0	[8:0]	DMC_PHY_STAT0[8:0]
0	[11:9]	DMC_PHY_STAT3[2:0]
0	[31:12]	DMC_CAL_PADCTL0[31:12]
1	[8:0]	DMC_PHY_STAT0[8:0]
1	[11:9]	DMC_PHY_STAT3[2:0]
1	[31:12]	DMC_PHY_STAT4[31:12]

ulDDR_CAL_PADCTL2

Content of DMC_CAL_PADCTL2 if ROM_WUA_DMC_PADRESTORE is disabled. Content of DMC_PHY_STAT5 if ROM_WUA_DMC_PADRESTORE is enabled. When saving or restoring content from the DPM_RESTORE[n] registers, the contents of this parameter are located in DPM_RESTORE15 .

struct ROM_BOOTMODES_TYPE

Structure Type Declaration: ROM_BOOTMODES_TYPE

The complete boot modes lookup table.

Contains a ROM_BOOTMODE_TYPE structure for the default supported boot mode configuration used by the boot software. Users may gain access to the parameters used by the boot software via the adi_rom_GetAddress() function allowing users to use the default values when wishing to test booting via the API

Table 36-64: ROM_BOOTMODES_TYPE Members

Type	Name	Description
ROM_BOOTMODE_TYPE[ROM_BK_NUM_BOOT_ MODES]	bmode	Default boot setting for every boot mode

bmode

Default boot setting for every boot mode

struct ROM_BOOTMODE_TYPE

Structure Type Declaration: ROM_BOOTMODE_TYPE

Boot Mode Lookup table structure.

Contains the default settings of the boot mode for arguments used with the adi_rom_Boot() routine

Table 36-65: ROM_BOOTMODE_TYPE Members

Type	Name	Description
void *	pAddress	Boot Start Address
uint32_t	flags	Boot Flags, refer to adi_rom_Boot() for supported flag details
ROM_BOOT_HOOK_FUNC *	pHook	Hook Routine
uint32_t	command	The boot mode specific command parameter that specifies the peripheral instance and configuration to use

pAddress

Boot Start Address

flags

Boot Flags, refer to adi_rom_Boot() for supported flag details

pHook

Hook Routine

command

The boot mode specific command parameter that specifies the peripheral instance and configuration to use

struct ROM_BOOT_DMA_INSTANCE

Structure Type Declaration: `ROM_BOOT_DMA_INSTANCE`

DMA Channel Instance.

Specifies the base MMR address of the DMA channel as well as trigger and interrupt IDs

Table 36-66: ROM_BOOT_DMA_INSTANCE Members

Type	Name	Description
`ADI_DMA_TypeDef *`	`pReg`	Pointer to the base address of the DMA channel MMR registers
`DMA_CHANn_TypeDef`	`eDmaChannelId`	The actual DMA channel ID in the system
`uint8_t`	`TriggerId`	The trigger ID associated with the DMA channel
`uint8_t`	`InterruptId`	The interrupt ID associated with the DMA channel

pReg

Pointer to the base address of the DMA channel MMR registers

eDmaChannelId

The actual DMA channel ID in the system

TriggerId

The trigger ID associated with the DMA channel

InterruptId

The interrupt ID associated with the DMA channel

struct ROM_BOOT_MDMA

Structure Type Declaration: `ROM_BOOT_MDMA`

MDMA Channels.

Provides access to all the MDMA channels and CRC peripherals supported by the processor

Table 36-67: ROM_BOOT_MDMA Members

Type	Name	Description
`ROM_BOOT_MDMA_REGS[ PARAM_SYS0_NUM_MDMA_ STREAMS]`	`Stream`	Array of MDMA channel configurations supported by the processor

Stream

Array of MDMA channel configurations supported by the processor

struct ROM_BOOT_MDMA_REGS

Structure Type Declaration: `ROM_BOOT_MDMA_REGS`

MDMA Channel Registers.

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.

Table 36-68: ROM_BOOT_MDMA_REGS Members

Type	Name	Description
`ROM_BOOT_DMA_INSTANCE`	`Src`	The source DMA Channel in the MDMA pair
`ROM_BOOT_DMA_INSTANCE`	`Dst`	The destination DMA Channel in the MDMA pair
`ADI_CRC_TypeDef *`	`pCrc`	The base MMR address of the associated CRC peripheral if one exists
`ROM_BOOT_MDMA_CRC_SUPPORT`	`eCrcSupport`	Indicates if the MDMA channel supports CRC or not

Src

The source DMA Channel in the MDMA pair

Dst

The destination DMA Channel in the MDMA pair

pCrc

The base MMR address of the associated CRC peripheral if one exists

eCrcSupport

Indicates if the MDMA channel supports CRC or not

struct ROM_DMA_MDMA_CONFIG

Structure Type Declaration: `ROM_DMA_MDMA_CONFIG`

MDMA Configuration Object.

The user configurable structure for controlling the MDMA operation to be supplied to the `adi_rom_MemDma()` routine.

Table 36-69: ROM_DMA_MDMA_CONFIG Members

Type	Name	Description
	`eOperation`	Type of operation to perform

Table 36-69: ROM_DMA_MDMA_CONFIG Members (Continued)

Type	Name	Description
ROM_DMA_MDMA_OPERATION		
ROM_DMA_MDMA_ID	eId	MDMA Channel ID
void *	pSource	Source Pointer
void *	pDestination	Destination Pointer
uint32_t	ByteCount	Byte Count
ROM_DMA_DONE_DETECT_METHOD	eDoneDetect	DMA Done Detection Method
uint32_t	CrcCtl	CRC_CTL value when CRC operations are required
uint32_t	FillVal	Fill value for memory fill operations
uint32_t	CrcPoly	CRC Polynomial for CRC operations
uint32_t	CrcCompare	Value used for CRC compare operations or for a CRC32 result compare

eOperation

Type of operation to perform

eId

MDMA Channel ID

pSource

Source Pointer

pDestination

Destination Pointer

ByteCount

Byte Count

eDoneDetect

DMA Done Detection Method

CrcCtl

CRC_CTL value when CRC operations are required

FillVal

Fill value for memory fill operations

CrcPoly

CRC Polynomial for CRC operations

CrcCompare

Value used for CRC compare operations or for a CRC32 result compare

struct ROM_DMA_PDMA_CONFIG

Structure Type Declaration: `ROM_DMA_PDMA_CONFIG`

PDMA Configuration Object.

The user configurable structure for controlling the PDMA operation via the `adi_rom_PeriphDma()` function.

Table 36-70: ROM_DMA_PDMA_CONFIG Members

Type	Name	Description
`ROM_DMA_PDMA_OPERATION`	`eOperation`	Type of operation to perform
`ADI_DMA_TypeDef *`	`pRegs`	Pointer to the base address of the DMA channel MMR registers
`uint16_t`	`dataWidth`	The maximum supported data width of the DMA channel. Used to configure the `DMA_CFG.PSIZE PSIZE` field in `DMA_CFG`
`uint16_t`	`dstModifyMult`	The modify multiplier to be applied, usually set to 1
`void *`	`pSource`	Source Pointer used for transmit operations
`void *`	`pDestination`	Destination Pointer used for receive operations
`uint32_t`	`byteCount`	Number of bytes to transfer
`ROM_DMA_DONE_DETECT_METHOD`	`eDoneDetect`	DMA Done Detection method to be used for the transfer

eOperation

Type of operation to perform

pRegs

Pointer to the base address of the DMA channel MMR registers

dataWidth

The maximum supported data width of the DMA channel. Used to configure the `DMA_CFG.PSIZE PSIZE` field in `DMA_CFG`

dstModifyMult

The modify multiplier to be applied, usually set to 1

pSource

Source Pointer used for transmit operations

pDestination

Destination Pointer used for receive operations

byteCount

Number of bytes to transfer

eDoneDetect

DMA Done Detection method to be used for the transfer

struct ROM_ECDSA_DOMAIN

Structure Type Declaration: `ROM_ECDSA_DOMAIN`

NIST P-224 Parameter Curves.

Table 36-71: ROM_ECDSA_DOMAIN Members

Type	Name	Description
`uint8_t[28]`	`p`	p. parameters
`uint8_t[28]`	`a`	a. parameters
`uint8_t[28]`	`b`	b. parameters
`uint8_t[28]`	`n`	n. parameters
`uint8_t[28]`	`Gx`	Gx. parameters
`uint8_t[28]`	`Gy`	Gy. parameters

p

p. parameters

a

a. parameters

b

b. parameters

n

n. parameters

Gx

Gx. parameters

Gy

Gy. parameters

struct ROM_SPI_LUTENTRY

Structure Type Declaration: `ROM_SPI_LUTENTRY`

SPI Master Boot Default auto-detection settings for a given BCODE value used during auto-detection.

This structure is used to initialize the SPI configuration for given BCODE when auto-detection is used. BCODE values of 0 through 16 are available and boot supports 1 through 15. 0 and 16 cannot be used but an entry is provided. An array containing 16 of these items can be generated to provide a full lookup table in the ROM of default SPI configuration settings to be used for each of the boot codes. The first nibble of the boot stream is used as the index into the array to retrieve the required configuration.

Table 36-72: ROM_SPI_LUTENTRY Members

Type	Name	Description
`uint8_t`	`ubDummyBytes`	The number of Dummy bytes to be issued after the command and address sequence and before data can be received
`uint8_t`	`ubReadCommand`	The read command to be used to read the boot data from the flash
`uint8_t`	`ubDataBits`	The bus width to be used when reading the data. 0 for single bit, 1 for dual, 2 for quad
`uint8_t`	`ubAddressBytes`	The number of address bytes to send after the read command
`uint16_t`	`uwClkLower`	The SPI clock divider to be used
`uint16_t`	`reserved0`	Reserved
`uint32_t`	`nTxCtl`	The value to be written to the `SPI_TXCTL` register that is used for the address transmit operations such as address cycles
`uint32_t`	`nRxCtl`	The value to be written to the `SPI_RXCTL` register that is used for all receive operations
`uint32_t`	`nCmdCtl`	The value to be written to the `SPI_TXCTL` register that is used for sending the read command to the SPI Flash
`ROM_BOOT_SPIM_IO_ENABLE_FUNC *`	`pMIOEnFunction`	Pointer to the function used to enable quad mode on the SPI flash
`uint8_t`	`nDummy`	The Dummy Byte value to be used if dummy byte transfers are required and the bus is not tri-stated

ubDummyBytes

The number of Dummy bytes to be issued after the command and address sequence and before data can be received

ubReadCommand

The read command to be used to read the boot data from the flash

ubDataBits

The bus width to be used when reading the data. 0 for single bit, 1 for dual, 2 for quad

ubAddressBytes

The number of address bytes to send after the read command

uwClkLower

The SPI clock divider to be used

nTxCtl

The value to be written to the `SPI_TXCTL` register that is used for the address transmit operations such as address cycles

nRxCtl

The value to be written to the `SPI_RXCTL` register that is used for all receive operations

nCmdCtl

The value to be written to the `SPI_TXCTL` register that is used for sending the read command to the SPI Flash

pMIOEnFunction

Pointer to the function used to enable quad mode on the SPI flash

nDummy

The Dummy Byte value to be used if dummy byte transfers are required and the bus is not tri-stated

struct otp_data

Structure Type Declaration: `otp_data`

Container for accessing data to be written to `OTP` via the `adi_rom_otp_pgm()` routine.

Table 36-73: otp_data Members

Type	Name	Description
`uint32_t(*) [ROM_OTP_SZ_huk]`	`huk`	256-bit Hardware Unique Key
`uint32_t(*) [ROM_OTP_SZ_gp0]`	`gp0`	2624-bit General purpose user space
`uint32_t(*) [ROM_OTP_SZ_pvt_128key0]`	`pvt_128key0`	128-bit AES Key
`uint32_t(*) [ROM_OTP_SZ_pvt_128key1]`	`pvt_128key1`	128-bit AES Key

Table 36-73: otp_data Members (Continued)

Type	Name	Description
uint32_t(*) [ROM_OTP_SZ_pvt_128key2]	pvt_128key2	128-bit AES Key
uint32_t(*) [ROM_OTP_SZ_pvt_128key3]	pvt_128key3	128-bit AES Key
uint32_t(*) [ROM_OTP_SZ_ek]	ek	256-bit endorsement key
uint32_t(*) [ROM_OTP_SZ_secure_emu_key]	secure_emu_key	128-bit Secure Debug Key
uint32_t(*) [ROM_OTP_SZ_public_key0]	public_key0	512-bit public key used for boot stream authentication
uint32_t(*) [ROM_OTP_SZ_public_key1]	public_key1	512-bit public key used for boot stream authentication
uint32_t(*) [ROM_OTP_SZ_boot_info]	boot_info	512-bit boot customization structure, see also ADI_ROM_OTP_BOOT_INFO
uint8_t	antiroll_nv_cntr	512-bit anti-rollback counter to prevent loading of older firmware during secure boot
uint32_t(*) [ROM_OTP_SZ_gp1]	gp1	512-bit General purpose user space
uint32_t	bootModeDisable:8 (bit-field)	Boot mode disable for permanently disabling specific boot modes
uint32_t(*) [ROM_OTP_SZ_preboot_ddr_cfg]	preboot_ddr_cfg	384-bit DMC Configuration. See also ADI_ROM_OTP_DMC_CONFIG
uint32_t(*) [ROM_OTP_SZ_stageID]	stageID	64-bit staging ID

huk

256-bit Hardware Unique Key

gp0

2624-bit General purpose user space

pvt_128key0

128-bit AES Key

pvt_128key1

128-bit AES Key

pvt_128key2

128-bit AES Key

pvt_128key3

128-bit AES Key

ek

256-bit endorsement key

secure_emu_key

128-bit Secure Debug Key

public_key0

512-bit public key used for boot stream authentication

public_key1

512-bit public key used for boot stream authentication

boot_info

512-bit boot customization structure, see also `ADI_ROM_OTP_BOOT_INFO`

antiroll_nv_cntr

512-bit anti-rollback counter to prevent loading of older firmware during secure boot

gp1

512-bit General purpose user space

bootModeDisable

Boot mode disable for permanently disabling specific boot modes

preboot_ddr_cfg

384-bit DMC Configuration. See also `ADI_ROM_OTP_DMC_CONFIG`

stageID

64-bit staging ID

Enumerations

The programming model for booting the processor uses the enumerations defined in this section.

enum ADI_ROM_BOOT_BUFFER_STATE

Enumeration Type Declaration: `ADI_ROM_BOOT_BUFFER_STATE`

Indicates the state of the two buffers used when page mode is enabled.

Page mode is a feature of the boot kernel that allows boot stream data to be fetched in 1024 bytes at a time in a single load request by the kernel. The data is loaded to an internal buffer then processed by the boot kernel from this internal memory. In order to improve boot performance a second buffer is also used and is filled by using a non-blocking DMA. This allows the boot kernel to perform operation on one block of boot data while simultaneously fetching the next block of data from the boot source.

This item indicates which buffer is the currently active buffer that the boot kernel is processing the boot data from of if the buffers are currently empty indicating all data has been processed or no data has been loaded yet

Table 36-74: ADI_ROM_BOOT_BUFFER_STATE Members

Enumerator	Description
`ADI_ROM_BOOT_BUFFERS_EMPTY`	Both the buffers are empty
`ADI_ROM_BOOT_BUFFER0_ACTIVE`	Buffer 0 is the currently active buffer in which the boot kernel is processing the boot stream from
`ADI_ROM_BOOT_BUFFER1_ACTIVE`	Buffer 1 is the currently active buffer in which the boot kernel is processing the boot stream from

ADI_ROM_BOOT_BUFFERS_EMPTY

Both the buffers are empty

ADI_ROM_BOOT_BUFFER0_ACTIVE

Buffer 0 is the currently active buffer in which the boot kernel is processing the boot stream from

ADI_ROM_BOOT_BUFFER1_ACTIVE

Buffer 1 is the currently active buffer in which the boot kernel is processing the boot stream from

enum ADI_ROM_BOOT_KEY_TYPE

Enumeration Type Declaration: `ADI_ROM_BOOT_KEY_TYPE`

Indicates if custom security keys are to be used for evaluation of secure boot.

By default the boot process will fetch all security keys from `OTP` for use during secure boot. Enabling custom security allows for the user to set their own keys in the `ADI_ROM_BOOT_CONFIG` item and not have them taken from `OTP`. Allowing evaluation of secure boot through the use of the `adi_rom_Boot()` function without provisioning keys in `OTP`.

Table 36-75: ADI_ROM_BOOT_KEY_TYPE Members

Enumerator	Description
ADI_ROM_CUSTOM_SECURITY	Enable use of custom security keys for authentication and decryption

ADI_ROM_CUSTOM_SECURITY

Enable use of custom security keys for authentication and decryption

enum ADI_ROM_BOOT_TYPE

Enumeration Type Declaration: ADI_ROM_BOOT_TYPE

Used to indicate to the boot kernel in an open processor if secure or non secure boot is required.

The boot kernel defaults to a secure boot unless the boot structure has been configured to indicate Non-Secure Boot

Table 36-76: ADI_ROM_BOOT_TYPE Members

Enumerator	Description
ADI_ROM_SECURE_BOOT_DIS	Non-Secure Boot
ADI_ROM_SECURE_BOOT	Secure Boot

ADI_ROM_SECURE_BOOT_DIS

Non-Secure Boot

ADI_ROM_SECURE_BOOT

Secure Boot

enum OTPCMD

Enumeration Type Declaration: OTPCMD

Comnmands required by the adi_rom_otp_get() routine to retrieve specific fields from the OTP memory.

Table 36-77: OTPCMD Members

Enumerator	Description
otpcmd_reserved0	Reserved
otpcmd_huk	Hardware Unique Key
otpcmd_gp0	General Purpose Block 0
otpcmd_pvt_128key0	Customer Private Key 0 128bits
otpcmd_pvt_128key1	Customer Private Key 1 128bits
otpcmd_pvt_128key2	Customer Private Key 2 128bits

Table 36-77: OTPCMD Members (Continued)

Enumerator	Description
otpcmd_pvt_128key3	Customer Private Key 3 128bits
otpcmd_ek	Endorsement Key
otpcmd_secure_emu_key	Secure Emulation Key
otpcmd_public_key0	Customer Public Key0
otpcmd_public_key1	Customer Public Key1
otpcmd_boot_info	Customer Programmable Boot Information
otpcmd_otpTiming	OTP Read timing override
otpcmd_antiroll_nv_cntr	AntiRollback NV Counter
otpcmd_gp1	General Purpose Block 1
otpcmd_bootModeDisable	Boot Mode Disable Bits
otpcmd_preboot_ddr_cfg	User Preboot DMC configuration
otpcmd_stageID	StageID
otpcmd_reserved1	Reserved

otpcmd_reserved0

Reserved

otpcmd_huk

Hardware Unique Key

otpcmd_gp0

General Purpose Block 0

otpcmd_pvt_128key0

Customer Private Key 0 128bits

otpcmd_pvt_128key1

Customer Private Key 1 128bits

otpcmd_pvt_128key2

Customer Private Key 2 128bits

otpcmd_pvt_128key3

Customer Private Key 3 128bits

otpcmd_ek

Endorsement Key

otpcmd_secure_emu_key

Secure Emulation Key

otpcmd_public_key0

Customer Public Key0

otpcmd_public_key1

Customer Public Key1

otpcmd_boot_info

Customer Programmable Boot Information

otpcmd_otpTiming

OTP Read timing override

otpcmd_antiroll_nv_cntr

AntiRollback NV Counter

otpcmd_gp1

General Purpose Block 1

otpcmd_bootModeDisable

Boot Mode Disable Bits

otpcmd_preboot_ddr_cfg

User Preboot DMC configuration

otpcmd_stageID

StageID

otpcmd_reserved1

Reserved

enum ROM_BOOT_MDMA_CRC_SUPPORT

Enumeration Type Declaration: `ROM_BOOT_MDMA_CRC_SUPPORT`

MDMA Channel CRC support.

Specifies whether the MDMA channel supports CRC operations or not

Table 36-78: ROM_BOOT_MDMA_CRC_SUPPORT Members

Enumerator	Description
ROM_BOOT_DMA_CRC_SUPPORTED	MDMA Channel supports CRC
ROM_BOOT_DMA_CRC_NOT_SUPPORTED	MDMA Channel does not support CRC

ROM_BOOT_DMA_CRC_SUPPORTED

MDMA Channel supports CRC

ROM_BOOT_DMA_CRC_NOT_SUPPORTED

MDMA Channel does not support CRC

enum ROM_BOOT_RESULT

Enumeration Type Declaration: ROM_BOOT_RESULT

Table 36-79: ROM_BOOT_RESULT Members

Enumerator	Description
ROM_BOOT_FAILURE	Failure.
ROM_BOOT_SUCCESS	Success.
ROM_BOOT_HDR_CHKSUM_ERR	Boot Stream Block Header Checksum Error.
ROM_BOOT_HDR_SIGN_ERR	Boot Stream Block Header Sign Failure.
ROM_BOOT_HDR_DEST_ERR	Boot Stream Block Payload Destination Error.
RESERVED0	Reserved.
RESERVED1	Reserved.
RESERVED2	Reserved.
RESERVED3	Reserved.
ROM_BOOT_CGU_WRITE_ERR	CGU Write Error
RESERVED4	Reserved.
ROM_BOOT_DMA_FAILURE	DMA Failure.
ROM_BOOT_DMA_ACTIVE	DMA Channel is Active
RESERVED5	Reserved details.
ROM_BOOT_MDMA_ID_ERR	Illegal MDMA Channel ID.
ROM_BOOT_MDMA_OPERATION_ERR	Illegal MDMA operation Specified.
RESERVED6	Reserved.
ROM_BOOT_MDMA_SRC_ERR	MDMA Source Channel Configuration Error.
ROM_BOOT_MDMA_DST_ERR	MDMA Destination Channel Configuration Error.

Table 36-79: ROM_BOOT_RESULT Members (Continued)

Enumerator	Description
RESERVED7	Reserved.
RESERVED8	Reserved.
RESERVED9	Reserved.
RESERVED10	Reserved.
ROM_BOOT_CRC_FAILURE	MDMA CRC32 Failure.
RESERVED11	Reserved.
ROM_BOOT_CRC_COUNT_ERR	CRC Byte Count was not a multiple 4.
ROM_BOOT_CRC_SUPPORTED_ERR	CRC Not Supported Error.
ROM_BOOT_CRC_INITCODE_ERR	CRC32 Enable Failure During Boot.
ROM_BOOT_CRC_CALLBACK_ERR	Error in Execution of the CRC Callback.
RESERVED12	Reserved.
RESERVED13	Reserved.
RESERVED14	Reserved.
RESERVED15	Reserved.
RESERVED16	Reserved.
RESERVED17	Reserved.
ROM_BOOT_SB_IMAGE_VERSION_ERR	Secure Boot Header Version Error.
ROM_BOOT_SB_IMAGE_TYPE_ERR	Secure Boot Header Type Error.
RESERVED18	Reserved.
ROM_BOOT_SB_CERT_COUNT_ERR	Secure Boot Header Certificate Count Error.
RESERVED19	Reserved.
RESERVED20	Reserved.
RESERVED21	Reserved.
ROM_BOOT_SB_KEY_UNWRAP_ERR	Decryption Key Unwrap Error.
RESERVED22	Reserved.
ROM_BOOT_ROLLBACK_ID_ERR	Secure Boot Anti-Rollback Protection Error.
ROM_BOOT_OTP_NVCNTR_READ_ERR	Non-Volatile Counter Read Error.
ROM_BOOT_OTP_NVCNTR_PGM_ERR	Non-Volatile Counter Program Error.
ROM_BOOT_WAKEUP_NO_CGU_INIT	Wakeup Actions CGU Programming Status.
ROM_BOOT_WAKEUP_NO_DMC_INIT	Wakeup Actions DMC Programming Status.
RESERVED23	Reserved.

Table 36-79: ROM_BOOT_RESULT Members (Continued)

Enumerator	Description
ROM_BMODE_ILLEGAL_DEVNUM	Illegal Device Enumeration Error.
ROM_BMODE_EXIT	Boot Stream Block Payload Destination Error.
ROM_BMODE_FAILURE	Boot Mode Error.
ROM_BMODE_SUCCESS	Boot Mode Specific Operation Success.
RESERVED24	Reserved.
RESERVED25	Reserved.

ROM_BOOT_FAILURE

Failure.

General failure can be used to indicate any general failure throughout the boot process

ROM_BOOT_SUCCESS

Success.

General success can be used to indicate any general functional success for an operation during the boot process.

NOTE: This must be the return result for a boot mode drivers initialization, configuration, load and cleanup routines when overriding their functionality in second stage boot loaders to use custom functions.

ROM_BOOT_HDR_CHKSUM_ERR

Boot Stream Block Header Checksum Error.

Indicates that the 8-bit XOR checksum of the 16-byte block header failed to generated the expected result.

ROM_BOOT_HDR_SIGN_ERR

Boot Stream Block Header Sign Failure.

The 0xAD block required as byte 4 of the boot block header was not found.

ROM_BOOT_HDR_DEST_ERR

Boot Stream Block Payload Destination Error.

The target address field of the block header indicates that the payload for the block is destined towards an address that is not supported. This would typically indicate an attempt to load data to the reserved 8KB region of memory reserved by the boot process as non-bootable.

ROM_BOOT_CGU_WRITE_ERR

CGU Write Error

Returned by the CGU Configuration routine if a CGU error is set during the initialization of the CGU from settings provisioned in the OTP.

ROM_BOOT_DMA_FAILURE

DMA Failure.

Returned by the DMA routines if an error was detected in the `DMA_STAT.IRQERR` prior to setting up a new DMA operation with the newly supplied configuration.

ROM_BOOT_DMA_ACTIVE

DMA Channel is Active

Returned only by the peripheral DMA routine when an attempt to run another peripheral DMA operation is attempted and the DMA channel is already running.

NOTE: This is not currently implemented for MDMA operations.

ROM_BOOT_MDMA_ID_ERR

Illegal MDMA Channel ID.

Returned by `adi_rom_MemDma()` if the MDMA channel ID is not supported. For supported channel IDs, please refer to `ROM_DMA_MDMA_ID`

ROM_BOOT_MDMA_OPERATION_ERR

Illegal MDMA operation Specified.

Returned by `adi_rom_MemDma()` if the MDMA operation to be performed is not supported. For supported operations, please refer to `ROM_DMA_MDMA_OPERATION`

ROM_BOOT_MDMA_SRC_ERR

MDMA Source Channel Configuration Error.

Set by the MDMA routines if after configuring the MDMA source channel to start a DMA operation, an error is generated in the source channels `DMA_STAT.IRQERR`

ROM_BOOT_MDMA_DST_ERR

MDMA Destination Channel Configuration Error.

Set by the MDMA routines if after configuring the MDMA source channel to start a DMA operation, an error is generated in the destination channels `DMA_STAT.IRQERR` .

ROM_BOOT_CRC_FAILURE

MDMA CRC32 Failure.

Returned by the higher level `adi_rom_MemDma()` routine and the underlying `adi_rom_MemCompare()` routine if the CRC32 result of the block of data did not match the expected result.

ROM_BOOT_CRC_COUNT_ERR

CRC Byte Count was not a multiple 4.

The CRC peripheral operates on 32-bit data only and as such all CRC operations must have a byte count that is a multiple of 4. This result is returned by the higher level `adi_rom_MemDma()` routine and the underlying `adi_rom_MemCompare()` and `adi_rom_MemFill()` routines if the byte count is not a multiple of 4 bytes.

ROM_BOOT_CRC_SUPPORTED_ERR

CRC Not Supported Error.

Returned by `adi_rom_MemDma()`, `adi_rom_MemFill()`, `adi_rom_MemCompare()` and `adi_rom_Crc32Poly()` if the supplied DMA configuration specified an MDMA channel that does not support CRC operations.

ROM_BOOT_CRC_INITCODE_ERR

CRC32 Enable Failure During Boot.

Returned by `adi_rom_Crc32Init()` if the boot process cannot enable the CRC32 functionality due to a `NULL` `ADI_ROM_BOOT_CONFIG` pointer or `NULL` `ADI_ROM_BOOT_HEADER` pointer located in `ADI_ROM_BOOT_CONFIG::pHeader`

ROM_BOOT_CRC_CALLBACK_ERR

Error in Execution of the CRC Callback.

Returned by the default CRC callback function located in the boot rom if any of the following conditions are met:

- The `ADI_ROM_BOOT_CONFIG` pointer passed to the callback is a `NULL` pointer
- The `ADI_ROM_BOOT_BUFFER` pointer pointing to the buffer to run CRC validation on is a `NULL` pointer
- The `ROM_CBFLAG_DIRECT` flag is not set in the supplied flags parameter indicating it was a direct callback
- The `ADI_ROM_BOOT_HEADER` pointer located in `ADI_ROM_BOOT_CONFIG::pHeader` is a `NULL` pointer

ROM_BOOT_SB_IMAGE_VERSION_ERR

Secure Boot Header Version Error.

Returned by the routine responsible for verifying the secure boot header if the version field is not the version supported by the processor. The version field is automatically set by the `signtool` utility.

ROM_BOOT_SB_IMAGE_TYPE_ERR

Secure Boot Header Type Error.

Returned by the routine responsible for verifying the secure boot header if the secure boot image type field is not one of the types supported by the processor. The type field is used to indicate if the image is a BLp, BLx, or BLw image secure boot image.

ROM_BOOT_SB_CERT_COUNT_ERR

Secure Boot Header Certificate Count Error.

Returned by the routine responsible for verifying the secure boot header if the number of certificate count is greater than 0 as the processor does not support the use of certificates in the secure boot implementation.

ROM_BOOT_SB_KEY_UNWRAP_ERR

Decryption Key Unwrap Error.

Indicates failure to unwrap the decryption key in BLw secure boot images.

ROM_BOOT_ROLLBACK_ID_ERR

Secure Boot Anti-Rollback Protection Error.

Indicates an attempt to boot a secure boot image with a lower firmware version than is currently stored in OTP or if the rollback ID read from OTP is out of bounds of the supported firmware update limit.

ROM_BOOT_OTP_NVCNTR_READ_ERR

Non-Volatile Counter Read Error.

A failure occurred in the reading of the non-volatile counter in OTP that contains the current Anti-Rollback protection value

ROM_BOOT_OTP_NVCNTR_PGM_ERR

Non-Volatile Counter Program Error.

A failure occurred in the programming of the new Anti-Rollback firmware version to the Non-Volatile Counter in OTP.

ROM_BOOT_WAKEUP_NO_CGU_INIT

Wakeup Actions CGU Programming Status.

Indicates that the routine responsible for programming the CGU during wakeup events did not perform any CGU configuration as there was no wakeup action request to do so.

ROM_BOOT_WAKEUP_NO_DMC_INIT

Wakeup Actions DMC Programming Status.

Indicates that the routine responsible for programming the DMC during wakeup events did not perform any DMC configuration as there was no wakeup action request to do so.

ROM_BMODE_ILLEGAL_DEVNUM

Illegal Device Enumeration Error.

Set when the boot process attempts to boot from a peripheral enumeration that is not supported or does not exist on the product. The peripheral enumeration is checked multiple times by all boot mode drivers in the ROM to ensure that peripheral instance to boot from is supported.

ROM_BMODE_EXIT

Boot Stream Block Payload Destination Error.

The target address field of the block header indicates that the payload for the block is destined towards an address that is not supported. This would typically indicate an attempt to load data to the reserved 8KB region of memory reserved by the boot process as non-bootable.

ROM_BMODE_FAILURE

Boot Mode Error.

A general error that can be returned by any of the boot mode drivers functions to indicate an error occurred. If an error occurs during the initialization, configuration or loading of boot data from the boot source then this error result may be used in the event a more concise error is is not available.

ROM_BMODE_SUCCESS

Boot Mode Specific Operation Success.

May be used by boot mode drivers as indication of a general successful operation.

NOTE: This must not be used as the successful return result for the boot modes initialization, configuration, load or cleanup function however which is required to return `ROM_BOOT_RESULT::ROM_BOOT_SUCCESS`

enum ROM_CORE_ID

Enumeration Type Declaration: `ROM_CORE_ID`

Core ID.

An enumeration for referencing a particular core

Table 36-80: ROM_CORE_ID Members

Enumerator	Description
`ROM_CORE_ID0`	Core 0
`ROM_CORE_NUM_CORES`	Number of Cores

ROM_CORE_ID0

Core 0

ROM_CORE_NUM_CORES

Number of Cores

enum ROM_DMA_DONE_DETECT_METHOD

Enumeration Type Declaration: `ROM_DMA_DONE_DETECT_METHOD`

DMA Done Detection Method.

Specifies the method to be used for detecting the completion of the requested DMA operation.

When a user requests a non-blocking DMA operation then separate software is required to check the status of the DMA channel. The boot rom does not provide an API for use for this operation.

NOTE: The trigger mode is not supported on this product

Table 36-81: ROM_DMA_DONE_DETECT_METHOD Members

Enumerator	Description
`ROM_DMA_DONE_NON_BLOCKING`	Return without waiting for the DMA to complete
`ROM_DMA_DONE_POLL_IRQDONE`	Poll on the IRQDONE bit in the DMA Status register
`ROM_DMA_DONE_WAKEUP_TRIGGER`	Configure a trigger to wakeup the core on completion

ROM_DMA_DONE_NON_BLOCKING

Return without waiting for the DMA to complete

ROM_DMA_DONE_POLL_IRQDONE

Poll on the IRQDONE bit in the DMA Status register

ROM_DMA_DONE_WAKEUP_TRIGGER

Configure a trigger to wakeup the core on completion

enum ROM_DMA_MDMA_ID

Enumeration Type Declaration: `ROM_DMA_MDMA_ID`

MDMA Channel ID.

The ID of the Memory DMA channel to be used. This item is used in the `ROM_DMA_MDMA_CONFIG` configuration to specify the Memory DMA channel to use for operations accessible via the `adi_rom_MemDma()` routine

Table 36-82: ROM_DMA_MDMA_ID Members

Enumerator	Description
`ROM_DMA_MDMA0`	Memory DMA Stream 0

Table 36-82: ROM_DMA_MDMA_ID Members (Continued)

Enumerator	Description
ROM_DMA_MDMA1	Memory DMA Stream 1
ROM_DMA_MDMA2	Memory DMA Stream 2
ROM_DMA_MEMDMA_END_COUNT	Number of Memory DMA Streams

ROM_DMA_MDMA0

Memory DMA Stream 0

ROM_DMA_MDMA1

Memory DMA Stream 1

ROM_DMA_MDMA2

Memory DMA Stream 2

ROM_DMA_MEMDMA_END_COUNT

Number of Memory DMA Streams

enum ROM_DMA_MDMA_OPERATION

Enumeration Type Declaration: ROM_DMA_MDMA_OPERATION

MDMA Operation to be performed.

The operation determines if only an MDMA is required to be configured, or whether a CRC operation must be used in conjunction with the MDMA.

Table 36-83: ROM_DMA_MDMA_OPERATION Members

Enumerator	Description
ROM_DMA_MEM_COPY	Standard MDMA transfer from a source to a destination
ROM_DMA_MEM_CRC	Performs a CRC32 MDMA read operation and compares the result with an expected result
ROM_DMA_MEM_FILL	Uses the CRC peripheral to perform a fill operation with a 32-bit value
ROM_DMA_MEM_COMPARE	Uses the CRC peripheral to compare data with a constant 32-bit value
ROM_DMA_CRC_LUT_INIT	Initializes the CRC LUT from the supplied CRC Polynomial

ROM_DMA_MEM_COPY

Standard MDMA transfer from a source to a destination

ROM_DMA_MEM_CRC

Performs a CRC32 MDMA read operation and compares the result with an expected result

ROM_DMA_MEM_FILL

Uses the CRC peripheral to perform a fill operation with a 32-bit value

ROM_DMA_MEM_COMPARE

Uses the CRC peripheral to compare data with a constant 32-bit value

ROM_DMA_CRC_LUT_INIT

Initializes the CRC LUT from the supplied CRC Polynomial

enum ROM_GETADDR_VALUE

Enumeration Type Declaration: `ROM_GETADDR_VALUE`

Data items for which the storage location can be retrieved using the `adi_rom_GetAddress()` function.

A number of data items are located in the boot ROM and used by the boot process, for example cache configuration information and default boot mode parameters used when preboot calls a boot mode using the `adi_rom_Boot()` API.

Table 36-84: ROM_GETADDR_VALUE Members

Enumerator	Description
`ROM_GETADDR_CONSTANTS`	`ROM_CONSTANTS_TYPE` object containing ROM version information
`ROM_GETADDR_BMODE`	`ROM_BOOTMODES_TYPE` object containing default boot mode parameters when calling `adi_rom_Boot()` from preboot
`ROM_GETADDR_MDMAREGS`	`ROM_BOOT_MDMA` object containing all the MDMA configuration details for use by the MDMA API routines
`ROM_GETADDR_SPILUT`	First `ROM_SPI_LUTENTRY` object in the list of 16 containing all the configuration details for SPI Master boot defined by each `BCODE` value during autodetection
`ROM_GETADDR_ECDSA_DOMAIN`	NIST- P-224 Parameter Curves
`ROM_GETADDR_ICPLB_DATA`	First 32-bit item of an array of 10 containing the `ICPLB_DATA[n]` items for cache configuration in an open device
`ROM_GETADDR_ICPLB_ADDR`	First 32-bit item of an array of 10 containing the `ICPLB_ADDR[n]` items for cache configuration in an open device
`ROM_GETADDR_ICPLB_DATA_SECURE`	First 32-bit item of an array of 6 containing the `ICPLB_DATA[n]` items for cache configuration in a locked device
`ROM_GETADDR_ICPLB_ADDR_SECURE`	First 32-bit item of an array of 6 containing the `ICPLB_ADDR[n]` items for cache configuration in a locked device
`ROM_GETADDR_DCPLB_DATA`	First 32-bit item of an array of 12 containing the `DCPLB_DATA[n]` items for cache configuration in an open device
`ROM_GETADDR_DCLB_ADDR`	First 32-bit item of an array of 12 containing the `DCPLB_DATA[n]` items for cache configuration in an open device

ROM_GETADDR_CONSTANTS

ROM_CONSTANTS_TYPE object containing ROM version information

ROM_GETADDR_BMODE

ROM_BOOTMODES_TYPE object containing default boot mode parameters when calling adi_rom_Boot() from preboot

ROM_GETADDR_MDMAREGS

ROM_BOOT_MDMA object containing all the MDMA configuration details for use by the MDMA API routines

ROM_GETADDR_SPILUT

First ROM_SPI_LUTENTRY object in the list of 16 containing all the configuration details for SPI Master boot defined by each BCODE value during autodetection

ROM_GETADDR_ECDSA_DOMAIN

NIST- P-224 Parameter Curves

ROM_GETADDR_ICPLB_DATA

First 32-bit item of an array of 10 containing the ICPLB_DATA[n] items for cache configuration in an open device

ROM_GETADDR_ICPLB_ADDR

First 32-bit item of an array of 10 containing the ICPLB_ADDR[n] items for cache configuration in an open device

ROM_GETADDR_ICPLB_DATA_SECURE

First 32-bit item of an array of 6 containing the ICPLB_DATA[n] items for cache configuration in a locked device

ROM_GETADDR_ICPLB_ADDR_SECURE

First 32-bit item of an array of 6 containing the ICPLB_ADDR[n] items for cache configuration in a locked device

ROM_GETADDR_DCPLB_DATA

First 32-bit item of an array of 12 containing the DCPLB_DATA[n] items for cache configuration in an open device

ROM_GETADDR_DCLB_ADDR

First 32-bit item of an array of 12 containing the DCPLB_DATA[n] items for cache configuration in an open device

enum ROM_HOOK_CALL_CAUSE

Enumeration Type Declaration: `ROM_HOOK_CALL_CAUSE`

Passed to a user hook routine to indicate the reason of the call.

When calling a boot mode via adi_rom_Boot, the user may provide an optional hook routine as a callback. This hook routine is called by the boot software firstly after the execution of the boot modes initialization routine then again after execution of the boot modes configuration routine. This parameter allows the users routine to identify at which point the call was made allowing the user to perform different actions for each call.

Table 36-85: ROM_HOOK_CALL_CAUSE Members

Enumerator	Description
`ROM_HOOK_CALL_INIT_COMPLETE`	Call was as a result of completion of the boot modes initialization function
`ROM_HOOK_CALL_CONFIG_COMPLETE`	Call was as a result of the completion of the boot modes configuration function

ROM_HOOK_CALL_INIT_COMPLETE

Call was as a result of completion of the boot modes initialization function

ROM_HOOK_CALL_CONFIG_COMPLETE

Call was as a result of the completion of the boot modes configuration function

enum ROM_SB_IMAGE_TYPE

Enumeration Type Declaration: `ROM_SB_IMAGE_TYPE`

Secure Boot Image Types.

The secure boot header contains a type field for the secure boot image type, this enumeration provides a complete list of all image types.

NOTE: The secure boot process does not necessarily support all image types defined.

Table 36-86: ROM_SB_IMAGE_TYPE Members

Enumerator	Description
`ROM_SB_IMAGE_UNKNOWN`	Unknown Secure Boot image type, used by software to initialize the type before detection of boot image type takes place
`ROM_SB_IMAGE_BLP`	Plain text `BLp` secure boot image supporting authentication only with no decryption
`ROM_SB_IMAGE_BLW`	Keywrapped `BLw` secure boot image supporting authentication and decryption, boot stream decryption key wrapped in the secure header
`ROM_SB_IMAGE_BLE`	Not supported by any Secure boot products. Secure boot image with key stored in plain text form in the secure header

Table 36-86: ROM_SB_IMAGE_TYPE Members (Continued)

Enumerator	Description
ROM_SB_IMAGE_BLX	BLx Secure boot image supporting authentication and decryption, boot stream decryption key wrapped located in OTP
ROM_SB_IMAGE_UNSUPPORTED	May be used by software to indicate any other unsupported image type

ROM_SB_IMAGE_UNKNOWN

Unknown Secure Boot image type, used by software to initialize the type before detection of boot image type takes place

ROM_SB_IMAGE_BLP

Plain text BLp secure boot image supporting authentication only with no decryption

ROM_SB_IMAGE_BLW

Keywrapped BLw secure boot image supporting authentication and decryption, boot stream decryption key wrapped in the secure header

ROM_SB_IMAGE_BLE

Not supported by any Secure boot products. Secure boot image with key stored in plain text form in the secure header

ROM_SB_IMAGE_BLX

BLx Secure boot image supporting authentication and decryption, boot stream decryption key wrapped located in OTP

ROM_SB_IMAGE_UNSUPPORTED

May be used by software to indicate any other unsupported image type

enum ROM_SPI_PROTOCOL

Enumeration Type Declaration: ROM_SPI_PROTOCOL

The SPI Protocol to use.

SPI Flash devices are now capable of supporting multiple protocols for the sending of the command to the SPI flash. Typically the command would be sent over the single bit bus, however a number of newer devices also support the sending of the command over the dual or quad bit bus.

WARNING: Enabling of dual and quad modes for the command cycles runs the risk of the boot process being unable to communicate with the SPI flash, especially in system reset type event where the processor will attempt to reboot and the flash may not have been reset. It is not recommended to enable such features on SPI Flash devices if they are also the primary boot source used for booting from hardware reset and system reset events.

Table 36-87: ROM_SPI_PROTOCOL Members

Enumerator	Description
ROM_SPI_EXT_PROTOCOL	Extended protocol where the command cycle is sent on the single bit bus
ROM_SPI_DUALIO_PROTOCOL	DualIO protocol where the command cycle is sent on the dual bit bus
ROM_SPI_QUADIO_PROTOCOL	QuadIO protocol where the command cycle is sent on the quad bit bus

ROM_SPI_EXT_PROTOCOL

Extended protocol where the command cycle is sent on the single bit bus

ROM_SPI_DUALIO_PROTOCOL

DualIO protocol where the command cycle is sent on the dual bit bus

ROM_SPI_QUADIO_PROTOCOL

QuadIO protocol where the command cycle is sent on the quad bit bus

37 System Debug and Trace Unit (DBG)

The system debug and trace unit is based on ARM Core Sight technology. CoreSight™ is a set of architecture specifications defining debug and trace architecture. The processor uses CoreSight infrastructure to provide industry standard debug and trace capabilities.

This document describes the Core Sight technology integration on the processor, but does not provide detailed about the Core sight features. For more information about the CoreSight Debug and Trace Component, visit the ARM Information Center:

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

The applicable documentation for more details about the ARM CoreSight feature includes:

- ARM Debug Interface v5, Architecture Specification, ARM IHI 00031A (Debug)
- CoreSight PFT Architecture Specification, ARM IHI 0035B (PFT)
- System Trace Macrocell, Programmers' Model Architecture Specification, ARM IHI 0054A (STM)
- CoreSight Trace Memory Controller, ARM DDI0461B (TMC)
- CoreSight Components Technical Reference Manual, ARM DDI 0314H (TPIU)
- Embedded Cross Trigger Technical Reference Manual, ARM DDI 0291A

DBG Features

The system debug and trace unit contains the following features.

- System JTAG TAP controller for system debug features, boundary scan, and public JTAG features
- A debug interface to cores, and other system resources
- Direct and run-time access to the memory system and system MMRs
- Direct control over system reset
- Support for debug immediately after reset (boot debug)
- Group halt (debug event immediately halts all specified endpoints)
- Real-time on-chip visibility is made available to all developers, including software developers

DBG Functional Description

The following sections provide functional descriptions of the DBG unit.

ADSP-BF70x CSPFT Register List

Table 37-1: ADSP-BF70x CSPFT Register List

Name	Description
CSPFT_ACTR[n]	Address Comparator Access Type Register
CSPFT_ACVR[n]	Address Comparator Value Register
CSPFT_AUTHSTATUS	Authentication Status Register
CSPFT_CCER	Configuration Code Extension Register
CSPFT_CID0	Component ID0 Register
CSPFT_CID1	Component ID1 Register
CSPFT_CID2	Component ID2 Register
CSPFT_CID3	Component ID3 Register
CSPFT_CIDCMR	Context ID Comparator Mask Register
CSPFT_CIDCVR[n]	Context ID Comparator Value
CSPFT_CLAIMCLR	Claim Tag Clear Register
CSPFT_CLAIMSET	Claim Tag Set Register
CSPFT_CNTENR[n]	Counter Enable Event Register
CSPFT_CNTRLDEVR[n]	Counter Reload Event Register
CSPFT_CNTRLDVR[n]	Counter Reload Value Register
CSPFT_CNTVR[n]	Counter Value Register
CSPFT_CTL	Main Control Register
CSPFT_DEVTYPE	Device Type Identifier Register
CSPFT_EXTOUTEVR[n]	External Output Event Register
CSPFT_HWFEAT	Hardware Feature Register
CSPFT_LAR	Lock Access Register
CSPFT_LSR	Lock Status Register
CSPFT_PID0	Peripheral ID0 Register
CSPFT_PID1	Peripheral ID1 Register
CSPFT_PID2	Peripheral ID2 Register
CSPFT_PID3	Peripheral ID3 Register
CSPFT_PID4	Peripheral ID4 Register

Table 37-1: ADSP-BF70x CSPFT Register List (Continued)

Name	Description
CSPFT_STAT	Status Register
CSPFT_SYNCFR	Synchronization Frequency Register
CSPFT_TECTL	TraceEnable Control Register
CSPFT_TEEVENT	TraceEnable Event Register
CSPFT_TRACEIDR	CoreSight Trace ID Register
CSPFT_TRIGGER	Trigger Event Register
CSPFT_TSSCTL	TraceEnable Start/Stop Control Register

ADSP-BF70x TAPC Register List

The Test Access Port Controller (TAPC) provides access to debug features. A set of registers governs TAPC operations. For more information on TAPC functionality, see the TAPC register descriptions.

Table 37-2: ADSP-BF70x TAPC Register List

Name	Description
TAPC_DBGCTL	Debug Control
TAPC_IDCODE	IDCODE 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_SDBGKEY_CTL	Secure Debug Key Control Register
TAPC_SDBGKEY_STAT	Secure Debug Key Status Register
TAPC_USERCODE	USERCODE Register

DBG Block Diagram

The block diagram is shown below.

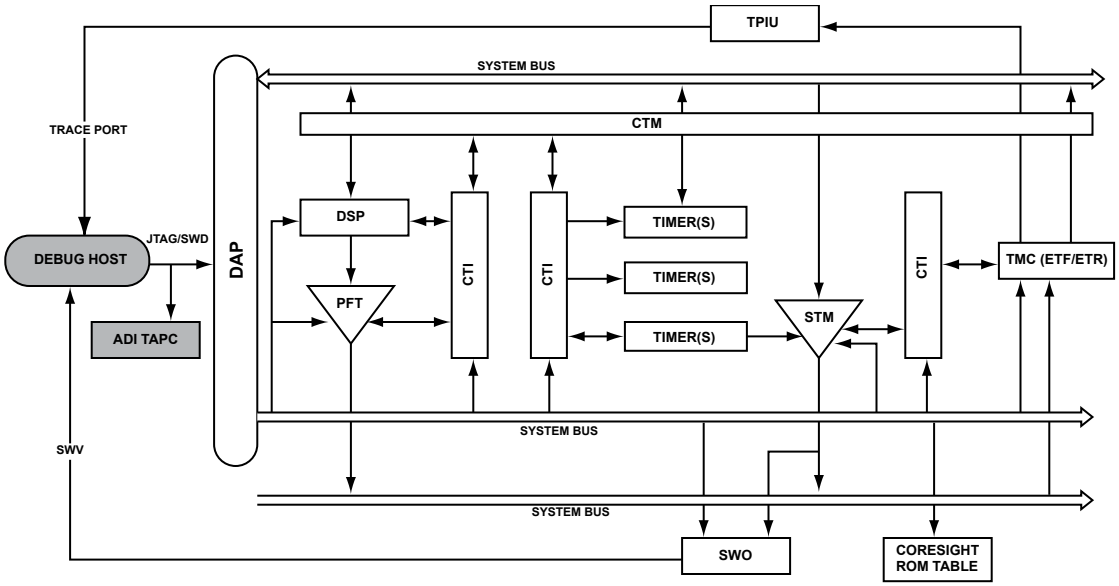

Figure 37-1: ADSP-BF70x Block Diagram

DBG Definitions

The following terms are useful when working with the debug features of the processor and programming tools.

Test Access Port Controller (ADI TAPC)

Provides IDCODE and SDBGKEY features.

Debug Access Port (DAP)

Core Sight Interface providing a single port for two debug options: JTAG-DP (JTAP Debug Port), SW-DP (Serial Wire Debug Port)

Program Trace Macrocell (PTM)

Provides DSP (Blackfin+)Core Trace.

Standard Trace Macrocell (STM)

Provides capability to trace up to 32 hardware events and supports 32 software stimulus for data transfer between the user and emulator.

Embedded Cross Trigger (ECT)

The ECTs are responsible managing events and triggers as follows.

- CTI (Cross Trigger Interface) is a CoreSight component for enabling cross triggering of events across a system. From the view of the ECT, it is responsible for combining and mapping trigger requests.

- CTM (Cross Trigger Matrix) is a CoreSight component for connecting multiple CTIs. From the view of the ECT, it is responsible for connecting CTIs and distribution of events.

NOTE: An embedded cross trigger is not the same as the master and slave trigger in the trigger routing unit.

Trace Capture Devices

The trace capture devices capture and format the trace data. There are two trace capture devices in the system:

- ETF - Embedded Trace Funnel - Provides a buffer for burst trace data.
- ETR - Embedded Trace Router - Provides interface for trace data to be stored in system memories.

Trace Port Interface Unit (TPIU)

The TPIU acts as a bridge between the on-chip trace data, with separate IDs, to a data stream. It encapsulates IDs, when required, that the Trace Port Analyzer (TPA) captures.

Serial Wire Output (SWO)

SWO is a trace data drain that acts as a bridge between the on-chip trace data and a data stream that the Trace Port Analyzer (TPA) captures.

Test Access Port Controller (TAPC)

TAPC is an independent component daisy-chained to the DAP in the JTAG scan path. The TAPC component provides the product IDCODE (Chip ID) and a security feature.

The TAPC component provides security features to the chip using a debug key match features. This feature permits only those components which have the SDBGKEY to connect and debug the chip. Initially, a 128-bits user key is programmed in the `SDBGKEYx` registers (`SDBGKEY0`, `SDBGKEY1`, `SDBGKEY2`, `SDBGKEY3`) in TAPC by a secure master. The `SDBGKEY_VALID` bit in `SDBGKEY_CTL` register is set.

Then, a user key is entered through the emulator for a match to the initially programmed-entered user key. On a successful match, the chip connects to the emulator. It can be debugged on a failed user key match. Further attempts at keys matching are disabled. JTAG reset or system reset is required to reenable the user key match logic.

Debug Access Ports

The Debug Access Port (DAP) is an implementation of an ARM Debug Interface version 5 (ADIv5) comprising a number of components supplied in a single configuration. All the supplied components fit into the various architectural components for Debug Ports (DPs). The components are used to access the DAP from an external debugger and APs, to access on-chip system resources.

The DAP provides access to all debug and trace capabilities through a single external interface. DAP has some additions to support BSCA and IDCODE features that Core Sight DAP does not support. The DAP provides a combined single debug interface port called SWJ_DP that includes:

- JTAG Debug Port (JTAG-DP). The JTAG-DP is based on the IEEE 1149.1 Test Access Port (TAP) and Boundary Scan Architecture
- Serial Wire Debug Port (SW-DP). The serial wire debug port provides a bidirectional serial connection to the ARM debug interface

The SWJ-DP is a combined JTAG-DP and SW-DP that allows connections to either a Serial Wire Debug (SWD) or JTAG probe to a target. It is the standard CoreSight debug port, and enables access either to the JTAG-DP or SW-DP blocks. The JTAG-DP is selected by default.

Trace Unit

The trace module provides instruction, data tracing, and system activity tracing for the processor. The program trace module on the processor is similar to the embedded trace macrocell module provided by the ARM processor. System trace is provided using the system trace macrocell as part of the CoreSight debug and trace interface. The trace module uses an interface based on the AMBA Trace Bus (ATB) standard to output its trace data. The trace data can be either exported to an off-chip trace port analyzer or captured on an on-chip buffer. The PFT and STM modules capture information on the processor both before and after a specific event. The modules add no burden to the processor performance when it runs at full speed.

- Programmable Flow Trace (CSPFT)
- System Trace Module (STM)

Programmable Flow Trace (CSPFT)

When tracing processor execution, trace information can be generated for every instruction the processor executes. This information would be easy to interpret, but would require a prohibitively high trace bandwidth to get the trace data off the chip. With program flow tracing, only branch points are traced. The debugger uses the source code to infer the rest of the executed code.

Certain instructions in the program and events are identified as waypoints. A waypoint is a point where instruction execution involves a change of program flow. The CSPFT only traces those waypoints. These waypoints are:

- All indirect branches
- All direct branches
- Exceptions or interrupts
- Emulator debug entry and exit

When a waypoint occurs, trace data is generated to describe it. From this data and the source code, a trace decompressor can determine what instructions were executed and recreate the instruction flow. To allow the decompressor to calculate where it is in the source code, conditional instructions are marked as waypoints, regardless of whether they pass or fail their condition test. Events like interrupts or debug entry and exit can be promoted from non-waypoint instructions to waypoints to trace the interrupted program flow.

Tracing a waypoint implies the execution of all instructions from the target address of the previous waypoint up to the current waypoint. Non-waypoint instructions are not explicitly traced but the debugger must infer them using the source code. The concept of an instruction block is used throughout this manual and refers to the contiguous block of instructions between two waypoints.

The programming model and function is a subset of the PTM (Programmable Trace Module) of ARM.

System Trace Module (STM)

The STM is a trace source that is integrated into a CoreSight system, and is designed primarily for high-bandwidth trace of instrumentation embedded into software. The STM enables tracing of system activity from various sources:

- Instrumented software, using memory-mapped stimulus ports
- Hardware events

The STM supports the following features:

- Multiple software masters writing software instrumentation independently. Each master can use multiple stimulus ports.
- Time stamping of the system activity. The time stamp is a global time stamp which can be shared with other trace sources in the system to enable correlation of activity from multiple trace sources.
- Indication that specific events have occurred, such as a particular hardware event or a piece of software instrumentation. These events are known as triggers and can be indicated in the trace stream, or through signals to other system components.

32 hardware event resources are connected as output from the TRU which allows monitoring all of the hardware events that can generate a trigger.

Embedded Cross Trigger (ECT)

ECT provides an interface to the CoreSight debug system enabling the subsystems to interact (cross trigger) with each other. ECT provides a mechanism to forward debug events from one connected subsystem to another connected subsystem. The different subsystems connected to the ECT depend on the processor design. For example, in a multiprocessor system, the interface can be connected to each of the cores and one to the trace subsystem. For a uniprocessor system, the interface can include just the core and trace subsystem connection.

- CTI Cross trigger interface. A CoreSight component for enabling cross triggering of events across a system.
- CTM Cross trigger matrix. A CoreSight component for connecting multiple Cross Trigger Interfaces.

The main function of the ECT (CTI and CTM) is to pass debug events from one connected subsystem to another connected subsystem.

For example, the ECT can communicate debug state information from the core to trace subsystem for a single processor system or to another core in a multiprocessor-based system. Program execution on both the subsystem can be stopped at the same time.

The ***Trigger Flow*** figure shows a simple debug trigger flow sequence. On each CTI, there are four channel, eight input, and eight output debug triggers. All the eight inputs and outputs can be mapped to a single channel or different channels based on the debug trigger to channel mapping. When a trigger input occurs, it creates a channel event. The channel event causes all the output debug triggers to be triggered. The embedded cross trigger depends on the debug trigger it connects to.

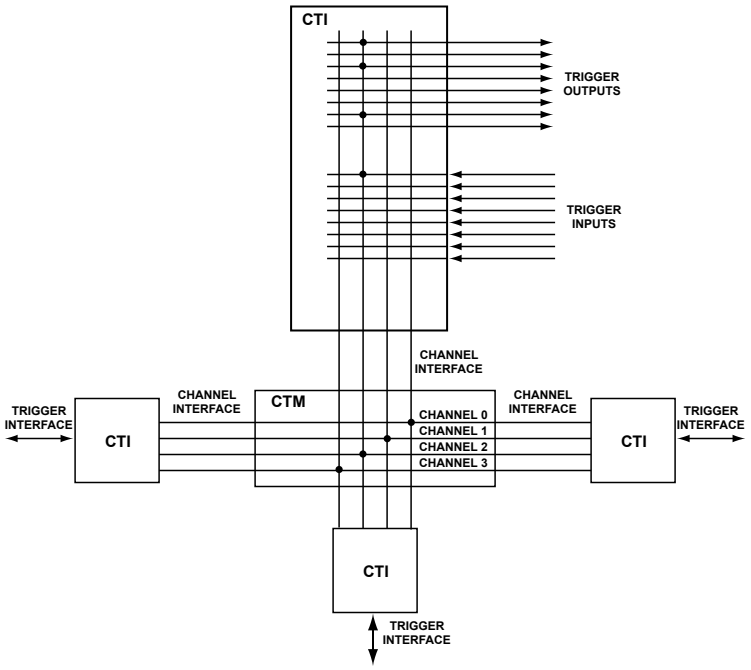

Figure 37-2: Trigger Flow

Refer to the ***ECT Integration*** figure. There are 3 CTIs in the system that connect to the core, trace, and system module, respectively. The CTIs all interconnect through the CTM. This configuration allows the core to trigger debug events on the trace, on the system and on the core itself. CTI0 handles all the PFT and core debug triggers. CTI1 handles all the trace components debug triggers. CTI2 handles all the system debug triggers.

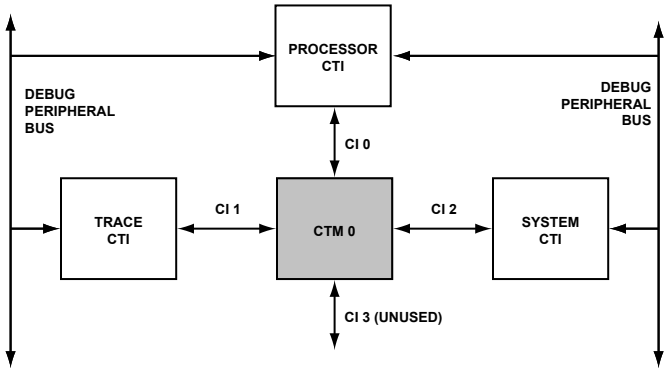

Figure 37-3: ECT Integration

CTI Debug Trigger Tables

The ***System CTI Trigger Connection*** tables show the debug trigger that connects to each CTI.

Table 37-3: System CTI Trigger Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	From TRU	CTITRIGOUT[7]	To TRU and also as SYS_DBGRESTART
CTITRIGIN [6:2]	From TRU	CTITRIGOUT[6:2]	To TRU
CTITRIGIN [1]	From TRU	CTITRIGOUT[1]	To TRU and also as Peripheral Halt
CTITRIGIN[0]	From TRU	CTITRIGOUT[0]	To TRU and also as System fabric Halt

Table 37-4: Trace CTI Trigger Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	STM ASYNCOUT	CTITRIGOUT[7]	Unused
CTITRIGIN[6]	STM TRIGOUTHETE	CTITRIGOUT [6]	TPIU FLUSHIN
CTITRIGIN[5]	STM TRIGOUTSW	CTITRIGOUT [5]	TPIU TRIGIN
CTITRIGIN[4]	STM TRIGOUTSPTE	CTITRIGOUT [4]	ETR FLUSHIN
CTITRIGIN[3]	ETR FULL	CTITRIGOUT [3]	ETR TRIGIN
CTITRIGIN[2]	ETR ACQCOMP	CTITRIGOUT [2]	ETF FLUSHIN
CTITRIGIN[1]	ETF FULL	CTITRIGOUT [1]	ETF TRIGIN
CTITRIGIN[0]	ETF ACQCOMP	CTITRIGOUT [0]	Unused

Table 37-5: ADSP-BF70x Core CTI Trace Connection

CTI Port	Input	CTI Port	Output
CTITRIGIN[7]	Tied Low	CTITRIGOUT[7]	DBGRESTART
CTITRIGIN[6]	PTM TRIGGER	CTITRIGOUT[6]	SEC
CTITRIGIN[5:2]	PTM EXTOUT[3:0]	CTITRIGOUT[5:2]	PTM EXTIN[3:0]
CTITRIGIN[1]	Tied Low	CTITRIGOUT[1]	Unused
CTITRIGIN[0]	DBGTRIGGER	CTITRIGOUT[0]	EDBGRQ

Table 37-6: Trigger Descriptions

Signal Name	Description
STM ASYNCOUT	Alignment synchronization output. This signal is asserted for one clock cycle when an ASYNC-VERSION-FREQ sequence is completely output on the ATB, and can be used for cross-triggering.
STM TRIGOUTHETE	Trigger output. This signal is asserted for one clock cycle when a trigger event is detected on match using STMHETER.

Table 37-6: Trigger Descriptions (Continued)

Signal Name	Description
STM TRIGOUTSW	Trigger output. This signal is asserted for one clock cycle when a trigger event is generated on writes to a TRIG location in the extended stimulus port registers.
STM TRIGOUTSPTE	Trigger output. This signal is asserted for one clock cycle when a trigger event is detected on match using STMSPTER.
ETR / ETF FULL	This output indicates the value of the Full bit in the ETR/ETF Status Register. Full Bit indicates the amount of data in ETF/ETR. An output signal indicating when the Circular Buffer or FIFO is full, or within a programmable amount of being full.
ETR / ETF ACQCOMP	This output indicates the value of the FtEmpty bit in the ETR/ETF Status Register. Bit is set when trace capture has stopped. An output signal indicating when trace capture has stopped, usually following a trigger condition.
ETR/ETF/ TPIU TRIGIN	This input can cause a Trigger Event (Start /Stop Trigger). An input signal indicating when a trigger condition has occurred.
ETR/ETF /TPIU FLUSHIN	This input can cause a Trace flush. An input signal indicating a flush request
PTM/ETM TRIGGER	Trigger Input. Trigger event specifies the conditions that must be met to generate a trigger.
PTM/ETM EXTOUT[3:0]	PTM Output.
PTM/ETM EXTIN[3:0]	PTM Input
SEC	CTI Interrupt
DBGRESTART	This is an output from the CTI to a processor core or to system to return from debug mode.
DBGTRIGGER	This is a processor core output signal indicating that the core has moved to debug mode. If the CTIs are setup for synchronous halt, it will generate EDBGRQ to everyone else.
EDBGRQ	This is an output from CTI and an input (as debug halt) to a processor core and to the system in general. It can assert as a result of another core going to emulation space (DBGTRIGGER) or by setting the corresponding bit in the CTI.
SYS_DBGRESTART	This is an output from the CTI to system to return from debug mode.
To TRU	TRU Master Event
From TRU	TRU Slave Event

ADSP-BF70x CSPFT Register Descriptions

Program Flow Trace (CSPFT) contains the following registers.

Table 37-7: ADSP-BF70x CSPFT Register List

Name	Description
CSPFT_ACTR[n]	Address Comparator Access Type Register
CSPFT_ACVR[n]	Address Comparator Value Register
CSPFT_AUTHSTATUS	Authentication Status Register
CSPFT_CCER	Configuration Code Extension Register
CSPFT_CID0	Component ID0 Register
CSPFT_CID1	Component ID1 Register
CSPFT_CID2	Component ID2 Register
CSPFT_CID3	Component ID3 Register
CSPFT_CIDCMR	Context ID Comparator Mask Register
CSPFT_CIDCVR[n]	Context ID Comparator Value
CSPFT_CLAIMCLR	Claim Tag Clear Register
CSPFT_CLAIMSET	Claim Tag Set Register
CSPFT_CNTENR[n]	Counter Enable Event Register
CSPFT_CNTRLDEVR[n]	Counter Reload Event Register
CSPFT_CNTRLDVR[n]	Counter Reload Value Register
CSPFT_CNTVR[n]	Counter Value Register
CSPFT_CTL	Main Control Register
CSPFT_DEVTYPE	Device Type Identifier Register
CSPFT_EXTOUTEVR[n]	External Output Event Register
CSPFT_HWFEAT	Hardware Feature Register
CSPFT_LAR	Lock Access Register
CSPFT_LSR	Lock Status Register
CSPFT_PID0	Peripheral ID0 Register
CSPFT_PID1	Peripheral ID1 Register
CSPFT_PID2	Peripheral ID2 Register
CSPFT_PID3	Peripheral ID3 Register
CSPFT_PID4	Peripheral ID4 Register
CSPFT_STAT	Status Register
CSPFT_SYNCFR	Synchronization Frequency Register
CSPFT_TECTL	TraceEnable Control Register
CSPFT_TEEVENT	TraceEnable Event Register

Table 37-7: ADSP-BF70x CSPFT Register List (Continued)

Name	Description
CSPFT_TRACEIDR	CoreSight Trace ID Register
CSPFT_TRIGGER	Trigger Event Register
CSPFT_TSSCTL	TraceEnable Start/Stop Control Register

Address Comparator Access Type Register

The `CSPFT_ACTR[n]` register specifies whether the context ID needs to match.

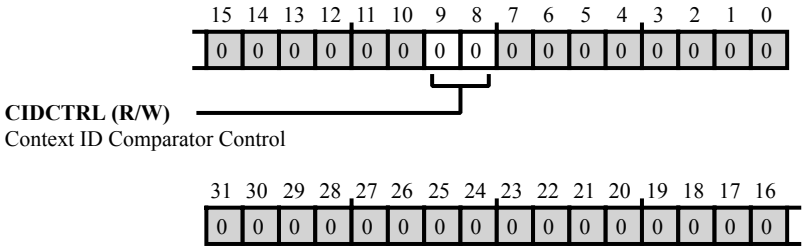

Figure 37-4: CSPFT_ACTR[n] Register Diagram

Table 37-8: CSPFT_ACTR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
9:8 (R/W)	CIDCTRL	Context ID Comparator Control. The `CSPFT_ACTR[n].CIDCTRL` contains the ID comparator control value.	
		0	Ignore Context ID
		1	Match if Context ID Comparator 0 Matches
		2	Match if Context ID Comparator 1 Matches
		3	Match if Context ID Comparator 2 Matches

Address Comparator Value Register

The CSPFT_ACVR[n] register holds an address for comparison.

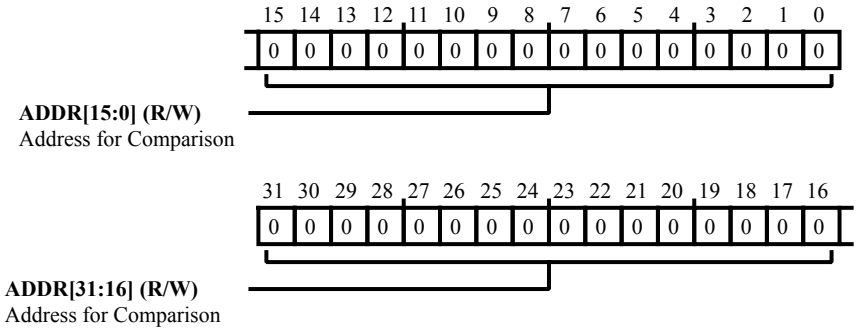

Figure 37-5: CSPFT_ACVR[n] Register Diagram

Table 37-9: CSPFT_ACVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Address for Comparison. The CSPFT_ACVR[n].ADDR bit field contains the address used for comparison.

Authentication Status Register

The CSPFT_AUTHSTATUS register reports the level of tracing currently permitted based on the DBGEN signal.

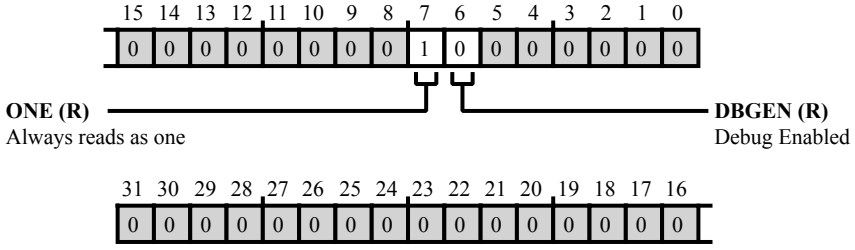

Figure 37-6: CSPFT_AUTHSTATUS Register Diagram

Table 37-10: CSPFT_AUTHSTATUS Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7 (R/NW)	ONE	Always reads as one.
6 (R/NW)	DBGEN	Debug Enabled. The CSPFT_AUTHSTATUS.DBGEN bit indicates that invasive debug is enabled. Normally, NIDEN is used in conjunction with a signal that enables invasive debug, DBGEN. Non-invasive debug is disabled only if both NIDEN and DBGEN signals are LOW. In a PTM, typically these signals are ORed together and the result is used to determine whether non-invasive debug is enabled.

Configuration Code Extension Register

The CSPFT_CCER register holds extra feature information. (See CSPFT_HWFEAT.)

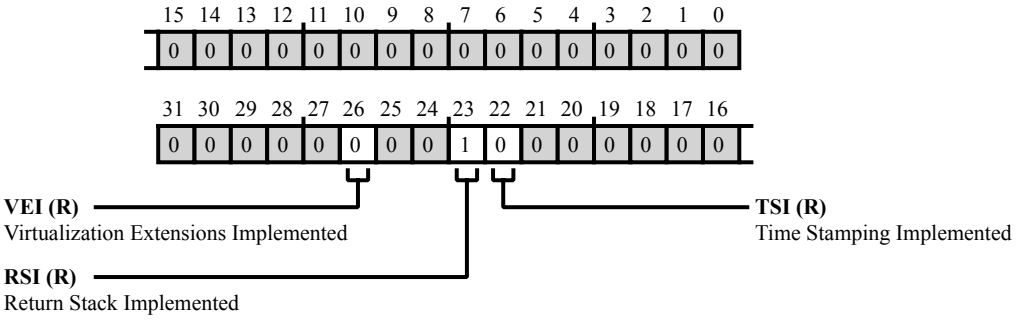

Figure 37-7: CSPFT_CCER Register Diagram

Table 37-11: CSPFT_CCER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
26 (R/NW)	VEI	Virtualization Extensions Implemented. The CSPFT_CCER.VEI bit indicates if the virtualization extensions are implemented	
		0	Not Implemented
		1	Implemented
23 (R/NW)	RSI	Return Stack Implemented. The CSPFT_CCER.RSI bit indicates if a return stack is implemented.	
		0	Not Implemented
		1	Implemented
22 (R/NW)	TSI	Time Stamping Implemented. The CSPFT_CCER.TSI bit indicates if time stamping is implemented.	
		0	Disabled
		1	Enabled

Component ID0 Register

The `CSPFT_CID0` register holds sections of the CoreSight Component ID for CSPFT.

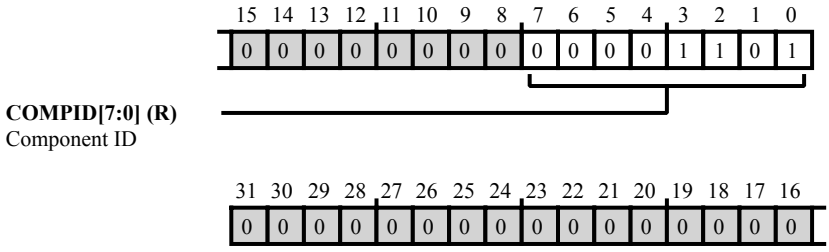

Figure 37-8: CSPFT_CID0 Register Diagram

Table 37-12: CSPFT_CID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The `CSPFT_CID0.COMPID` bit field identifies this component as a CoreSight component.

Component ID1 Register

The `CSPFT_CID1` register holds sections of the CoreSight Component ID for CSPFT.

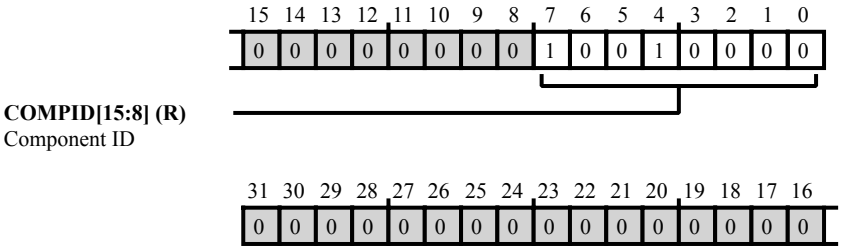

Figure 37-9: CSPFT_CID1 Register Diagram

Table 37-13: CSPFT_CID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The `CSPFT_CID1.COMPID` bit field identifies this component as a CoreSight component.

Component ID2 Register

The CSPFT_CID2 register holds sections of the CoreSight Component ID for CSPFT.

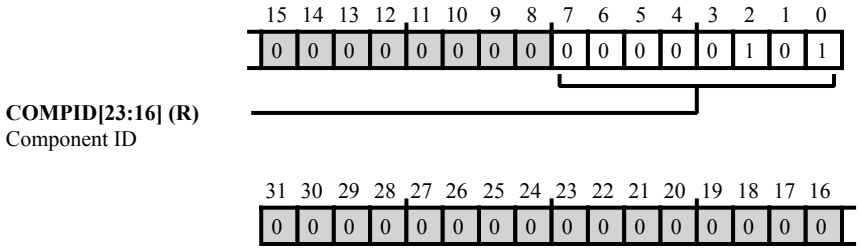

Figure 37-10: CSPFT_CID2 Register Diagram

Table 37-14: CSPFT_CID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The CSPFT_CID2.COMPID bit field identifies this component as a CoreSight component.

Component ID3 Register

The CSPFT_CID3 register holds sections of the CoreSight Component ID for CSPFT.

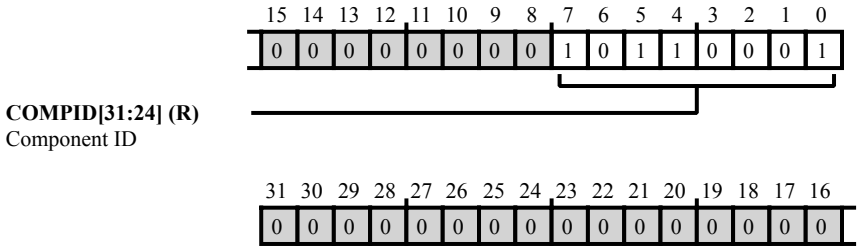

Figure 37-11: CSPFT_CID3 Register Diagram

Table 37-15: CSPFT_CID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	COMPID	Component ID. The CSPFT_CID3.COMPID bit field identifies this component as a CoreSight component.

Context ID Comparator Mask Register

The CSPFT_CIDCMR register holds a 32-bit mask for use for all context ID comparisons.

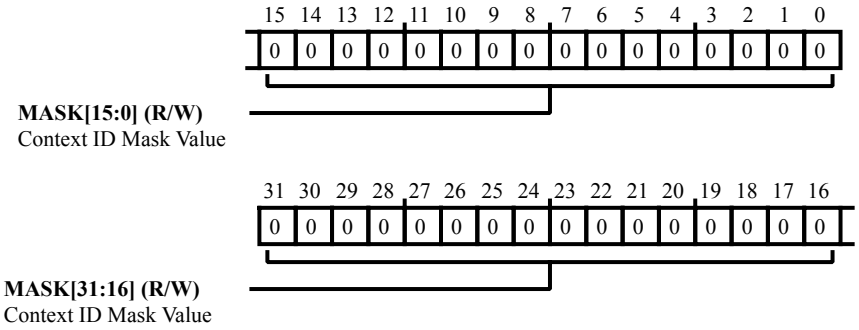

Figure 37-12: CSPFT_CIDCMR Register Diagram

Table 37-16: CSPFT_CIDCMR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	MASK	Context ID Mask Value. The CSPFT_CIDCMR.MASK bit field holds a 32-bit mask for use in all context ID comparisons.

Context ID Comparator Value

The CSPFT_CIDCVR[n] register holds a context ID value for comparison.

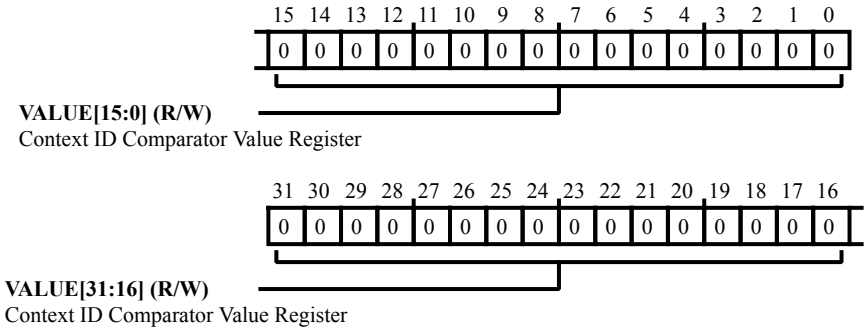

Figure 37-13: CSPFT_CIDCVR[n] Register Diagram

Table 37-17: CSPFT_CIDCVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	Context ID Comparator Value Register. The CSPFT_CIDCVR[n].VALUE bit field holds a context ID value for comparison.

Claim Tag Clear Register

The CSPFT_CLAIMCLR register is used to clear bits in the claim tag or get the current value of the claim tag.

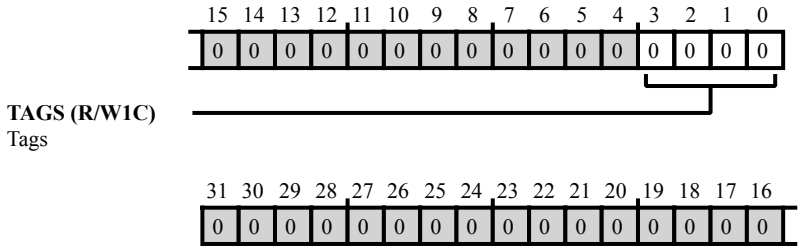

Figure 37-14: CSPFT_CLAIMCLR Register Diagram

Table 37-18: CSPFT_CLAIMCLR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R/W1C)	TAGS	Tags. A read of the CSPFT_CLAIMCLR.TAGS bit field returns the current value, a write clears bits.

Claim Tag Set Register

The CSPFT_CLAIMSET register is used to set bits in the claim tag and find the number of bits supported by the claim tag.

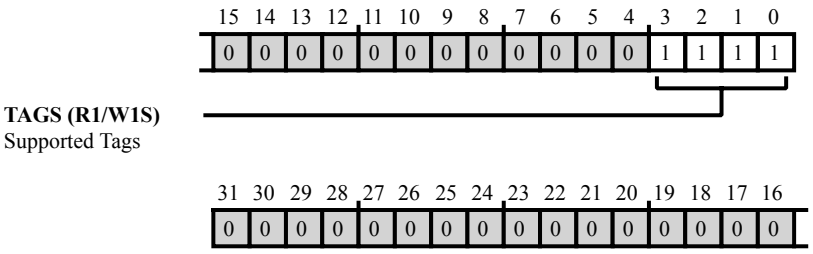

Figure 37-15: CSPFT_CLAIMSET Register Diagram

Table 37-19: CSPFT_CLAIMSET Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3:0 (R1/W1S)	TAGS	Supported Tags. The CSPFT_CLAIMSET.TAGS bit field sets bits in the claim tag and finds the number of bits supported by the claim tag.

Counter Enable Event Register

The CSPFT_CNTENR[n] register describes the event that enables the corresponding counter.

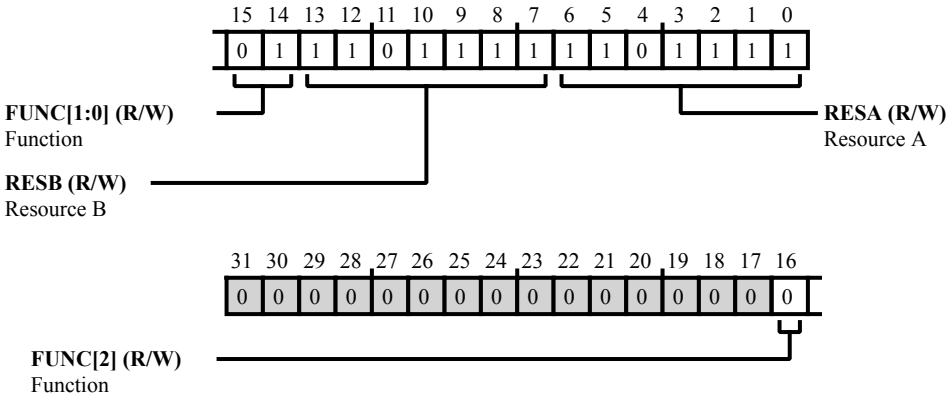

Figure 37-16: CSPFT_CNTENR[n] Register Diagram

Table 37-20: CSPFT_CNTENR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. The CSPFT_CNTENR[n].FUNC bit field specifies the logical operation that combines the two resources that define the event.	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The CSPFT_CNTENR[n].RESB bit field specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_CNTENR[n].FUNC field (See CSPFT_CNTENR[n].RESA for list of possible values).	
6:0 (R/W)	RESA	Resource A. The CSPFT_CNTENR[n].RESA bit field specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_CNTENR[n].FUNC field.	
		0	Single Addr Comparator 0

Table 37-20: CSPFT_CNTENR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at Zero
		65	Counter 1 at Zero
		66	Counter 2 at Zero
		67	Counter 3 at Zero
		88	Context ID Comparator 0
		89	Context ID Comparator 1
		90	Context ID Comparator 2

Table 37-20: CSPFT_CNTENR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

Counter Reload Event Register

The CSPFT_CNTRLDEVR[n] register defines the event that causes the corresponding counter to be reloaded with the value held in the corresponding CSPFT_CNTRLDVR[n] register.

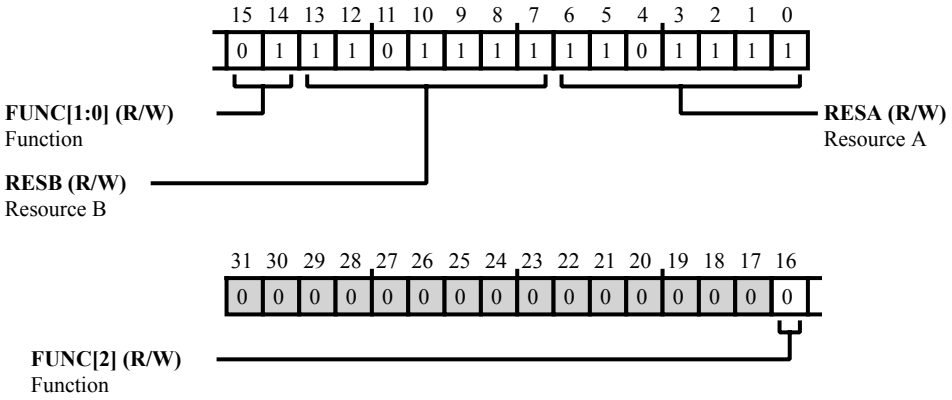

Figure 37-17: CSPFT_CNTRLDEVR[n] Register Diagram

Table 37-21: CSPFT_CNTRLDEVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. The CSPFT_CNTRLDEVR[n].FUNC bit field specifies the logical operation that combines the two resources that define the event.	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The CSPFT_CNTRLDEVR[n].RESB bit field specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_CNTRLDEVR[n].FUNC field (See CSPFT_CNTRLDEVR[n].RESA for list of possible values).	
6:0 (R/W)	RESA	Resource A.	

Table 37-21: CSPFT_CNTRLDEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The `CSPFT_CNTRLDEVR[n].RESA` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_CNTRLDEVR[n].FUNC` field.	
		0	Single Addr Comparator 0
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at zero
		65	Counter 1 at zero
		66	Counter 2 at zero
		67	Counter 3 at zero

Table 37-21: CSPFT_CNTRLDEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		88	Context ID comparator 0
		89	Context ID comparator 1
		90	Context ID comparator 2
		95	TraceEnable start/stop resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace prohibited
		111	Always TRUE

Counter Reload Value Register

The CSPFT_CNTRLDVR[n] register specifies the starting value of the corresponding counter.

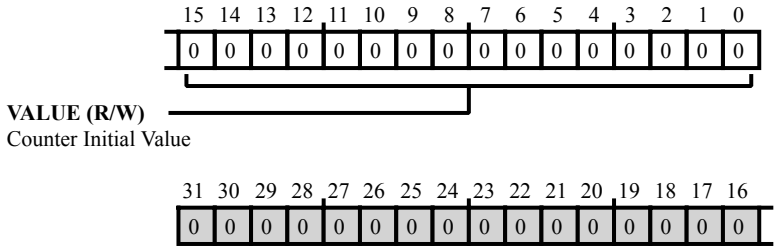

Figure 37-18: CSPFT_CNTRLDVR[n] Register Diagram

Table 37-22: CSPFT_CNTRLDVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Counter Initial Value. The CSPFT_CNTRLDVR[n].VALUE bit field specifies the starting value of the corresponding counter.

Counter Value Register

The CSPFT_CNTVR[n] register holds the current value of the corresponding counter.

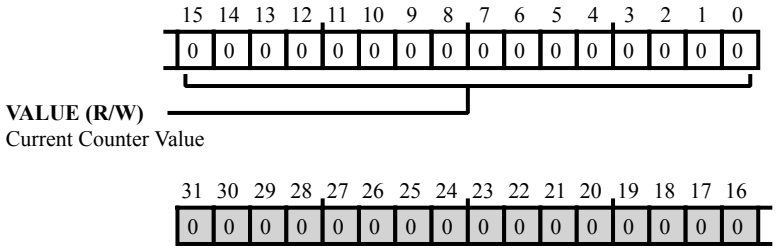

Figure 37-19: CSPFT_CNTVR[n] Register Diagram

Table 37-23: CSPFT_CNTVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
15:0 (R/W)	VALUE	Current Counter Value. The CSPFT_CNTVR[n].VALUE bit field specifies the current value of the corresponding counter.

Main Control Register

The CSPFT_CTL register controls general operation of the PTM, such as whether tracing is enabled.

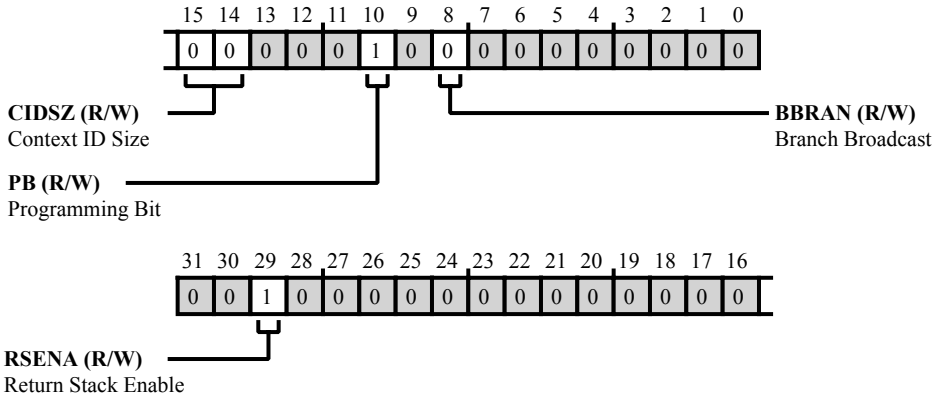

Figure 37-20: CSPFT_CTL Register Diagram

Table 37-24: CSPFT_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
29 (R/W)	RSENA	Return Stack Enable. When the CSPFT_CTL.RSENA bit is set, the first indirect branch back to an address generates a branch without exception packet, and subsequent branches back to the same address generate E Atoms. This compresses inner loops of HW loops and code that indirectly branches back.	
15:14 (R/W)	CIDSZ	Context ID Size. The CSPFT_CTL.CIDSZ bit field specifies the byte size to trace. Only the bytes specified are traced, even if the new Context ID value is larger than this.	
		0	No Context ID Tracing
		1	One byte Traced
		2	Two Bytes Traced
		3	Three Bytes Traced

Table 37-24: CSPFT_CTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
10 (R/W)	PB	Programming Bit. To program the CSPFT, use the following procedure. 1. Set the CSPFT_CTL.PB bit to disable all trace functionality. 2. Poll the CSPFT_STAT.PB bit waiting for it to be 1 (FIFO drained, trace halted). 3. Program the trace registers, counter and other registers, as required. 4. Set this bit to 0. 5. Poll the CSPFT_STAT.PB bit until it reads 0 (trace status reset, trace restarted). When the CSPFT_CTL.PB bit is set, the FIFO is drained and no more trace is produced. All counters are held in their present state and the external outputs are forced low. After the FIFO is drained, the CSPFT_STAT.PB is set to reflect that the part is ready to program. When this bit is cleared, the trace status is cleared and trace is restarted.	
		0	Trace Enabled
		1	Trace Disabled
8 (R/W)	BBRAN	Branch Broadcast. Set the CSPFT_CTL.BBRAN bit to 1 to enable branch broadcasting. Branch broadcasting traces the address of direct branch instructions rather than producing E atoms.	

Device Type Identifier Register

The CSPFT_DEVTYPE register is read-only. It provides a debugger with information about the component when the part number field is not recognized. The debugger can then report this information.

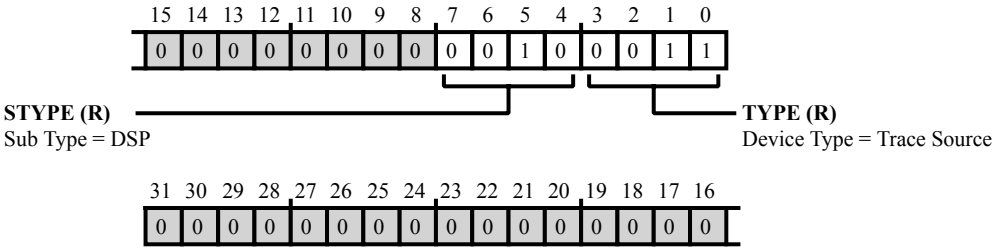

Figure 37-21: CSPFT_DEVTYPE Register Diagram

Table 37-25: CSPFT_DEVTYPE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	STYPE	Sub Type = DSP.
3:0 (R/NW)	TYPE	Device Type = Trace Source.

External Output Event Register

The CSPFT_EXTOUTEVR[n] register defines the event that controls the corresponding EXTOUT external output signal.

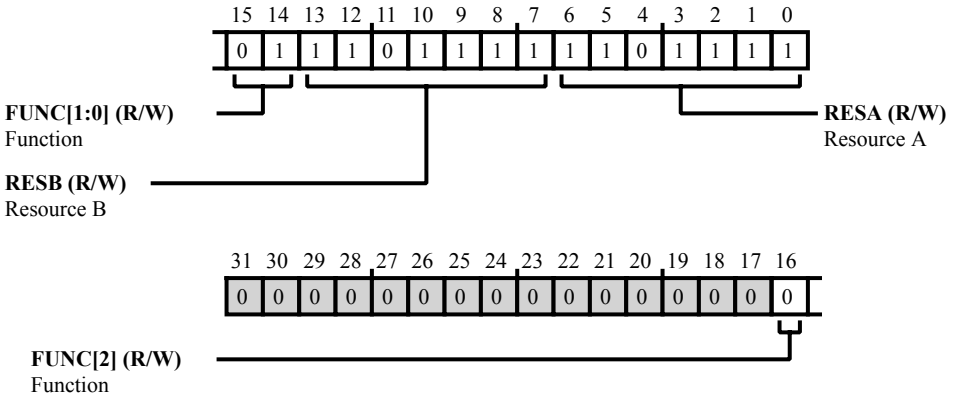

Figure 37-22: CSPFT_EXTOUTEVR[n] Register Diagram

Table 37-26: CSPFT_EXTOUTEVR[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. The CSPFT_EXTOUTEVR[n].FUNC bit field specifies the logical operation that combines the two resources that define the event.	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The CSPFT_EXTOUTEVR[n].RESB bit field specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_EXTOUTEVR[n].FUNC field (See CSPFT_EXTOUTEVR[n].RESA for list of possible values).	
6:0 (R/W)	RESA	Resource A.	

Table 37-26: CSPFT_EXTOUTEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		The `CSPFT_EXTOUTEVR[n].RESA` bit field specifies one of the two resources that can be combined by the logical operation specified in the `CSPFT_EXTOUTEVR[n].FUNC` field.	
		0	Single Addr Comparator 0
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at Zero
		65	Counter 1 at Zero
		66	Counter 2 at Zero
		67	Counter 3 at Zero

Table 37-26: CSPFT_EXTOUTEVR[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		88	Context ID Comparator 0
		89	Context ID Comparator 1
		90	Context ID Comparator 2
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

Hardware Feature Register

The CSPFT_HWFEAT register enables software to read the implementation defined configuration of the PTM, giving the number of each type of hardware resource. Each field indicates the number of instances of a particular resource, zero indicates that there are no implemented resources of that type.

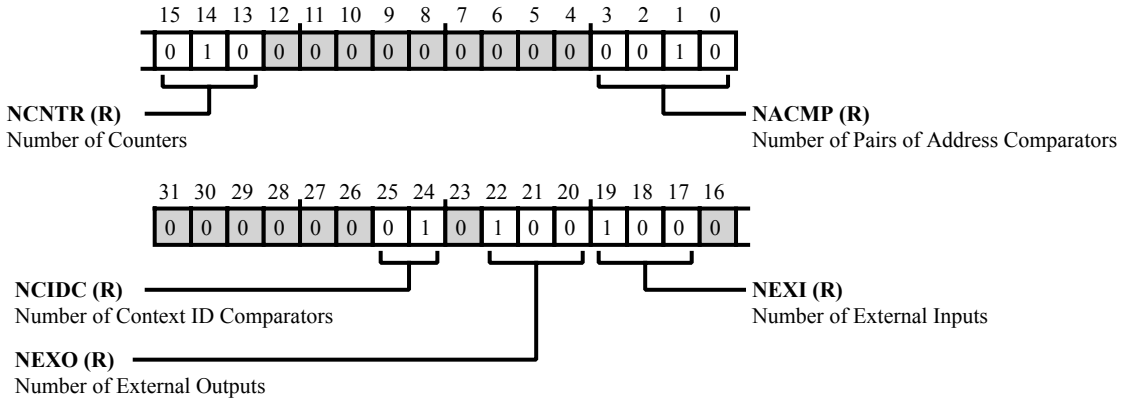

Figure 37-23: CSPFT_HWFEAT Register Diagram

Table 37-27: CSPFT_HWFEAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
25:24 (R/NW)	NCIDC	Number of Context ID Comparators. The CSPFT_HWFEAT.NCIDC bit field identifies the number of context ID comparators.
22:20 (R/NW)	NEXO	Number of External Outputs. The CSPFT_HWFEAT.NEXO bit field identifies the number of external outputs (up to four).
19:17 (R/NW)	NEXI	Number of External Inputs. The CSPFT_HWFEAT.NEXI bit field identifies the number of external inputs (up to four).
15:13 (R/NW)	NCNTR	Number of Counters. The CSPFT_HWFEAT.NCNTR bit field identifies the number of counters (up to four) that are configured using the counter registers.
3:0 (R/NW)	NACMP	Number of Pairs of Address Comparators. The CSPFT_HWFEAT.NACMP bit field identifies the number of pairs of address comparators as address range comparators (ARCs). In this case, two adjacent address comparators form the ARC, so you can use address comparators 1 and 2 to define the first ARC. An ARC matches when any instruction in the specified range is committed for execution, regardless of whether the instruction passes its condition code test.

Table 37-27: CSPFT_HWFEAT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		0	No Address Comparators
		1	1 Pair of Address Comparators
		2	2 Pairs of Address Comparators
		3	3 Pairs of Address Comparators
		4	4 Pairs of Address Comparators
		5	5 Pairs of Address Comparators
		6	6 Pairs of Address Comparators
		7	7 Pairs of Address Comparators
		8	8 Pairs of Address Comparators

Lock Access Register

The CSPFT_LAR register is used to provide lock and unlock access to all other CSPFT registers.

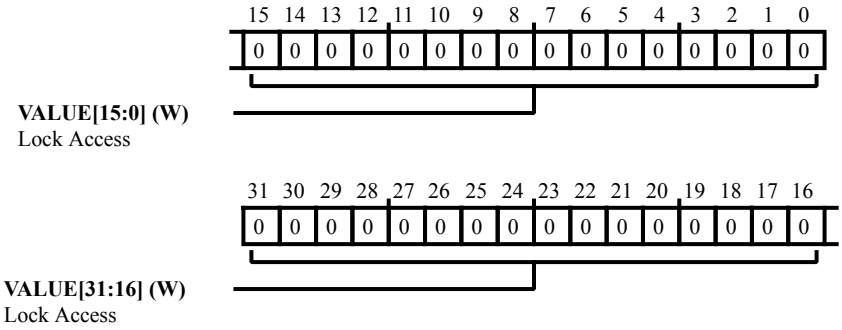

Figure 37-24: CSPFT_LAR Register Diagram

Table 37-28: CSPFT_LAR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (RX/W)	VALUE	Lock Access. Write 0xC5ACCE55 to the CSPFT_LAR.VALUE bit field to unlock. Write any other value to lock.

Lock Status Register

The CSPFT_LSR register is used to detect if the lock registers are implemented and if they are currently locked.

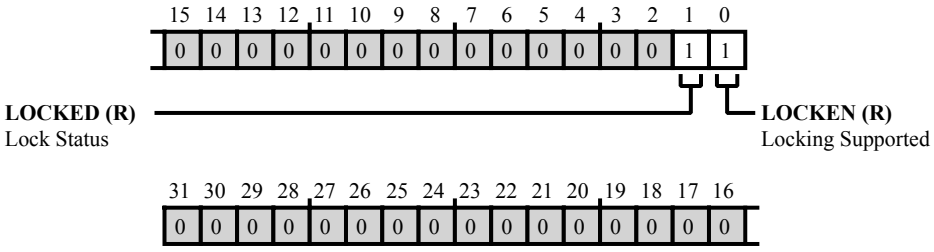

Figure 37-25: CSPFT_LSR Register Diagram

Table 37-29: CSPFT_LSR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
1 (R/NW)	LOCKED	Lock Status. The CSPFT_LSR.LOCKED bit indicates whether the PFT is locked.	
		0	Writes are permitted
		1	Locked. Writes are ignored
0 (R/NW)	LOCKEN	Locking Supported. The CSPFT_LSR.LOCKEN bit indicates whether the lock registers are implemented for this interface.	
		0	Locking is Not Required. This access is from an interface that ignores the lock registers.
		1	Locking is Required. This access is from an interface that requires the PFT to be unlocked.

Peripheral ID0 Register

The CSPFT_PID0 register holds peripheral identification information.

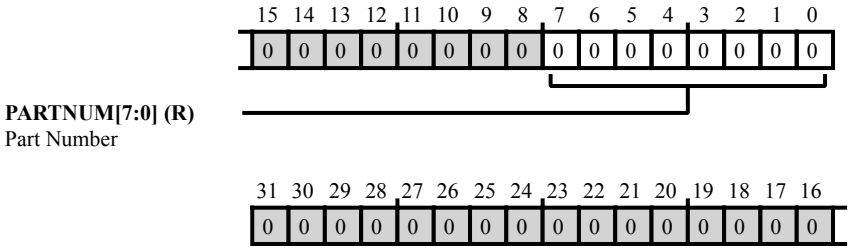

Figure 37-26: CSPFT_PID0 Register Diagram

Table 37-30: CSPFT_PID0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:0 (R/NW)	PARTNUM	Part Number. The CSPFT_PID0.PARTNUM bit field holds the peripheral identification number.

Peripheral ID1 Register

The CSPFT_PID1 register holds peripheral identification information.

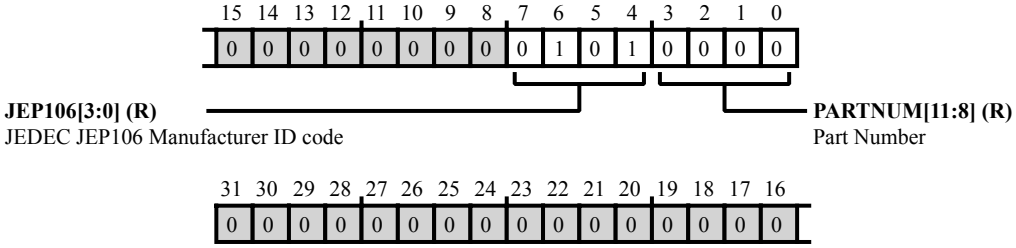

Figure 37-27: CSPFT_PID1 Register Diagram

Table 37-31: CSPFT_PID1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	JEP106	JEDEC JEP106 Manufacturer ID code.
3:0 (R/NW)	PARTNUM	Part Number. The CSPFT_PID1.PARTNUM bit field holds the peripheral identification number.

Peripheral ID2 Register

The CSPFT_PID2 register holds peripheral identification information.

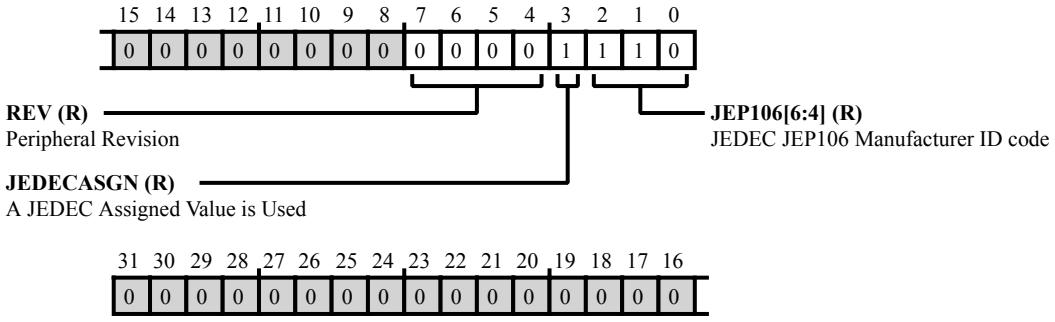

Figure 37-28: CSPFT_PID2 Register Diagram

Table 37-32: CSPFT_PID2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REV	Peripheral Revision.
3 (R/NW)	JEDECASGN	A JEDEC Assigned Value is Used. The CSPFT_PID2.JEDECASGN bit indicates that a JEDEC assigned value is used.
2:0 (R/NW)	JEP106	JEDEC JEP106 Manufacturer ID code.

Peripheral ID3 Register

The CSPFT_PID3 register holds peripheral identification information.

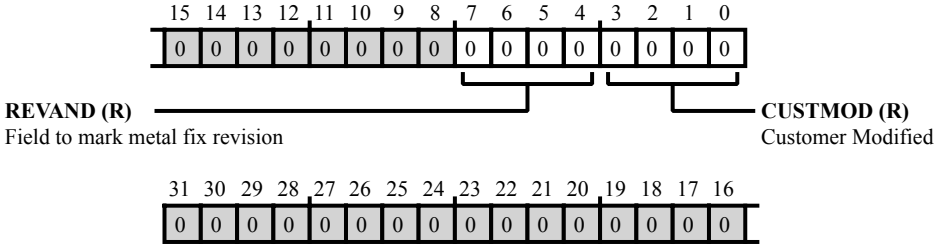

Figure 37-29: CSPFT_PID3 Register Diagram

Table 37-33: CSPFT_PID3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	REVAND	Field to mark metal fix revision.
3:0 (R/NW)	CUSTMOD	Customer Modified.

Peripheral ID4 Register

The CSPFT_PID4 register holds peripheral identification information.

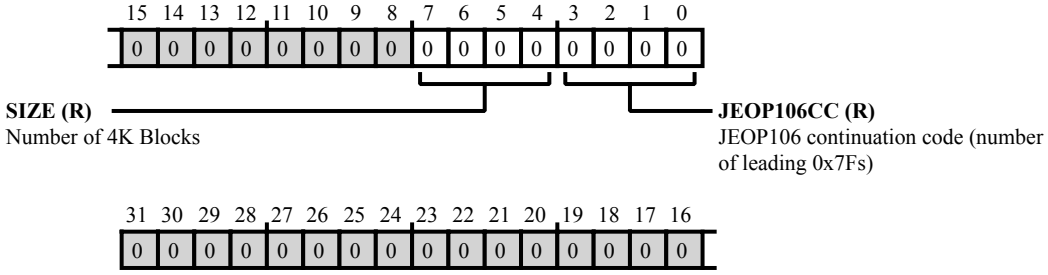

Figure 37-30: CSPFT_PID4 Register Diagram

Table 37-34: CSPFT_PID4 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
7:4 (R/NW)	SIZE	Number of 4K Blocks. The CSPFT_PID4.SIZE bit field contains the size of the component in 4K chunks minus 1 (for example 0=4K).
3:0 (R/NW)	JEOP106CC	JEOP106 continuation code (number of leading 0x7Fs).

Status Register

The CSPFT_STAT register provides information about the current status of the trace and trigger logic.

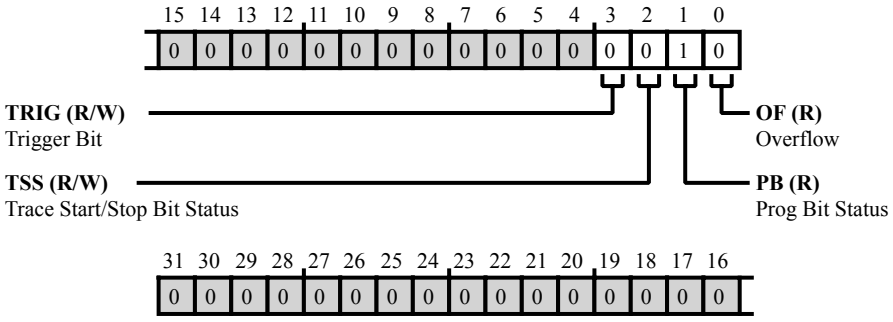

Figure 37-31: CSPFT_STAT Register Diagram

Table 37-35: CSPFT_STAT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
3 (R/W)	TRIG	Trigger Bit. The CSPFT_STAT.TRIG bit is set when the trigger occurs, and prevents the trigger from being output until the CSPFT is next programmed. This bit is reset when the CSPFT_CTL.PB bit transitions from 1 to 0.
2 (R/W)	TSS	Trace Start/Stop Bit Status. The CSPFT_STAT.TSS bit holds the current status of the trace start/stop resource. If = 1, it indicates that a trace start address has been matched, without a corresponding trace stop address match. This bit =0 when trace is restarted (the CSPFT_CTL.PB bit transitions from 1 to 0).
1 (R/NW)	PB	Prog Bit Status. The CSPFT_STAT.PB bit indicates the current effective value of the CSPFT_CTL.PB bit. The program must wait for this bit to =1 before programming the CSPFT. (See the CSPFT_CTL.PB bit description).
0 (R/NW)	OF	Overflow. If the CSPFT_STAT.OF bit is =1, there is an overflow. This bit is cleared =0 when the trace is restarted (CSPFT_CTL.PB transitions from 1 to 0)

Synchronization Frequency Register

The CSPFT_SYNCFR register holds the trace synchronization frequency value.

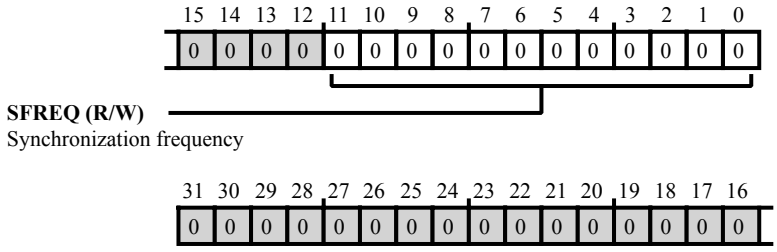

Figure 37-32: CSPFT_SYNCFR Register Diagram

Table 37-36: CSPFT_SYNCFR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
11:0 (R/W)	SFREQ	Synchronization frequency. The CSPFT_SYNCFR.SFREQ bit field is the number of 128-byte blocks of trace data after which you want to drop an address synchronization packet. If the circular buffer size is 16k, ensure that there are a few A-syncs in the buffer, so setting this to 16 means that every 2k there is an A-Sync packet.

TraceEnable Control Register

The CSPFT_TECTL register controls the start stop logic, whether resources specified are used for include or exclude, and specifies the address range comparators to use.

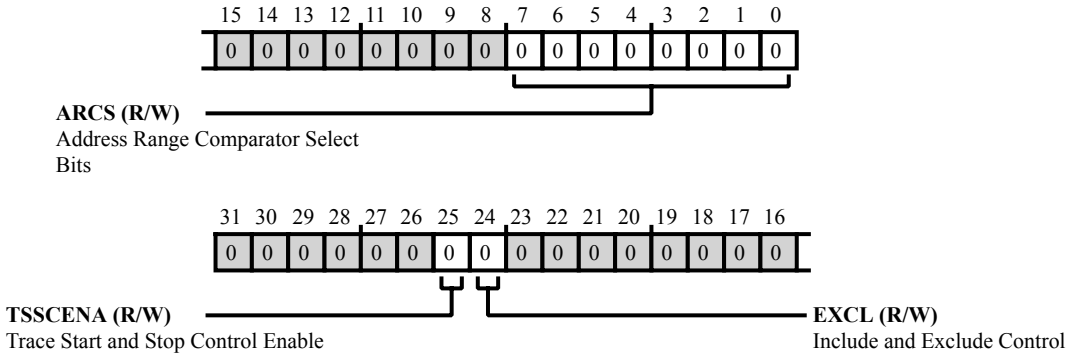

Figure 37-33: CSPFT_TECTL Register Diagram

Table 37-37: CSPFT_TECTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
25 (R/W)	TSSCENA	Trace Start and Stop Control Enable.	
		0	Tracing is not affected by the trace start/stop logic
		1	Tracing is controlled by the trace on and off address comparators
24 (R/W)	EXCL	Include and Exclude Control.	
		0	Include. The specified address range comparators indicate the regions where tracing can occur. When outside the region trace is prevented.
		1	Exclude. The specified address range comparators indicate regions to be excluded from the trace. When outside the range tracing is enabled.
7:0 (R/W)	ARCS	Address Range Comparator Select Bits. When a bit in the CSPFT_TECTL.ARCS bit field is set to 1, it selects an address range comparator, for include/exclude control.	

TraceEnable Event Register

The CSPFT_TEEVENT register defines the TraceEnable enabling event.

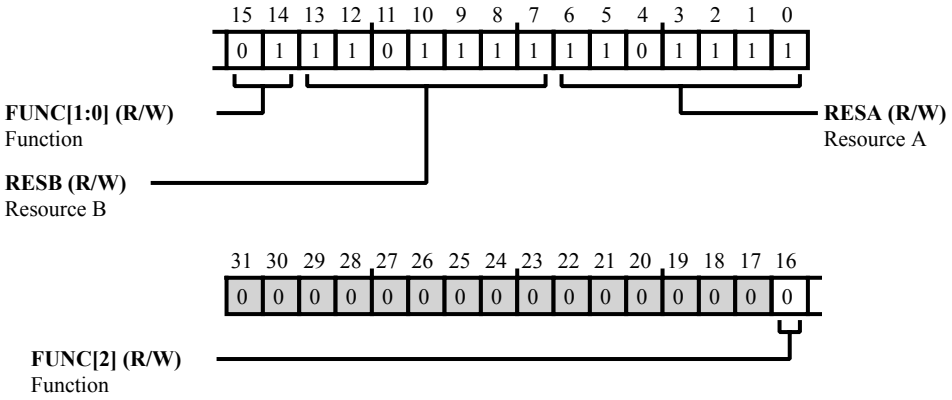

Figure 37-34: CSPFT_TEEVENT Register Diagram

Table 37-38: CSPFT_TEEVENT Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. The CSPFT_TEEVENT.FUNC bit field specifies the logical operation that combines the two resources that define the event.	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. The CSPFT_TEEVENT.RESB bit field specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_TEEVENT.FUNC field. (See CSPFT_TEEVENT.RESA for list of possible values).	
6:0 (R/W)	RESA	Resource A. The CSPFT_TEEVENT.RESA bit field specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_TEEVENT.FUNC field.	
		0	Single Addr Comparator 0

Table 37-38: CSPFT_TEEVENT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at Zero
		65	Counter 1 at Zero
		66	Counter 2 at Zero
		67	Counter 3 at Zero
		88	Context ID Comparator 0
		89	Context ID Comparator 1
		90	Context ID Comparator 2

Table 37-38: CSPFT_TEEVENT Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

CoreSight Trace ID Register

The CSPFT_TRACEIDR register defines the 7-bit trace ID, for output to the trace bus.

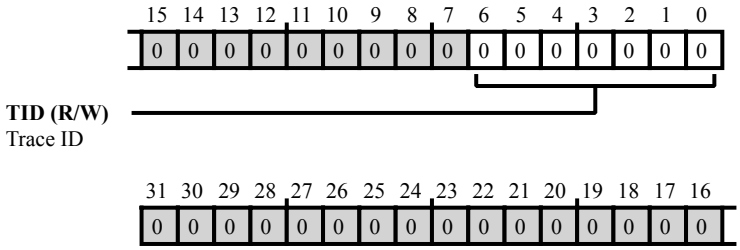

Figure 37-35: CSPFT_TRACEIDR Register Diagram

Table 37-39: CSPFT_TRACEIDR Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
6:0 (R/W)	TID	Trace ID. Defines the 7-bit trace ID, for output to the trace bus. Used in systems where multiple trace sources are present and tracing simultaneously. For example, when outputs trace onto the AMBA 3 Advanced Trace Bus, a unique ID is required for each trace source.

Trigger Event Register

The CSPFT_TRIGGER register defines the event that controls the trigger. This event creates the trigger output signal that is in the ATCLK domain.

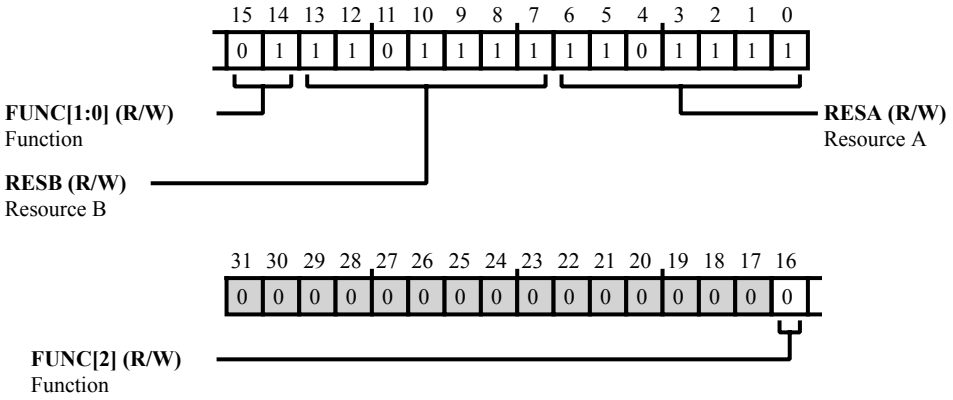

Figure 37-36: CSPFT_TRIGGER Register Diagram

Table 37-40: CSPFT_TRIGGER Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
16:14 (R/W)	FUNC	Function. Specifies the logical operation that combines the two resources that define the event	
		0	A
		1	NOT(A)
		2	A AND B
		3	NOT(A) AND B
		4	NOT(A) AND NOT(B)
		5	A OR B
		6	NOT(A) OR B
		7	NOT(A) OR NOT(B)
13:7 (R/W)	RESB	Resource B. Specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_TRIGGER.FUNC field. (See CSPFT_TRIGGER.RESA for list of possible values.)	
6:0 (R/W)	RESA	Resource A. Specifies one of the two resources that can be combined by the logical operation specified in the CSPFT_TRIGGER.FUNC field	
		0	Single Addr Comparator 0

Table 37-40: CSPFT_TRIGGER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		1	Single Addr Comparator 1
		2	Single Addr Comparator 2
		3	Single Addr Comparator 3
		4	Single Addr Comparator 4
		5	Single Addr Comparator 5
		6	Single Addr Comparator 6
		7	Single Addr Comparator 7
		8	Single Addr Comparator 8
		9	Single Addr Comparator 9
		10	Single Addr Comparator 10
		11	Single Addr Comparator 11
		12	Single Addr Comparator 12
		13	Single Addr Comparator 13
		14	Single Addr Comparator 14
		15	Single Addr Comparator 15
		16	Addr Range Comparator 0
		17	Addr Range Comparator 1
		18	Addr Range Comparator 2
		19	Addr Range Comparator 3
		20	Addr Range Comparator 4
		21	Addr Range Comparator 5
		22	Addr Range Comparator 6
		23	Addr Range Comparator 7
		64	Counter 0 at Zero
		65	Counter 1 at Zero
		66	Counter 2 at Zero
		67	Counter 3 at Zero
		88	Context ID Comparator 0
		89	Context ID Comparator 1
		90	Context ID Comparator 2

Table 37-40: CSPFT_TRIGGER Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		95	TraceEnable Start/Stop Resource 0 or 1
		96	External Inputs 0
		97	External Inputs 1
		98	External Inputs 2
		99	External Inputs 3
		110	Trace Prohibited
		111	Always TRUE

TraceEnable Start/Stop Control Register

The CSPFT_TSSCTL register specifies the single address comparators that hold the trace start and stop addresses.

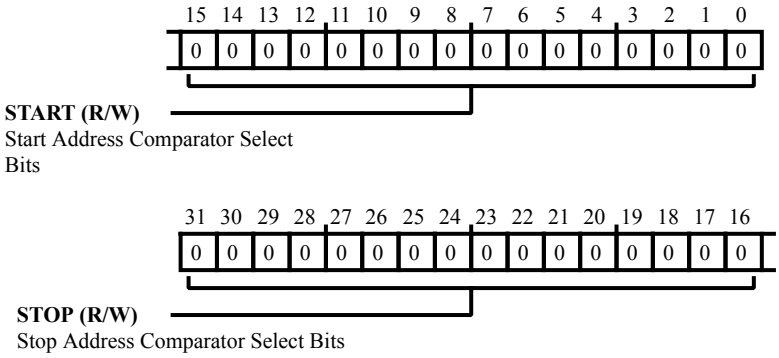

Figure 37-37: CSPFT_TSSCTL Register Diagram

Table 37-41: CSPFT_TSSCTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration	
31:16 (R/W)	STOP	Stop Address Comparator Select Bits. When a bit is set to 1, it selects a single address comparator as a stop address for the TraceEnable start/stop block.	
		0	disabled
		1	Address Comparator 1
		2	Address Comparator 2
		4	Address Comparator 3
		8	Address Comparator 4
		16	Address Comparator 5
		32	Address Comparator 6
		64	Address Comparator 7
		128	Address Comparator 8
		256	Address Comparator 9
		512	Address Comparator 10
		1024	Address Comparator 11
		2048	Address Comparator 12
		4096	Address Comparator 13
		8192	Address Comparator 14
		16384	Address Comparator 15

Table 37-41: CSPFT_TSSCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
		32768	Address Comparator 16
15:0 (R/W)	START	Start Address Comparator Select Bits. When a bit is set to 1, it selects a single address comparator as a start address for the TraceEnable start/stop block.	
		0	Disabled
		1	Address Comparator 1
		2	Address Comparator 2
		4	Address Comparator 3
		8	Address Comparator 4
		16	Address Comparator 5
		32	Address Comparator 6
		64	Address Comparator 7
		128	Address Comparator 8
		256	Address Comparator 9
		512	Address Comparator 10
		1024	Address Comparator 11
		2048	Address Comparator 12
		4096	Address Comparator 13
		8192	Address Comparator 14
		16384	Address Comparator 15
		32768	Address Comparator 16

ADSP-BF70x TAPC Register Descriptions

TAPC (TAPC) contains the following registers.

Table 37-42: ADSP-BF70x TAPC Register List

Name	Description
TAPC_DBGCTL	Debug Control
TAPC_IDCODE	IDCODE Register
TAPC_SDBGKEY0	Secure Debug Key 0 Register
TAPC_SDBGKEY1	Secure Debug Key 1 Register

Table 37-42: ADSP-BF70x TAPC Register List (Continued)

Name	Description
TAPC_SDBGKEY2	Secure Debug Key 2 Register
TAPC_SDBGKEY3	Secure Debug Key 3 Register
TAPC_SDBGKEY_CTL	Secure Debug Key Control Register
TAPC_SDBGKEY_STAT	Secure Debug Key Status Register
TAPC_USERCODE	USERCODE Register

Debug Control

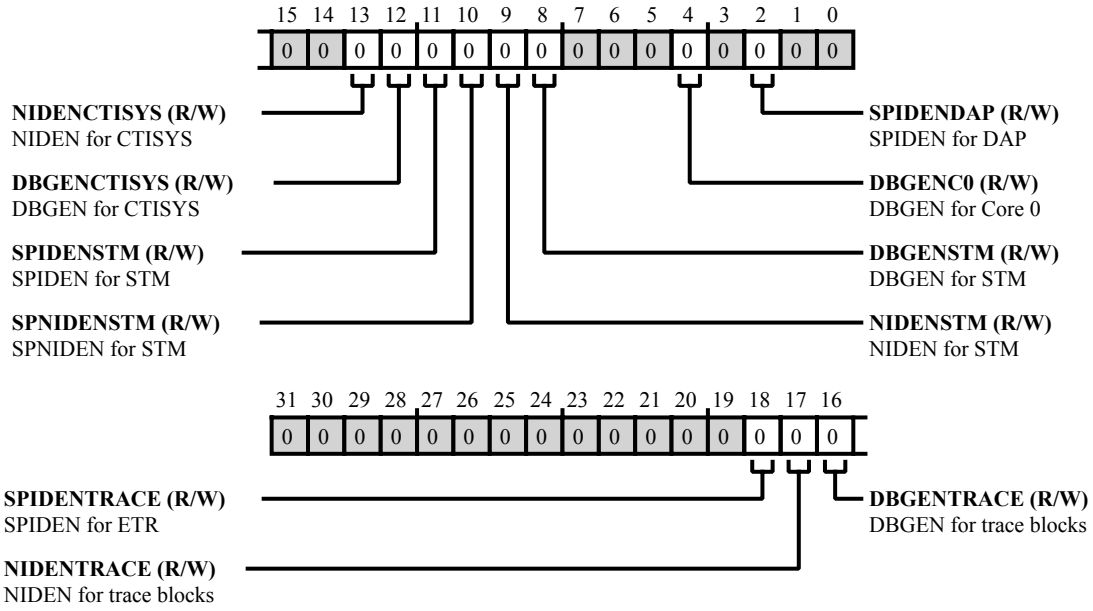

Figure 37-38: TAPC_DBGCTL Register Diagram

Table 37-43: 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
10 (R/W)	SPNIDENSTM	SPNIDEN for STM. SPINIDEN for STM
9	NIDENSTM	NIDEN for STM.

Table 37-43: TAPC_DBGCTL Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration
(R/W)		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

IDCODE Register

The TAPC_IDCODE register holds the IDCODE.

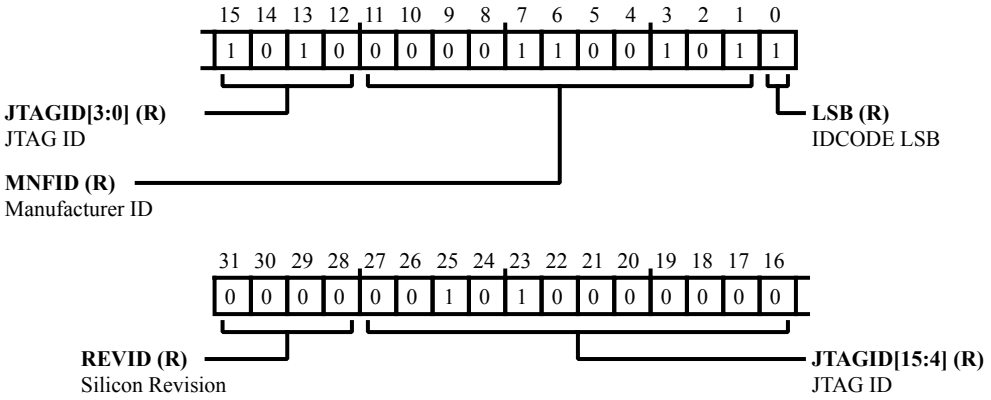

Figure 37-39: TAPC_IDCODE Register Diagram

Table 37-44: TAPC_IDCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:28 (R/NW)	REVID	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.

Secure Debug Key 0 Register

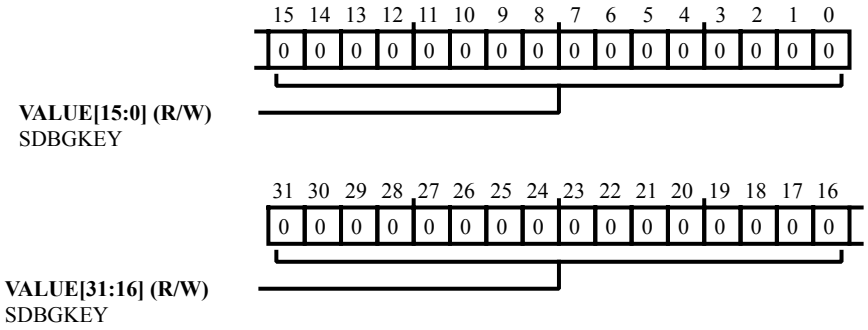

Figure 37-40: TAPC_SDBGKEY0 Register Diagram

Table 37-45: TAPC_SDBGKEY0 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY.

Secure Debug Key 1 Register

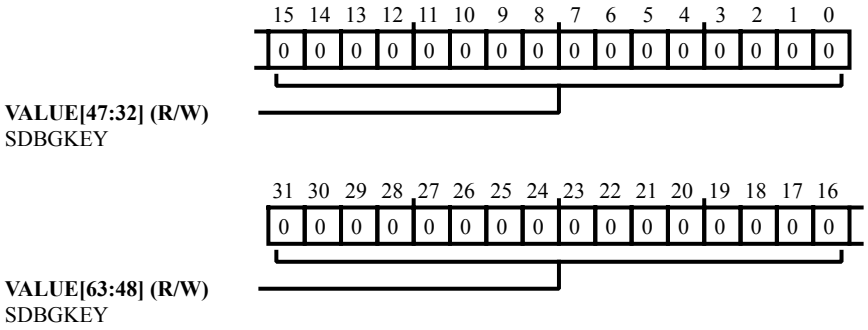

Figure 37-41: TAPC_SDBGKEY1 Register Diagram

Table 37-46: TAPC_SDBGKEY1 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY.

Secure Debug Key 2 Register

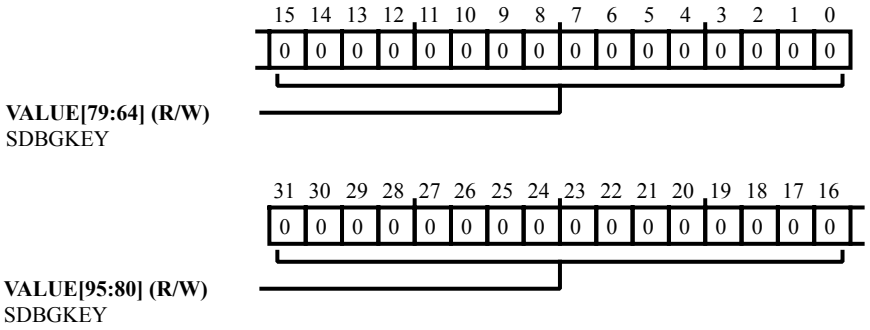

Figure 37-42: TAPC_SDBGKEY2 Register Diagram

Table 37-47: TAPC_SDBGKEY2 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY.

Secure Debug Key 3 Register

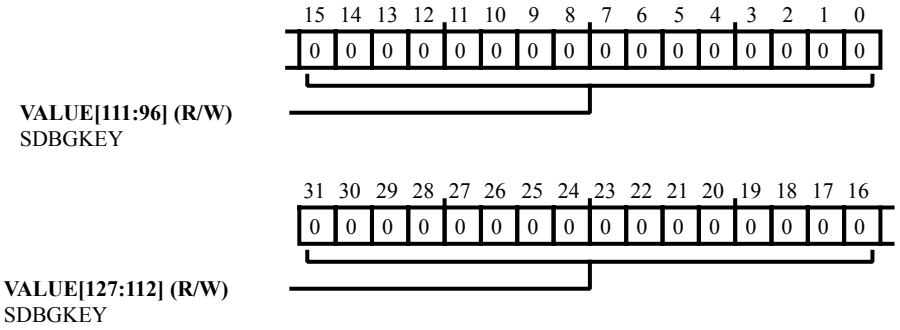

Figure 37-43: TAPC_SDBGKEY3 Register Diagram

Table 37-48: TAPC_SDBGKEY3 Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	VALUE	SDBGKEY.

Secure Debug Key Control Register

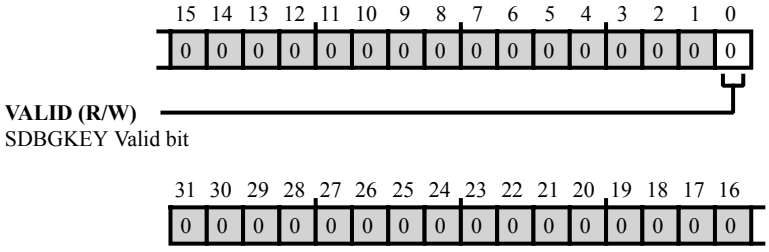

Figure 37-44: TAPC_SDBGKEY_CTL Register Diagram

Table 37-49: TAPC_SDBGKEY_CTL Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
0 (R/W)	VALID	SDBGKEY Valid bit.

Secure Debug Key Status Register

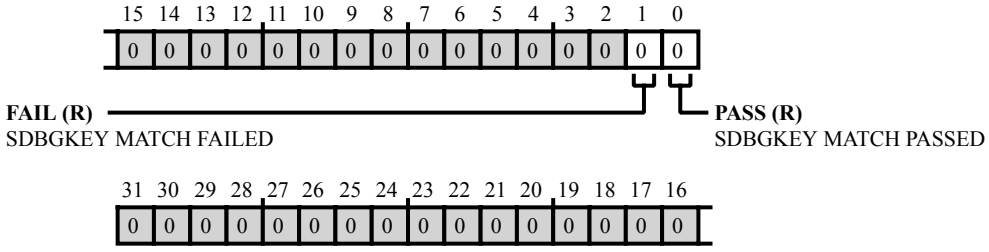

Figure 37-45: TAPC_SDBGKEY_STAT Register Diagram

Table 37-50: 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

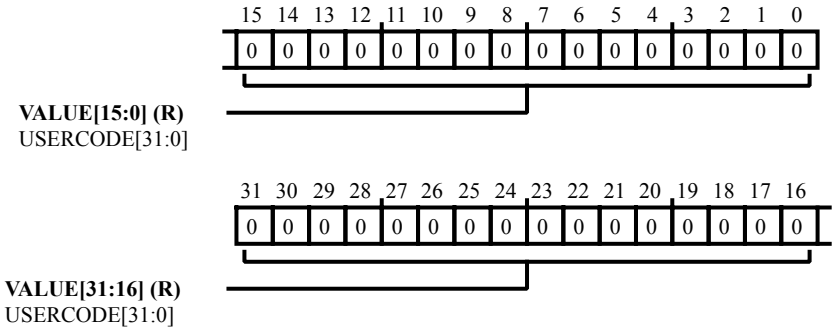

Figure 37-46: TAPC_USERCODE Register Diagram

Table 37-51: TAPC_USERCODE Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/NW)	VALUE	USERCODE[31:0].

38 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.

ADSP-BF70x 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 38-1: ADSP-BF70x SWU Register List

Name	Description
SWU_CNT[n]	Count Register n
SWU_CTL[n]	Control Register n

Table 38-1: ADSP-BF70x SWU Register List (Continued)

Name	Description
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

ADSP-BF70x SWU Interrupt List

Table 38-2: ADSP-BF70x SWU Interrupt List

Interrupt ID	Name	Description	Sensitivity	DMA Channel
97	SWU0_EVT	SWU0 Event (L1)	None	
98	SWU1_EVT	SWU1 Event (Core L2)	None	
99	SWU2_EVT	SWU2 Event (DMA L2)	None	
100	SWU3_EVT	SWU3 Event (MMR)	None	
101	SWU4_EVT	SWU4 Event (DMC)	None	
102	SWU5_EVT	SWU5 Event (SMC)	None	
103	SWU6_EVT	SWU6 Event (SPIF)	None	
104	SWU7_EVT	SWU7 Event (OTP)	None	

ADSP-BF70x SWU Trigger List

Table 38-3: ADSP-BF70x SWU Trigger List Masters

Trigger ID	Name	Description	Sensitivity
50	SWU0_EVT	SWU0 Event (L1)	None
51	SWU1_EVT	SWU1 Event (Core L2)	None
52	SWU2_EVT	SWU2 Event (DMA L2)	None
53	SWU3_EVT	SWU3 Event (MMR)	None
54	SWU4_EVT	SWU4 Event (DMC)	None
55	SWU5_EVT	SWU5 Event (SMC)	None

Table 38-3: ADSP-BF70x SWU Trigger List Masters (Continued)

Trigger ID	Name	Description	Sensitivity
56	SWU6_EVT	SWU6 Event (SPIF)	None
57	SWU7_EVT	SWU7 Event (OTP)	None
58	SWU0_DBG	SWU0 Debug (L1)	Edge
59	SWU1_DBG	SWU1 Debug (Core L2)	Edge
60	SWU2_DBG	SWU2 Debug (DMA L2)	Edge
61	SWU3_DBG	SWU3 Debug (MMR)	Edge
62	SWU4_DBG	SWU4 Debug (DMC)	Edge
63	SWU5_DBG	SWU5 Debug (SMC)	Edge
64	SWU6_DBG	SWU6 Debug (SPIF)	Edge
65	SWU7_DBG	SWU7 Debug (OTP)	Edge

Table 38-4: ADSP-BF70x SWU Trigger List Slaves

Trigger ID	Name	Description	Sensitivity
51	SWU0_EN	SWU0 Enable	Pulse
52	SWU1_EN	SWU1 Enable	Pulse
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
58	SWU7_EN	SWU7 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 Flow Diagram

The *SWU Logical Flow* diagram shows the logical program flow of the SWU.

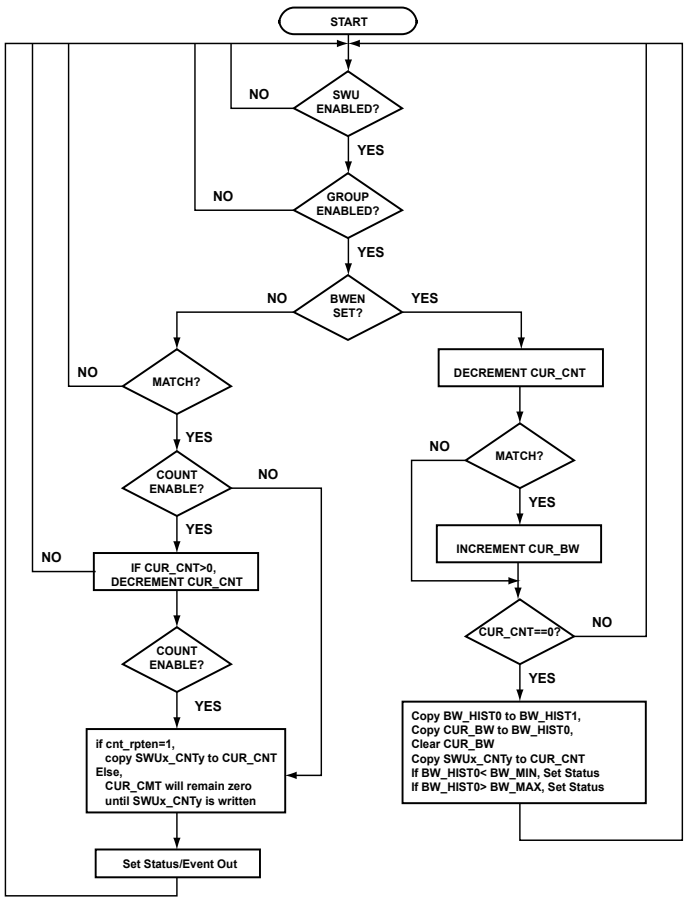

Figure 38-1: SWU Logical Flow

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.

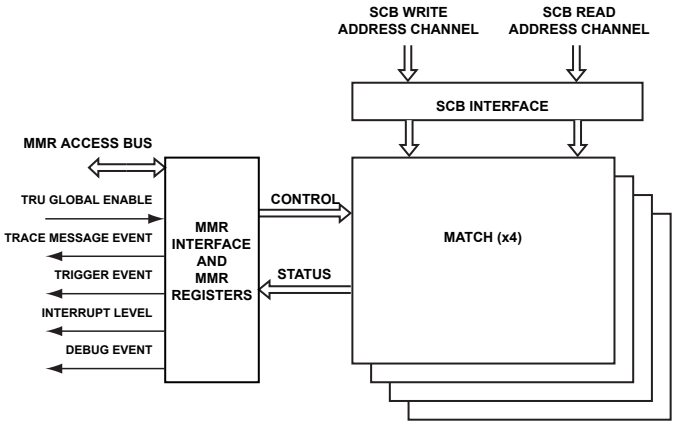

Figure 38-2: System Watchpoint Unit Top-Level Block Diagram

System Crossbar 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 Block

The SWU MMR block contains the peripheral bus interface and the SWU MMR registers. It also merges all interrupts 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
4. Debug

SWU Interrupts

All interrupts 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 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

The SWU is used by programming the appropriate registers. Each control register is used to configure various 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

Supplemental registers such as the lower and upper address boundaries are also configured before enabling.

Once the SWU has been enabled and monitoring conditions are met, events are generated if the SWU was configured to do so.

The global status register can be read to observe the status of the units.

SWU Mode Configuration

The following sections show the steps for configuring the SWU into 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.

ADSP-BF70x SWU Register Descriptions

System Watchpoint Unit (SWU) contains the following registers.

Table 38-5: ADSP-BF70x 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

Table 38-5: ADSP-BF70x SWU Register List (Continued)

Name	Description
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.

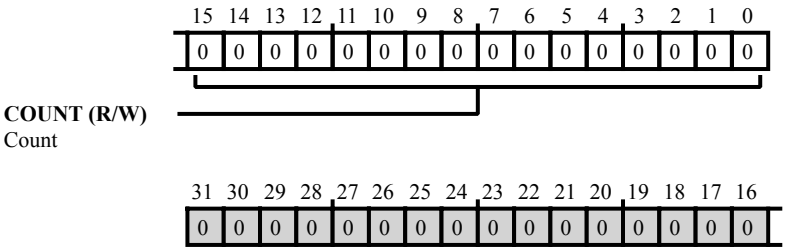

Figure 38-3: SWU_CNT[n] Register Diagram

Table 38-6: 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.

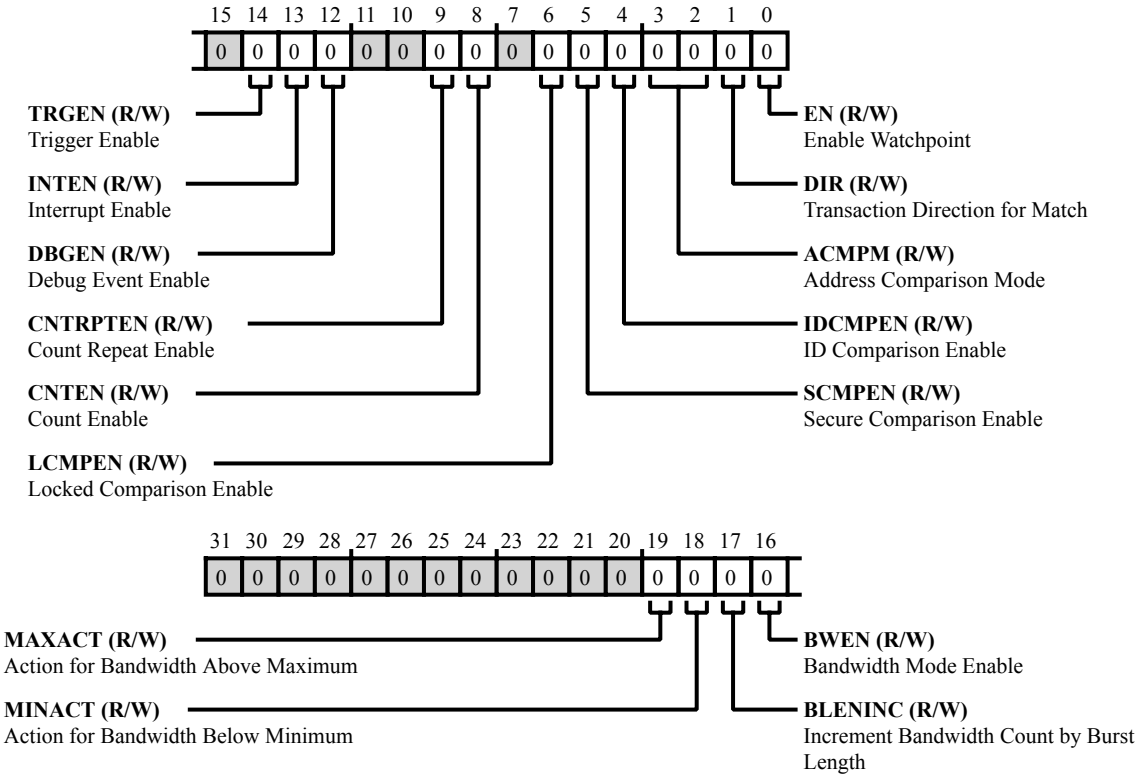

Figure 38-4: SWU_CTL[n] Register Diagram

Table 38-7: 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

Table 38-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
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

Table 38-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
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
12 (R/W)	DBGEN	Debug Event Enable. Each SWU_CTL[n].DBGEN bit controls debug event comparison for a watchpoint group, permitting matches based on debug status.	
		0	Disable
		1	Enable
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
6 (R/W)	LCMPEN	Locked Comparison Enable. Each SWU_CTL[n].LCMPEN bit controls locked comparison operation of an SWU watchpoint group, permitting matches based on lock status.	
		0	Match on all transaction
		1	Match only locked transactions

Table 38-7: SWU_CTL[n] Register Fields (Continued)

Bit No. (Access)	Bit Name	Description/Enumeration	
5 (R/W)	SCMPEN	Secure Comparison Enable. Each SWU_CTL[n].SCMPEN bit controls secure transaction comparison operation of an SWU watchpoint group, permitting matches based on transaction security.	
		0	Match on all transaction
		1	Match only secure transactions
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
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.

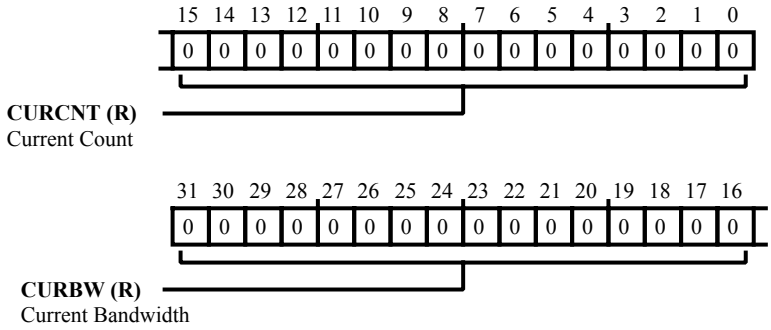

Figure 38-5: SWU_CUR[n] Register Diagram

Table 38-8: 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.

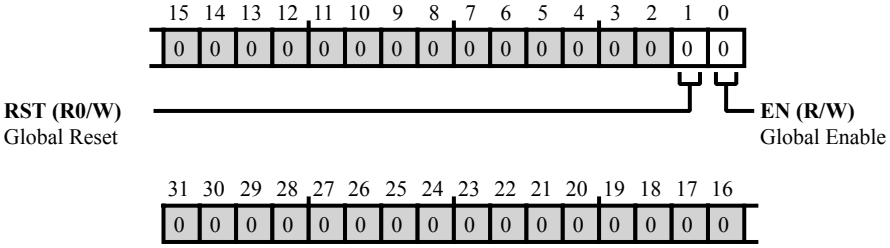

Figure 38-6: SWU_GCTL Register Diagram

Table 38-9: 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.

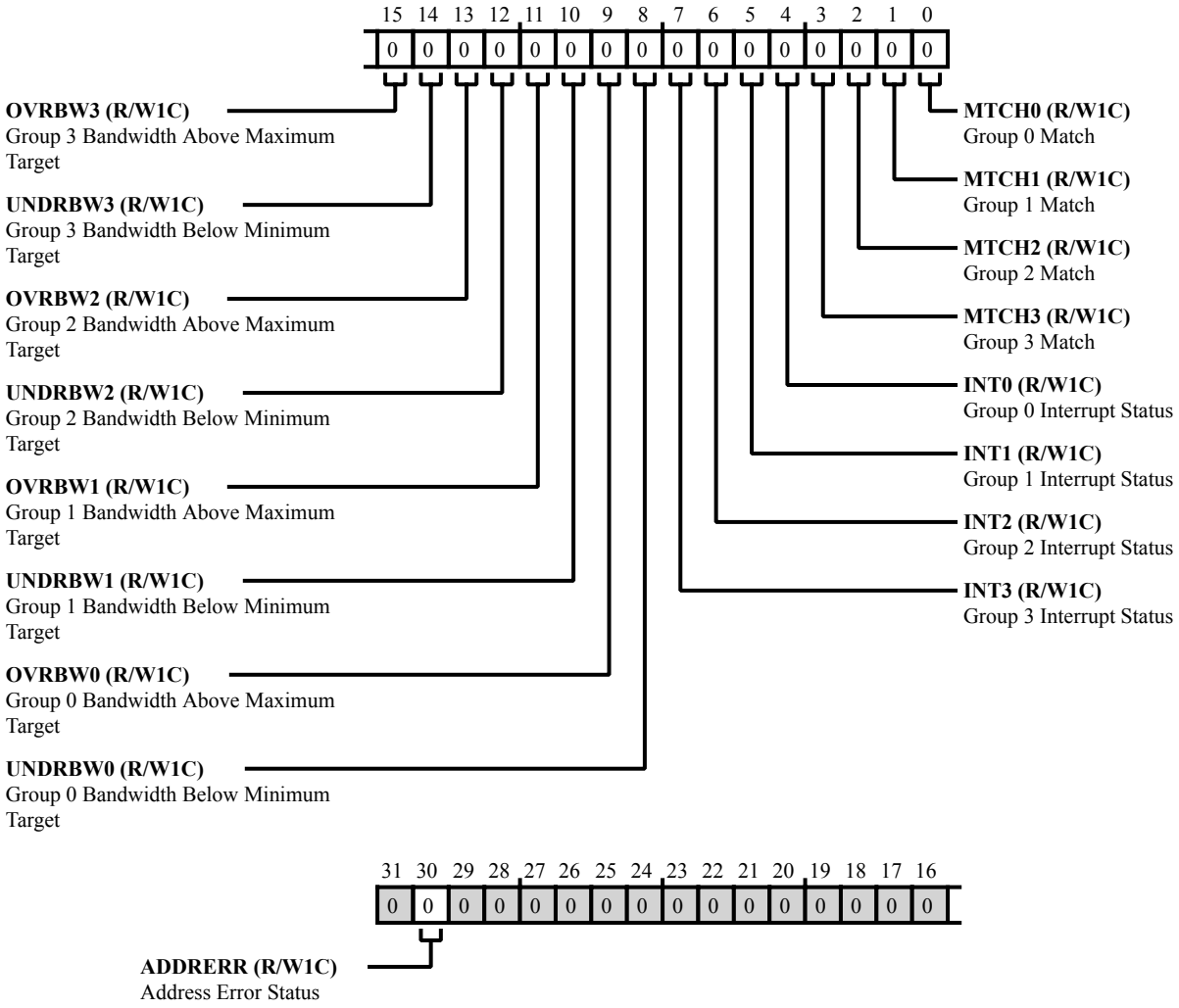

Figure 38-7: SWU_GSTAT Register Diagram

Table 38-10: 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 38-10: 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 watchpoint 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 38-10: 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 38-10: 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.

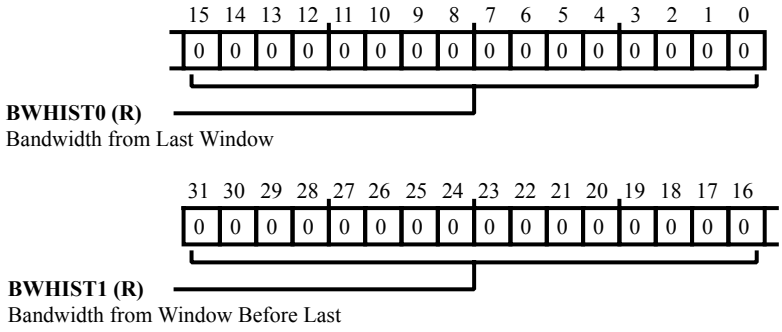

Figure 38-8: SWU_HIST[n] Register Diagram

Table 38-11: 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.

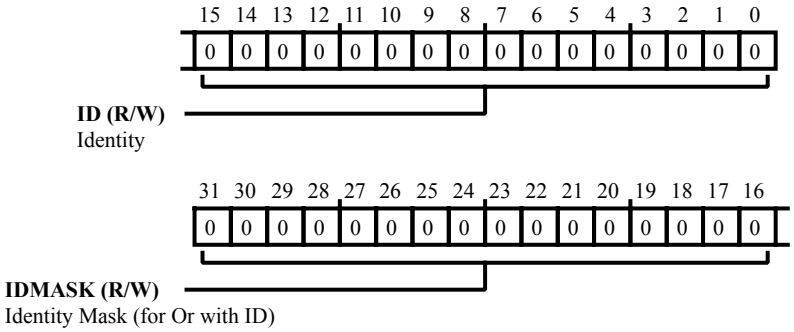

Figure 38-9: SWU_ID[n] Register Diagram

Table 38-12: 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.

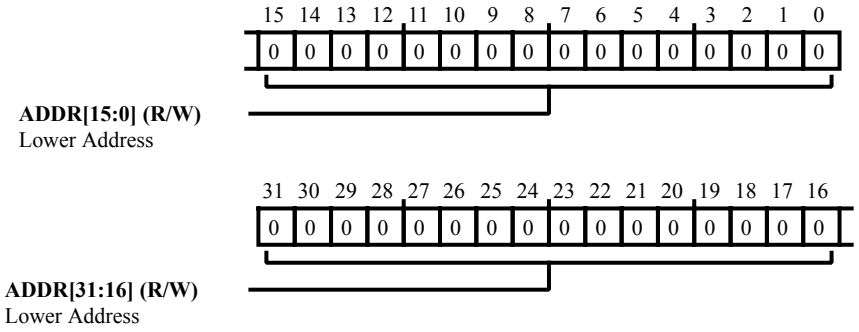

Figure 38-10: SWU_LA[n] Register Diagram

Table 38-13: 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.

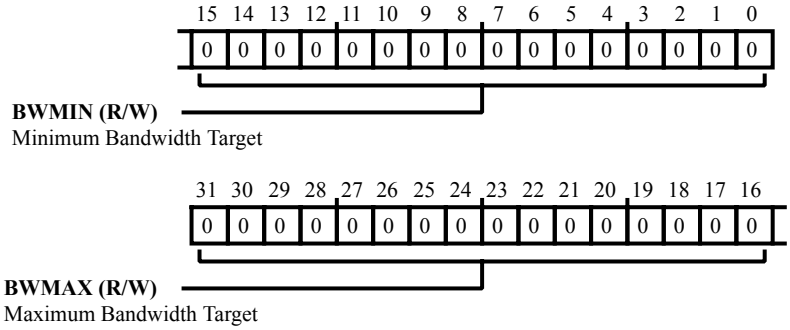

Figure 38-11: SWU_TARG[n] Register Diagram

Table 38-14: 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.

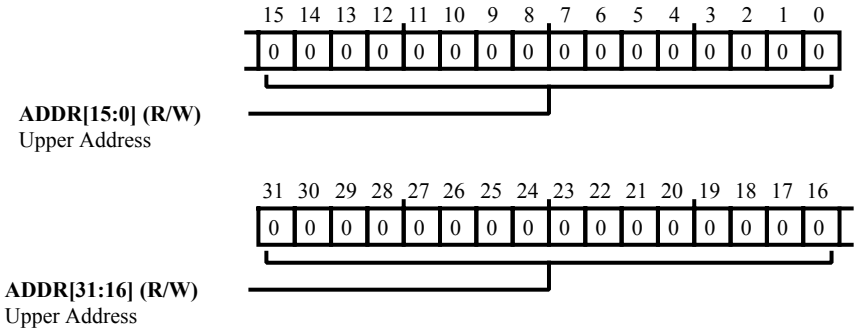

Figure 38-12: SWU_UA[n] Register Diagram

Table 38-15: SWU_UA[n] Register Fields

Bit No. (Access)	Bit Name	Description/Enumeration
31:0 (R/W)	ADDR	Upper Address.

Appendix A ADSP-BF70x Register List

This appendix lists Memory Mapped Register address and register names. The modules are presented in alphabetical order.

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C200	CAN0_MC1	CAN0 Mailbox Configuration 1 Register	0x00000000
0x2006C204	CAN0_MD1	CAN0 Mailbox Direction 1 Register	0x000000FF
0x2006C208	CAN0_TRS1	CAN0 Transmission Request Set 1 Register	0x00000000
0x2006C20C	CAN0_TRR1	CAN0 Transmission Request Reset 1 Register	0x00000000
0x2006C210	CAN0_TA1	CAN0 Transmission Acknowledge 1 Register	0x00000000
0x2006C214	CAN0_AA1	CAN0 Abort Acknowledge 1 Register	0x00000000
0x2006C218	CAN0_RMP1	CAN0 Receive Message Pending 1 Register	0x00000000
0x2006C21C	CAN0_RML1	CAN0 Receive Message Lost 1 Register	0x00000000
0x2006C220	CAN0_MBTIF1	CAN0 Mailbox Transmit Interrupt Flag 1 Register	0x00000000
0x2006C224	CAN0_MBRIF1	CAN0 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x2006C228	CAN0_MBIM1	CAN0 Mailbox Interrupt Mask 1 Register	0x00000000
0x2006C22C	CAN0_RFH1	CAN0 Remote Frame Handling 1 Register	0x00000000
0x2006C230	CAN0_OPSS1	CAN0 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x2006C240	CAN0_MC2	CAN0 Mailbox Configuration 2 Register	0x00000000
0x2006C244	CAN0_MD2	CAN0 Mailbox Direction 2 Register	0x00000000
0x2006C248	CAN0_TRS2	CAN0 Transmission Request Set 2 Register	0x00000000
0x2006C24C	CAN0_TRR2	CAN0 Transmission Request Reset 2 Register	0x00000000
0x2006C250	CAN0_TA2	CAN0 Transmission Acknowledge 2 Register	0x00000000
0x2006C254	CAN0_AA2	CAN0 Abort Acknowledge 2 Register	0x00000000
0x2006C258	CAN0_RMP2	CAN0 Receive Message Pending 2 Register	0x00000000
0x2006C25C	CAN0_RML2	CAN0 Receive Message Lost 2 Register	0x00000000
0x2006C260	CAN0_MBTIF2	CAN0 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x2006C264	CAN0_MBRIF2	CAN0 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x2006C268	CAN0_MBIM2	CAN0 Mailbox Interrupt Mask 2 Register	0x00000000
0x2006C26C	CAN0_RFH2	CAN0 Remote Frame Handling 2 Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C270	CAN0_OPSS2	CAN0 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x2006C280	CAN0_CLK	CAN0 Clock Register	0x00000000
0x2006C284	CAN0_TIMING	CAN0 Timing Register	0x00000000
0x2006C288	CAN0_DBG	CAN0 Debug Register	0x00000008
0x2006C28C	CAN0_STAT	CAN0 Status Register	0x00000080
0x2006C290	CAN0_CEC	CAN0 Error Counter Register	0x00000000
0x2006C294	CAN0_GIS	CAN0 Global CAN Interrupt Status Register	0x00000000
0x2006C298	CAN0_GIM	CAN0 Global CAN Interrupt Mask Register	0x00000000
0x2006C29C	CAN0_GIF	CAN0 Global CAN Interrupt Flag Register	0x00000000
0x2006C2A0	CAN0_CTL	CAN0 CAN Master Control Register	0x00000080
0x2006C2A4	CAN0_INT	CAN0 Interrupt Pending Register	0x00000000
0x2006C2AC	CAN0_MBTD	CAN0 Temporary Mailbox Disable Register	0x00000000
0x2006C2B0	CAN0_EWR	CAN0 Error Counter Warning Level Register	0x00006060
0x2006C2B4	CAN0_ESR	CAN0 Error Status Register	0x00000020
0x2006C2C4	CAN0_UCCNT	CAN0 Universal Counter Register	0x00000000
0x2006C2C8	CAN0_UCRC	CAN0 Universal Counter Reload/Capture Register	0x00000000
0x2006C2CC	CAN0_UCCNF	CAN0 Universal Counter Configuration Mode Register	0x00000000
0x2006C300	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C304	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C308	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C30C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C310	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C314	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C318	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C31C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C320	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C324	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C328	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C32C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C330	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C334	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C338	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C33C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C340	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C344	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C348	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C34C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C350	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C354	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C358	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C35C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C360	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C364	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C368	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C36C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C370	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C374	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C378	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C37C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C380	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C384	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C388	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C38C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C390	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C394	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C398	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C39C	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3A0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3A4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3A8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3AC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C3B0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3B4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3B8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3BC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3C0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3C4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3C8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3CC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3D0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3D4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3D8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3DC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3E0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3E4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3E8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3EC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3F0	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3F4	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C3F8	CAN0_AM[nn]L	CAN0 Acceptance Mask (L) Register	0x00000000
0x2006C3FC	CAN0_AM[nn]H	CAN0 Acceptance Mask (H) Register	0x00000000
0x2006C400	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C404	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C408	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C40C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C410	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C414	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C418	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C41C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C420	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C424	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C428	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C42C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C430	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C434	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C438	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C43C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C440	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C444	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C448	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C44C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C450	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C454	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C458	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C45C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C460	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C464	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C468	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C46C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C470	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C474	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C478	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C47C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C480	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C484	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C488	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C48C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C490	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C494	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C498	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C49C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C4A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C4A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C4A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C4AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C4B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C4B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C4B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C4BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C4C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C4C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C4C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C4CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C4D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C4D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C4D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C4DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C4E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C4E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C4E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C4EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C4F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C4F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C4F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C4FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C500	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C504	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C508	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C50C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C510	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C514	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C518	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C51C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C520	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C524	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C528	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C52C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C530	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C534	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C538	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C53C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C540	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C544	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C548	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C54C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C550	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C554	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C558	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C55C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C560	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C564	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C568	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C56C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C570	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C574	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C578	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C57C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C580	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C584	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C588	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C58C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C590	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C594	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C598	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C59C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C5A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C5A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C5A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C5AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C5B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C5B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C5B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C5BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C5C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C5C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C5C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C5CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C5D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C5D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C5D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C5DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C5E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C5E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C5E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C5EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C5F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C5F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C5F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C5FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C600	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C604	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C608	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C60C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C610	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C614	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C618	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C61C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C620	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C624	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C628	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C62C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C630	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C634	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C638	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C63C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C640	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C644	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C648	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C64C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C650	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C654	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C658	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C65C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C660	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C664	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C668	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C66C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C670	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C674	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C678	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C67C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C680	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C684	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C688	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C68C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C690	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C694	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C698	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C69C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C6A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C6A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C6A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C6AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C6B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C6B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C6B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C6BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C6C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C6C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C6C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C6CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C6D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C6D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C6D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C6DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C6E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C6E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C6E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C6EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C6F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C6F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C6F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C6FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C700	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C704	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C708	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C70C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C710	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C714	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C718	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C71C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C720	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C724	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C728	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C72C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C730	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C734	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C738	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C73C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C740	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C744	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C748	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C74C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C750	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C754	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C758	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C75C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C760	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C764	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C768	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C76C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C770	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C774	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C778	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C77C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C780	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C784	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C788	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C78C	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C790	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C794	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C798	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C79C	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C7A0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C7A4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C7A8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C7AC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C7B0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000

Table A-1: ADSP-BF70x CAN0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006C7B4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C7B8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C7BC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C7C0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C7C4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C7C8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C7CC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C7D0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C7D4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C7D8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C7DC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000
0x2006C7E0	CAN0_MB[nn]_DATA0	CAN0 Mailbox Word 0 Register	0x00000000
0x2006C7E4	CAN0_MB[nn]_DATA1	CAN0 Mailbox Word 1 Register	0x00000000
0x2006C7E8	CAN0_MB[nn]_DATA2	CAN0 Mailbox Word 2 Register	0x00000000
0x2006C7EC	CAN0_MB[nn]_DATA3	CAN0 Mailbox Word 3 Register	0x00000000
0x2006C7F0	CAN0_MB[nn]_LENGTH	CAN0 Mailbox Length Register	0x00000000
0x2006C7F4	CAN0_MB[nn]_TIMESTAMP	CAN0 Mailbox Time Stamp Register	0x00000000
0x2006C7F8	CAN0_MB[nn]_ID0	CAN0 Mailbox ID 0 Register	0x00000000
0x2006C7FC	CAN0_MB[nn]_ID1	CAN0 Mailbox ID 1 Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D200	CAN1_MC1	CAN1 Mailbox Configuration 1 Register	0x00000000
0x2006D204	CAN1_MD1	CAN1 Mailbox Direction 1 Register	0x000000FF
0x2006D208	CAN1_TRS1	CAN1 Transmission Request Set 1 Register	0x00000000
0x2006D20C	CAN1_TRR1	CAN1 Transmission Request Reset 1 Register	0x00000000
0x2006D210	CAN1_TA1	CAN1 Transmission Acknowledge 1 Register	0x00000000
0x2006D214	CAN1_AA1	CAN1 Abort Acknowledge 1 Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D218	CAN1_RMP1	CAN1 Receive Message Pending 1 Register	0x00000000
0x2006D21C	CAN1_RML1	CAN1 Receive Message Lost 1 Register	0x00000000
0x2006D220	CAN1_MBTIF1	CAN1 Mailbox Transmit Interrupt Flag 1 Register	0x00000000
0x2006D224	CAN1_MBRIF1	CAN1 Mailbox Receive Interrupt Flag 1 Register	0x00000000
0x2006D228	CAN1_MBIM1	CAN1 Mailbox Interrupt Mask 1 Register	0x00000000
0x2006D22C	CAN1_RFH1	CAN1 Remote Frame Handling 1 Register	0x00000000
0x2006D230	CAN1_OPSS1	CAN1 Overwrite Protection/Single Shot Transmission 1 Register	0x00000000
0x2006D240	CAN1_MC2	CAN1 Mailbox Configuration 2 Register	0x00000000
0x2006D244	CAN1_MD2	CAN1 Mailbox Direction 2 Register	0x00000000
0x2006D248	CAN1_TRS2	CAN1 Transmission Request Set 2 Register	0x00000000
0x2006D24C	CAN1_TRR2	CAN1 Transmission Request Reset 2 Register	0x00000000
0x2006D250	CAN1_TA2	CAN1 Transmission Acknowledge 2 Register	0x00000000
0x2006D254	CAN1_AA2	CAN1 Abort Acknowledge 2 Register	0x00000000
0x2006D258	CAN1_RMP2	CAN1 Receive Message Pending 2 Register	0x00000000
0x2006D25C	CAN1_RML2	CAN1 Receive Message Lost 2 Register	0x00000000
0x2006D260	CAN1_MBTIF2	CAN1 Mailbox Transmit Interrupt Flag 2 Register	0x00000000
0x2006D264	CAN1_MBRIF2	CAN1 Mailbox Receive Interrupt Flag 2 Register	0x00000000
0x2006D268	CAN1_MBIM2	CAN1 Mailbox Interrupt Mask 2 Register	0x00000000
0x2006D26C	CAN1_RFH2	CAN1 Remote Frame Handling 2 Register	0x00000000
0x2006D270	CAN1_OPSS2	CAN1 Overwrite Protection/Single Shot Transmission 2 Register	0x00000000
0x2006D280	CAN1_CLK	CAN1 Clock Register	0x00000000
0x2006D284	CAN1_TIMING	CAN1 Timing Register	0x00000000
0x2006D288	CAN1_DBG	CAN1 Debug Register	0x00000008
0x2006D28C	CAN1_STAT	CAN1 Status Register	0x00000080
0x2006D290	CAN1_CEC	CAN1 Error Counter Register	0x00000000
0x2006D294	CAN1_GIS	CAN1 Global CAN Interrupt Status Register	0x00000000
0x2006D298	CAN1_GIM	CAN1 Global CAN Interrupt Mask Register	0x00000000
0x2006D29C	CAN1_GIF	CAN1 Global CAN Interrupt Flag Register	0x00000000
0x2006D2A0	CAN1_CTL	CAN1 CAN Master Control Register	0x00000080

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D2A4	CAN1_INT	CAN1 Interrupt Pending Register	0x00000000
0x2006D2AC	CAN1_MBTD	CAN1 Temporary Mailbox Disable Register	0x00000000
0x2006D2B0	CAN1_EWR	CAN1 Error Counter Warning Level Register	0x00006060
0x2006D2B4	CAN1_ESR	CAN1 Error Status Register	0x00000020
0x2006D2C4	CAN1_UCCNT	CAN1 Universal Counter Register	0x00000000
0x2006D2C8	CAN1_UCRC	CAN1 Universal Counter Reload/Capture Register	0x00000000
0x2006D2CC	CAN1_UCCNF	CAN1 Universal Counter Configuration Mode Register	0x00000000
0x2006D300	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D304	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D308	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D30C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D310	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D314	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D318	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D31C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D320	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D324	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D328	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D32C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D330	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D334	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D338	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D33C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D340	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D344	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D348	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D34C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D350	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D354	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D358	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D35C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D360	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D364	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D368	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D36C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D370	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D374	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D378	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D37C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D380	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D384	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D388	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D38C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D390	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D394	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D398	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D39C	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3A0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3A4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3A8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3AC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3B0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3B4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3B8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3BC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3C0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3C4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3C8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3CC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3D0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3D4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3D8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D3DC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3E0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3E4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3E8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3EC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3F0	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3F4	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D3F8	CAN1_AM[nn]L	CAN1 Acceptance Mask (L) Register	0x00000000
0x2006D3FC	CAN1_AM[nn]H	CAN1 Acceptance Mask (H) Register	0x00000000
0x2006D400	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D404	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D408	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D40C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D410	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D414	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D418	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D41C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D420	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D424	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D428	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D42C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D430	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D434	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D438	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D43C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D440	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D444	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D448	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D44C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D450	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D454	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D458	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D45C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D460	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D464	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D468	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D46C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D470	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D474	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D478	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D47C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D480	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D484	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D488	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D48C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D490	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D494	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D498	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D49C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D4A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D4A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D4A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D4AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D4B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D4B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D4B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D4BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D4C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D4C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D4C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D4CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D4D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D4D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D4D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D4DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D4E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D4E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D4E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D4EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D4F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D4F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D4F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D4FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D500	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D504	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D508	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D50C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D510	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D514	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D518	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D51C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D520	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D524	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D528	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D52C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D530	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D534	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D538	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D53C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D540	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D544	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D548	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D54C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D550	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D554	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D558	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D55C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D560	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D564	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D568	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D56C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D570	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D574	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D578	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D57C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D580	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D584	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D588	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D58C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D590	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D594	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D598	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D59C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D5A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D5A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D5A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D5AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D5B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D5B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D5B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D5BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D5C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D5C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D5C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D5CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D5D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D5D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D5D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D5DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D5E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D5E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D5E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D5EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D5F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D5F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D5F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D5FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D600	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D604	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D608	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D60C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D610	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D614	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D618	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D61C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D620	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D624	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D628	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D62C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D630	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D634	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D638	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D63C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D640	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D644	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D648	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D64C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D650	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D654	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D658	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D65C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D660	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D664	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D668	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D66C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D670	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D674	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D678	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D67C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D680	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D684	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D688	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D68C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D690	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D694	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D698	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D69C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D6A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D6A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D6A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D6AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D6B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D6B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D6B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D6BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D6C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D6C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D6C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D6CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D6D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D6D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D6D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D6DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D6E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D6E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D6E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D6EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D6F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D6F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D6F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D6FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D700	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D704	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D708	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D70C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D710	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D714	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D718	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D71C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D720	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D724	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D728	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D72C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D730	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D734	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D738	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D73C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D740	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D744	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D748	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D74C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D750	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D754	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D758	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D75C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D760	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D764	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D768	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D76C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D770	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D774	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D778	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D77C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D780	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D784	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D788	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D78C	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D790	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D794	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D798	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D79C	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D7A0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D7A4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D7A8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D7AC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D7B0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D7B4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D7B8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D7BC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D7C0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D7C4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D7C8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D7CC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D7D0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000

Table A-2: ADSP-BF70x CAN1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2006D7D4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D7D8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D7DC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000
0x2006D7E0	CAN1_MB[nn]_DATA0	CAN1 Mailbox Word 0 Register	0x00000000
0x2006D7E4	CAN1_MB[nn]_DATA1	CAN1 Mailbox Word 1 Register	0x00000000
0x2006D7E8	CAN1_MB[nn]_DATA2	CAN1 Mailbox Word 2 Register	0x00000000
0x2006D7EC	CAN1_MB[nn]_DATA3	CAN1 Mailbox Word 3 Register	0x00000000
0x2006D7F0	CAN1_MB[nn]_LENGTH	CAN1 Mailbox Length Register	0x00000000
0x2006D7F4	CAN1_MB[nn]_TIMESTAMP	CAN1 Mailbox Time Stamp Register	0x00000000
0x2006D7F8	CAN1_MB[nn]_ID0	CAN1 Mailbox ID 0 Register	0x00000000
0x2006D7FC	CAN1_MB[nn]_ID1	CAN1 Mailbox ID 1 Register	0x00000000

Table A-3: ADSP-BF70x CGU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20002000	CGU0_CTL	CGU0 Control Register	0x00000C00
0x20002004	CGU0_PLLCTL	CGU0 PLL Control Register	0x00000000
0x20002008	CGU0_STAT	CGU0 Status Register	0x00000005
0x2000200C	CGU0_DIV	CGU0 Clocks Divisor Register	0x03084844
0x20002010	CGU0_CLKOUTSEL	CGU0 CLKOUT Select Register	0x00000000
0x20002018	CGU0_TSCTL	CGU0 Time Stamp Control Register	0x00000000
0x2000201C	CGU0_TSVALUE0	CGU0 Time Stamp Counter Initial 32 LSB Value Register	0x00000000
0x20002020	CGU0_TSVALUE1	CGU0 Time Stamp Counter Initial MSB Value Register	0x00000000
0x20002024	CGU0_TSCOUNT0	CGU0 Time Stamp Counter 32 LSB Register	0x00000000
0x20002028	CGU0_TSCOUNT1	CGU0 Time Stamp Counter 32 MSB Register	0x00000000
0x2000202C	CGU0_CCBF_DIS	CGU0 Core Clock Buffer Disable Register	0x00000000
0x20002030	CGU0_CCBF_STAT	CGU0 Core Clock Buffer Status Register	0x00000000
0x20002038	CGU0_SCBF_DIS	CGU0 System Clock Buffer Disable Register	0x00000000
0x2000203C	CGU0_SCBF_STAT	CGU0 System Clock Buffer Status Register	0x00000000
0x20002048	CGU0_REVID	CGU0 Revision ID Register	0x00000020

Table A-4: ADSP-BF70x CNT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20070000	CNT0_CFG	CNT0 Configuration Register	0x00000000
0x20070004	CNT0_IMSK	CNT0 Interrupt Mask Register	0x00000000
0x20070008	CNT0_STAT	CNT0 Status Register	0x00000000
0x2007000C	CNT0_CMD	CNT0 Command Register	0x00000000
0x20070010	CNT0_DEBNCE	CNT0 Debounce Register	0x00000000
0x20070014	CNT0_CNTR	CNT0 Counter Register	0x00000000
0x20070018	CNT0_MAX	CNT0 Maximum Count Register	0x00000000
0x2007001C	CNT0_MIN	CNT0 Minimum Count Register	0x00000000

Table A-5: ADSP-BF70x CRC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200B0000	CRC0_CTL	CRC0 Control Register	0x00000000
0x200B0004	CRC0_DCNT	CRC0 Data Word Count Register	0x00000000
0x200B0008	CRC0_DCNTRLD	CRC0 Data Word Count Reload Register	0x00000000
0x200B0014	CRC0_COMP	CRC0 Data Compare Register	0x00000000
0x200B0018	CRC0_FILLVAL	CRC0 Fill Value Register	0x00000000
0x200B001C	CRC0_DFIFO	CRC0 Data FIFO Register	0x00000000
0x200B0020	CRC0_INEN	CRC0 Interrupt Enable Register	0x00000000
0x200B0024	CRC0_INEN_SET	CRC0 Interrupt Enable Set Register	0x00000000
0x200B0028	CRC0_INEN_CLR	CRC0 Interrupt Enable Clear Register	0x00000000
0x200B002C	CRC0_POLY	CRC0 Polynomial Register	0x00000000
0x200B0040	CRC0_STAT	CRC0 Status Register	0x00000000
0x200B0044	CRC0_DCNTCAP	CRC0 Data Count Capture Register	0x00000000
0x200B004C	CRC0_RESULT_FIN	CRC0 CRC Final Result Register	0x00000000
0x200B0050	CRC0_RESULT_CUR	CRC0 CRC Current Result Register	0x00000000

Table A-6: ADSP-BF70x CRC1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200B1000	CRC1_CTL	CRC1 Control Register	0x00000000
0x200B1004	CRC1_DCNT	CRC1 Data Word Count Register	0x00000000

Table A-6: ADSP-BF70x CRC1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200B1008	CRC1_DCNTRLD	CRC1 Data Word Count Reload Register	0x00000000
0x200B1014	CRC1_COMP	CRC1 Data Compare Register	0x00000000
0x200B1018	CRC1_FILLVAL	CRC1 Fill Value Register	0x00000000
0x200B101C	CRC1_DFIFO	CRC1 Data FIFO Register	0x00000000
0x200B1020	CRC1_INEN	CRC1 Interrupt Enable Register	0x00000000
0x200B1024	CRC1_INEN_SET	CRC1 Interrupt Enable Set Register	0x00000000
0x200B1028	CRC1_INEN_CLR	CRC1 Interrupt Enable Clear Register	0x00000000
0x200B102C	CRC1_POLY	CRC1 Polynomial Register	0x00000000
0x200B1040	CRC1_STAT	CRC1 Status Register	0x00000000
0x200B1044	CRC1_DCNTCAP	CRC1 Data Count Capture Register	0x00000000
0x200B104C	CRC1_RESULT_FIN	CRC1 CRC Final Result Register	0x00000000
0x200B1050	CRC1_RESULT_CUR	CRC1 CRC Current Result Register	0x00000000

Table A-7: ADSP-BF70x CSPFT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20012000	CSPFT0_CTL	CSPFT0 Main Control Register	0x20000400
0x20012004	CSPFT0_HWFEAT	CSPFT0 Hardware Feature Register	0x01484002
0x20012008	CSPFT0_TRIGGER	CSPFT0 Trigger Event Register	0x000077EF
0x20012010	CSPFT0_STAT	CSPFT0 Status Register	0x00000002
0x20012018	CSPFT0_TSSCTL	CSPFT0 TraceEnable Start/Stop Control Register	0x00000000
0x20012020	CSPFT0_TEEVENT	CSPFT0 TraceEnable Event Register	0x000077EF
0x20012024	CSPFT0_TECTL	CSPFT0 TraceEnable Control Register	0x00000000
0x20012040	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x20012044	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x20012048	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x2001204C	CSPFT0_ACVR[n]	CSPFT0 Address Comparator Value Register	0x00000000
0x20012080	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x20012084	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x20012088	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000
0x2001208C	CSPFT0_ACTR[n]	CSPFT0 Address Comparator Access Type Register	0x00000000

Table A-7: ADSP-BF70x CSPFT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20012140	CSPFT0_CNTRLDVR[n]	CSPFT0 Counter Reload Value Register	0x00000000
0x20012144	CSPFT0_CNTRLDVR[n]	CSPFT0 Counter Reload Value Register	0x00000000
0x20012150	CSPFT0_CNTENR[n]	CSPFT0 Counter Enable Event Register	0x000077EF
0x20012154	CSPFT0_CNTENR[n]	CSPFT0 Counter Enable Event Register	0x000077EF
0x20012160	CSPFT0_CNTRLDEVR[n]	CSPFT0 Counter Reload Event Register	0x000077EF
0x20012164	CSPFT0_CNTRLDEVR[n]	CSPFT0 Counter Reload Event Register	0x000077EF
0x20012170	CSPFT0_CNTVR[n]	CSPFT0 Counter Value Register	0x00000000
0x20012174	CSPFT0_CNTVR[n]	CSPFT0 Counter Value Register	0x00000000
0x200121A0	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x200121A4	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x200121A8	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x200121AC	CSPFT0_EXTOUTEVR[n]	CSPFT0 External Output Event Register	0x000077EF
0x200121B0	CSPFT0_CIDCVR[n]	CSPFT0 Context ID Comparator Value	0x00000000
0x200121BC	CSPFT0_CIDCMR	CSPFT0 Context ID Comparator Mask Register	0x00000000
0x200121E0	CSPFT0_SYNCFR	CSPFT0 Synchronization Frequency Register	0x00000000
0x200121E8	CSPFT0_CCER	CSPFT0 Configuration Code Extension Register	0x00800000
0x20012200	CSPFT0_TRACEIDR	CSPFT0 CoreSight Trace ID Register	0x00000000
0x20012FA0	CSPFT0_CLAIMSET	CSPFT0 Claim Tag Set Register	0x0000000F
0x20012FA4	CSPFT0_CLAIMCLR	CSPFT0 Claim Tag Clear Register	0x00000000
0x20012FB0	CSPFT0_LAR	CSPFT0 Lock Access Register	0x00000000
0x20012FB4	CSPFT0_LSR	CSPFT0 Lock Status Register	0x00000003
0x20012FB8	CSPFT0_AUTHSTATUS	CSPFT0 Authentication Status Register	0x00000080
0x20012FCC	CSPFT0_DEVTYPE	CSPFT0 Device Type Identifier Register	0x00000023
0x20012FD0	CSPFT0_PID4	CSPFT0 Peripheral ID4 Register	0x00000000
0x20012FE0	CSPFT0_PID0	CSPFT0 Peripheral ID0 Register	0x00000000
0x20012FE4	CSPFT0_PID1	CSPFT0 Peripheral ID1 Register	0x00000050
0x20012FE8	CSPFT0_PID2	CSPFT0 Peripheral ID2 Register	0x0000000E
0x20012FEC	CSPFT0_PID3	CSPFT0 Peripheral ID3 Register	0x00000000
0x20012FF0	CSPFT0_CID0	CSPFT0 Component ID0 Register	0x0000000D
0x20012FF4	CSPFT0_CID1	CSPFT0 Component ID1 Register	0x00000090
0x20012FF8	CSPFT0_CID2	CSPFT0 Component ID2 Register	0x00000005

Table A-7: ADSP-BF70x CSPFT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20012FFC	CSPFT0_CID3	CSPFT0 Component ID3 Register	0x000000B1

Table A-8: ADSP-BF70x DMA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2007C000	DMA0_DSCPTR_NXT	DMA0 Pointer to Next Initial Descriptor Register	0x00000000
0x2007C004	DMA0_ADDRSTART	DMA0 Start Address of Current Buffer Register	0x00000000
0x2007C008	DMA0_CFG	DMA0 Configuration Register	0x00000000
0x2007C00C	DMA0_XCNT	DMA0 Inner Loop Count Start Value Register	0x00000000
0x2007C010	DMA0_XMOD	DMA0 Inner Loop Address Increment Register	0x00000000
0x2007C014	DMA0_YCNT	DMA0 Outer Loop Count Start Value (2D only) Register	0x00000000
0x2007C018	DMA0_YMOD	DMA0 Outer Loop Address Increment (2D only) Register	0x00000000
0x2007C024	DMA0_DSCPTR_CUR	DMA0 Current Descriptor Pointer Register	0x00000000
0x2007C028	DMA0_DSCPTR_PRV	DMA0 Previous Initial Descriptor Pointer Register	0x00000000
0x2007C02C	DMA0_ADDR_CUR	DMA0 Current Address Register	0x00000000
0x2007C030	DMA0_STAT	DMA0 Status Register	0x00006000
0x2007C034	DMA0_XCNT_CUR	DMA0 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x2007C038	DMA0_YCNT_CUR	DMA0 Current Row Count (2D only) Register	0x00000000

Table A-9: ADSP-BF70x DMA1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2007D000	DMA1_DSCPTR_NXT	DMA1 Pointer to Next Initial Descriptor Register	0x00000000
0x2007D004	DMA1_ADDRSTART	DMA1 Start Address of Current Buffer Register	0x00000000
0x2007D008	DMA1_CFG	DMA1 Configuration Register	0x00000000
0x2007D00C	DMA1_XCNT	DMA1 Inner Loop Count Start Value Register	0x00000000
0x2007D010	DMA1_XMOD	DMA1 Inner Loop Address Increment Register	0x00000000
0x2007D014	DMA1_YCNT	DMA1 Outer Loop Count Start Value (2D only) Register	0x00000000
0x2007D018	DMA1_YMOD	DMA1 Outer Loop Address Increment (2D only) Register	0x00000000
0x2007D024	DMA1_DSCPTR_CUR	DMA1 Current Descriptor Pointer Register	0x00000000
0x2007D028	DMA1_DSCPTR_PRV	DMA1 Previous Initial Descriptor Pointer Register	0x00000000

Table A-9: ADSP-BF70x DMA1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2007D02C	DMA1_ADDR_CUR	DMA1 Current Address Register	0x00000000
0x2007D030	DMA1_STAT	DMA1 Status Register	0x00006000
0x2007D034	DMA1_XCNT_CUR	DMA1 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x2007D038	DMA1_YCNT_CUR	DMA1 Current Row Count (2D only) Register	0x00000000

Table A-10: ADSP-BF70x DMA10 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20086000	DMA10_DSCPTR_NXT	DMA10 Pointer to Next Initial Descriptor Register	0x00000000
0x20086004	DMA10_ADDRSTART	DMA10 Start Address of Current Buffer Register	0x00000000
0x20086008	DMA10_CFG	DMA10 Configuration Register	0x00000000
0x2008600C	DMA10_XCNT	DMA10 Inner Loop Count Start Value Register	0x00000000
0x20086010	DMA10_XMOD	DMA10 Inner Loop Address Increment Register	0x00000000
0x20086014	DMA10_YCNT	DMA10 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20086018	DMA10_YMOD	DMA10 Outer Loop Address Increment (2D only) Register	0x00000000
0x20086024	DMA10_DSCPTR_CUR	DMA10 Current Descriptor Pointer Register	0x00000000
0x20086028	DMA10_DSCPTR_PRV	DMA10 Previous Initial Descriptor Pointer Register	0x00000000
0x2008602C	DMA10_ADDR_CUR	DMA10 Current Address Register	0x00000000
0x20086030	DMA10_STAT	DMA10 Status Register	0x00006000
0x20086034	DMA10_XCNT_CUR	DMA10 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20086038	DMA10_YCNT_CUR	DMA10 Current Row Count (2D only) Register	0x00000000

Table A-11: ADSP-BF70x DMA11 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20087000	DMA11_DSCPTR_NXT	DMA11 Pointer to Next Initial Descriptor Register	0x00000000
0x20087004	DMA11_ADDRSTART	DMA11 Start Address of Current Buffer Register	0x00000000
0x20087008	DMA11_CFG	DMA11 Configuration Register	0x00000000
0x2008700C	DMA11_XCNT	DMA11 Inner Loop Count Start Value Register	0x00000000

Table A-11: ADSP-BF70x DMA11 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20087010	DMA11_XMOD	DMA11 Inner Loop Address Increment Register	0x00000000
0x20087014	DMA11_YCNT	DMA11 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20087018	DMA11_YMOD	DMA11 Outer Loop Address Increment (2D only) Register	0x00000000
0x20087024	DMA11_DSCPTR_CUR	DMA11 Current Descriptor Pointer Register	0x00000000
0x20087028	DMA11_DSCPTR_PRV	DMA11 Previous Initial Descriptor Pointer Register	0x00000000
0x2008702C	DMA11_ADDR_CUR	DMA11 Current Address Register	0x00000000
0x20087030	DMA11_STAT	DMA11 Status Register	0x00006000
0x20087034	DMA11_XCNT_CUR	DMA11 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20087038	DMA11_YCNT_CUR	DMA11 Current Row Count (2D only) Register	0x00000000

Table A-12: ADSP-BF70x DMA12 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20088000	DMA12_DSCPTR_NXT	DMA12 Pointer to Next Initial Descriptor Register	0x00000000
0x20088004	DMA12_ADDRSTART	DMA12 Start Address of Current Buffer Register	0x00000000
0x20088008	DMA12_CFG	DMA12 Configuration Register	0x00000000
0x2008800C	DMA12_XCNT	DMA12 Inner Loop Count Start Value Register	0x00000000
0x20088010	DMA12_XMOD	DMA12 Inner Loop Address Increment Register	0x00000000
0x20088014	DMA12_YCNT	DMA12 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20088018	DMA12_YMOD	DMA12 Outer Loop Address Increment (2D only) Register	0x00000000
0x20088024	DMA12_DSCPTR_CUR	DMA12 Current Descriptor Pointer Register	0x00000000
0x20088028	DMA12_DSCPTR_PRV	DMA12 Previous Initial Descriptor Pointer Register	0x00000000
0x2008802C	DMA12_ADDR_CUR	DMA12 Current Address Register	0x00000000
0x20088030	DMA12_STAT	DMA12 Status Register	0x00006000
0x20088034	DMA12_XCNT_CUR	DMA12 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20088038	DMA12_YCNT_CUR	DMA12 Current Row Count (2D only) Register	0x00000000

Table A-13: ADSP-BF70x DMA13 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20089000	DMA13_DSCPTR_NXT	DMA13 Pointer to Next Initial Descriptor Register	0x00000000
0x20089004	DMA13_ADDRSTART	DMA13 Start Address of Current Buffer Register	0x00000000
0x20089008	DMA13_CFG	DMA13 Configuration Register	0x00000000
0x2008900C	DMA13_XCNT	DMA13 Inner Loop Count Start Value Register	0x00000000
0x20089010	DMA13_XMOD	DMA13 Inner Loop Address Increment Register	0x00000000
0x20089014	DMA13_YCNT	DMA13 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20089018	DMA13_YMOD	DMA13 Outer Loop Address Increment (2D only) Register	0x00000000
0x20089024	DMA13_DSCPTR_CUR	DMA13 Current Descriptor Pointer Register	0x00000000
0x20089028	DMA13_DSCPTR_PRV	DMA13 Previous Initial Descriptor Pointer Register	0x00000000
0x2008902C	DMA13_ADDR_CUR	DMA13 Current Address Register	0x00000000
0x20089030	DMA13_STAT	DMA13 Status Register	0x00006000
0x20089034	DMA13_XCNT_CUR	DMA13 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20089038	DMA13_YCNT_CUR	DMA13 Current Row Count (2D only) Register	0x00000000

Table A-14: ADSP-BF70x DMA14 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2008A000	DMA14_DSCPTR_NXT	DMA14 Pointer to Next Initial Descriptor Register	0x00000000
0x2008A004	DMA14_ADDRSTART	DMA14 Start Address of Current Buffer Register	0x00000000
0x2008A008	DMA14_CFG	DMA14 Configuration Register	0x00000000
0x2008A00C	DMA14_XCNT	DMA14 Inner Loop Count Start Value Register	0x00000000
0x2008A010	DMA14_XMOD	DMA14 Inner Loop Address Increment Register	0x00000000
0x2008A014	DMA14_YCNT	DMA14 Outer Loop Count Start Value (2D only) Register	0x00000000
0x2008A018	DMA14_YMOD	DMA14 Outer Loop Address Increment (2D only) Register	0x00000000
0x2008A024	DMA14_DSCPTR_CUR	DMA14 Current Descriptor Pointer Register	0x00000000
0x2008A028	DMA14_DSCPTR_PRV	DMA14 Previous Initial Descriptor Pointer Register	0x00000000
0x2008A02C	DMA14_ADDR_CUR	DMA14 Current Address Register	0x00000000
0x2008A030	DMA14_STAT	DMA14 Status Register	0x00006000

Table A-14: ADSP-BF70x DMA14 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2008A034	DMA14_XCNT_CUR	DMA14 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x2008A038	DMA14_YCNT_CUR	DMA14 Current Row Count (2D only) Register	0x00000000

Table A-15: ADSP-BF70x DMA15 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2008B000	DMA15_DSCPTR_NXT	DMA15 Pointer to Next Initial Descriptor Register	0x00000000
0x2008B004	DMA15_ADDRSTART	DMA15 Start Address of Current Buffer Register	0x00000000
0x2008B008	DMA15_CFG	DMA15 Configuration Register	0x00000000
0x2008B00C	DMA15_XCNT	DMA15 Inner Loop Count Start Value Register	0x00000000
0x2008B010	DMA15_XMOD	DMA15 Inner Loop Address Increment Register	0x00000000
0x2008B014	DMA15_YCNT	DMA15 Outer Loop Count Start Value (2D only) Register	0x00000000
0x2008B018	DMA15_YMOD	DMA15 Outer Loop Address Increment (2D only) Register	0x00000000
0x2008B024	DMA15_DSCPTR_CUR	DMA15 Current Descriptor Pointer Register	0x00000000
0x2008B028	DMA15_DSCPTR_PRV	DMA15 Previous Initial Descriptor Pointer Register	0x00000000
0x2008B02C	DMA15_ADDR_CUR	DMA15 Current Address Register	0x00000000
0x2008B030	DMA15_STAT	DMA15 Status Register	0x00006000
0x2008B034	DMA15_XCNT_CUR	DMA15 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x2008B038	DMA15_YCNT_CUR	DMA15 Current Row Count (2D only) Register	0x00000000

Table A-16: ADSP-BF70x DMA16 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2008C000	DMA16_DSCPTR_NXT	DMA16 Pointer to Next Initial Descriptor Register	0x00000000
0x2008C004	DMA16_ADDRSTART	DMA16 Start Address of Current Buffer Register	0x00000000
0x2008C008	DMA16_CFG	DMA16 Configuration Register	0x00000000
0x2008C00C	DMA16_XCNT	DMA16 Inner Loop Count Start Value Register	0x00000000
0x2008C010	DMA16_XMOD	DMA16 Inner Loop Address Increment Register	0x00000000
0x2008C014	DMA16_YCNT	DMA16 Outer Loop Count Start Value (2D only) Register	0x00000000

Table A-16: ADSP-BF70x DMA16 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2008C018	DMA16_YMOD	DMA16 Outer Loop Address Increment (2D only) Register	0x00000000
0x2008C024	DMA16_DSCPTR_CUR	DMA16 Current Descriptor Pointer Register	0x00000000
0x2008C028	DMA16_DSCPTR_PRV	DMA16 Previous Initial Descriptor Pointer Register	0x00000000
0x2008C02C	DMA16_ADDR_CUR	DMA16 Current Address Register	0x00000000
0x2008C030	DMA16_STAT	DMA16 Status Register	0x00006000
0x2008C034	DMA16_XCNT_CUR	DMA16 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x2008C038	DMA16_YCNT_CUR	DMA16 Current Row Count (2D only) Register	0x00000000
0x2008C040	DMA16_BWLCNT	DMA16 Bandwidth Limit Count Register	0x00000000
0x2008C044	DMA16_BWLCNT_CUR	DMA16 Bandwidth Limit Count Current Register	0x00000000
0x2008C048	DMA16_BWMCNT	DMA16 Bandwidth Monitor Count Register	0x00000000
0x2008C04C	DMA16_BWMCNT_CUR	DMA16 Bandwidth Monitor Count Current Register	0x00000000

Table A-17: ADSP-BF70x DMA17 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2008D000	DMA17_DSCPTR_NXT	DMA17 Pointer to Next Initial Descriptor Register	0x00000000
0x2008D004	DMA17_ADDRSTART	DMA17 Start Address of Current Buffer Register	0x00000000
0x2008D008	DMA17_CFG	DMA17 Configuration Register	0x00000000
0x2008D00C	DMA17_XCNT	DMA17 Inner Loop Count Start Value Register	0x00000000
0x2008D010	DMA17_XMOD	DMA17 Inner Loop Address Increment Register	0x00000000
0x2008D014	DMA17_YCNT	DMA17 Outer Loop Count Start Value (2D only) Register	0x00000000
0x2008D018	DMA17_YMOD	DMA17 Outer Loop Address Increment (2D only) Register	0x00000000
0x2008D024	DMA17_DSCPTR_CUR	DMA17 Current Descriptor Pointer Register	0x00000000
0x2008D028	DMA17_DSCPTR_PRV	DMA17 Previous Initial Descriptor Pointer Register	0x00000000
0x2008D02C	DMA17_ADDR_CUR	DMA17 Current Address Register	0x00000000
0x2008D030	DMA17_STAT	DMA17 Status Register	0x00006000
0x2008D034	DMA17_XCNT_CUR	DMA17 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x2008D038	DMA17_YCNT_CUR	DMA17 Current Row Count (2D only) Register	0x00000000

Table A-17: ADSP-BF70x DMA17 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2008D040	DMA17_BWLCNT	DMA17 Bandwidth Limit Count Register	0x00000000
0x2008D044	DMA17_BWLCNT_CUR	DMA17 Bandwidth Limit Count Current Register	0x00000000
0x2008D048	DMA17_BWMCNT	DMA17 Bandwidth Monitor Count Register	0x00000000
0x2008D04C	DMA17_BWMCNT_CUR	DMA17 Bandwidth Monitor Count Current Register	0x00000000

Table A-18: ADSP-BF70x DMA18 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200A0000	DMA18_DSCPTR_NXT	DMA18 Pointer to Next Initial Descriptor Register	0x00000000
0x200A0004	DMA18_ADDRSTART	DMA18 Start Address of Current Buffer Register	0x00000000
0x200A0008	DMA18_CFG	DMA18 Configuration Register	0x00000000
0x200A000C	DMA18_XCNT	DMA18 Inner Loop Count Start Value Register	0x00000000
0x200A0010	DMA18_XMOD	DMA18 Inner Loop Address Increment Register	0x00000000
0x200A0014	DMA18_YCNT	DMA18 Outer Loop Count Start Value (2D only) Register	0x00000000
0x200A0018	DMA18_YMOD	DMA18 Outer Loop Address Increment (2D only) Register	0x00000000
0x200A0024	DMA18_DSCPTR_CUR	DMA18 Current Descriptor Pointer Register	0x00000000
0x200A0028	DMA18_DSCPTR_PRV	DMA18 Previous Initial Descriptor Pointer Register	0x00000000
0x200A002C	DMA18_ADDR_CUR	DMA18 Current Address Register	0x00000000
0x200A0030	DMA18_STAT	DMA18 Status Register	0x00006000
0x200A0034	DMA18_XCNT_CUR	DMA18 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x200A0038	DMA18_YCNT_CUR	DMA18 Current Row Count (2D only) Register	0x00000000
0x200A0040	DMA18_BWLCNT	DMA18 Bandwidth Limit Count Register	0x00000000
0x200A0044	DMA18_BWLCNT_CUR	DMA18 Bandwidth Limit Count Current Register	0x00000000
0x200A0048	DMA18_BWMCNT	DMA18 Bandwidth Monitor Count Register	0x00000000
0x200A004C	DMA18_BWMCNT_CUR	DMA18 Bandwidth Monitor Count Current Register	0x00000000

Table A-19: ADSP-BF70x DMA19 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200A1000	DMA19_DSCPTR_NXT	DMA19 Pointer to Next Initial Descriptor Register	0x00000000

Table A-19: ADSP-BF70x DMA19 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200A1004	DMA19_ADDRSTART	DMA19 Start Address of Current Buffer Register	0x00000000
0x200A1008	DMA19_CFG	DMA19 Configuration Register	0x00000000
0x200A100C	DMA19_XCNT	DMA19 Inner Loop Count Start Value Register	0x00000000
0x200A1010	DMA19_XMOD	DMA19 Inner Loop Address Increment Register	0x00000000
0x200A1014	DMA19_YCNT	DMA19 Outer Loop Count Start Value (2D only) Register	0x00000000
0x200A1018	DMA19_YMOD	DMA19 Outer Loop Address Increment (2D only) Register	0x00000000
0x200A1024	DMA19_DSCPTR_CUR	DMA19 Current Descriptor Pointer Register	0x00000000
0x200A1028	DMA19_DSCPTR_PRV	DMA19 Previous Initial Descriptor Pointer Register	0x00000000
0x200A102C	DMA19_ADDR_CUR	DMA19 Current Address Register	0x00000000
0x200A1030	DMA19_STAT	DMA19 Status Register	0x00006000
0x200A1034	DMA19_XCNT_CUR	DMA19 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x200A1038	DMA19_YCNT_CUR	DMA19 Current Row Count (2D only) Register	0x00000000
0x200A1040	DMA19_BWLCNT	DMA19 Bandwidth Limit Count Register	0x00000000
0x200A1044	DMA19_BWLCNT_CUR	DMA19 Bandwidth Limit Count Current Register	0x00000000
0x200A1048	DMA19_BWMCNT	DMA19 Bandwidth Monitor Count Register	0x00000000
0x200A104C	DMA19_BWMCNT_CUR	DMA19 Bandwidth Monitor Count Current Register	0x00000000

Table A-20: ADSP-BF70x DMA2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2007E000	DMA2_DSCPTR_NXT	DMA2 Pointer to Next Initial Descriptor Register	0x00000000
0x2007E004	DMA2_ADDRSTART	DMA2 Start Address of Current Buffer Register	0x00000000
0x2007E008	DMA2_CFG	DMA2 Configuration Register	0x00000000
0x2007E00C	DMA2_XCNT	DMA2 Inner Loop Count Start Value Register	0x00000000
0x2007E010	DMA2_XMOD	DMA2 Inner Loop Address Increment Register	0x00000000
0x2007E014	DMA2_YCNT	DMA2 Outer Loop Count Start Value (2D only) Register	0x00000000
0x2007E018	DMA2_YMOD	DMA2 Outer Loop Address Increment (2D only) Register	0x00000000
0x2007E024	DMA2_DSCPTR_CUR	DMA2 Current Descriptor Pointer Register	0x00000000
0x2007E028	DMA2_DSCPTR_PRV	DMA2 Previous Initial Descriptor Pointer Register	0x00000000

Table A-20: ADSP-BF70x DMA2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2007E02C	DMA2_ADDR_CUR	DMA2 Current Address Register	0x00000000
0x2007E030	DMA2_STAT	DMA2 Status Register	0x00006000
0x2007E034	DMA2_XCNT_CUR	DMA2 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x2007E038	DMA2_YCNT_CUR	DMA2 Current Row Count (2D only) Register	0x00000000

Table A-21: ADSP-BF70x DMA20 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200A2000	DMA20_DSCPTR_NXT	DMA20 Pointer to Next Initial Descriptor Register	0x00000000
0x200A2004	DMA20_ADDRSTART	DMA20 Start Address of Current Buffer Register	0x00000000
0x200A2008	DMA20_CFG	DMA20 Configuration Register	0x00000000
0x200A200C	DMA20_XCNT	DMA20 Inner Loop Count Start Value Register	0x00000000
0x200A2010	DMA20_XMOD	DMA20 Inner Loop Address Increment Register	0x00000000
0x200A2014	DMA20_YCNT	DMA20 Outer Loop Count Start Value (2D only) Register	0x00000000
0x200A2018	DMA20_YMOD	DMA20 Outer Loop Address Increment (2D only) Register	0x00000000
0x200A2024	DMA20_DSCPTR_CUR	DMA20 Current Descriptor Pointer Register	0x00000000
0x200A2028	DMA20_DSCPTR_PRV	DMA20 Previous Initial Descriptor Pointer Register	0x00000000
0x200A202C	DMA20_ADDR_CUR	DMA20 Current Address Register	0x00000000
0x200A2030	DMA20_STAT	DMA20 Status Register	0x00006000
0x200A2034	DMA20_XCNT_CUR	DMA20 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x200A2038	DMA20_YCNT_CUR	DMA20 Current Row Count (2D only) Register	0x00000000
0x200A2040	DMA20_BWLCNT	DMA20 Bandwidth Limit Count Register	0x00000000
0x200A2044	DMA20_BWLCNT_CUR	DMA20 Bandwidth Limit Count Current Register	0x00000000
0x200A2048	DMA20_BWMCNT	DMA20 Bandwidth Monitor Count Register	0x00000000
0x200A204C	DMA20_BWMCNT_CUR	DMA20 Bandwidth Monitor Count Current Register	0x00000000

Table A-22: ADSP-BF70x DMA21 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200A3000	DMA21_DSCPTR_NXT	DMA21 Pointer to Next Initial Descriptor Register	0x00000000
0x200A3004	DMA21_ADDRSTART	DMA21 Start Address of Current Buffer Register	0x00000000
0x200A3008	DMA21_CFG	DMA21 Configuration Register	0x00000000
0x200A300C	DMA21_XCNT	DMA21 Inner Loop Count Start Value Register	0x00000000
0x200A3010	DMA21_XMOD	DMA21 Inner Loop Address Increment Register	0x00000000
0x200A3014	DMA21_YCNT	DMA21 Outer Loop Count Start Value (2D only) Register	0x00000000
0x200A3018	DMA21_YMOD	DMA21 Outer Loop Address Increment (2D only) Register	0x00000000
0x200A3024	DMA21_DSCPTR_CUR	DMA21 Current Descriptor Pointer Register	0x00000000
0x200A3028	DMA21_DSCPTR_PRV	DMA21 Previous Initial Descriptor Pointer Register	0x00000000
0x200A302C	DMA21_ADDR_CUR	DMA21 Current Address Register	0x00000000
0x200A3030	DMA21_STAT	DMA21 Status Register	0x00006000
0x200A3034	DMA21_XCNT_CUR	DMA21 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x200A3038	DMA21_YCNT_CUR	DMA21 Current Row Count (2D only) Register	0x00000000
0x200A3040	DMA21_BWLCNT	DMA21 Bandwidth Limit Count Register	0x00000000
0x200A3044	DMA21_BWLCNT_CUR	DMA21 Bandwidth Limit Count Current Register	0x00000000
0x200A3048	DMA21_BWMCNT	DMA21 Bandwidth Monitor Count Register	0x00000000
0x200A304C	DMA21_BWMCNT_CUR	DMA21 Bandwidth Monitor Count Current Register	0x00000000

Table A-23: ADSP-BF70x DMA3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2007F000	DMA3_DSCPTR_NXT	DMA3 Pointer to Next Initial Descriptor Register	0x00000000
0x2007F004	DMA3_ADDRSTART	DMA3 Start Address of Current Buffer Register	0x00000000
0x2007F008	DMA3_CFG	DMA3 Configuration Register	0x00000000
0x2007F00C	DMA3_XCNT	DMA3 Inner Loop Count Start Value Register	0x00000000
0x2007F010	DMA3_XMOD	DMA3 Inner Loop Address Increment Register	0x00000000
0x2007F014	DMA3_YCNT	DMA3 Outer Loop Count Start Value (2D only) Register	0x00000000
0x2007F018	DMA3_YMOD	DMA3 Outer Loop Address Increment (2D only) Register	0x00000000
0x2007F024	DMA3_DSCPTR_CUR	DMA3 Current Descriptor Pointer Register	0x00000000

Table A-23: ADSP-BF70x DMA3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2007F028	DMA3_DSCPTR_PRV	DMA3 Previous Initial Descriptor Pointer Register	0x00000000
0x2007F02C	DMA3_ADDR_CUR	DMA3 Current Address Register	0x00000000
0x2007F030	DMA3_STAT	DMA3 Status Register	0x00006000
0x2007F034	DMA3_XCNT_CUR	DMA3 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x2007F038	DMA3_YCNT_CUR	DMA3 Current Row Count (2D only) Register	0x00000000

Table A-24: ADSP-BF70x DMA4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20080000	DMA4_DSCPTR_NXT	DMA4 Pointer to Next Initial Descriptor Register	0x00000000
0x20080004	DMA4_ADDRSTART	DMA4 Start Address of Current Buffer Register	0x00000000
0x20080008	DMA4_CFG	DMA4 Configuration Register	0x00000000
0x2008000C	DMA4_XCNT	DMA4 Inner Loop Count Start Value Register	0x00000000
0x20080010	DMA4_XMOD	DMA4 Inner Loop Address Increment Register	0x00000000
0x20080014	DMA4_YCNT	DMA4 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20080018	DMA4_YMOD	DMA4 Outer Loop Address Increment (2D only) Register	0x00000000
0x20080024	DMA4_DSCPTR_CUR	DMA4 Current Descriptor Pointer Register	0x00000000
0x20080028	DMA4_DSCPTR_PRV	DMA4 Previous Initial Descriptor Pointer Register	0x00000000
0x2008002C	DMA4_ADDR_CUR	DMA4 Current Address Register	0x00000000
0x20080030	DMA4_STAT	DMA4 Status Register	0x00006000
0x20080034	DMA4_XCNT_CUR	DMA4 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20080038	DMA4_YCNT_CUR	DMA4 Current Row Count (2D only) Register	0x00000000

Table A-25: ADSP-BF70x DMA5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20081000	DMA5_DSCPTR_NXT	DMA5 Pointer to Next Initial Descriptor Register	0x00000000
0x20081004	DMA5_ADDRSTART	DMA5 Start Address of Current Buffer Register	0x00000000
0x20081008	DMA5_CFG	DMA5 Configuration Register	0x00000000
0x2008100C	DMA5_XCNT	DMA5 Inner Loop Count Start Value Register	0x00000000
0x20081010	DMA5_XMOD	DMA5 Inner Loop Address Increment Register	0x00000000

Table A-25: ADSP-BF70x DMA5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20081014	DMA5_YCNT	DMA5 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20081018	DMA5_YMOD	DMA5 Outer Loop Address Increment (2D only) Register	0x00000000
0x20081024	DMA5_DSCPTR_CUR	DMA5 Current Descriptor Pointer Register	0x00000000
0x20081028	DMA5_DSCPTR_PRV	DMA5 Previous Initial Descriptor Pointer Register	0x00000000
0x2008102C	DMA5_ADDR_CUR	DMA5 Current Address Register	0x00000000
0x20081030	DMA5_STAT	DMA5 Status Register	0x00006000
0x20081034	DMA5_XCNT_CUR	DMA5 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20081038	DMA5_YCNT_CUR	DMA5 Current Row Count (2D only) Register	0x00000000

Table A-26: ADSP-BF70x DMA6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20082000	DMA6_DSCPTR_NXT	DMA6 Pointer to Next Initial Descriptor Register	0x00000000
0x20082004	DMA6_ADDRSTART	DMA6 Start Address of Current Buffer Register	0x00000000
0x20082008	DMA6_CFG	DMA6 Configuration Register	0x00000000
0x2008200C	DMA6_XCNT	DMA6 Inner Loop Count Start Value Register	0x00000000
0x20082010	DMA6_XMOD	DMA6 Inner Loop Address Increment Register	0x00000000
0x20082014	DMA6_YCNT	DMA6 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20082018	DMA6_YMOD	DMA6 Outer Loop Address Increment (2D only) Register	0x00000000
0x20082024	DMA6_DSCPTR_CUR	DMA6 Current Descriptor Pointer Register	0x00000000
0x20082028	DMA6_DSCPTR_PRV	DMA6 Previous Initial Descriptor Pointer Register	0x00000000
0x2008202C	DMA6_ADDR_CUR	DMA6 Current Address Register	0x00000000
0x20082030	DMA6_STAT	DMA6 Status Register	0x00006000
0x20082034	DMA6_XCNT_CUR	DMA6 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20082038	DMA6_YCNT_CUR	DMA6 Current Row Count (2D only) Register	0x00000000

Table A-27: ADSP-BF70x DMA7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20083000	DMA7_DSCPTR_NXT	DMA7 Pointer to Next Initial Descriptor Register	0x00000000
0x20083004	DMA7_ADDRSTART	DMA7 Start Address of Current Buffer Register	0x00000000

Table A-27: ADSP-BF70x DMA7 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20083008	DMA7_CFG	DMA7 Configuration Register	0x00000000
0x2008300C	DMA7_XCNT	DMA7 Inner Loop Count Start Value Register	0x00000000
0x20083010	DMA7_XMOD	DMA7 Inner Loop Address Increment Register	0x00000000
0x20083014	DMA7_YCNT	DMA7 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20083018	DMA7_YMOD	DMA7 Outer Loop Address Increment (2D only) Register	0x00000000
0x20083024	DMA7_DSCPTR_CUR	DMA7 Current Descriptor Pointer Register	0x00000000
0x20083028	DMA7_DSCPTR_PRV	DMA7 Previous Initial Descriptor Pointer Register	0x00000000
0x2008302C	DMA7_ADDR_CUR	DMA7 Current Address Register	0x00000000
0x20083030	DMA7_STAT	DMA7 Status Register	0x00006000
0x20083034	DMA7_XCNT_CUR	DMA7 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20083038	DMA7_YCNT_CUR	DMA7 Current Row Count (2D only) Register	0x00000000

Table A-28: ADSP-BF70x DMA8 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20084000	DMA8_DSCPTR_NXT	DMA8 Pointer to Next Initial Descriptor Register	0x00000000
0x20084004	DMA8_ADDRSTART	DMA8 Start Address of Current Buffer Register	0x00000000
0x20084008	DMA8_CFG	DMA8 Configuration Register	0x00000000
0x2008400C	DMA8_XCNT	DMA8 Inner Loop Count Start Value Register	0x00000000
0x20084010	DMA8_XMOD	DMA8 Inner Loop Address Increment Register	0x00000000
0x20084014	DMA8_YCNT	DMA8 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20084018	DMA8_YMOD	DMA8 Outer Loop Address Increment (2D only) Register	0x00000000
0x20084024	DMA8_DSCPTR_CUR	DMA8 Current Descriptor Pointer Register	0x00000000
0x20084028	DMA8_DSCPTR_PRV	DMA8 Previous Initial Descriptor Pointer Register	0x00000000
0x2008402C	DMA8_ADDR_CUR	DMA8 Current Address Register	0x00000000
0x20084030	DMA8_STAT	DMA8 Status Register	0x00006000
0x20084034	DMA8_XCNT_CUR	DMA8 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20084038	DMA8_YCNT_CUR	DMA8 Current Row Count (2D only) Register	0x00000000

Table A-29: ADSP-BF70x DMA9 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20085000	DMA9_DSCPTR_NXT	DMA9 Pointer to Next Initial Descriptor Register	0x00000000
0x20085004	DMA9_ADDRSTART	DMA9 Start Address of Current Buffer Register	0x00000000
0x20085008	DMA9_CFG	DMA9 Configuration Register	0x00000000
0x2008500C	DMA9_XCNT	DMA9 Inner Loop Count Start Value Register	0x00000000
0x20085010	DMA9_XMOD	DMA9 Inner Loop Address Increment Register	0x00000000
0x20085014	DMA9_YCNT	DMA9 Outer Loop Count Start Value (2D only) Register	0x00000000
0x20085018	DMA9_YMOD	DMA9 Outer Loop Address Increment (2D only) Register	0x00000000
0x20085024	DMA9_DSCPTR_CUR	DMA9 Current Descriptor Pointer Register	0x00000000
0x20085028	DMA9_DSCPTR_PRV	DMA9 Previous Initial Descriptor Pointer Register	0x00000000
0x2008502C	DMA9_ADDR_CUR	DMA9 Current Address Register	0x00000000
0x20085030	DMA9_STAT	DMA9 Status Register	0x00006000
0x20085034	DMA9_XCNT_CUR	DMA9 Current Count (1D) or Intra-row XCNT (2D) Register	0x00000000
0x20085038	DMA9_YCNT_CUR	DMA9 Current Row Count (2D only) Register	0x00000000

Table A-30: ADSP-BF70x DMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200C0004	DMC0_CTL	DMC0 Control Register	0x00000000
0x200C0008	DMC0_STAT	DMC0 Status Register	0x00000001
0x200C000C	DMC0_EFFCTL	DMC0 Efficiency Control Register	0x00440000
0x200C0010	DMC0_PRIO	DMC0 Priority ID Register 1	0x00000000
0x200C0014	DMC0_PRIOMSK	DMC0 Priority ID Mask Register 1	0x00000000
0x200C0018	DMC0_PRIO2	DMC0 Priority ID Register 2	0x00000000
0x200C001C	DMC0_PRIOMSK2	DMC0 Priority ID Mask Register 2	0x00000000
0x200C0040	DMC0_CFG	DMC0 Configuration Register	0x00000000
0x200C0044	DMC0_TR0	DMC0 Timing 0 Register	0x00000000
0x200C0048	DMC0_TR1	DMC0 Timing 1 Register	0x00000000
0x200C004C	DMC0_TR2	DMC0 Timing 2 Register	0x00000000
0x200C005C	DMC0_MSK	DMC0 Mask (Mode Register Shadow) Register	0x00000000
0x200C0060	DMC0_MR	DMC0 Shadow MR Register	0x00000000
0x200C0064	DMC0_EMR1	DMC0 Shadow EMR1 Register	0x00000000

Table A-30: ADSP-BF70x DMC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200C0068	DMC0_EMR2	DMC0 Shadow EMR2 Register (DDR2)/Shadow EMR Register (LPDDR)	0x00000000
0x200C0080	DMC0_DLLCTL	DMC0 DLL Control Register	0x0000054B
0x200C0100	DMC0_RDDATABUFID1	DMC0 DMC Read Data Buffer ID Register 1	0x00000000
0x200C0104	DMC0_RDDATABUFMSK1	DMC0 DMC Read Data Buffer Mask Register 1	0x00000000
0x200C0108	DMC0_RDDATABUFID2	DMC0 DMC Read Data Buffer ID Register 2	0x00000000
0x200C010C	DMC0_RDDATABUFMSK2	DMC0 DMC Read Data Buffer Mask Register 2	0x00000000
0x200C01C0	DMC0_CPHY_CTL	DMC0 Controller to PHY Interface Register	0x00000000

Table A-31: ADSP-BF70x DMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200C1000	DMC0_PHY_CTL0	DMC0 PHY Control 0 Register	0x00000000
0x200C1004	DMC0_PHY_CTL1	DMC0 PHY Control 1 Register	0x00000000
0x200C1008	DMC0_PHY_CTL2	DMC0 PHY Control 2 Register	0x00000000
0x200C100C	DMC0_PHY_CTL3	DMC0 PHY Control 3 Register	0x00000000
0x200C1010	DMC0_PHY_CTL4	DMC0 PHY Control 4 Register	0x00000000
0x200C1014	DMC0_PHY_CTL5	DMC0 PHY Control 5 Register	0x00000000
0x200C101C	DMC0_PHY_STAT0	DMC0 PHY Status 0 Register	0x00000000
0x200C1028	DMC0_PHY_STAT3	DMC0 PHY Status 3 Register	0x00000000
0x200C102C	DMC0_PHY_STAT4	DMC0 PHY Status 4 Register	0x00000000
0x200C1030	DMC0_PHY_STAT5	DMC0 PHY Status 5 Register	0x00000000
0x200C1034	DMC0_CAL_PADCTL0	DMC0 Calibration PAD Control 0 Register	0xE0000000
0x200C1038	DMC0_CAL_PADCTL1	DMC0 Calibration PAD Control 1 Register	0x00000000
0x200C103C	DMC0_CAL_PADCTL2	DMC0 Calibration PAD Control 2 Register	0x00000000

Table A-32: ADSP-BF70x DPM0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20003000	DPM0_CTL	DPM0 Control Register	0x00000000
0x20003004	DPM0_STAT	DPM0 Status Register	0x00000001
0x2000301C	DPM0_WAKE_EN	DPM0 Wakeup Enable Register	0x00000000

Table A-32: ADSP-BF70x DPM0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20003020	DPM0_WAKE_POL	DPM0 Wakeup Polarity Register	0x00000000
0x20003024	DPM0_WAKE_STAT	DPM0 Wakeup Status Register	0x00000000
0x20003028	DPM0_HIB_DIS	DPM0 Hibernate Disable Register	0x00000000
0x2000302C	DPM0_PGCNTR	DPM0 Power Good Counter Register	0x00000004
0x20003030	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003034	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003038	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x2000303C	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003040	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003044	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003048	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x2000304C	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003050	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003054	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003058	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x2000305C	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003060	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003064	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003068	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x2000306C	DPM0_RESTORE[n]	DPM0 Restore Registers	0x00000000
0x20003084	DPM0_REVID	DPM0 Revision ID	0x00000020

Table A-33: ADSP-BF70x EPPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20063000	EPPI0_STAT	EPPI0 Status Register	0x00000000
0x20063004	EPPI0_HCNT	EPPI0 Horizontal Transfer Count Register	0x00000000
0x20063008	EPPI0_HDLY	EPPI0 Horizontal Delay Count Register	0x00000000
0x2006300C	EPPI0_VCNT	EPPI0 Vertical Transfer Count Register	0x00000000
0x20063010	EPPI0_VDLY	EPPI0 Vertical Delay Count Register	0x00000000
0x20063014	EPPI0_FRAME	EPPI0 Lines Per Frame Register	0x00000000

Table A-33: ADSP-BF70x EPPI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20063018	EPPI0_LINE	EPPI0 Samples Per Line Register	0x00000000
0x2006301C	EPPI0_CLKDIV	EPPI0 Clock Divide Register	0x00000000
0x20063020	EPPI0_CTL	EPPI0 Control Register	0x00000000
0x20063024	EPPI0_FS1_WLHB	EPPI0 FS1 Width Register / EPPI Horizontal Blanking Samples Per Line Register	0x00000000
0x20063028	EPPI0_FS1_PASPL	EPPI0 FS1 Period Register / EPPI Active Samples Per Line Register	0x00000000
0x2006302C	EPPI0_FS2_WLVB	EPPI0 FS2 Width Register / EPPI Lines Of Vertical Blanking Register	0x00000000
0x20063030	EPPI0_FS2_PALPF	EPPI0 FS2 Period Register / EPPI Active Lines Per Field Register	0x00000000
0x20063034	EPPI0_IMSK	EPPI0 Interrupt Mask Register	0x00000000
0x2006303C	EPPI0_ODDCLIP	EPPI0 Clipping Register for ODD (Chroma) Data Register	0xFFFF0000
0x20063040	EPPI0_EVENCLIP	EPPI0 Clipping Register for EVEN (Luma) Data Register	0xFFFF0000
0x20063044	EPPI0_FS1_DLY	EPPI0 Frame Sync 1 Delay Value Register	0x00000000
0x20063048	EPPI0_FS2_DLY	EPPI0 Frame Sync 2 Delay Value Register	0x00000000
0x2006304C	EPPI0_CTL2	EPPI0 Control Register 2 Register	0x00000000

Table A-34: ADSP-BF70x HADC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20074000	HADC0_CTL	HADC0 Control Register	0x000001AA
0x20074004	HADC0_CHAN_MSK	HADC0 Channel Mask Register	0x0000FF00
0x20074008	HADC0_IMSK	HADC0 Interrupt Mask Register	0x00000000
0x2007400C	HADC0_STAT	HADC0 Status Register	0x00000000
0x20074010	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074014	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074018	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x2007401C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074020	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074024	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074028	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000

Table A-34: ADSP-BF70x HADC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2007402C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074030	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074034	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074038	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x2007403C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074040	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074044	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x20074048	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000
0x2007404C	HADC0_DATA[nn]	HADC0 Channel Data Registers	0x00000000

Table A-35: ADSP-BF70x L2CTL0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20008000	L2CTL0_CTL	L2CTL0 Control Register	0x00000000
0x20008004	L2CTL0_ACTL_C0	L2CTL0 Access Control Core 0 Register	0x00000000
0x2000800C	L2CTL0_ACTL_SYS	L2CTL0 Access Control System Register	0x00000000
0x20008010	L2CTL0_STAT	L2CTL0 Status Register	0x00000000
0x20008014	L2CTL0_RPCR	L2CTL0 Read Priority Count Register	0x00000F0F
0x20008018	L2CTL0_WPCR	L2CTL0 Write Priority Count Register	0x00000F0F
0x20008024	L2CTL0_INIT	L2CTL0 Initialization Register	0x00000000
0x20008038	L2CTL0_ISTAT	L2CTL0 Initialization Status Register	0x00000000
0x2000803C	L2CTL0_PCTL	L2CTL0 Power Control Register	0x00000000
0x20008040	L2CTL0_ERRADDR0	L2CTL0 ECC Error Address 0 Register	0x08000000
0x20008044	L2CTL0_ERRADDR1	L2CTL0 ECC Error Address 1 Register	0x08020000
0x20008048	L2CTL0_ERRADDR2	L2CTL0 ECC Error Address 2 Register	0x08040000
0x2000804C	L2CTL0_ERRADDR3	L2CTL0 ECC Error Address 3 Register	0x08060000
0x20008050	L2CTL0_ERRADDR4	L2CTL0 ECC Error Address 4 Register	0x08080000
0x20008054	L2CTL0_ERRADDR5	L2CTL0 ECC Error Address 5 Register	0x080A0000
0x20008058	L2CTL0_ERRADDR6	L2CTL0 ECC Error Address 6 Register	0x080C0000
0x2000805C	L2CTL0_ERRADDR7	L2CTL0 ECC Error Address 7 Register	0x080E0000
0x20008060	L2CTL0_ERRADDR8	L2CTL0 ECC Error Address 8 Register	0x04000000

Table A-35: ADSP-BF70x L2CTL0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20008080	L2CTL0_ET0	L2CTL0 Error Type 0 Register	0x00000000
0x20008084	L2CTL0_EADDR0	L2CTL0 Error Type 0 Address Register	0x00000000
0x20008088	L2CTL0_ET1	L2CTL0 Error Type 1 Register	0x00000000
0x2000808C	L2CTL0_EADDR1	L2CTL0 Error Type 1 Address Register	0x00000000
0x200080EC	L2CTL0_SCTL	L2CTL0 Scrub Control Register	0x00000000
0x200080F0	L2CTL0_SADR	L2CTL0 Scrub Start Address Register	0x00000000
0x200080F4	L2CTL0_SCNT	L2CTL0 Scrub Count Register	0x00000000
0x200080FC	L2CTL0_REVID	L2CTL0 Revision ID Register	0x00000001

Table A-36: ADSP-BF70x MSI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20076000	MSI0_CTL	MSI0 Control Register	0x00000000
0x20076008	MSI0_CLKDIV	MSI0 Clock Divider Register	0x00000000
0x20076010	MSI0_CLKEN	MSI0 Clock Enable Register	0x00000000
0x20076014	MSI0_TMOUT	MSI0 Timeout Register	0xFFFFFF40
0x20076018	MSI0_CTYPE	MSI0 Card Type Register	0x00000000
0x2007601C	MSI0_BLKSIZ	MSI0 Block Size Register	0x00000200
0x20076020	MSI0_BYTCNT	MSI0 Byte Count Register	0x00000200
0x20076024	MSI0_IMSK	MSI0 Interrupt Mask Register	0x00000000
0x20076028	MSI0_CMDARG	MSI0 Command Argument Register	0x00000000
0x2007602C	MSI0_CMD	MSI0 Command Register	0x20000000
0x20076030	MSI0_RESP0	MSI0 Response Register 0	0x00000000
0x20076034	MSI0_RESP1	MSI0 Response Register 1	0x00000000
0x20076038	MSI0_RESP2	MSI0 Response Register 2	0x00000000
0x2007603C	MSI0_RESP3	MSI0 Response Register 3	0x00000000
0x20076040	MSI0_MSKISTAT	MSI0 Masked Interrupt Status Register	0x00000000
0x20076044	MSI0_ISTAT	MSI0 Raw Interrupt Status Register	0x00000000
0x20076048	MSI0_STAT	MSI0 Status Register	0x00000106
0x2007604C	MSI0_FIFOTH	MSI0 FIFO Threshold Watermark Register	0x00FF0000
0x20076050	MSI0_CDETECT	MSI0 Card Detect Register	0x00000001

Table A-36: ADSP-BF70x MSI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2007605C	MSI0_TCBCNT	MSI0 Transferred CIU Card Byte Count Register	0x00000000
0x20076060	MSI0_TBBCNT	MSI0 Transferred Host to BIU-FIFO Byte Count Register	0x00000000
0x20076064	MSI0_DEBNCE	MSI0 Debounce Count Register	0x00FFFFFF
0x20076080	MSI0_BUSMODE	MSI0 Bus Mode Register	0x00000000
0x20076084	MSI0_PLDMND	MSI0 Poll Demand Register	0x00000000
0x20076088	MSI0_DBADDR	MSI0 Descriptor List Base Address Register	0x00000000
0x2007608C	MSI0_IDSTS	MSI0 Internal DMA Status Register	0x00000000
0x20076090	MSI0_IDINTEN	MSI0 Internal DMA Interrupt Enable Register	0x00000000
0x20076094	MSI0_DSCADDR	MSI0 Current Host Descriptor Address Register	0x00000000
0x20076098	MSI0_BUFADDR	MSI0 Current Buffer Descriptor Address Register	0x00000000
0x20076100	MSI0_CDTHRCTL	MSI0 Card Threshold Control Register	0x00000000
0x20076108	MSI0_UHS_EXT	MSI0 Ultra High Speed Register Extension	0x00000000
0x20076110	MSI0_ENSHIFT	MSI0 Enable Phase Shift Register	0x00000000

Table A-37: ADSP-BF70x OTPC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20075004	OTPC0_STAT	OTPC0 OTP Status Register	0x00000000
0x2007502C	OTPC0_SECU_STATE	OTPC0 OTP Security State Register	0x00000001

Table A-38: ADSP-BF70x PADS0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20040404	PADS0_PCFG0	PADS0 Peripheral PAD Configuration0 Register	0x00000000

Table A-39: ADSP-BF70x PINT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20041000	PINT0_MSK_SET	PINT0 PINT Mask Set Register	0x00000000
0x20041004	PINT0_MSK_CLR	PINT0 PINT Mask Clear Register	0x00000000
0x20041008	PINT0_REQ	PINT0 PINT Request Register	0x00000000
0x2004100C	PINT0_ASSIGN	PINT0 PINT Assign Register	0x00000101
0x20041010	PINT0_EDGE_SET	PINT0 PINT Edge Set Register	0x00000000

Table A-39: ADSP-BF70x PINT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20041014	PINT0_EDGE_CLR	PINT0 PINT Edge Clear Register	0x00000000
0x20041018	PINT0_INV_SET	PINT0 PINT Invert Set Register	0x00000000
0x2004101C	PINT0_INV_CLR	PINT0 PINT Invert Clear Register	0x00000000
0x20041020	PINT0_PINSTATE	PINT0 PINT Pin State Register	0x00000000
0x20041024	PINT0_LATCH	PINT0 PINT Latch Register	0x00000000

Table A-40: ADSP-BF70x PINT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20041100	PINT1_MSK_SET	PINT1 PINT Mask Set Register	0x00000000
0x20041104	PINT1_MSK_CLR	PINT1 PINT Mask Clear Register	0x00000000
0x20041108	PINT1_REQ	PINT1 PINT Request Register	0x00000000
0x2004110C	PINT1_ASSIGN	PINT1 PINT Assign Register	0x00000101
0x20041110	PINT1_EDGE_SET	PINT1 PINT Edge Set Register	0x00000000
0x20041114	PINT1_EDGE_CLR	PINT1 PINT Edge Clear Register	0x00000000
0x20041118	PINT1_INV_SET	PINT1 PINT Invert Set Register	0x00000000
0x2004111C	PINT1_INV_CLR	PINT1 PINT Invert Clear Register	0x00000000
0x20041120	PINT1_PINSTATE	PINT1 PINT Pin State Register	0x00000000
0x20041124	PINT1_LATCH	PINT1 PINT Latch Register	0x00000000

Table A-41: ADSP-BF70x PINT2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20041200	PINT2_MSK_SET	PINT2 PINT Mask Set Register	0x00000000
0x20041204	PINT2_MSK_CLR	PINT2 PINT Mask Clear Register	0x00000000
0x20041208	PINT2_REQ	PINT2 PINT Request Register	0x00000000
0x2004120C	PINT2_ASSIGN	PINT2 PINT Assign Register	0x00000101
0x20041210	PINT2_EDGE_SET	PINT2 PINT Edge Set Register	0x00000000
0x20041214	PINT2_EDGE_CLR	PINT2 PINT Edge Clear Register	0x00000000
0x20041218	PINT2_INV_SET	PINT2 PINT Invert Set Register	0x00000000
0x2004121C	PINT2_INV_CLR	PINT2 PINT Invert Clear Register	0x00000000
0x20041220	PINT2_PINSTATE	PINT2 PINT Pin State Register	0x00000000

Table A-41: ADSP-BF70x PINT2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20041224	PINT2_LATCH	PINT2 PINT Latch Register	0x00000000

Table A-42: ADSP-BF70x PKA0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200F4000	PKA0_APTR	PKA0 PKA Vector_A Address	0x00000000
0x200F4004	PKA0_BPTR	PKA0 PKA Vector_B Address	0x00000000
0x200F4008	PKA0_CPTR	PKA0 PKA Vector_C Address	0x00000000
0x200F400C	PKA0_DPTR	PKA0 PKA Vector_D Address	0x00000000
0x200F4010	PKA0_ALEN	PKA0 PKA Vector_A Length	0x00000000
0x200F4014	PKA0_BLEN	PKA0 PKA Vector_B Length	0x00000000
0x200F4018	PKA0_SHIFT	PKA0 PKA Bit Shift Value	0x00000000
0x200F401C	PKA0_FUNC	PKA0 PKA Function	0x00000000
0x200F4020	PKA0_COMPARE	PKA0 PKA Compare Result	0x00000001
0x200F4024	PKA0_RESULTMSW	PKA0 PKA Most-Significant-Word of Result Vector	0x00008000
0x200F4028	PKA0_DIVMSW	PKA0 PKA Most-Significant-Word of Divide Remainder	0x00008000
0x200F6000	PKA0_RAM	PKA0 Start of PKA RAM space	0x00000000

Table A-43: ADSP-BF70x PKIC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200F8000	PKIC0_POL_CTL	PKIC0 Polarity Control Register	0x00000000
0x200F8004	PKIC0_TYPE_CTL	PKIC0 Type Control Register	0x00000000
0x200F8008	PKIC0_EN_CTL	PKIC0 Enable Control Register	0x00000000
0x200F800C	PKIC0_RAW_STAT	PKIC0 Raw Status Register	0x00000000
0x200F800C	PKIC0_EN_SET	PKIC0 Enable Set Register	0x00000000
0x200F8010	PKIC0_ACK	PKIC0 Acknowledge Register	0x00000000
0x200F8010	PKIC0_EN_STAT	PKIC0 Enabled Status Register	0x00000000
0x200F8014	PKIC0_EN_CLR	PKIC0 Enable Clear Register	0x00000000

Table A-44: ADSP-BF70x PKTE0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200E0000	PKTE0_CTL_STAT	PKTE0 Packet Engine Control Register	0x00000002
0x200E0004	PKTE0_SRC_ADDR	PKTE0 Packet Engine Source Address	0x00000000
0x200E0008	PKTE0_DEST_ADDR	PKTE0 Packet Engine Destination Address	0x00000000
0x200E000C	PKTE0_SA_ADDR	PKTE0 Packet Engine SA Address	0x00000000
0x200E0010	PKTE0_STATE_ADDR	PKTE0 Packet Engine State Record Address	0x00000000
0x200E0018	PKTE0_USERID	PKTE0 Packet Engine User ID	0x00000000
0x200E001C	PKTE0_LEN	PKTE0 Packet Engine Length Register	0x00800000
0x200E0080	PKTE0_CDRBASE_ADDR	PKTE0 Packet Engine Command Descriptor Ring Base Address	0x00000000
0x200E0084	PKTE0_RDRBASE_ADDR	PKTE0 Packet Engine Result Descriptor Ring Base Address	0x00000000
0x200E0088	PKTE0_RING_CFG	PKTE0 Packet Engine Ring Configuration	0x00000000
0x200E008C	PKTE0_RING_THRESH	PKTE0 Packet Engine Ring Threshold Registers	0x00000000
0x200E0090	PKTE0_CDSC_CNT	PKTE0 Packet Engine Command Descriptor Count Register	0x00000000
0x200E0090	PKTE0_CDSC_INCR	PKTE0 Packet Engine Command Descriptor Count Increment Register	0x00000000
0x200E0094	PKTE0_RDSC_DECR	PKTE0 Packet Engine Result Descriptor Count Decrement Registers	0x00000000
0x200E0094	PKTE0_RDSC_CNT	PKTE0 Packet Engine Result Descriptor Count Registers	0x00000000
0x200E0098	PKTE0_RING_PTR	PKTE0 Packet Engine Ring Pointer Status	0x00000000
0x200E009C	PKTE0_RING_STAT	PKTE0 Packet Engine Ring Status	0x00000000
0x200E0100	PKTE0_CFG	PKTE0 Packet Engine Configuration Register	0x00000000
0x200E0104	PKTE0_STAT	PKTE0 Packet Engine Status Register	0x00040402
0x200E010C	PKTE0_BUF_THRESH	PKTE0 Packet Engine Buffer Threshold Register	0x00800080
0x200E0110	PKTE0_INBUF_CNT	PKTE0 Packet Engine Input Buffer Count Register	0x00000000
0x200E0110	PKTE0_INBUF_INCR	PKTE0 Packet Engine Input Buffer Count Increment Register	0x00000000
0x200E0114	PKTE0_OUTBUF_DECR	PKTE0 Packet Engine Output Buffer Count Decrement Register	0x00000000
0x200E0114	PKTE0_OUTBUF_CNT	PKTE0 Packet Engine Output Buffer Count Register	0x00000000
0x200E0118	PKTE0_BUF_PTR	PKTE0 Packet Engine Buffer Pointer Register	0x00000000
0x200E0120	PKTE0_DMA_CFG	PKTE0 Packet Engine DMA Configuration Register	0x00180006

Table A-44: ADSP-BF70x PKTE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200E01D0	PKTE0_ENDIAN_CFG	PKTE0 Packet Engine Endian Configuration Register	0x00E400E4
0x200E01E0	PKTE0_HLT_CTL	PKTE0 Packet Engine Halt Control Register	0x00000000
0x200E01E0	PKTE0_HLT_STAT	PKTE0 Packet Engine Halt Status Register	0x00000000
0x200E01E4	PKTE0_CONT	PKTE0 PKTE Continue Register	0x00000000
0x200E01E8	PKTE0_CLK_CTL	PKTE0 PE Clock Control Register	0x00000017
0x200E0200	PKTE0_IUMSK_STAT	PKTE0 Interrupt Unmasked Status Register	0x00000000
0x200E0204	PKTE0_IMSK_STAT	PKTE0 Interrupt Masked Status Register	0x00000000
0x200E0204	PKTE0_INT_CLR	PKTE0 Interrupt Clear Register	0x00000000
0x200E0208	PKTE0_INT_EN	PKTE0 Interrupt Enable Register	0x00000000
0x200E020C	PKTE0_INT_CFG	PKTE0 Interrupt Configuration Register	0x00000000
0x200E0210	PKTE0_IMSK_EN	PKTE0 Interrupt Mask Enable Register	0x00000000
0x200E0214	PKTE0_IMSK_DIS	PKTE0 Interrupt Mask Disable Register	0x00000000
0x200E0400	PKTE0_SA_CMD0	PKTE0 SA Command 0	0x00000000
0x200E0404	PKTE0_SA_CMD1	PKTE0 SA Command 1	0x00000000
0x200E0408	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x200E040C	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x200E0410	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x200E0414	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x200E0418	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x200E041C	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x200E0420	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x200E0424	PKTE0_SA_KEY[n]	PKTE0 SA Key Registers	0x00000000
0x200E0428	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x200E042C	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x200E0430	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x200E0434	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x200E0438	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x200E043C	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x200E0440	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x200E0444	PKTE0_SA_IDIGEST[n]	PKTE0 SA Inner Hash Digest Registers	0x00000000
0x200E0448	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000

Table A-44: ADSP-BF70x PKTE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200E044C	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x200E0450	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x200E0454	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x200E0458	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x200E045C	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x200E0460	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x200E0464	PKTE0_SA_ODIGEST[n]	PKTE0 SA Outer Hash Digest Registers	0x00000000
0x200E0468	PKTE0_SA_SPI	PKTE0 SA SPI Register	0x00000000
0x200E046C	PKTE0_SA_SEQNUM[n]	PKTE0 SA Sequence Number Register	0x00000000
0x200E0470	PKTE0_SA_SEQNUM[n]	PKTE0 SA Sequence Number Register	0x00000000
0x200E0474	PKTE0_SA_SEQ-NUM_MSK[n]	PKTE0 SA Sequence Number Mask Registers	0x00000000
0x200E0478	PKTE0_SA_SEQ-NUM_MSK[n]	PKTE0 SA Sequence Number Mask Registers	0x00000000
0x200E047C	PKTE0_SA_RDY	PKTE0 SA Ready Indicator	0x00000000
0x200E047C	PKTE0_SA_NONCE	PKTE0 SA Initialization Vector Register	0x00000000
0x200E0500	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x200E0504	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x200E0508	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x200E050C	PKTE0_STATE_IV[n]	PKTE0 State Initialization Vector Registers	0x00000000
0x200E0510	PKTE0_STATE_BYTE_CNT[n]	PKTE0 State Hash Byte Count Registers	0x00000000
0x200E0514	PKTE0_STATE_BYTE_CNT[n]	PKTE0 State Hash Byte Count Registers	0x00000000
0x200E0518	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x200E051C	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x200E0520	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x200E0524	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x200E0528	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x200E052C	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x200E0530	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000
0x200E0534	PKTE0_STATE_IDIGEST[n]	PKTE0 State Inner Digest Registers	0x00000000

Table A-44: ADSP-BF70x PKTE0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200E0800	PKTE0_DATAIO_BUF	PKTE0 Starting Entry of 256-byte Data Input/Output Buffer	0x00000000

Table A-45: ADSP-BF70x PORTA MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20040000	PORTA_FER	PORTA Port x Function Enable Register	0x00000000
0x20040004	PORTA_FER_SET	PORTA Port x Function Enable Set Register	0x00000000
0x20040008	PORTA_FER_CLR	PORTA Port x Function Enable Clear Register	0x00000000
0x2004000C	PORTA_DATA	PORTA Port x GPIO Data Register	0x00000000
0x20040010	PORTA_DATA_SET	PORTA Port x GPIO Data Set Register	0x00000000
0x20040014	PORTA_DATA_CLR	PORTA Port x GPIO Data Clear Register	0x00000000
0x20040018	PORTA_DIR	PORTA Port x GPIO Direction Register	0x00000000
0x2004001C	PORTA_DIR_SET	PORTA Port x GPIO Direction Set Register	0x00000000
0x20040020	PORTA_DIR_CLR	PORTA Port x GPIO Direction Clear Register	0x00000000
0x20040024	PORTA_INEN	PORTA Port x GPIO Input Enable Register	0x00000000
0x20040028	PORTA_INEN_SET	PORTA Port x GPIO Input Enable Set Register	0x00000000
0x2004002C	PORTA_INEN_CLR	PORTA Port x GPIO Input Enable Clear Register	0x00000000
0x20040030	PORTA_MUX	PORTA Port x Multiplexer Control Register	0x00000000
0x20040034	PORTA_DATA_TGL	PORTA Port x GPIO Output Toggle Register	0x00000000
0x20040038	PORTA_POL	PORTA Port x GPIO Polarity Invert Register	0x00000000
0x2004003C	PORTA_POL_SET	PORTA Port x GPIO Polarity Invert Set Register	0x00000000
0x20040040	PORTA_POL_CLR	PORTA Port x GPIO Polarity Invert Clear Register	0x00000000
0x20040044	PORTA_LOCK	PORTA Port x GPIO Lock Register	0x00000000
0x20040048	PORTA_TRIG_TGL	PORTA Port x GPIO Trigger Toggle Register	0x00000000

Table A-46: ADSP-BF70x PORTB MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20040080	PORTB_FER	PORTB Port x Function Enable Register	0x00000000
0x20040084	PORTB_FER_SET	PORTB Port x Function Enable Set Register	0x00000000
0x20040088	PORTB_FER_CLR	PORTB Port x Function Enable Clear Register	0x00000000

Table A-46: ADSP-BF70x PORTB MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2004008C	PORTB_DATA	PORTB Port x GPIO Data Register	0x00000000
0x20040090	PORTB_DATA_SET	PORTB Port x GPIO Data Set Register	0x00000000
0x20040094	PORTB_DATA_CLR	PORTB Port x GPIO Data Clear Register	0x00000000
0x20040098	PORTB_DIR	PORTB Port x GPIO Direction Register	0x00000000
0x2004009C	PORTB_DIR_SET	PORTB Port x GPIO Direction Set Register	0x00000000
0x200400A0	PORTB_DIR_CLR	PORTB Port x GPIO Direction Clear Register	0x00000000
0x200400A4	PORTB_INEN	PORTB Port x GPIO Input Enable Register	0x00000000
0x200400A8	PORTB_INEN_SET	PORTB Port x GPIO Input Enable Set Register	0x00000000
0x200400AC	PORTB_INEN_CLR	PORTB Port x GPIO Input Enable Clear Register	0x00000000
0x200400B0	PORTB_MUX	PORTB Port x Multiplexer Control Register	0x00000000
0x200400B4	PORTB_DATA_TGL	PORTB Port x GPIO Output Toggle Register	0x00000000
0x200400B8	PORTB_POL	PORTB Port x GPIO Polarity Invert Register	0x00000000
0x200400BC	PORTB_POL_SET	PORTB Port x GPIO Polarity Invert Set Register	0x00000000
0x200400C0	PORTB_POL_CLR	PORTB Port x GPIO Polarity Invert Clear Register	0x00000000
0x200400C4	PORTB_LOCK	PORTB Port x GPIO Lock Register	0x00000000
0x200400C8	PORTB_TRIG_TGL	PORTB Port x GPIO Trigger Toggle Register	0x00000000

Table A-47: ADSP-BF70x PORTC MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20040100	PORTC_FER	PORTC Port x Function Enable Register	0x00000000
0x20040104	PORTC_FER_SET	PORTC Port x Function Enable Set Register	0x00000000
0x20040108	PORTC_FER_CLR	PORTC Port x Function Enable Clear Register	0x00000000
0x2004010C	PORTC_DATA	PORTC Port x GPIO Data Register	0x00000000
0x20040110	PORTC_DATA_SET	PORTC Port x GPIO Data Set Register	0x00000000
0x20040114	PORTC_DATA_CLR	PORTC Port x GPIO Data Clear Register	0x00000000
0x20040118	PORTC_DIR	PORTC Port x GPIO Direction Register	0x00000000
0x2004011C	PORTC_DIR_SET	PORTC Port x GPIO Direction Set Register	0x00000000
0x20040120	PORTC_DIR_CLR	PORTC Port x GPIO Direction Clear Register	0x00000000
0x20040124	PORTC_INEN	PORTC Port x GPIO Input Enable Register	0x00000000
0x20040128	PORTC_INEN_SET	PORTC Port x GPIO Input Enable Set Register	0x00000000

Table A-47: ADSP-BF70x PORTC MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2004012C	PORTC_INEN_CLR	PORTC Port x GPIO Input Enable Clear Register	0x00000000
0x20040130	PORTC_MUX	PORTC Port x Multiplexer Control Register	0x00000000
0x20040134	PORTC_DATA_TGL	PORTC Port x GPIO Output Toggle Register	0x00000000
0x20040138	PORTC_POL	PORTC Port x GPIO Polarity Invert Register	0x00000000
0x2004013C	PORTC_POL_SET	PORTC Port x GPIO Polarity Invert Set Register	0x00000000
0x20040140	PORTC_POL_CLR	PORTC Port x GPIO Polarity Invert Clear Register	0x00000000
0x20040144	PORTC_LOCK	PORTC Port x GPIO Lock Register	0x00000000
0x20040148	PORTC_TRIG_TGL	PORTC Port x GPIO Trigger Toggle Register	0x00000000

Table A-48: ADSP-BF70x RCU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20000000	RCU0_CTL	RCU0 Control Register	0x00000300
0x20000004	RCU0_STAT	RCU0 Status Register	0x00000021
0x20000008	RCU0_CRCTL	RCU0 Core Reset Outputs Control Register	0x00000002
0x2000000C	RCU0_CRSTAT	RCU0 Core Reset Outputs Status Register	0x00000001
0x20000010	RCU0_SIDIS	RCU0 System Interface Disable Register	0x00000000
0x20000014	RCU0_SISTAT	RCU0 System Interface Status Register	0x00000000
0x20000018	RCU0_SVECT_LCK	RCU0 SVECT Lock Register	0x00000000
0x2000001C	RCU0_BCODE	RCU0 Boot Code Register	0x00000000
0x20000020	RCU0_SVECT0	RCU0 Software Vector Register 0	0x00000020
0x20000024	RCU0_SVECT1	RCU0 Software Vector Register 1	0x00C81010
0x20000060	RCU0_MSG	RCU0 Message Register	0x00000000
0x20000064	RCU0_MSG_SET	RCU0 Message Set Bits Register	0x00000000
0x20000068	RCU0_MSG_CLR	RCU0 Message Clear Bits Register	0x00000000
0x20000070	RCU0_REVID	RCU0 Revision ID Register	0x00000020

Table A-49: ADSP-BF70x RTC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200D1280	RTC0_CLK	RTC0 RTC Clock Register	0x00000000
0x200D1284	RTC0_ALM	RTC0 RTC Alarm Register	0x00000000

Table A-49: ADSP-BF70x RTC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D1288	RTC0_IEN	RTC0 Interrupt Enable Register	0x00000000
0x200D128C	RTC0_STAT	RTC0 RTC Status Register	0x00000000
0x200D1290	RTC0_STPWTCH	RTC0 RTC Stop Watch Register	0x00000000
0x200D1298	RTC0_INIT	RTC0 RTC Initialization Register	0x00000000
0x200D129C	RTC0_INITSTAT	RTC0 RTC Initialization Status Register	0x00000000

Table A-50: ADSP-BF70x SCB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20205020	SCB0_IB7_SYNC	SCB0 DDR Interface Block Sync Mode	0x00000004
0x20242100	SCB0_MST00_RQOS	SCB0 Master 00 Read Quality of Service Register	0x00000001
0x20242104	SCB0_MST00_WQOS	SCB0 Master 00 Write Quality of Service Register	0x00000001
0x20243100	SCB0_MST01_RQOS	SCB0 Master 01 Read Quality of Service Register	0x00000001
0x20243104	SCB0_MST01_WQOS	SCB0 Master 01 Write Quality of Service Register	0x00000001
0x20244100	SCB0_MST02_RQOS	SCB0 Master 02 Read Quality of Service Register	0x00000001
0x20244104	SCB0_MST02_WQOS	SCB0 Master 02 Write Quality of Service Register	0x00000001
0x20245100	SCB0_MST03_RQOS	SCB0 Master 03 Read Quality of Service Register	0x00000001
0x20245104	SCB0_MST03_WQOS	SCB0 Master 03 Write Quality of Service Register	0x00000001
0x20246100	SCB0_MST04_RQOS	SCB0 Master 04 Read Quality of Service Register	0x00000001
0x20246104	SCB0_MST04_WQOS	SCB0 Master 04 Write Quality of Service Register	0x00000001
0x20247100	SCB0_MST05_RQOS	SCB0 Master 05 Read Quality of Service Register	0x00000001
0x20247104	SCB0_MST05_WQOS	SCB0 Master 05 Write Quality of Service Register	0x00000001
0x20248100	SCB0_MST06_RQOS	SCB0 Master 06 Read Quality of Service Register	0x00000001
0x20248104	SCB0_MST06_WQOS	SCB0 Master 06 Write Quality of Service Register	0x00000001
0x20249100	SCB0_MST07_RQOS	SCB0 Master 07 Read Quality of Service Register	0x00000001
0x20249104	SCB0_MST07_WQOS	SCB0 Master 07 Write Quality of Service Register	0x00000001
0x2024A100	SCB0_MST08_RQOS	SCB0 Master 08 Read Quality of Service Register	0x00000001
0x2024A104	SCB0_MST08_WQOS	SCB0 Master 08 Write Quality of Service Register	0x00000001
0x2024B100	SCB0_MST09_RQOS	SCB0 Master 09 Read Quality of Service Register	0x00000001
0x2024B104	SCB0_MST09_WQOS	SCB0 Master 09 Write Quality of Service Register	0x00000001
0x2024C100	SCB0_MST10_RQOS	SCB0 Master 10 Read Quality of Service Register	0x00000001

Table A-50: ADSP-BF70x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2024C104	SCB0_MST10_WQOS	SCB0 Master 10 Write Quality of Service Register	0x00000001
0x2024D100	SCB0_MST11_RQOS	SCB0 Master 11 Read Quality of Service Register	0x00000001
0x2024D104	SCB0_MST11_WQOS	SCB0 Master 11 Write Quality of Service Register	0x00000001
0x2024E100	SCB0_MST12_RQOS	SCB0 Master 12 Read Quality of Service Register	0x00000001
0x2024E104	SCB0_MST12_WQOS	SCB0 Master 12 Write Quality of Service Register	0x00000001
0x2024F100	SCB0_MST13_RQOS	SCB0 Master 13 Read Quality of Service Register	0x00000001
0x2024F104	SCB0_MST13_WQOS	SCB0 Master 13 Write Quality of Service Register	0x00000001
0x20250100	SCB0_MST14_RQOS	SCB0 Master 14 Read Quality of Service Register	0x00000001
0x20250104	SCB0_MST14_WQOS	SCB0 Master 14 Write Quality of Service Register	0x00000001
0x20251100	SCB0_MST15_RQOS	SCB0 Master 15 Read Quality of Service Register	0x00000001
0x20251104	SCB0_MST15_WQOS	SCB0 Master 15 Write Quality of Service Register	0x00000001
0x20252100	SCB0_MST16_RQOS	SCB0 Master 16 Read Quality of Service Register	0x00000000
0x20252104	SCB0_MST16_WQOS	SCB0 Master 16 Write Quality of Service Register	0x00000000
0x20253100	SCB0_MST17_RQOS	SCB0 Master 17 Read Quality of Service Register	0x00000000
0x20253104	SCB0_MST17_WQOS	SCB0 Master 17 Write Quality of Service Register	0x00000000
0x20254100	SCB0_MST22_RQOS	SCB0 Master 22 Read Quality of Service Register	0x00000001
0x20254104	SCB0_MST22_WQOS	SCB0 Master 22 Write Quality of Service Register	0x00000001
0x20255100	SCB0_MST25_RQOS	SCB0 Master 25 Read Quality of Service Register	0x00000001
0x20255104	SCB0_MST25_WQOS	SCB0 Master 25 Write Quality of Service Register	0x00000001
0x20256020	SCB0_IB6_SYNC	SCB0 Interface Block Sync Register	0x00000004
0x20256100	SCB0_MST26_RQOS	SCB0 Master 26 Read Quality of Service Register	0x00000001
0x20256104	SCB0_MST26_WQOS	SCB0 Master 26 Write Quality of Service Register	0x00000001
0x20257100	SCB0_MST23_RQOS	SCB0 Master 23 Read Quality of Service Register	0x00000001
0x20257104	SCB0_MST23_WQOS	SCB0 Master 23 Write Quality of Service Register	0x00000001
0x20258100	SCB0_MST24_RQOS	SCB0 Master 24 Read Quality of Service Register	0x00000001
0x20258104	SCB0_MST24_WQOS	SCB0 Master 24 Write Quality of Service Register	0x00000001
0x20259100	SCB0_MST18_RQOS	SCB0 Master 18 Read Quality of Service Register	0x00000000
0x20259104	SCB0_MST18_WQOS	SCB0 Master 18 Write Quality of Service Register	0x00000000
0x2025A100	SCB0_MST19_RQOS	SCB0 Master 19 Read Quality of Service Register	0x00000000
0x2025A104	SCB0_MST19_WQOS	SCB0 Master 19 Write Quality of Service Register	0x00000000
0x2025B100	SCB0_MST20_RQOS	SCB0 Master 20 Read Quality of Service Register	0x00000000

Table A-50: ADSP-BF70x SCB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2025B104	SCB0_MST20_WQOS	SCB0 Master 20 Write Quality of Service Register	0x00000000
0x2025C100	SCB0_MST21_RQOS	SCB0 Master 21 Read Quality of Service Register	0x00000000
0x2025C104	SCB0_MST21_WQOS	SCB0 Master 21 Write Quality of Service Register	0x00000000
0x2025D100	SCB0_MST27_RQOS	SCB0 Master 27 Read Quality of Service Register	0x00000001
0x2025D104	SCB0_MST27_WQOS	SCB0 Master 27 Write Quality of Service Register	0x00000001
0x2025E100	SCB0_MST28_RQOS	SCB0 Master 28 Read Quality of Service Register	0x00000001
0x2025E104	SCB0_MST28_WQOS	SCB0 Master 28 Write Quality of Service Register	0x00000001
0x2025F100	SCB0_MST29_RQOS	SCB0 Master 29 Read Quality of Service Register	0x00000001
0x2025F104	SCB0_MST29_WQOS	SCB0 Master 29 Write Quality of Service Register	0x00000001
0x202C2020	SCB0_MST30_SYNC	SCB0 Interface Block IB4 Sync Mode	0x00000004
0x202C3020	SCB0_MST31_SYNC	SCB0 Interface Block IB5 Sync Mode	0x00000004

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20004000	SEC0_GCTL	SEC0 Global Control Register	0x00000000
0x20004004	SEC0_GSTAT	SEC0 Global Status Register	0x00000000
0x20004008	SEC0_RAISE	SEC0 Global Raise Register	0x00000000
0x2000400C	SEC0_END	SEC0 Global End Register	0x00000000
0x20004010	SEC0_FCTL	SEC0 Fault Control Register	0x00000000
0x20004014	SEC0_FSTAT	SEC0 Fault Status Register	0x00000000
0x20004018	SEC0_FSID	SEC0 Fault Source ID Register	0x00000000
0x2000401C	SEC0_FEND	SEC0 Fault End Register	0x00000000
0x20004020	SEC0_FDLY	SEC0 Fault Delay Register	0x00000000
0x20004024	SEC0_FDLY_CUR	SEC0 Fault Delay Current Register	0x00000000
0x20004028	SEC0_FSRDLY	SEC0 Fault System Reset Delay Register	0x00000000
0x2000402C	SEC0_FSRDLY_CUR	SEC0 Fault System Reset Delay Current Register	0x00000000
0x20004030	SEC0_FCOPP	SEC0 Fault COP Period Register	0x00000000
0x20004034	SEC0_FCOPP_CUR	SEC0 Fault COP Period Current Register	0x00000000
0x20004400	SEC0_CCTL[n]	SEC0 SCI Control Register n	0x00000000
0x20004404	SEC0_CSTAT[n]	SEC0 SCI Status Register n	0x00000000

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20004408	SEC0_CPND[n]	SEC0 Core Pending Register n	0x00000000
0x2000440C	SEC0_CACT[n]	SEC0 SCI Active Register n	0x00000000
0x20004410	SEC0_CPMSK[n]	SEC0 SCI Priority Mask Register n	0x000000FF
0x20004414	SEC0_CGMSK[n]	SEC0 SCI Group Mask Register n	0x00000000
0x20004418	SEC0_CPLVL[n]	SEC0 SCI Priority Level Register n	0x00000007
0x2000441C	SEC0_CSID[n]	SEC0 SCI Source ID Register n	0x00000000
0x20004800	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004804	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004808	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000480C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004810	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004814	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004818	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000481C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004820	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004824	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004828	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000482C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004830	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004834	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004838	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000483C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004840	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004844	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004848	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000484C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004850	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004854	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004858	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000485C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004860	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20004864	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004868	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000486C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004870	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004874	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004878	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000487C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004880	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004884	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004888	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000488C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004890	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004894	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004898	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000489C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200048E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200048F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200048FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004900	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004904	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004908	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000490C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004910	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004914	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004918	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000491C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004920	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004924	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004928	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000492C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004930	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004934	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004938	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000493C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004940	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004944	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004948	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000494C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004950	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004954	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004958	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2000495C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004960	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004964	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004968	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000496C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004970	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004974	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004978	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000497C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004980	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004984	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004988	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000498C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004990	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004994	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004998	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x2000499C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049A0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049A4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049A8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049AC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049B0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049B4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049B8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049BC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049C0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049C4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049C8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049CC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049D0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049D4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200049D8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049DC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049E0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049E4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049E8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049EC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049F0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049F4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x200049F8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x200049FC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A50	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20004A54	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A58	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A5C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A60	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A64	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A68	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A6C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A70	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A74	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A78	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A7C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A80	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A84	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A88	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A8C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A90	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A94	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004A98	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004A9C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AA0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AA4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AA8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AAC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AB0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AB4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AB8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004ABC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AC0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AC4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AC8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004ACC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20004AD0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AD4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AD8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004ADC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AE0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AE4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AE8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AEC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AF0	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AF4	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004AF8	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004AFC	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B00	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B04	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B08	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B0C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B10	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B14	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B18	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B1C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B20	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B24	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B28	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B2C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B30	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B34	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B38	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B3C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B40	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000
0x20004B44	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000
0x20004B48	SEC0_SCTL[n]	SEC0 Source Control Register n	0x00000000

Table A-51: ADSP-BF70x SEC0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20004B4C	SEC0_SSTAT[n]	SEC0 Source Status Register n	0x00000000

Table A-52: ADSP-BF70x SMC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2005D00C	SMC0_B0CTL	SMC0 Bank 0 Control Register	0x01000000
0x2005D010	SMC0_B0TIM	SMC0 Bank 0 Timing Register	0x01010101
0x2005D014	SMC0_B0ETIM	SMC0 Bank 0 Extended Timing Register	0x00020200
0x2005D01C	SMC0_B1CTL	SMC0 Bank 1 Control Register	0x00000000
0x2005D020	SMC0_B1TIM	SMC0 Bank 1 Timing Register	0x01010101
0x2005D024	SMC0_B1ETIM	SMC0 Bank 1 Extended Timing Register	0x00020200
0x2005D02C	SMC0_B2CTL	SMC0 Bank 2 Control Register	0x00000000
0x2005D030	SMC0_B2TIM	SMC0 Bank 2 Timing Register	0x01010101
0x2005D034	SMC0_B2ETIM	SMC0 Bank 2 Extended Timing Register	0x00020200
0x2005D03C	SMC0_B3CTL	SMC0 Bank 3 Control Register	0x00000000
0x2005D040	SMC0_B3TIM	SMC0 Bank 3 Timing Register	0x01010101
0x2005D044	SMC0_B3ETIM	SMC0 Bank 3 Extended Timing Register	0x00020200

Table A-53: ADSP-BF70x SMPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20006000	SMPU0_CTL	SMPU0 SMPU Control Register	0x00000000
0x20006004	SMPU0_STAT	SMPU0 SMPU Status Register	0x00000000
0x20006008	SMPU0_IADDR	SMPU0 Interrupt Address Register	0x00000000
0x2000600C	SMPU0_IDTLS	SMPU0 Interrupt Details Register	0x00000000
0x20006010	SMPU0_BADDR	SMPU0 Bus Error Address Register	0x00000000
0x20006014	SMPU0_BDTLS	SMPU0 Bus Error Details Register	0x00000000
0x20006020	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x20006024	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x20006028	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x2000602C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x20006030	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000

Table A-53: ADSP-BF70x SMPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20006034	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x20006038	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x2000603C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x20006040	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x20006044	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x20006048	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x2000604C	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x20006050	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x20006054	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x20006058	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x2000605C	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x20006060	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x20006064	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x20006068	SMPU0_RCTL[n]	SMPU0 Region n Control Register	0x00000000
0x2000606C	SMPU0_RADDR[n]	SMPU0 Region n Address Register	0x00000000
0x20006070	SMPU0_RIDA[n]	SMPU0 Region n ID A Register	0x00000000
0x20006074	SMPU0_RIDMSKA[n]	SMPU0 Region n ID Mask A Register	0x00000000
0x20006078	SMPU0_RIDB[n]	SMPU0 Region n ID B Register	0x00000000
0x2000607C	SMPU0_RIDMSKB[n]	SMPU0 Region n ID Mask B Register	0x00000000
0x20006220	SMPU0_REVID	SMPU0 SMPU Revision ID Register	0x00000010
0x20006800	SMPU0_SECURECTL	SMPU0 SMPU Control Secure Accesses Register	0x00000000
0x20006820	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x20006824	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x20006828	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000
0x2000682C	SMPU0_SECURERCTL[n]	SMPU0 Region n Control Secure Accesses Register	0x00000000

Table A-54: ADSP-BF70x SMPU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200C8000	SMPU1_CTL	SMPU1 SMPU Control Register	0x00000000
0x200C8004	SMPU1_STAT	SMPU1 SMPU Status Register	0x00000000

Table A-54: ADSP-BF70x SMPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200C8008	SMPU1_IADDR	SMPU1 Interrupt Address Register	0x00000000
0x200C800C	SMPU1_IDTLS	SMPU1 Interrupt Details Register	0x00000000
0x200C8010	SMPU1_BADDR	SMPU1 Bus Error Address Register	0x00000000
0x200C8014	SMPU1_BDTLS	SMPU1 Bus Error Details Register	0x00000000
0x200C8020	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x200C8024	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x200C8028	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x200C802C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x200C8030	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x200C8034	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x200C8038	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x200C803C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x200C8040	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x200C8044	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x200C8048	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x200C804C	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x200C8050	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x200C8054	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x200C8058	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x200C805C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x200C8060	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x200C8064	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x200C8068	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x200C806C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x200C8070	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x200C8074	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x200C8078	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x200C807C	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x200C8080	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x200C8084	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x200C8088	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000

Table A-54: ADSP-BF70x SMPU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200C808C	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x200C8090	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x200C8094	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x200C8098	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x200C809C	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x200C80A0	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x200C80A4	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x200C80A8	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x200C80AC	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x200C80B0	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x200C80B4	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x200C80B8	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x200C80BC	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x200C80C0	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x200C80C4	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x200C80C8	SMPU1_RCTL[n]	SMPU1 Region n Control Register	0x00000000
0x200C80CC	SMPU1_RADDR[n]	SMPU1 Region n Address Register	0x00000000
0x200C80D0	SMPU1_RIDA[n]	SMPU1 Region n ID A Register	0x00000000
0x200C80D4	SMPU1_RIDMSKA[n]	SMPU1 Region n ID Mask A Register	0x00000000
0x200C80D8	SMPU1_RIDB[n]	SMPU1 Region n ID B Register	0x00000000
0x200C80DC	SMPU1_RIDMSKB[n]	SMPU1 Region n ID Mask B Register	0x00000000
0x200C8220	SMPU1_REVID	SMPU1 SMPU Revision ID Register	0x00000010
0x200C8800	SMPU1_SECURECTL	SMPU1 SMPU Control Secure Accesses Register	0x00000000
0x200C8820	SMPU1_SECURERCTL[n]	SMPU1 Region n Control Secure Accesses Register	0x00000000
0x200C8824	SMPU1_SECURERCTL[n]	SMPU1 Region n Control Secure Accesses Register	0x00000000
0x200C8828	SMPU1_SECURERCTL[n]	SMPU1 Region n Control Secure Accesses Register	0x00000000
0x200C882C	SMPU1_SECURERCTL[n]	SMPU1 Region n Control Secure Accesses Register	0x00000000
0x200C8830	SMPU1_SECURERCTL[n]	SMPU1 Region n Control Secure Accesses Register	0x00000000
0x200C8834	SMPU1_SECURERCTL[n]	SMPU1 Region n Control Secure Accesses Register	0x00000000
0x200C8838	SMPU1_SECURERCTL[n]	SMPU1 Region n Control Secure Accesses Register	0x00000000
0x200C883C	SMPU1_SECURERCTL[n]	SMPU1 Region n Control Secure Accesses Register	0x00000000

Table A-55: ADSP-BF70x SPI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20055004	SPI0_CTL	SPI0 Control Register	0x00000050
0x20055008	SPI0_RXCTL	SPI0 Receive Control Register	0x00000000
0x2005500C	SPI0_TXCTL	SPI0 Transmit Control Register	0x00000000
0x20055010	SPI0_CLK	SPI0 Clock Rate Register	0x00000000
0x20055014	SPI0_DLY	SPI0 Delay Register	0x00000301
0x20055018	SPI0_SLVSEL	SPI0 Slave Select Register	0x0000FE00
0x2005501C	SPI0_RWC	SPI0 Received Word Count Register	0x00000000
0x20055020	SPI0_RWCR	SPI0 Received Word Count Reload Register	0x00000000
0x20055024	SPI0_TWC	SPI0 Transmitted Word Count Register	0x00000000
0x20055028	SPI0_TWCR	SPI0 Transmitted Word Count Reload Register	0x00000000
0x20055030	SPI0_IMSK	SPI0 Interrupt Mask Register	0x00000000
0x20055034	SPI0_IMSK_CLR	SPI0 Interrupt Mask Clear Register	0x00000000
0x20055038	SPI0_IMSK_SET	SPI0 Interrupt Mask Set Register	0x00000000
0x20055040	SPI0_STAT	SPI0 Status Register	0x00440001
0x20055044	SPI0_ILAT	SPI0 Masked Interrupt Condition Register	0x00000000
0x20055048	SPI0_ILAT_CLR	SPI0 Masked Interrupt Clear Register	0x00000000
0x20055050	SPI0_RFIFO	SPI0 Receive FIFO Data Register	0x00000000
0x20055058	SPI0_TFIFO	SPI0 Transmit FIFO Data Register	0x00000000
0x20055060	SPI0_MMRDH	SPI0 Memory Mapped Read Header	0x00000000
0x20055064	SPI0_MMTOP	SPI0 SPI Memory Top Address	0x00000000

Table A-56: ADSP-BF70x SPI1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20056004	SPI1_CTL	SPI1 Control Register	0x00000050
0x20056008	SPI1_RXCTL	SPI1 Receive Control Register	0x00000000
0x2005600C	SPI1_TXCTL	SPI1 Transmit Control Register	0x00000000
0x20056010	SPI1_CLK	SPI1 Clock Rate Register	0x00000000
0x20056014	SPI1_DLY	SPI1 Delay Register	0x00000301
0x20056018	SPI1_SLVSEL	SPI1 Slave Select Register	0x0000FE00
0x2005601C	SPI1_RWC	SPI1 Received Word Count Register	0x00000000

Table A-56: ADSP-BF70x SPI1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20056020	SPI1_RWCR	SPI1 Received Word Count Reload Register	0x00000000
0x20056024	SPI1_TWC	SPI1 Transmitted Word Count Register	0x00000000
0x20056028	SPI1_TWCR	SPI1 Transmitted Word Count Reload Register	0x00000000
0x20056030	SPI1_IMSK	SPI1 Interrupt Mask Register	0x00000000
0x20056034	SPI1_IMSK_CLR	SPI1 Interrupt Mask Clear Register	0x00000000
0x20056038	SPI1_IMSK_SET	SPI1 Interrupt Mask Set Register	0x00000000
0x20056040	SPI1_STAT	SPI1 Status Register	0x00440001
0x20056044	SPI1_ILAT	SPI1 Masked Interrupt Condition Register	0x00000000
0x20056048	SPI1_ILAT_CLR	SPI1 Masked Interrupt Clear Register	0x00000000
0x20056050	SPI1_RFIFO	SPI1 Receive FIFO Data Register	0x00000000
0x20056058	SPI1_TFIFO	SPI1 Transmit FIFO Data Register	0x00000000
0x20056060	SPI1_MMRDH	SPI1 Memory Mapped Read Header	0x00000000
0x20056064	SPI1_MMTOP	SPI1 SPI Memory Top Address	0x00000000

Table A-57: ADSP-BF70x SPI2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20057004	SPI2_CTL	SPI2 Control Register	0x00000050
0x20057008	SPI2_RXCTL	SPI2 Receive Control Register	0x00000000
0x2005700C	SPI2_TXCTL	SPI2 Transmit Control Register	0x00000000
0x20057010	SPI2_CLK	SPI2 Clock Rate Register	0x00000000
0x20057014	SPI2_DLY	SPI2 Delay Register	0x00000301
0x20057018	SPI2_SLVSEL	SPI2 Slave Select Register	0x0000FE00
0x2005701C	SPI2_RWC	SPI2 Received Word Count Register	0x00000000
0x20057020	SPI2_RWCR	SPI2 Received Word Count Reload Register	0x00000000
0x20057024	SPI2_TWC	SPI2 Transmitted Word Count Register	0x00000000
0x20057028	SPI2_TWCR	SPI2 Transmitted Word Count Reload Register	0x00000000
0x20057030	SPI2_IMSK	SPI2 Interrupt Mask Register	0x00000000
0x20057034	SPI2_IMSK_CLR	SPI2 Interrupt Mask Clear Register	0x00000000
0x20057038	SPI2_IMSK_SET	SPI2 Interrupt Mask Set Register	0x00000000
0x20057040	SPI2_STAT	SPI2 Status Register	0x00440001

Table A-57: ADSP-BF70x SPI2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20057044	SPI2_ILAT	SPI2 Masked Interrupt Condition Register	0x00000000
0x20057048	SPI2_ILAT_CLR	SPI2 Masked Interrupt Clear Register	0x00000000
0x20057050	SPI2_RFIFO	SPI2 Receive FIFO Data Register	0x00000000
0x20057058	SPI2_TFIFO	SPI2 Transmit FIFO Data Register	0x00000000
0x20057060	SPI2_MMRDH	SPI2 Memory Mapped Read Header	0x00000000
0x20057064	SPI2_MMTOP	SPI2 SPI Memory Top Address	0x00000000

Table A-58: ADSP-BF70x SPIHP0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20058000	SPIHP0_CTL	SPIHP0 Control Register	0x00000000
0x20058004	SPIHP0_STAT	SPIHP0 Status Register	0x00000000
0x20058008	SPIHP0_RDPF	SPIHP0 Read Prefetch Register	0x00000000
0x20058010	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058014	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058018	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x2005801C	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058020	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058024	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058028	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x2005802C	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058030	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058034	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058038	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x2005803C	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058040	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058044	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058048	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x2005804C	SPIHP0_BAR[n]	SPIHP0 Base Address Register	0x00000000
0x20058050	SPIHP0_AUX[n]	SPIHP0 Auxiliary Register	0x00000000
0x20058054	SPIHP0_AUX[n]	SPIHP0 Auxiliary Register	0x00000000

Table A-58: ADSP-BF70x SPIHP0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20058058	SPIHP0_AUX[n]	SPIHP0 Auxiliary Register	0x00000000
0x2005805C	SPIHP0_AUX[n]	SPIHP0 Auxiliary Register	0x00000000

Table A-59: ADSP-BF70x SPORT0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2004D000	SPORT0_CTL_A	SPORT0 Half SPORT 'A' Control Register	0x00000000
0x2004D004	SPORT0_DIV_A	SPORT0 Half SPORT 'A' Divisor Register	0x00000000
0x2004D008	SPORT0_MCTL_A	SPORT0 Half SPORT 'A' Multichannel Control Register	0x00000000
0x2004D00C	SPORT0_CS0_A	SPORT0 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x2004D010	SPORT0_CS1_A	SPORT0 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x2004D014	SPORT0_CS2_A	SPORT0 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x2004D018	SPORT0_CS3_A	SPORT0 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x2004D020	SPORT0_ERR_A	SPORT0 Half SPORT 'A' Error Register	0x00000000
0x2004D024	SPORT0_MSTAT_A	SPORT0 Half SPORT 'A' Multichannel Status Register	0x00000000
0x2004D028	SPORT0_CTL2_A	SPORT0 Half SPORT 'A' Control 2 Register	0x00000000
0x2004D040	SPORT0_TXPRI_A	SPORT0 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x2004D044	SPORT0_RXPRI_A	SPORT0 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x2004D048	SPORT0_TXSEC_A	SPORT0 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x2004D04C	SPORT0_RXSEC_A	SPORT0 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x2004D080	SPORT0_CTL_B	SPORT0 Half SPORT 'B' Control Register	0x00000000
0x2004D084	SPORT0_DIV_B	SPORT0 Half SPORT 'B' Divisor Register	0x00000000
0x2004D088	SPORT0_MCTL_B	SPORT0 Half SPORT 'B' Multichannel Control Register	0x00000000
0x2004D08C	SPORT0_CS0_B	SPORT0 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x2004D090	SPORT0_CS1_B	SPORT0 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x2004D094	SPORT0_CS2_B	SPORT0 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000

Table A-59: ADSP-BF70x SPORT0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2004D098	SPORT0_CS3_B	SPORT0 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x2004D0A0	SPORT0_ERR_B	SPORT0 Half SPORT 'B' Error Register	0x00000000
0x2004D0A4	SPORT0_MSTAT_B	SPORT0 Half SPORT 'B' Multichannel Status Register	0x00000000
0x2004D0A8	SPORT0_CTL2_B	SPORT0 Half SPORT 'B' Control 2 Register	0x00000000
0x2004D0C0	SPORT0_TXPRI_B	SPORT0 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x2004D0C4	SPORT0_RXPRI_B	SPORT0 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x2004D0C8	SPORT0_TXSEC_B	SPORT0 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x2004D0CC	SPORT0_RXSEC_B	SPORT0 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-60: ADSP-BF70x SPORT1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2004E000	SPORT1_CTL_A	SPORT1 Half SPORT 'A' Control Register	0x00000000
0x2004E004	SPORT1_DIV_A	SPORT1 Half SPORT 'A' Divisor Register	0x00000000
0x2004E008	SPORT1_MCTL_A	SPORT1 Half SPORT 'A' Multichannel Control Register	0x00000000
0x2004E00C	SPORT1_CS0_A	SPORT1 Half SPORT 'A' Multichannel 0-31 Select Register	0x00000000
0x2004E010	SPORT1_CS1_A	SPORT1 Half SPORT 'A' Multichannel 32-63 Select Register	0x00000000
0x2004E014	SPORT1_CS2_A	SPORT1 Half SPORT 'A' Multichannel 64-95 Select Register	0x00000000
0x2004E018	SPORT1_CS3_A	SPORT1 Half SPORT 'A' Multichannel 96-127 Select Register	0x00000000
0x2004E020	SPORT1_ERR_A	SPORT1 Half SPORT 'A' Error Register	0x00000000
0x2004E024	SPORT1_MSTAT_A	SPORT1 Half SPORT 'A' Multichannel Status Register	0x00000000
0x2004E028	SPORT1_CTL2_A	SPORT1 Half SPORT 'A' Control 2 Register	0x00000000
0x2004E040	SPORT1_TXPRI_A	SPORT1 Half SPORT 'A' Tx Buffer (Primary) Register	0x00000000
0x2004E044	SPORT1_RXPRI_A	SPORT1 Half SPORT 'A' Rx Buffer (Primary) Register	0x00000000
0x2004E048	SPORT1_TXSEC_A	SPORT1 Half SPORT 'A' Tx Buffer (Secondary) Register	0x00000000
0x2004E04C	SPORT1_RXSEC_A	SPORT1 Half SPORT 'A' Rx Buffer (Secondary) Register	0x00000000
0x2004E080	SPORT1_CTL_B	SPORT1 Half SPORT 'B' Control Register	0x00000000
0x2004E084	SPORT1_DIV_B	SPORT1 Half SPORT 'B' Divisor Register	0x00000000

Table A-60: ADSP-BF70x SPORT1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2004E088	SPORT1_MCTL_B	SPORT1 Half SPORT 'B' Multichannel Control Register	0x00000000
0x2004E08C	SPORT1_CS0_B	SPORT1 Half SPORT 'B' Multichannel 0-31 Select Register	0x00000000
0x2004E090	SPORT1_CS1_B	SPORT1 Half SPORT 'B' Multichannel 32-63 Select Register	0x00000000
0x2004E094	SPORT1_CS2_B	SPORT1 Half SPORT 'B' Multichannel 64-95 Select Register	0x00000000
0x2004E098	SPORT1_CS3_B	SPORT1 Half SPORT 'B' Multichannel 96-127 Select Register	0x00000000
0x2004E0A0	SPORT1_ERR_B	SPORT1 Half SPORT 'B' Error Register	0x00000000
0x2004E0A4	SPORT1_MSTAT_B	SPORT1 Half SPORT 'B' Multichannel Status Register	0x00000000
0x2004E0A8	SPORT1_CTL2_B	SPORT1 Half SPORT 'B' Control 2 Register	0x00000000
0x2004E0C0	SPORT1_TXPRI_B	SPORT1 Half SPORT 'B' Tx Buffer (Primary) Register	0x00000000
0x2004E0C4	SPORT1_RXPRI_B	SPORT1 Half SPORT 'B' Rx Buffer (Primary) Register	0x00000000
0x2004E0C8	SPORT1_TXSEC_B	SPORT1 Half SPORT 'B' Tx Buffer (Secondary) Register	0x00000000
0x2004E0CC	SPORT1_RXSEC_B	SPORT1 Half SPORT 'B' Rx Buffer (Secondary) Register	0x00000000

Table A-61: ADSP-BF70x SPU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20005000	SPU0_CTL	SPU0 Control Register	0x000000AD
0x20005004	SPU0_STAT	SPU0 Status Register	0x00000000
0x20005400	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005404	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005408	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000540C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005410	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005414	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005418	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000541C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005420	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005424	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005428	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-61: ADSP-BF70x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2000542C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005430	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005434	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005438	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000543C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005440	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005444	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005448	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000544C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005450	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005454	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005458	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000545C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005460	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005464	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005468	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000546C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005470	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005474	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005478	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000547C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005480	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005484	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005488	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000548C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005490	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005494	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005498	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000549C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054A0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054A4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-61: ADSP-BF70x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200054A8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054AC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054B0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054B4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054B8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054BC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054C0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054C4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054C8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054CC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054D0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054D4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054D8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054DC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054E0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054E4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054E8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054EC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054F0	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054F4	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054F8	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x200054FC	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005500	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005504	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005508	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000550C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005510	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005514	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005518	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000551C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005520	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000

Table A-61: ADSP-BF70x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20005524	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005528	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x2000552C	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005530	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005534	SPU0_WP[n]	SPU0 Write Protect Register n	0x00000000
0x20005840	SPU0_SECURECTL	SPU0 Secure Control Register	0x00000000
0x2000584C	SPU0_SECURECHK	SPU0 Secure Check Register	0xFFFFFFFF
0x20005980	SPU0_SECUREC[n]	SPU0 Secure Core Registers	0x00000001
0x20005A00	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A04	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A08	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A0C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A10	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A14	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A18	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A1C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A20	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A24	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A28	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A2C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A30	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A34	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A38	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A3C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A40	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A44	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A48	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A4C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A50	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A54	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A58	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-61: ADSP-BF70x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20005A5C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A60	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A64	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A68	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A6C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A70	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A74	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A78	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A7C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A80	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A84	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A88	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A8C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A90	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A94	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A98	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005A9C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AA0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AA4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AA8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AAC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AB0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AB4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AB8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005ABC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AC0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AC4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AC8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005ACC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AD0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AD4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-61: ADSP-BF70x SPU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20005AD8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005ADC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AE0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AE4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AE8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AEC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AF0	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AF4	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AF8	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005AFC	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B00	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B04	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B08	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B0C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B10	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B14	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B18	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B1C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B20	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B24	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B28	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B2C	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B30	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001
0x20005B34	SPU0_SECUREP[n]	SPU0 Secure Peripheral Register	0x00000001

Table A-62: ADSP-BF70x SWU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2000C000	SWU0_GCTL	SWU0 Global Control Register	0x00000000
0x2000C004	SWU0_GSTAT	SWU0 Global Status Register	0x00000000
0x2000C010	SWU0_CTL[n]	SWU0 Control Register n	0x00000000

Table A-62: ADSP-BF70x SWU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2000C014	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x2000C018	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x2000C01C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x2000C020	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x2000C024	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x2000C028	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x2000C02C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x2000C030	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x2000C034	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x2000C038	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x2000C03C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x2000C040	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x2000C044	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x2000C048	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x2000C04C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x2000C050	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x2000C054	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x2000C058	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x2000C05C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x2000C060	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x2000C064	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x2000C068	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x2000C06C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000
0x2000C070	SWU0_CTL[n]	SWU0 Control Register n	0x00000000
0x2000C074	SWU0_LA[n]	SWU0 Lower Address Register n	0x00000000
0x2000C078	SWU0_UA[n]	SWU0 Upper Address Register n	0x00000000
0x2000C07C	SWU0_ID[n]	SWU0 ID Register n	0x00000000
0x2000C080	SWU0_CNT[n]	SWU0 Count Register n	0x00000000
0x2000C084	SWU0_TARG[n]	SWU0 Target Register n	0x00000000
0x2000C088	SWU0_HIST[n]	SWU0 Bandwidth History Register n	0x00000000
0x2000C08C	SWU0_CUR[n]	SWU0 Current Register n	0x00000000

Table A-63: ADSP-BF70x SWU1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2000E000	SWU1_GCTL	SWU1 Global Control Register	0x00000000
0x2000E004	SWU1_GSTAT	SWU1 Global Status Register	0x00000000
0x2000E010	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x2000E014	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x2000E018	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x2000E01C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x2000E020	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x2000E024	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x2000E028	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x2000E02C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x2000E030	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x2000E034	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x2000E038	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x2000E03C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x2000E040	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x2000E044	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x2000E048	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x2000E04C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x2000E050	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x2000E054	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x2000E058	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x2000E05C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x2000E060	SWU1_CNT[n]	SWU1 Count Register n	0x00000000
0x2000E064	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x2000E068	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x2000E06C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000
0x2000E070	SWU1_CTL[n]	SWU1 Control Register n	0x00000000
0x2000E074	SWU1_LA[n]	SWU1 Lower Address Register n	0x00000000
0x2000E078	SWU1_UA[n]	SWU1 Upper Address Register n	0x00000000
0x2000E07C	SWU1_ID[n]	SWU1 ID Register n	0x00000000
0x2000E080	SWU1_CNT[n]	SWU1 Count Register n	0x00000000

Table A-63: ADSP-BF70x SWU1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2000E084	SWU1_TARG[n]	SWU1 Target Register n	0x00000000
0x2000E088	SWU1_HIST[n]	SWU1 Bandwidth History Register n	0x00000000
0x2000E08C	SWU1_CUR[n]	SWU1 Current Register n	0x00000000

Table A-64: ADSP-BF70x SWU2 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2000F000	SWU2_GCTL	SWU2 Global Control Register	0x00000000
0x2000F004	SWU2_GSTAT	SWU2 Global Status Register	0x00000000
0x2000F010	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x2000F014	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x2000F018	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x2000F01C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x2000F020	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x2000F024	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x2000F028	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x2000F02C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x2000F030	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x2000F034	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x2000F038	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x2000F03C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x2000F040	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x2000F044	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x2000F048	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x2000F04C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x2000F050	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x2000F054	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x2000F058	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x2000F05C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x2000F060	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x2000F064	SWU2_TARG[n]	SWU2 Target Register n	0x00000000

Table A-64: ADSP-BF70x SWU2 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2000F068	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x2000F06C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000
0x2000F070	SWU2_CTL[n]	SWU2 Control Register n	0x00000000
0x2000F074	SWU2_LA[n]	SWU2 Lower Address Register n	0x00000000
0x2000F078	SWU2_UA[n]	SWU2 Upper Address Register n	0x00000000
0x2000F07C	SWU2_ID[n]	SWU2 ID Register n	0x00000000
0x2000F080	SWU2_CNT[n]	SWU2 Count Register n	0x00000000
0x2000F084	SWU2_TARG[n]	SWU2 Target Register n	0x00000000
0x2000F088	SWU2_HIST[n]	SWU2 Bandwidth History Register n	0x00000000
0x2000F08C	SWU2_CUR[n]	SWU2 Current Register n	0x00000000

Table A-65: ADSP-BF70x SWU3 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2000D000	SWU3_GCTL	SWU3 Global Control Register	0x00000000
0x2000D004	SWU3_GSTAT	SWU3 Global Status Register	0x00000000
0x2000D010	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x2000D014	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x2000D018	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x2000D01C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x2000D020	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x2000D024	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x2000D028	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x2000D02C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x2000D030	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x2000D034	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x2000D038	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x2000D03C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x2000D040	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x2000D044	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x2000D048	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000

Table A-65: ADSP-BF70x SWU3 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2000D04C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x2000D050	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x2000D054	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x2000D058	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x2000D05C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x2000D060	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x2000D064	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x2000D068	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x2000D06C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000
0x2000D070	SWU3_CTL[n]	SWU3 Control Register n	0x00000000
0x2000D074	SWU3_LA[n]	SWU3 Lower Address Register n	0x00000000
0x2000D078	SWU3_UA[n]	SWU3 Upper Address Register n	0x00000000
0x2000D07C	SWU3_ID[n]	SWU3 ID Register n	0x00000000
0x2000D080	SWU3_CNT[n]	SWU3 Count Register n	0x00000000
0x2000D084	SWU3_TARG[n]	SWU3 Target Register n	0x00000000
0x2000D088	SWU3_HIST[n]	SWU3 Bandwidth History Register n	0x00000000
0x2000D08C	SWU3_CUR[n]	SWU3 Current Register n	0x00000000

Table A-66: ADSP-BF70x SWU4 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200CC000	SWU4_GCTL	SWU4 Global Control Register	0x00000000
0x200CC004	SWU4_GSTAT	SWU4 Global Status Register	0x00000000
0x200CC010	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x200CC014	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x200CC018	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x200CC01C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x200CC020	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x200CC024	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x200CC028	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x200CC02C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000

Table A-66: ADSP-BF70x SWU4 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200CC030	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x200CC034	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x200CC038	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x200CC03C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x200CC040	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x200CC044	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x200CC048	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x200CC04C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x200CC050	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x200CC054	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x200CC058	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x200CC05C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x200CC060	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x200CC064	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x200CC068	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x200CC06C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000
0x200CC070	SWU4_CTL[n]	SWU4 Control Register n	0x00000000
0x200CC074	SWU4_LA[n]	SWU4 Lower Address Register n	0x00000000
0x200CC078	SWU4_UA[n]	SWU4 Upper Address Register n	0x00000000
0x200CC07C	SWU4_ID[n]	SWU4 ID Register n	0x00000000
0x200CC080	SWU4_CNT[n]	SWU4 Count Register n	0x00000000
0x200CC084	SWU4_TARG[n]	SWU4 Target Register n	0x00000000
0x200CC088	SWU4_HIST[n]	SWU4 Bandwidth History Register n	0x00000000
0x200CC08C	SWU4_CUR[n]	SWU4 Current Register n	0x00000000

Table A-67: ADSP-BF70x SWU5 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20077000	SWU5_GCTL	SWU5 Global Control Register	0x00000000
0x20077004	SWU5_GSTAT	SWU5 Global Status Register	0x00000000
0x20077010	SWU5_CTL[n]	SWU5 Control Register n	0x00000000

Table A-67: ADSP-BF70x SWU5 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20077014	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x20077018	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x2007701C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x20077020	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x20077024	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x20077028	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x2007702C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x20077030	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x20077034	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x20077038	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x2007703C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x20077040	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x20077044	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x20077048	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x2007704C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x20077050	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x20077054	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x20077058	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x2007705C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x20077060	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x20077064	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x20077068	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x2007706C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000
0x20077070	SWU5_CTL[n]	SWU5 Control Register n	0x00000000
0x20077074	SWU5_LA[n]	SWU5 Lower Address Register n	0x00000000
0x20077078	SWU5_UA[n]	SWU5 Upper Address Register n	0x00000000
0x2007707C	SWU5_ID[n]	SWU5 ID Register n	0x00000000
0x20077080	SWU5_CNT[n]	SWU5 Count Register n	0x00000000
0x20077084	SWU5_TARG[n]	SWU5 Target Register n	0x00000000
0x20077088	SWU5_HIST[n]	SWU5 Bandwidth History Register n	0x00000000
0x2007708C	SWU5_CUR[n]	SWU5 Current Register n	0x00000000

Table A-68: ADSP-BF70x SWU6 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20078000	SWU6_GCTL	SWU6 Global Control Register	0x00000000
0x20078004	SWU6_GSTAT	SWU6 Global Status Register	0x00000000
0x20078010	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x20078014	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x20078018	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x2007801C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x20078020	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x20078024	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x20078028	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x2007802C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x20078030	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x20078034	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x20078038	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x2007803C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x20078040	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x20078044	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x20078048	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x2007804C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x20078050	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x20078054	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x20078058	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x2007805C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x20078060	SWU6_CNT[n]	SWU6 Count Register n	0x00000000
0x20078064	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x20078068	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x2007806C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000
0x20078070	SWU6_CTL[n]	SWU6 Control Register n	0x00000000
0x20078074	SWU6_LA[n]	SWU6 Lower Address Register n	0x00000000
0x20078078	SWU6_UA[n]	SWU6 Upper Address Register n	0x00000000
0x2007807C	SWU6_ID[n]	SWU6 ID Register n	0x00000000
0x20078080	SWU6_CNT[n]	SWU6 Count Register n	0x00000000

Table A-68: ADSP-BF70x SWU6 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20078084	SWU6_TARG[n]	SWU6 Target Register n	0x00000000
0x20078088	SWU6_HIST[n]	SWU6 Bandwidth History Register n	0x00000000
0x2007808C	SWU6_CUR[n]	SWU6 Current Register n	0x00000000

Table A-69: ADSP-BF70x SWU7 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20079000	SWU7_GCTL	SWU7 Global Control Register	0x00000000
0x20079004	SWU7_GSTAT	SWU7 Global Status Register	0x00000000
0x20079010	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x20079014	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x20079018	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x2007901C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x20079020	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x20079024	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x20079028	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x2007902C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x20079030	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x20079034	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x20079038	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x2007903C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x20079040	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x20079044	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x20079048	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x2007904C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x20079050	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x20079054	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x20079058	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x2007905C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x20079060	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x20079064	SWU7_TARG[n]	SWU7 Target Register n	0x00000000

Table A-69: ADSP-BF70x SWU7 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20079068	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x2007906C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000
0x20079070	SWU7_CTL[n]	SWU7 Control Register n	0x00000000
0x20079074	SWU7_LA[n]	SWU7 Lower Address Register n	0x00000000
0x20079078	SWU7_UA[n]	SWU7 Upper Address Register n	0x00000000
0x2007907C	SWU7_ID[n]	SWU7 ID Register n	0x00000000
0x20079080	SWU7_CNT[n]	SWU7 Count Register n	0x00000000
0x20079084	SWU7_TARG[n]	SWU7 Target Register n	0x00000000
0x20079088	SWU7_HIST[n]	SWU7 Bandwidth History Register n	0x00000000
0x2007908C	SWU7_CUR[n]	SWU7 Current Register n	0x00000000

Table A-70: ADSP-BF70x TAPC0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20048000	TAPC0_IDCODE	TAPC0 IDCODE Register	0x0280A0CB
0x20048004	TAPC0_USERCODE	TAPC0 USERCODE Register	0x00000000
0x20048008	TAPC0_SDBGKEY_CTL	TAPC0 Secure Debug Key Control Register	0x00000000
0x2004800C	TAPC0_SDBGKEY_STAT	TAPC0 Secure Debug Key Status Register	0x00000000
0x20048010	TAPC0_SDBGKEY0	TAPC0 Secure Debug Key 0 Register	0x00000000
0x20048014	TAPC0_SDBGKEY1	TAPC0 Secure Debug Key 1 Register	0x00000000
0x20048018	TAPC0_SDBGKEY2	TAPC0 Secure Debug Key 2 Register	0x00000000
0x2004801C	TAPC0_SDBGKEY3	TAPC0 Secure Debug Key 3 Register	0x00000000
0x20048030	TAPC0_DBGCTL	TAPC0 Debug Control	0x00000000

Table A-71: ADSP-BF70x TIMER0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20043004	TIMER0_RUN	TIMER0 Run Register	0x00000000
0x20043008	TIMER0_RUN_SET	TIMER0 Run Set Register	0x00000000
0x2004300C	TIMER0_RUN_CLR	TIMER0 Run Clear Register	0x00000000
0x20043010	TIMER0_STOP_CFG	TIMER0 Stop Configuration Register	0x00000000
0x20043014	TIMER0_STOP_CFG_SET	TIMER0 Stop Configuration Set Register	0x00000000

Table A-71: ADSP-BF70x TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20043018	TIMER0_STOP_CFG_CLR	TIMER0 Stop Configuration Clear Register	0x00000000
0x2004301C	TIMER0_DATA_IMSK	TIMER0 Data Interrupt Mask Register	0x000000FF
0x20043020	TIMER0_STAT_IMSK	TIMER0 Status Interrupt Mask Register	0x000000FF
0x20043024	TIMER0_TRG_MSK	TIMER0 Trigger Master Mask Register	0x000000FF
0x20043028	TIMER0_TRG_IE	TIMER0 Trigger Slave Enable Register	0x00000000
0x2004302C	TIMER0_DATA_ILAT	TIMER0 Data Interrupt Latch Register	0x00000000
0x20043030	TIMER0_STAT_ILAT	TIMER0 Status Interrupt Latch Register	0x00000000
0x20043034	TIMER0_ERR_TYPE	TIMER0 Error Type Status Register	0x00000000
0x20043038	TIMER0_BCAST_PER	TIMER0 Broadcast Period Register	0x00000000
0x2004303C	TIMER0_BCAST_WID	TIMER0 Broadcast Width Register	0x00000000
0x20043040	TIMER0_BCAST_DLY	TIMER0 Broadcast Delay Register	0x00000000
0x20043060	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x20043064	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x20043068	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x2004306C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x20043070	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x20043080	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x20043084	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x20043088	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x2004308C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x20043090	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x200430A0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x200430A4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x200430A8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x200430AC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x200430B0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x200430C0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x200430C4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x200430C8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x200430CC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x200430D0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table A-71: ADSP-BF70x TIMER0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200430E0	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x200430E4	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x200430E8	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x200430EC	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x200430F0	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x20043100	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x20043104	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x20043108	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x2004310C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x20043110	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x20043120	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x20043124	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x20043128	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x2004312C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x20043130	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000
0x20043140	TIMER0_TMR[n]_CFG	TIMER0 Timer n Configuration Register	0x00000000
0x20043144	TIMER0_TMR[n]_CNT	TIMER0 Timer n Counter Register	0x00000001
0x20043148	TIMER0_TMR[n]_PER	TIMER0 Timer n Period Register	0x00000000
0x2004314C	TIMER0_TMR[n]_WID	TIMER0 Timer n Width Register	0x00000000
0x20043150	TIMER0_TMR[n]_DLY	TIMER0 Timer n Delay Register	0x00000000

Table A-72: ADSP-BF70x TRNG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200F0000	TRNG0_INPUT[n]	TRNG0 TRNG Input Registers	0x00000000
0x200F0000	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x200F0004	TRNG0_INPUT[n]	TRNG0 TRNG Input Registers	0x00000000
0x200F0004	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x200F0008	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x200F000C	TRNG0_OUTPUT[n]	TRNG0 TRNG Output Registers	0x00000000
0x200F0010	TRNG0_STAT	TRNG0 TRNG Status Register	0x00000000

Table A-72: ADSP-BF70x TRNG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200F0010	TRNG0_INTACK	TRNG0 TRNG Interrupt Acknowledge Register	0x00000000
0x200F0014	TRNG0_CTL	TRNG0 TRNG Control Register	0x00000000
0x200F0018	TRNG0_CFG	TRNG0 TRNG Configuration Register	0x00000000
0x200F001C	TRNG0_ALMCNT	TRNG0 TRNG Alarm Counter Register	0x000000FF
0x200F0020	TRNG0_FROEN	TRNG0 TRNG FRO Enable Register	0x000000FF
0x200F0024	TRNG0_FRODETUNE	TRNG0 TRNG FRO De-tune Register	0x00000000
0x200F0028	TRNG0_ALMMSK	TRNG0 TRNG Alarm Mask Register	0x00000000
0x200F002C	TRNG0_ALMSTP	TRNG0 TRNG Alarm Stop Register	0x00000000
0x200F0030	TRNG0_LFSR_L	TRNG0 TRNG LFSR Access Register	0x00000000
0x200F0034	TRNG0_LFSR_M	TRNG0 TRNG LFSR Access Register	0x00000000
0x200F0038	TRNG0_LFSR_H	TRNG0 TRNG LFSR Access Register	0x00000000
0x200F003C	TRNG0_CNT	TRNG0 Counter Register	0x00000000
0x200F0040	TRNG0_RUNCNT	TRNG0 TRNG Run Count Registers	0x00000000
0x200F0040	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x200F0044	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x200F0044	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x200F0048	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x200F0048	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x200F004C	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x200F004C	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x200F0050	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x200F0050	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x200F0054	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x200F0054	TRNG0_KEY[n]	TRNG0 Post-Process Key Registers	0x00000000
0x200F0058	TRNG0_RUN[n]	TRNG0 TRNG Run Test State and Result Registers	0x00000000
0x200F005C	TRNG0_MONOBITCNT	TRNG0 TRNG Monobit Test Result Register	0x00002710
0x200F0060	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x200F0060	TRNG0_V[n]	TRNG0 TRNG Post-Process "V" Value Registers	0x00000000
0x200F0064	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x200F0064	TRNG0_V[n]	TRNG0 TRNG Post-Process "V" Value Registers	0x00000000
0x200F0068	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000

Table A-72: ADSP-BF70x TRNG0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200F006C	TRNG0_POKER[n]	TRNG0 TRNG Poker Test Result Registers	0x00000000
0x200F0070	TRNG0_TEST	TRNG0 TRNG Test Register	0x00000000
0x200F0074	TRNG0_BLKCNT	TRNG0 TRNG Block Count Register	0x00000000

Table A-73: ADSP-BF70x TRU0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20001000	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001004	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001008	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000100C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001010	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001014	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001018	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000101C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001020	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001024	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001028	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000102C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001030	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001034	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001038	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000103C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001040	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001044	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001048	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000104C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001050	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001054	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001058	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000105C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-73: ADSP-BF70x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20001060	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001064	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001068	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000106C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001070	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001074	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001078	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000107C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001080	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001084	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001088	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000108C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001090	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001094	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001098	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000109C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010A0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010A4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010A8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010AC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010B0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010B4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010B8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010BC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010C0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010C4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010C8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010CC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010D0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010D4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010D8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-73: ADSP-BF70x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200010DC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010E0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010E4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010E8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010EC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010F0	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010F4	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010F8	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200010FC	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001100	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001104	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001108	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000110C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001110	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001114	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001118	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000111C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001120	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001124	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001128	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000112C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001130	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001134	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001138	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000113C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001140	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001144	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001148	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000114C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001150	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001154	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000

Table A-73: ADSP-BF70x TRU0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x20001158	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000115C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001160	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001164	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001168	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000116C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001170	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001174	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001178	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000117C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001180	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001184	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001188	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x2000118C	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001190	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x20001194	TRU0_SSR[n]	TRU0 Slave Select Register	0x00000000
0x200017E0	TRU0_MTR	TRU0 Master Trigger Register	0x00000000
0x200017E8	TRU0_ERRADDR	TRU0 Error Address Register	0x00000000
0x200017EC	TRU0_STAT	TRU0 Status Information Register	0x00000000
0x200017F4	TRU0_GCTL	TRU0 Global Control Register	0x00000000

Table A-74: ADSP-BF70x TWI0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2004A000	TWI0_CLKDIV	TWI0 SCL Clock Divider Register	0x00000000
0x2004A004	TWI0_CTL	TWI0 Control Register	0x00000000
0x2004A008	TWI0_SLVCTL	TWI0 Slave Mode Control Register	0x00000000
0x2004A00C	TWI0_SLVSTAT	TWI0 Slave Mode Status Register	0x00000000
0x2004A010	TWI0_SLVADDR	TWI0 Slave Mode Address Register	0x00000000
0x2004A014	TWI0_MSTRCTL	TWI0 Master Mode Control Registers	0x00000000
0x2004A018	TWI0_MSTRSTAT	TWI0 Master Mode Status Register	0x00000000

Table A-74: ADSP-BF70x TWI0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2004A01C	TWI0_MSTRADDR	TWI0 Master Mode Address Register	0x00000000
0x2004A020	TWI0_ISTAT	TWI0 Interrupt Status Register	0x00000000
0x2004A024	TWI0_IMSK	TWI0 Interrupt Mask Register	0x00000000
0x2004A028	TWI0_FIFOCTL	TWI0 FIFO Control Register	0x00000000
0x2004A02C	TWI0_FIFOSTAT	TWI0 FIFO Status Register	0x00000000
0x2004A080	TWI0_TXDATA8	TWI0 Tx Data Single-Byte Register	0x00000000
0x2004A084	TWI0_TXDATA16	TWI0 Tx Data Double-Byte Register	0x00000000
0x2004A088	TWI0_RXDATA8	TWI0 Rx Data Single-Byte Register	0x00000000
0x2004A08C	TWI0_RXDATA16	TWI0 Rx Data Double-Byte Register	0x00000000

Table A-75: ADSP-BF70x UART0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2005F004	UART0_CTL	UART0 Control Register	0x00000000
0x2005F008	UART0_STAT	UART0 Status Register	0x000000A0
0x2005F00C	UART0_SCR	UART0 Scratch Register	0x00000000
0x2005F010	UART0_CLK	UART0 Clock Rate Register	0x0000FFFF
0x2005F014	UART0_IMSK	UART0 Interrupt Mask Register	0x00000000
0x2005F018	UART0_IMSK_SET	UART0 Interrupt Mask Set Register	0x00000000
0x2005F01C	UART0_IMSK_CLR	UART0 Interrupt Mask Clear Register	0x00000000
0x2005F020	UART0_RBR	UART0 Receive Buffer Register	0x00000000
0x2005F024	UART0_THR	UART0 Transmit Hold Register	0x00000000
0x2005F028	UART0_TAIP	UART0 Transmit Address/Insert Pulse Register	0x00000000
0x2005F02C	UART0_TSR	UART0 Transmit Shift Register	0x000007FF
0x2005F030	UART0_RSR	UART0 Receive Shift Register	0x00000000
0x2005F034	UART0_TXCNT	UART0 Transmit Counter Register	0x00000000
0x2005F038	UART0_RXCNT	UART0 Receive Counter Register	0x00000000

Table A-76: ADSP-BF70x UART1 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x2005F404	UART1_CTL	UART1 Control Register	0x00000000

Table A-76: ADSP-BF70x UART1 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x2005F408	UART1_STAT	UART1 Status Register	0x000000A0
0x2005F40C	UART1_SCR	UART1 Scratch Register	0x00000000
0x2005F410	UART1_CLK	UART1 Clock Rate Register	0x0000FFFF
0x2005F414	UART1_IMSK	UART1 Interrupt Mask Register	0x00000000
0x2005F418	UART1_IMSK_SET	UART1 Interrupt Mask Set Register	0x00000000
0x2005F41C	UART1_IMSK_CLR	UART1 Interrupt Mask Clear Register	0x00000000
0x2005F420	UART1_RBR	UART1 Receive Buffer Register	0x00000000
0x2005F424	UART1_THR	UART1 Transmit Hold Register	0x00000000
0x2005F428	UART1_TAIP	UART1 Transmit Address/Insert Pulse Register	0x00000000
0x2005F42C	UART1_TSR	UART1 Transmit Shift Register	0x000007FF
0x2005F430	UART1_RSR	UART1 Receive Shift Register	0x00000000
0x2005F434	UART1_TXCNT	UART1 Transmit Counter Register	0x00000000
0x2005F438	UART1_RXCNT	UART1 Receive Counter Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0000	USB0_FADDR	USB0 Function Address Register	0x00000000
0x200D0001	USB0_POWER	USB0 Power and Device Control Register	0x00000020
0x200D0002	USB0_INTRTX	USB0 Transmit Interrupt Register	0x00000000
0x200D0004	USB0_INTRRX	USB0 Receive Interrupt Register	0x00000000
0x200D0006	USB0_INTRTXE	USB0 Transmit Interrupt Enable Register	0x00000FFF
0x200D0008	USB0_INTRRXE	USB0 Receive Interrupt Enable Register	0x00000FFE
0x200D000A	USB0_IRQ	USB0 Common Interrupts Register	0x00000000
0x200D000B	USB0_IEN	USB0 Common Interrupts Enable Register	0x00000000
0x200D000C	USB0_FRAME	USB0 Frame Number Register	0x00000000
0x200D000E	USB0_INDEX	USB0 Index Register	0x00000000
0x200D000F	USB0_TESTMODE	USB0 Testmode Register	0x00000000
0x200D0010	USB0_EPI[N]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0012	USB0_EP0I_CSR[N]_P	USB0 EP0 Configuration and Status (Peripheral) Register	0x00000000
0x200D0012	USB0_EP0I_CSR[N]_H	USB0 EP0 Configuration and Status (Host) Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0012	USB0_EPI[N]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0012	USB0_EPI[N]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0014	USB0_EPI[N]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0016	USB0_EPI[N]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0016	USB0_EPI[N]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0018	USB0_EP0I_CNT[N]	USB0 EP0 Number of Received Bytes Register	0x00000000
0x200D0018	USB0_EPI[N]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D001A	USB0_EP0I_TYPE[N]	USB0 EP0 Connection Type Register	0x00000000
0x200D001A	USB0_EPI[N]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D001B	USB0_EP0I_NAKLIMIT[N]	USB0 EP0 NAK Limit Register	0x00000000
0x200D001B	USB0_EPI[N]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D001C	USB0_EPI[N]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D001D	USB0_EPI[N]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D001F	USB0_EP0I_CFGDATA[N]	USB0 EP0 Configuration Information Register	0x0000001E
0x200D0020	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D0020	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0020	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0024	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D0024	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0024	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0028	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D0028	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0028	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D002C	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D002C	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D002C	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0030	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0030	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0030	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0034	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D0034	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0034	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0038	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D0038	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0038	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D003C	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D003C	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D003C	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0040	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D0040	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0040	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0044	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D0044	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0044	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0048	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D0048	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D0048	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D004C	USB0_FIFOB[n]	USB0 FIFO Byte (8-Bit) Register	0x00000000
0x200D004C	USB0_FIFO[n]	USB0 FIFO Word (32-Bit) Register	0x00000000
0x200D004C	USB0_FIFOH[n]	USB0 FIFO Half-Word (16-Bit) Register	0x00000000
0x200D0060	USB0_DEV_CTL	USB0 Device Control Register	0x00000000
0x200D0062	USB0_TXFIFOSZ	USB0 Transmit FIFO Size Register	0x00000000
0x200D0063	USB0_RXFIFOSZ	USB0 Receive FIFO Size Register	0x00000000
0x200D0064	USB0_TXFIFOADDR	USB0 Transmit FIFO Address Register	0x00000000
0x200D0066	USB0_RXFIFOADDR	USB0 Receive FIFO Address Register	0x00000000
0x200D0078	USB0_EPINFO	USB0 Endpoint Information Register	0x000000CC
0x200D0079	USB0_RAMINFO	USB0 RAM Information Register	0x0000008C
0x200D007A	USB0_LINKINFO	USB0 Link Information Register	0x0000005C

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D007B	USB0_VPLEN	USB0 VBUS Pulse Length Register	0x0000003C
0x200D007C	USB0_HS_EOF1	USB0 High-Speed EOF 1 Register	0x00000080
0x200D007D	USB0_FS_EOF1	USB0 Full-Speed EOF 1 Register	0x00000077
0x200D007E	USB0_LS_EOF1	USB0 Low-Speed EOF 1 Register	0x00000072
0x200D007F	USB0_SOFT_RST	USB0 Software Reset Register	0x00000000
0x200D0080	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D0082	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D0083	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x200D0084	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D0086	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D0087	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D0088	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D008A	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D008B	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x200D008C	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D008E	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D008F	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D0090	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D0092	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D0093	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0094	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D0096	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D0097	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D0098	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D009A	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D009B	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x200D009C	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D009E	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D009F	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D00A0	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D00A2	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D00A3	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x200D00A4	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D00A6	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D00A7	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D00A8	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D00AA	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D00AB	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D00AC	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D00AE	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D00AF	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D00B0	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D00B2	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D00B3	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x200D00B4	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D00B6	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D00B7	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D00B8	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D00BA	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D00BB	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x200D00BC	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D00BE	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D00BF	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D00C0	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D00C2	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D00C3	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D00C4	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D00C6	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D00C7	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D00C8	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D00CA	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D00CB	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x200D00CC	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D00CE	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D00CF	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D00D0	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D00D2	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D00D3	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000
0x200D00D4	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D00D6	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D00D7	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D00D8	USB0_MP[n]_TXFUNCADDR	USB0 MPn Transmit Function Address Register	0x00000000
0x200D00DA	USB0_MP[n]_TXHUBADDR	USB0 MPn Transmit Hub Address Register	0x00000000
0x200D00DB	USB0_MP[n]_TXHUBPORT	USB0 MPn Transmit Hub Port Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D00DC	USB0_MP[n]_RXFUNCADDR	USB0 MPn Receive Function Address Register	0x00000000
0x200D00DE	USB0_MP[n]_RXHUBADDR	USB0 MPn Receive Hub Address Register	0x00000000
0x200D00DF	USB0_MP[n]_RXHUBPORT	USB0 MPn Receive Hub Port Register	0x00000000
0x200D0100	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0102	USB0_EP0_CSR[n]_P	USB0 EP0 Configuration and Status (Peripheral) Register	0x00000000
0x200D0102	USB0_EP0_CSR[n]_H	USB0 EP0 Configuration and Status (Host) Register	0x00000000
0x200D0102	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0102	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0104	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0106	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0106	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0108	USB0_EP0_CNT[n]	USB0 EP0 Number of Received Bytes Register	0x00000000
0x200D0108	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D010A	USB0_EP0_TYPE[n]	USB0 EP0 Connection Type Register	0x00000000
0x200D010A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D010B	USB0_EP0_NAKLIMIT[n]	USB0 EP0 NAK Limit Register	0x00000000
0x200D010B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D010C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D010D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D010F	USB0_EP0_CFGDATA[n]	USB0 EP0 Configuration Information Register	0x0000001E
0x200D0110	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0112	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0112	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0114	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0116	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0116	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0118	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D011A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D011B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D011C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D011D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0120	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0122	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0122	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0124	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0126	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0126	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0128	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D012A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D012B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D012C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D012D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0130	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0132	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0132	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0134	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0136	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0136	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0138	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D013A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D013B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D013C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D013D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0140	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0142	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0142	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0144	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0146	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0146	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0148	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D014A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D014B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D014C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D014D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0150	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0152	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0152	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0154	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0156	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0156	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0158	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D015A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D015B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D015C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D015D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0160	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0162	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0162	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0164	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0166	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0166	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0168	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D016A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D016B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D016C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D016D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0170	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0172	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0172	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0174	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0176	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0176	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0178	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D017A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D017B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D017C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D017D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0180	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0182	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0182	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0184	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0186	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0186	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0188	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D018A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D018B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D018C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D018D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0190	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D0192	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D0192	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D0194	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D0196	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D0196	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D0198	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D019A	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D019B	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D019C	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D019D	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D01A0	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D01A2	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D01A2	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D01A4	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D01A6	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D01A6	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D01A8	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D01AA	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D01AB	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D01AC	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D01AD	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D01B0	USB0_EP[n]_TXMAXP	USB0 EPn Transmit Maximum Packet Length Register	0x00000000
0x200D01B2	USB0_EP[n]_TXCSR_H	USB0 EPn Transmit Configuration and Status (Host) Register	0x00000000
0x200D01B2	USB0_EP[n]_TXCSR_P	USB0 EPn Transmit Configuration and Status (Peripheral) Register	0x00000000
0x200D01B4	USB0_EP[n]_RXMAXP	USB0 EPn Receive Maximum Packet Length Register	0x00000000
0x200D01B6	USB0_EP[n]_RXCSR_P	USB0 EPn Receive Configuration and Status (Peripheral) Register	0x00000000
0x200D01B6	USB0_EP[n]_RXCSR_H	USB0 EPn Receive Configuration and Status (Host) Register	0x00000000
0x200D01B8	USB0_EP[n]_RXCNT	USB0 EPn Number of Bytes Received Register	0x00000000
0x200D01BA	USB0_EP[n]_TXTYPE	USB0 EPn Transmit Type Register	0x00000000
0x200D01BB	USB0_EP[n]_TXINTERVAL	USB0 EPn Transmit Polling Interval Register	0x00000000
0x200D01BC	USB0_EP[n]_RXTYPE	USB0 EPn Receive Type Register	0x00000000
0x200D01BD	USB0_EP[n]_RXINTERVAL	USB0 EPn Receive Polling Interval Register	0x00000000
0x200D0200	USB0_DMA_IRQ	USB0 DMA Interrupt Register	0x00000000
0x200D0204	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x200D0208	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x200D020C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x200D0214	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x200D0218	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x200D021C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x200D0224	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0228	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x200D022C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x200D0234	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x200D0238	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x200D023C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x200D0244	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x200D0248	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x200D024C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x200D0254	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x200D0258	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x200D025C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x200D0264	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x200D0268	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x200D026C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x200D0274	USB0_DMA[n]_CTL	USB0 DMA Channel n Control Register	0x00000000
0x200D0278	USB0_DMA[n]_ADDR	USB0 DMA Channel n Address Register	0x00000000
0x200D027C	USB0_DMA[n]_CNT	USB0 DMA Channel n Count Register	0x00000000
0x200D0300	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0304	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0308	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D030C	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0310	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0314	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0318	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D031C	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0320	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0324	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0328	USB0_RQPKTCNT[n]	USB0 EPn Request Packet Count Register	0x00000000
0x200D0344	USB0_CT_UCH	USB0 Chirp Timeout Register	0x00004074
0x200D0346	USB0_CT_HHSRTN	USB0 Host High-Speed Return to Normal Register	0x000005E6
0x200D0348	USB0_CT_HSBT	USB0 High-Speed Timeout Register	0x00000000

Table A-77: ADSP-BF70x USB0 MMR Register Addresses (Continued)

Memory Mapped Address	Register Name	Description	Reset Value
0x200D0360	USB0_LPM_ATTR	USB0 LPM Attribute Register	0x00000000
0x200D0362	USB0_LPM_CTL	USB0 LPM Control Register	0x00000000
0x200D0363	USB0_LPM_IEN	USB0 LPM Interrupt Enable Register	0x00000000
0x200D0364	USB0_LPM_IRQ	USB0 LPM Interrupt Status Register	0x00000000
0x200D0365	USB0_LPM_FADDR	USB0 LPM Function Address Register	0x00000000
0x200D0380	USB0_VBUS_CTL	USB0 VBUS Control Register	0x00000000
0x200D0381	USB0_BAT_CHG	USB0 Battery Charging Control Register	0x00000000
0x200D0394	USB0_PHY_CTL	USB0 PHY Control Register	0x00000000
0x200D0398	USB0_PLL_OSC	USB0 PLL and Oscillator Control Register	0x00000014

Table A-78: ADSP-BF70x WDOG0 MMR Register Addresses

Memory Mapped Address	Register Name	Description	Reset Value
0x20046000	WDOG0_CTL	WDOG0 Control Register	0x00000AD0
0x20046004	WDOG0_CNT	WDOG0 Count Register	0x00000000
0x20046008	WDOG0_STAT	WDOG0 Watchdog Timer Status Register	0x00000000