MPC8379E PowerQUICC™ II Pro Integrated Host Processor Family Reference Manual

Supports
MPC8379E
MPC8379
MPC8378E
MPC8378
MPC8377E
MPC8377

MPC8379ERM
Rev. 1
2/2009

freescale™ semiconductor

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: MPC8379ERM
Rev. 1, 2/2009

Contents

Paragraph Number	Title	Page Number

About This Book

Chapter 1 Overview

Chapter 2 Memory Map

Contents

Paragraph Number	Title	Page Number

Contents

Chapter 5
System Configuration

Contents

Contents

Contents

Contents

Paragraph Number	Title	Page Number

Chapter 6
Arbiter and Bus Monitor

Contents

Paragraph Number	Title	Page Number

Contents

Chapter 8
Integrated Programmable Interrupt Controller (IPIC)

Contents

Contents

Paragraph Number	Title	Page Number

Chapter 9
DDR Memory Controller

Contents

Chapter 10
Enhanced Local Bus Controller

Contents

Contents

Paragraph Number	Title	Page Number

Contents

Chapter 11
Enhanced Secure Digital Host Controller

Contents

Paragraph Number	Title	Page Number

Contents

Paragraph Number	Title	Page Number

Chapter 12
Sequencer

Contents

Paragraph Number	Title	Page Number

Chapter 13
DMA/Messaging Unit

Contents

Contents

Paragraph Number	Title	Page Number

Contents

Chapter 15
PCI Express Interface Controller

Contents

Contents

Contents

Contents

Contents

Paragraph Number	Title	Page Number

Contents

Contents

Paragraph Number	Title	Page Number

Chapter 17
Security Engine (SEC) 3.0

Contents

Contents

Contents

Contents

Contents

Paragraph Number	Title	Page Number

Contents

Contents

Paragraph Number	Title	Page Number

Contents

Contents

Paragraph Number	Title	Page Number

Contents

Contents

Paragraph Number	Title	Page Number

Chapter 20
Universal Serial Bus Interface

Contents

Paragraph Number	Title	Page Number

Contents

Contents

Contents

Contents

Paragraph Number	Title	Page Number

Contents

Chapter 22
DUART

Contents

Paragraph Number	Title	Page Number

Contents

Paragraph Number	Title	Page Number

Contents

Paragraph Number	Title	Page Number

Contents

Paragraph Number	Title	Page Number

Contents

Paragraph Number	Title	Page Number

Figures

Figures

Figures

Figures

Figures

Figures

Figure Number	Title	Page Number

Figures

Figures

Figures

Figures

Figures

Figures

Figure Number	Title	Page Number

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figure Number	Title	Page Number

Figures

Figure Number	Title	Page Number

Figures

Figures

Figures

Figures

Figure Number	Title	Page Number

Tables

Table Number	Title	Page Number

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Table Number	Title	Page Number

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Table Number	Title	Page Number

About This Book

This reference manual defines the functionality of the MPC8379E. It is written from the perspective of the MPC8379E, and unless otherwise noted, the information applies also to the MPC8379, MPC8378E, MPC8378, MPC8377E, and MPC8377. Note that the MPC8379, MPC8378, and MPC8377 do not support a security engine. Note also that only the MPC8378/E support SGMII, MPC8378/E and MPC8377/E support PCI Express, and MPC8379/E and MPC8377/E support SATA.

The MPC8379E is a cost-effective, highly integrated host processor that addresses the requirements of several printing and imaging, consumer, and industrial applications, including main CPUs and I/O processors in printing systems, networking switches and line cards, wireless LANs (WLANs), network access servers (NAS), VPN routers, intelligent NIC, and industrial controllers. The MPC8379E extends the PowerQUICC family, adding higher CPU performance, additional functionality, and faster interfaces while addressing the requirements related to time-to-market, price, power consumption, and package size.

Audience

It is assumed that the reader understands operating systems, microprocessor system design, and the basic principles of RISC processing.

Organization

Following is a summary and a brief description of the major parts of this reference manual:

- Chapter 1, "Overview," provides a high-level description of features and functionality of the MPC8379E integrated host processor. It describes the device, its interfaces, and the programming model. The functional operation of the device, with emphasis on peripheral functions, is also described.
- Chapter 2, "Memory Map," describes the memory map of the device. An overview of the local address map is provided. Next, a complete listing of all memory-mapped registers is provided, with cross references to the sections detailing descriptions of each.
- Chapter 3, "Signal Descriptions," provides a listing of all the external signals, cross-references for signals that serve multiple functions, their functional blocks, and I/O states. Also, these signals are listed by alphabetical order.
- Chapter 4, "Reset, Clocking, and Initialization," describes the hard and soft resets, the power-on reset (POR) sequence, power-on reset configuration, clocking, and initialization of the device.
- Chapter 5, "System Configuration," provides an overview of several functions that control the local access windows, system configuration, software watchdog, real time clock, periodic and general purpose timers, power management, protection, and general utilities.
- Chapter 6, "Arbiter and Bus Monitor," provides an overview of the arbiter in the device. Also, it describes the configuration, control, and status registers of the arbiter.

- Chapter 7, "e300 Processor Core Overview," provides an overview of the basic functionality of the processor core and briefly describes how the functional units interact.
- Chapter 8, "Integrated Programmable Interrupt Controller (IPIC)," describes the IPIC interrupt protocol, various types of interrupt sources controlled by the IPIC unit, and the IPIC registers with some programming guidelines. It also provides a definition of the external interrupt signals and their functions. In addition, the interrupt configuration, control, and status registers are described in this chapter.
- Chapter 9, "DDR Memory Controller," describes the 32-bit DDR SDRAM memory controller of the device. This fully programmable controller supports most DDR memories available today, including both buffered and unbuffered devices. Dynamic power management and auto-precharge modes simplify memory system design. The built-in error checking and correction (ECC) ensures very low bit-error rates for reliable high-frequency operation and ECC error injection support rapid system debug is provided.
- Chapter 10, "Enhanced Local Bus Controller," describes the enhanced local bus controller (eLBC), its external signals and the memory-mapped registers, as well as a functional description of the general-purpose chip-select machine (GPCM), flash control machine (FCM), and user-programmable machines (UPMs) of the eLBC. Also, it includes an initialization and applications information section with many specific examples of its use.
- Chapter 11, "Enhanced Secure Digital Host Controller," describes the enhanced SD Host Controller, which provides an interface between the host system and SD, SDIO, and MMC cards. It provides a functional description of the major system blocks and includes command information for the host.
- Chapter 12, "Sequencer," describes how the I/O sequencer (IOS) switches transactions among its ports, using a buffer pool to minimize blocking. It also provides address translation on outbound PCI transactions.
- Chapter 13, "DMA/Messaging Unit," describes the four-channel high speed general-purpose DMA controller of the device. The channels share buffer space in the IOS to facilitate the gathering and sending of data. The DMA/messaging unit supports communication between two processors on different buses, for example, a local processor and a processor on a PCI bus. This communication unit operates with generic messages and doorbell registers. This block also provides a DMA controller that transfers blocks of data independent of the local processor or PCI hosts.
- Chapter 14, "PCI Bus Interface," describes the PCI interface, which complies with the *PCI Local Bus Specification*, Rev. 2.3. This chapter provides a basic description of PCI bus operations. The specific emphasis is directed at how this device implements the PCI specification.
- Chapter 15, "PCI Express Interface Controller," describes the PCI Express interface controller, which connects the CSB to the PCI Express bus, a 2.5 GHz serial interface that supports up to a x2 lane. As both a master (initiator) and a target device, the PCI Express interface is capable of high bandwidth data transfer and is designed to support the next generation I/O devices.
- Chapter 16, "SATA Controller," describes the Serial ATA controller, a high-performance SATA solution incorporating some of the latest SATA-IO extensions. It is designed to operate in a system that supports command queuing and, in particular, switching scheme, based on a frame information

structure (FIS) using port multipliers. Overviews on the command, transport, link, and PHY control layers are provided.

- Chapter 17, "Security Engine (SEC) 3.0," describes the SEC 3.0, which is designed to offload computationally intensive security functions, such as key generation and exchange, authentication, and bulk encryption from the e300 core of the device. It is optimized to process all the algorithms associated with IPSec. The SEC 3.0 (implemented in the device) is derived from integrated security cores found in other members of the PowerQUICC family, including SEC 1.0, the version implemented in the MPC8272/MPC8248. Note that the MPC8379, MPC8378, and MPC8377 do not support a security engine.
- Chapter 18, "Enhanced Three-Speed Ethernet Controllers," describes the two enhanced three-speed Ethernet controllers on the device. These controllers provide 10/100/1Gb Ethernet support with a set of media-independent interface options including MII, RMII, GMII, and RTBI. The controllers provide two full-duplex FIFO interface modes and quality of service support. They are backward compatible with PowerQUICC III TSEC controllers.
- Chapter 19, "SerDes PHY," describes the block which includes the SerDes PHY, the protocol converter per protocol, the protocol mux, and the control registers and control logic. It supports one x2 PCI Express, two x1 PCI Express, two x1 SGMII, and two x1 SATA.
- Chapter 20, "Universal Serial Bus Interface," describes the universal serial bus (USB) interface. The USB DR module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The USB DR module can act as a host, as a device, or as an on-the-go (OTG) negotiable host/device on the USB bus.
- Chapter 21, "I^2C Interface," describes the inter-IC (IIC or I^2C) bus controllers of the device. These synchronous, serial, bidirectional, multiple-master buses allow two-wire connection of devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The device powers up in boot sequencer mode, which allows the I^2C controllers to initialize configuration registers.
- Chapter 22, "DUART," describes the (dual) universal asynchronous receiver/transmitters (UARTs) which feature a PC16552D-compatible programming model. These independent UARTs are provided specifically to support system debugging.
- Chapter 23, "Serial Peripheral Interface," describes the MPC8379E serial peripheral interface (SPI) that allows the exchange of data between MPC83xx family devices. The SPI can also be used to communicate with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.
- Chapter 24, "JTAG/Testing Support," describes the joint test action group (JTAG) interface of the MPC8349E to facilitate boundary-scan testing. The JTAG interface complies to the IEEE 1149.1 boundary-scan specification.
- Chapter 25, "General Purpose I/O (GPIO)," describes the general purpose I/O (GPIO) module in the MPC8313E device, including a definition of the external signals and functions they serve. Additionally, interrupt capabilities, pin description, and register settings are described.
- This reference manual also includes a glossary and an index.

Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about the architecture.

General Information

The following documentation, published by Morgan-Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Francisco, CA, provides useful information about the PowerPC architecture and computer architecture in general:

- *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Second Edition, by International Business Machines, Inc.
- *Computer Architecture: A Quantitative Approach*, Third Edition, by John L. Hennessy and David A. Patterson.
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, by David A. Patterson and John L. Hennessy.

Related Documentation

Freescale documentation is available from the sources listed on the back cover of this manual; the document order numbers are included in parentheses for ease in ordering:

- *e300 Power Architecture™ Core Family Reference Manual* (e300CORERM)—This book provides a more detailed description of the e300 core.
- Reference manuals (formerly called user's manuals)—These books provide details about individual implementations.
- Addenda/errata to reference or user's manuals—Because some processors have follow-on devices, an addendum may be provided that describes the additional features and functionality changes. These addenda are intended for use with the corresponding reference or user's manuals.
- Hardware specifications—Hardware specifications provide specific data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.
- Product briefs—Each device has a product brief that provides an overview of its features.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Freescale processors.

Additional literature is published as new processors become available. For a current list of documentation, refer to http://www.freescale.com.

Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
mnemonics	Instruction mnemonics are shown in lowercase bold
italics	Italics indicate variable command parameters, for example, **bcctr***x*

	Book titles in text are set in italics
	Internal signals are set in lowercase italics, for example, $\overline{\textit{core\_int}}$
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
rA, **r**B	Instruction syntax used to identify a source GPR
rD	Instruction syntax used to identify a destination GPR
REG[FIELD]	Abbreviations for registers are shown in uppercase text. Specific bits, fields, or ranges appear in brackets. For example, MSR[LE] refers to the little-endian mode enable bit in the machine state register.
x	In some contexts, such as signal encodings, an unitalicized x indicates a don’t care
x	An italicized *x* indicates an alphanumeric variable
n	An italicized *n* indicates a numeric variable
¬	NOT logical operator
&	AND logical operator
|	OR logical operator
||	Concatenation, for example TCR[WP]||TCR[WPEXT]
[—]	Indicates a reserved bit field in a register. Although these bits can be written to as ones or zeros, they are always read as zeros.
R [0] W [(shaded)]	Indicates a reserved bit field in a memory-mapped register. Although these bits can be written to as ones or zeros, they are always read as zeros.
R [FIELDNAME] W [(shaded)]	Indicates a read-only bit field in a memory-mapped register.
R [(shaded)] W [FIELDNAME]	Indicates a write-only bit field in a memory-mapped register. Although these bits can be written to as ones or zeros, they are always read as zeros.

Signal Conventions

$\overline{\text{OVERBAR}}$	An overbar indicates that a signal is active-low.
lowercase_italics	Lowercase italics is used to indicate internal signals.
lowercase_plaintext	Lowercase plain text is used to indicate signals that are used for configuration. For more information, see Section 3.2, “Configuration Signals Sampled at Reset.”

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
AESU	Advanced encryption standard unit
AFEU	ARC four execution unit
BD	Buffer descriptor
BIST	Built-in self test
CD	Collision detect
COL	Collision
CPM	Communication processor module
CRC	Cyclic redundancy check
CRS	Carrier sense
CSB	Coherent system bus
CSMA	Carrier-sense multiple access
DDR	Double data rate
DEU	Data encryption standard execution unit
DMA	Direct memory access
DRAM	Dynamic random access memory
DTLB	Data translation lookaside buffers
DUART	Dual universal asynchronous receiver/transmitter
EA	Effective address
ECC	Error checking and correction
EHCI	Enhanced host controller interface
EHPI	Enhanced host port interface
EPROM	Erasable programmable read-only memory
FS	Full-speed
FCS	Frame-check sequence
GMII	Gigabit media independent interface
GPCM	General-purpose chip-select machine
GPIO	General-purpose I/O
GPR	General-purpose register
GTM	General purpose timers
IAD	Internet access device

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
I^2C	Inter-integrated circuit
IEEE	Institute of Electrical and Electronics Engineers
IOS	I/O sequencer
IPG	Interpacket gap
ISDN	Integrated services digital network
ITLB	Instruction translation lookaside buffer
IU	Integer unit
JTAG	Joint Test Action Group
LALE	LBC external address latch enable
LBC	Local bus controller
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MAC	Multiply accumulate, media access control
MCP	Machine-check interrupt
MDI	Medium-dependent interface
MDEU	Message digest execution unit
MIB	Management information base
MII	Media independent interface
MMU	Memory management unit
MPH	Multi-port host
MSB	Most-significant byte
msb	Most-significant bit
OSI	Open systems interconnection
PCI	Peripheral component interconnect
PCS	Physical coding sublayer
PIC	Programmable interrupt controller
PIT	Periodic interval timer
PKEU	Public key execution unit
PMA	Physical medium attachment
PMD	Physical medium dependent
POR	Power-on reset

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
PRI	Primary rate interface
RGMII	Reduced gigabit media independent interface
RISC	Reduced instruction set computing
RMON	Remote monitoring
RMW	Read-modify-write
RNG	Random number generator
RTBI	Reduced ten-bit interface
RTC	Real time clock module
Rx	Receive
RxBD	Receive buffer descriptor
SCL	Serial clock
SDA	Serial data
SFD	Start frame delimiter
SI	Serial interface
SPI	Serial peripheral interface
SPR	Special-purpose register
SRAM	Static random access memory
TAP	Test access port
TBI	Ten-bit interface
TLB	Translation lookaside buffer
TSEC	Three-speed Ethernet controller
Tx	Transmit
TxBD	Transmit buffer descriptor
UART	Universal asynchronous receiver/transmitter
ULPI	USB low-pin count interface
UPM	User-programmable machine
USB	Universal serial bus
UTMI	USB transceiver macrocell interface
UTP	Unshielded twisted pair
WDT	Watchdog timer
ZBT	Zero bus turnaround

Chapter 1
Overview

This chapter provides an overview of the MPC8379E PowerQUICC II Pro processor features, including a block diagram showing the major functional components. The MPC8379E is a cost-effective, highly integrated host processor that addresses the requirements of several printing and imaging, consumer, and industrial applications, including main CPUs and I/O processors in printing systems, networking switches and line cards, wireless LANs (WLANs), network access servers (NAS), VPN routers, intelligent NIC, and industrial controllers. The MPC8379E extends the PowerQUICC family, adding higher CPU performance, additional functionality, and faster interfaces while addressing the requirements related to time-to-market, price, power consumption, and package size. This manual is written from the perspective of the MPC8379E, and unless otherwise noted, the information applies also to the MPC8379, MPC8378E, MPC8378, MPC8377E, and MPC8377. Note that the MPC8379, MPC8378, and MPC8377 do not support a security engine.

1.1 MPC8379E Family Product Distinctions

Table 1-1 highlights (in bold) the primary functional differences between the MPC8379E, MPC8378E, and MPC8377E.

Table 1-1. Functionality of the MPC8379E, MPC8378E, and MPC8377E

Descriptions	MPC8379E	MPC8378E	MPC8377E
Ethernet/PHY I/F	MII, RGMII, RMII, RTBI	MII, **SGMII**, RGMII, RMII, RTBI	MII, RGMII, RMII, RTBI
Security (DES/3DES, AES, SH1)	MPC8379E: Yes MPC8379: **No**	MPC8378E: Yes MPC8378: **No**	MPC8377E: Yes MPC8377: **No**
SGMII	0	**2**	0
PCI Express	**0**	2	2
SATA	4	**0**	**2**
SVR	MPC8379E: 80C2_0010 MPC8379: 80C3_0010	MPC8378E: 80C4_0010 MPC8378: 80C5_0010	MPC8377E: 80C6_0010 MPC8377: 80C7_0010
Packaging	PBGA	PBGA	PBGA

1.2 MPC8379E PowerQUICC II Pro Processor Overview

Figure 1-1 shows the major functional units within the MPC8379E. The e300 core in the MPC8379E, with its 32 Kbytes of instruction and 32 Kbytes of data cache, implements the user instruction set architecture and provides hardware and software debugging support. In addition, the MPC8379E offers dual three-speed 10, 100, 1000 Mbps Ethernet controllers (eTSECs), a DDR1/DDR2 SDRAM memory controller, an enhanced local bus controller (eLBC), four serial ATA (SATA) controllers, a 32-bit PCI 2.3 controller with

PME, PCI Express controller, an enhanced secure digital host controller (eSDHC), dual SerDes blocks, a dedicated security engine, a programmable interrupt controller, dual I^{2}C controllers, a 4-channel DMA controller, a general-purpose I/O port, and a USB 2.0 host and device controller. The high level of integration in the MPC8379E helps simplify board design and offers significant bandwidth and performance.

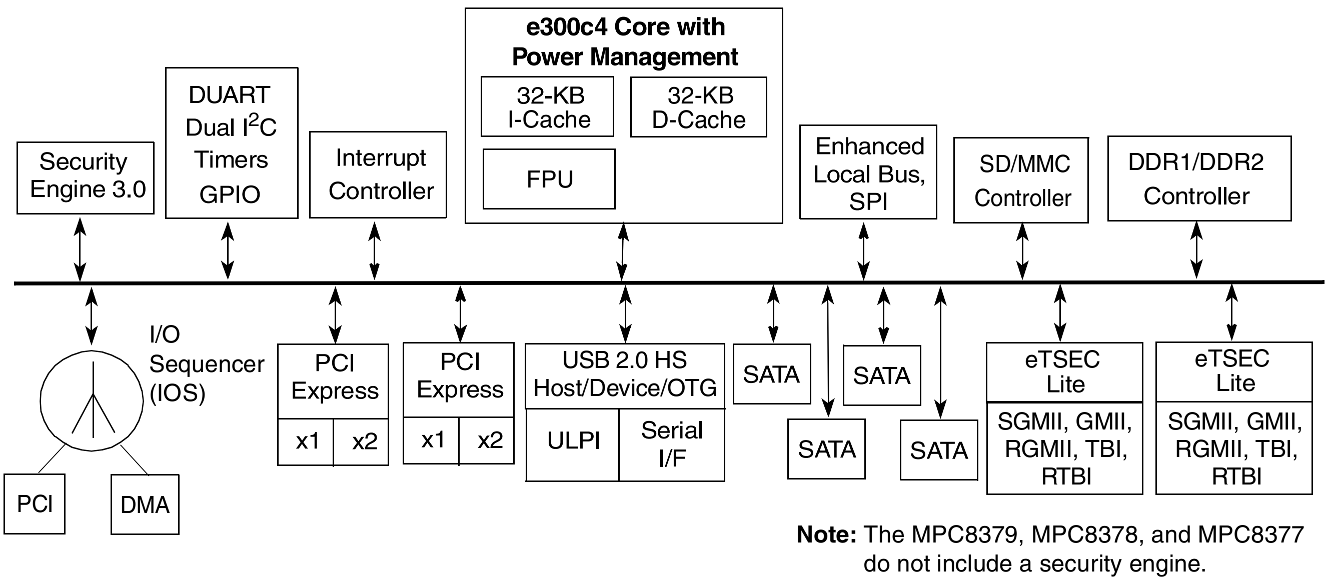


Figure 1-1. MPC8379E Block Diagram

The major features of this device are as follows:

- e300c4 Power Architecture™ processor core
  - High-performance, superscalar processor core with a four-stage pipeline and low interrupt latency times
  - Floating-point, dual integer units, load/store, system register, and branch processing units
  - 32-Kbyte instruction cache and 32-Kbyte data cache with lockable capabilities
  - Capable of completing two MACs every 3 cycles
  - Dynamic power management
  - Enhanced hardware program debug features
  - Software-compatible with Freescale processor families implementing Power Architecture technology
  - Separate PLL that is clocked by the system bus clock
- Security engine optimized to handle all the algorithms associated with IPSec, SSL/TLS, SRTP, IEEE 802.11i® standard, iSCSI, and IKE processing. The security engine contains four crypto-channels, a controller, and a set of crypto execution units (EUs). The execution units are:
  - Public key execution unit (PKEU) supporting the following:
    - RSA and Diffie-Hellman algorithms
    - Programmable field size up to 2048 bits
    - RSA programmable field size is up to 4096 bits; RSA modulus may not exceed $2^{4096}–1$

  - Elliptic curve cryptography
  - F2m and F(p) modes
  - Programmable field size up to 511 bits
- Data encryption standard execution unit (DEU)
  - DES and 3DES algorithms
  - Two key (K1, K2) or three key (K1, K2, K3) for 3DES
  - ECB and CBC modes for both DES and 3DES
- Advanced encryption standard unit (AESU)
  - Implements the Rijndael symmetric-key cipher
  - Key lengths of 128-, 192-, and 256-bits
  - ECB, CBC, CCM, GCM, OFB, CFB, CMAC, and counter (CTR) modes
  - XOR parity generation accelerator for RAID applications
- ARC four execution unit (AFEU)
  - Implements a stream cipher compatible with the RC4 algorithm
  - 40- to 128-bit programmable key
- Message digest execution unit (MDEU)
  - SHA with 160- or 256-bit message digest
  - MD5 with 128-bit message digest
  - HMAC with either algorithm; HMAC built into the MDEU
- Random number generator (RNG)
- Four crypto-channels, each supporting multi-command descriptor chains
  - Dynamic assignment of crypto-execution units through an integrated controller
  - Buffer size of 256 bytes for each execution unit, with flow control for large data sizes
  - Support for multiple outstanding bus transactions
  - Scatter/gather capability
  - Fetch FIFOs in the crypto-channel

- DDR SDRAM memory controller
  - Programmable timing supporting both DDR1 and DDR2 SDRAM
  - 32- or 64-bit data interface, up to 400-MHz data rate
    - 64-Mbit to 2-Gbit devices with x8/x16/x32 data ports (no direct x4 support)
  - Up to four physical banks (chip selects), each bank up to 1 Gbyte independently addressable
  - On-die termination (ODT) support when using DDR2
  - Support for up to 16 simultaneous open pages for DDR1 (up to 32 pages for DDR2)
  - Sleep-mode support for SDRAM self refresh
  - Supports auto refresh
  - On-the-fly power management using CKE
  - Registered DIMM support

- — 2.5-V SSTL2 compatible I/O for DDR1, 1.8-V SSTL_18 compatible I/O for DDR2
- Two enhanced three-speed Ethernet controllers (eTSECs)
  - — Backward compatible with MPC8349E (PowerQUICC II Pro) TSEC
  - — Three-speed support (10/100/1000 Mbps)
  - — Two controllers designed to comply with IEEE Std. 802.3®, 802.3u®, 802.3x®, 802.3z®, 802.3ac®, and 802.3ab® standards
  - — Support for IEEE Std. 1588™ standard
  - — Support for various Ethernet physical interfaces:
    - – 1000 Mbps full-duplex and IEEE Std. 802.3z RTBI/RGMII
    - – 10/100 Mbps full- and half-duplex IEEE Std. 802.3 MII and IEEE Std. 802.3 RGMII/RMII
  - — Inter-packet and intra-packet flow control
  - — Support Wake-on-LAN™, a method for bringing the device from standby to full operating mode
  - — TCP/IP acceleration and QoS features available
    - – IP v4 and IP v6 header recognition on receive
    - – IP v4 header checksum verification and generation
    - – TCP and UDP checksum verification and generation
    - – Per-packet configurable acceleration
    - – Recognition of stacked (queue in queue) VLAN, 802.2, PPPoE session, MPLS stacks, and ESP/AH IP-security headers
  - — Full- and half-duplex Ethernet support (1000 Mbps supports only full-duplex):
    - – IEEE Std. 802.3 full-duplex flow control (automatic PAUSE frame generation or software-programmed PAUSE frame generation and recognition)
  - — Programmable maximum frame length supports jumbo frames (up to 9.6 Kbytes) and IEEE Std 802.1® virtual local area network (VLAN) tags and priority
  - — VLAN insertion and deletion
    - – Per-frame VLAN control word or default VLAN for each eTSEC
    - – Extracted VLAN control word passed to software separately
  - — Retransmission following a collision
  - — CRC generation and verification of inbound/outbound packets
  - — Programmable Ethernet preamble insertion and extraction of up to 7 bytes
  - — MAC address recognition:
    - – Exact match on primary and virtual 48-bit unicast addresses
    - – VRRP and HSRP support for seamless router fail-over
    - – Up to 16 exact-match MAC addresses supported
    - – Broadcast address (accept/reject)
    - – Hash table match on up to 512 multicast addresses
    - – Promiscuous mode

- — Buffer descriptors backward compatible with MPC8260 and MPC860T 10/100 Ethernet programming models
- — RMON statistics support
- — 10-Kbyte internal transmit and 2-Kbyte receive FIFOs
- — MII management interface for control and status
- Two SGMII PHY interfaces
  - — Both interfaces share a single PLL
  - — Each interface supports a 4-wire differential (Tx, Rx) interface
  - — Support for auto-negotiation
- Enhanced secure digital host controller (eSDHC)
  - — Designed to comply with *SD Host Controller Standard Specification Version 2.0* with test event register support
  - — Designed to work with SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, MMC, MMC*plus*, and RS-MMC cards
  - — SD bus clock frequency up to 50 MHz
  - — Supports 1-/4-bit SD and SDIO modes, 1-/4-bit MMC modes, 4-bit devices
    - – Up to 200 Mbps data transfer for SD/SDIO/MMC cards using 4 parallel data lines
  - — Supports single- and multi-block read and write, write protection switch for write operations
  - — Supports synchronous and asynchronous abort
  - — Supports pause during the data transfer at a block gap
  - — Supports Auto CMD12 for multi-block transfer
  - — Host can initiate non-data transfer commands while the data transfer is in progress
  - — Allows cards to interrupt the host in 1- and 4-bit SDIO modes
  - — Embodies a fully configurable 128 $\times$ 32-bit FIFO for read/write data
  - — Supports internal DMA capabilities
- Two SerDes blocks with two lanes each
  - — Support for one x2 PCI Express, two x1 PCI Express, two x1 SGMII, and two x1 SATA
    - – Gen1i, Gen1m, Gen2i, and Gen2m electrical specifications are supported in SATA mode, compliant to *Serial ATA 2.5 Specification*
  - — Link-layer interfaces to IP controller
  - — SerDes power-down/reset state machine for cold (power-on) or warm (software-initiated) reset of SerDes, PHY, and controllers
  - — Provides *reset_done* indication
  - — Interface to clock controls

- PCI interface
  - Designed to comply with *PCI Local Bus Specification, Revision 2.3*
  - 32-bit PCI interface operating at up to 66 MHz
  - PCI 3.3-V compatible
  - Memory prefetching of PCI read accesses and support for delayed read transactions
  - On-chip arbitration, supporting five external PCI masters
  - Selectable hardware-enforced coherency
  - Mapping from an external 32-/64-bit address space to the internal 32-bit local space
  - Support for dual address cycle (DAC) 64-bit addressing mode
- PCI Express
  - Supports two interfaces supporting x1 and x2 widths
  - Compatible with the *PCI Express 1.0a Specification*
  - Selectable operation as root complex or endpoint
  - 32- and 64-bit addressing
  - 128-byte maximum payload size
  - Virtual channel 0 only
  - Full 64-bit decode with 32-bit wide windows
  - Power management
    - Active state power management (ASPM)
    - Supports the PCI Express Power Management L0, L0s, L2/L3 ready and L3 states
- Serial ATA (SATA) controller
  - Two ports
  - Controller is designed to be compliant to the *Serial ATA 2.5 Specification*
  - 1.5 and 3.0 Gbps operation for SATA and eSATA
  - Support for Gen1i, Gen1m, Gen2i, and Gen2m electricals per *Serial ATA 2.5 Specification*
  - Native command queuing
  - Staggered spin-up
  - Port multiplier support
  - Hot plug including asynchronous signal recovery
  - Support in each controller for 16-entry command queue
- Universal serial bus (USB) dual-role controller
  - Designed to comply with *Universal Serial Bus Revision 2.0 Specification*
  - Supports operation as a stand-alone USB host controller
    - Supports USB root hub with one downstream-facing port
    - Enhanced host controller interface (EHCI) compatible
  - Supports operation as a stand-alone USB device
    - Supports one upstream-facing port

    - Supports six programmable bidirectional USB endpoints
  - Supports high-speed (480-Mbps), full-speed (12-Mbps), and low-speed (1.5-Mbps) operations. Low speed is only supported in host mode.
  - Supports USB on-the-go mode, which includes both device and host functionality
  - Supports external PHY with serial and ULPI (UTMI+ low-pin interface)
- Enhanced local bus controller (eLBC)
  - Non-multiplexed 25-bit address and 8-/16-bit data bus at up to 133 MHz
  - Multiplexed 32-bit address and data operating at up to 133 MHz
  - Eight chip selects supporting eight external slaves
  - Variable memory block sizes (32 Kbytes to 4 Gbytes)
  - Up to 256-byte bursts, arbitrarily aligned
  - Up to eight-beat burst transfers
  - 32-, 16-, and 8-bit ports are controlled by an on-chip memory controller
  - Three protocol engines available on a per chip select basis:
    - General-purpose chip select machine (GPCM)
    - Three user programmable machines (UPMs)
    - NAND Flash control machine (FCM)
  - Parity support
  - Default boot ROM chip select with configurable bus width (8, 16, or 32 bits). Boot chip-select support for 8- and 16-bit devices on FCM only.
  - Supports segmentation of large transactions, such as accesses initiated by the DMA engine
- Integrated programmable interrupt controller (IPIC)
  - Functional and programming compatibility with the MPC8349 interrupt controller
  - Support for external and internal discrete interrupt sources
  - Support for one external (optional) and seven internal machine check interrupt sources
  - Support for programmable polarity of external interrupt active state
  - Programmable highest priority request
  - Four groups of interrupts with programmable priority
  - External and internal interrupts directed to host processor
  - Redirects interrupts to external $\overline{\text{PCI\_INTA}}$ signal when in core disable mode
  - Unique vector number for each interrupt source
- Dual I^{2}C interfaces
  - Two-wire interface
  - Multiple-master support
  - Master or slave I^{2}C mode support
  - On-chip digital filtering rejects spikes on the bus
  - System initialization data is optionally loaded from I^{2}C-1 EPROM by boot sequencer embedded hardware

- I/O sequencer
- DMA (Direct memory access) controller
  - Four independent fully programmable DMA channels
  - Concurrent execution across multiple channels with programmable bandwidth control
  - Handshaking (external control) signals supported for all channels: $\overline{\text{DMA\_DREQ}}$[0:3], $\overline{\text{DMA\_DACK}}$[0:3], $\overline{\text{DMA\_DDONE}}$[0:3]
  - All channels accessible by local core and remote PCI masters
  - Misaligned transfer capability for source/destination address
  - Data chaining and direct mode
  - Interrupt on completed segment, link, list, and error
  - Supports transfers to or from any local memory or I/O port
  - Supports transfers to or from external devices
  - Selectable hardware-enforced coherency (snoop/no-snoop)
  - Holds address for source/destination
- DUART
  - Two 4-wire interfaces (RxD, TxD, $\overline{\text{RTS}}$, $\overline{\text{CTS}}$)
  - Programming model compatible with the original 16450 UART and the PC16550D
- Serial peripheral interface (SPI)
  - Master or slave support
- Parallel I/O
  - General-purpose I/O (GPIO)
    - 52 parallel I/O pins multiplexed on various chip interfaces
  - Open drain capability
  - Interrupt capability
- System timers
  - Periodic interrupt timer
  - Real-time clock
  - Software watchdog timer
  - Eight general-purpose timers
- IEEE Std. 1149.1™ compliant JTAG boundary scan
- Integrated PCI bus and SDRAM clock generation

1.3 MPC8379E Architecture Overview

The following sections describe the major functional units of this device.

1.3.1 Power Architecture Core

The device contains the e300c4 Power Architecture processor core, which is an enhanced version of the MPC603e core (used in previous generations of PowerQUICC II processors). Enhancements include twice as much L1 cache (32-Kbyte data cache and 32-Kbyte instruction cache) with integrated parity checking, dual integer units, and other performance-enhancing features. The e300 core is upward software-compatible with existing MPC603e core-based products.

For detailed information regarding the processor core refer to the following:

- The *e300 Power Architecture™ Core Family Reference Manual* (chapters describing the programming model, cache model, memory management model, exception model, and instruction timing) (Document No. E300CORERM)
- The *Programming Environments Manual for 32-Bit Implementations of the PowerPC™ Architecture* (Document No. MPCFPE32B)

The e300 core is a low-power implementation of the family of microprocessors that implements Power Architecture technology. The core implements the 32-bit portion of the architecture, which provides 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

The core is a superscalar processor that can issue three instructions (two plus a branch) and completes and retires as many as two instructions per clock cycle. Instructions can execute out of order for increased performance; however, the core makes completion appear sequential.

The e300c4 core integrates six execution units—two integer units (IU1 and IU2) with full multiply and divides, a floating-point unit (FPU), a branch processing unit (BPU) with static branch prediction, a load/store unit (LSU) for data transfers, a performance monitor, and a system register unit (SRU). The ability to execute five instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput. Most integer instructions execute in one clock cycle. The FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle.

The e300c4 core provides independent on-chip, 32-Kbyte, eight-way set-associative, physically-addressed instruction and data caches with parity and integrated way lock capabilities. The processor also features independent on-chip instruction and data memory management units (MMUs). The MMUs contain 64-entry, two-way set-associative, data and instruction translation lookaside buffers (DTLB and ITLB) that provide support for demand-paged virtual memory address translation. The caches use a pseudo least recently used (PLRU) replacement algorithm; the TLBs use a least recently used (LRU) replacement algorithm. The processor also supports block address translation through the use of two independent instruction and data block address translation (IBAT and DBAT) arrays of eight entries each. Effective addresses are compared simultaneously with all eight entries in the BAT array during block translation. In accordance with the architecture, if an effective address hits in both the TLB and BAT array, the BAT translation takes priority.

As an added feature to the e300 core, the device can lock the contents of one to all ways in the instruction and data cache (or an entire cache). For example, this allows embedded applications to lock interrupt routines or other important (time-sensitive) instruction sequences into the instruction cache. It allows data to be locked into the data cache, which may be important to code that must have deterministic execution.

The e300 core has high-performance 64-bit data bus and 32-bit address bus interfaces to the rest of the device. The e300 core supports single-beat and burst data transfers for memory accesses, and memory-mapped I/O operations.

Figure 1-2 provides a block diagram of the e300 core that shows how the execution units (IU, FPU, BPU, LSU, and SRU) operate independently and in parallel. Note that this is a conceptual diagram and does not attempt to show how these features are physically implemented on the chip.

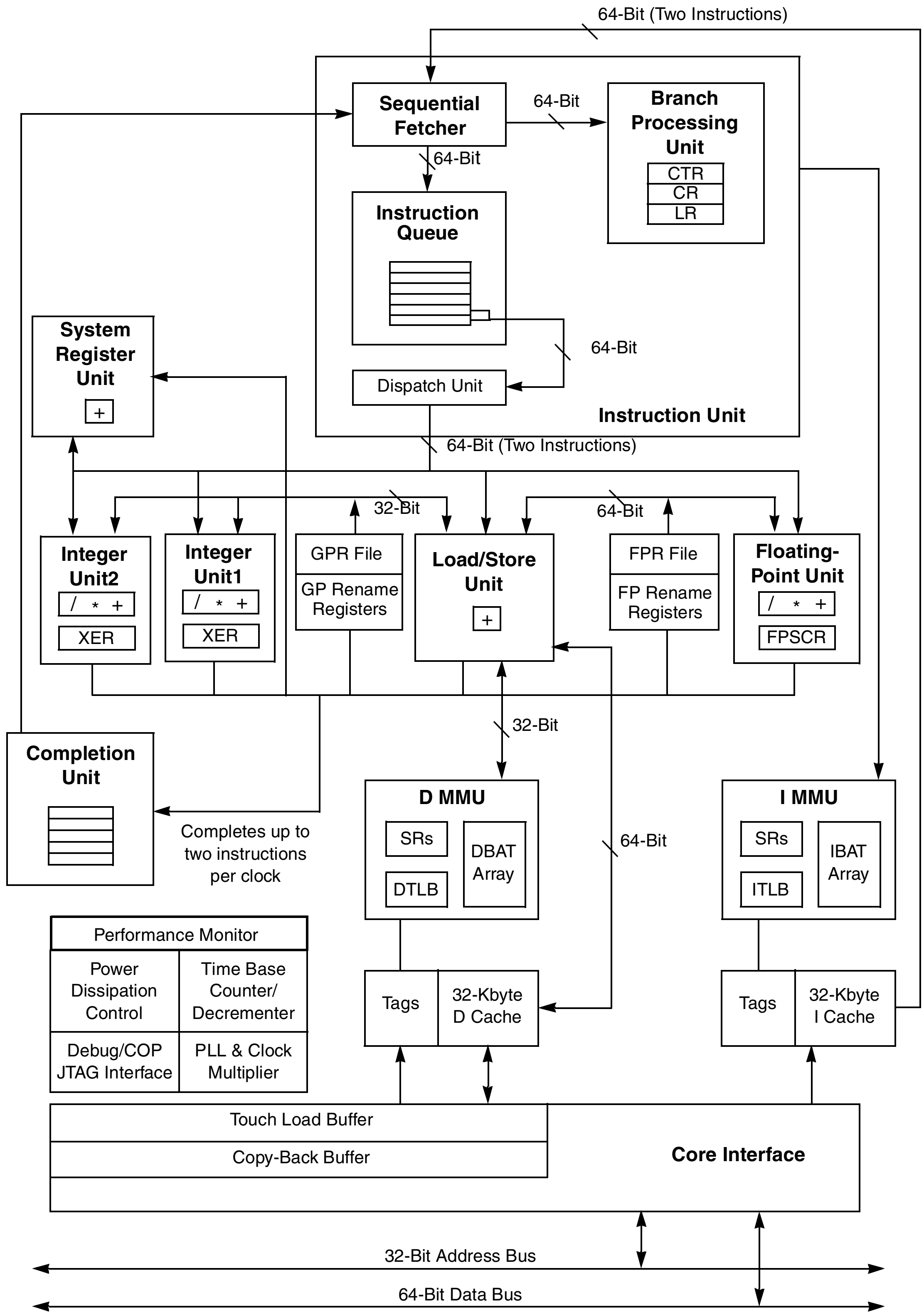


Figure 1-2. MPC8379E Integrated e300c4 Core Block Diagram

1.3.2 Security Engine

A hardware encryption block is also integrated in the device. It supports many encryption algorithms allowing for high performance data encryption and authentication as required in today's SoHo/RoBo routers. The encryption block is compatible with the corresponding block in the MPC8280.

The security engine supports DES, 3DES, MD-5, SHA-1, AES, RNG, and RC-4 encryption algorithms in hardware.

A block diagram of the security engine's internal architecture is shown in Figure 1-3. The bus interface module is designed to transfer 64-bit words between the internal bus and any register inside the security engine.

An operation begins with a write of a pointer to a crypto-channel fetch register that points to a data packet descriptor. The channel requests the descriptor and decodes the operation to be performed. The channel then requests the controller to assign crypto execution units and fetch the keys, IVs, and data needed to perform the given operation. The controller satisfies the requests by assigning execution units to the channel and by making requests to the master interface. As data is processed, it is written to the individual execution unit's output buffer and then back to system memory through the bus interface module.

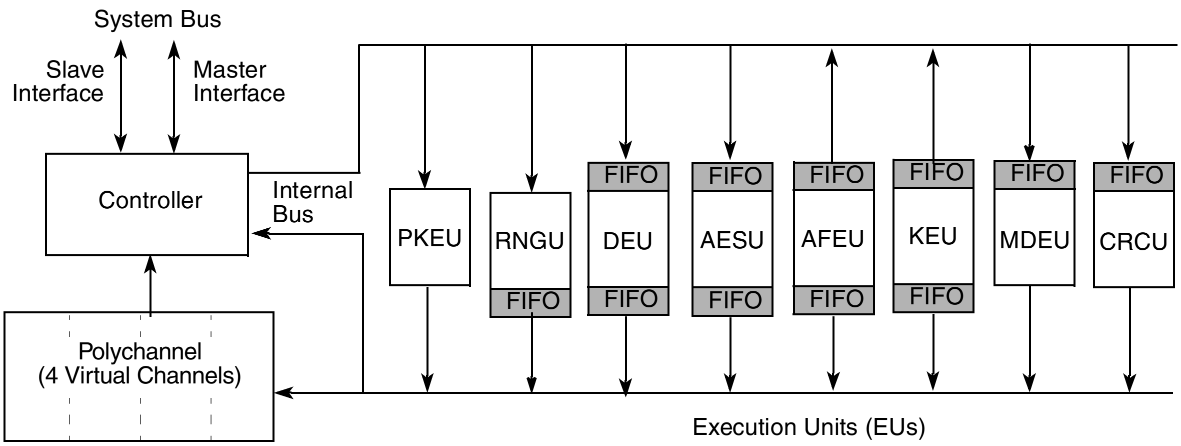


Figure 1-3. Integrated Security Engine Functional Blocks

1.3.3 Dual DDR Memory Controllers

These fully programmable dual DDR SDRAM controllers support most JEDEC standard x8 or x16 DDR1 or DDR2 memories available today, including buffered and unbuffered DIMMs. However, mixing nonregistered and registered DIMMs in the same system is not supported. The built-in error checking and correction (ECC) ensures very low bit-error rates for reliable high-frequency operation. Dynamic power management and auto-precharge modes simplify memory system design.

The DDR memory controllers include the following features:

- Support for DDR1 and DDR2 SDRAM
- 32-/40-bit and 64-/72-bit SDRAM data bus
- Programmable settings for meeting all SDRAM timing parameters
- Many different SDRAM configurations supported

— Support for as many as four physical banks (chip selects), each bank independently addressable
— Support for 64-Mbit to 4-Gbit devices with x8 or x16 data ports.
— Support for unbuffered and registered DIMMs

- Support for data mask signals and read-modify-write operations for sub-double word writes
- Support for double-bit error detection and single-bit error correction ECC (8-bit check word across 64- or 32-bit data when in 32-bit mode)
- Four-entry input request queue
- Open page management (dedicated entry for each sub-bank)
- Memory controller clock frequency of two times the SDRAM clock with support for sleep power management mode
- Support for error injection on ECC

1.3.4 Dual Enhanced Three-Speed Ethernet Controllers

The MPC8379E has two on-chip enhanced three-speed Ethernet controllers. The eTSECs incorporate a media access control (MAC) sublayer that supports 10- and 100-Mbps and 1-Gbps Ethernet/IEEE Std. 802.3 networks with MII, RMII, RGMII, RTBI, and SGMII physical interfaces. The eTSECs include 2-Kbyte receive and 10-Kbyte transmit FIFOs and DMA functions. They also support IEEE Std. 1588 standard.

The buffer descriptors are based on the MPC8260 and MPC860T 10/100 Ethernet programming models. Each eTSEC can emulate a PowerQUICC III TSEC, allowing existing driver software to be re-used with minimal change.

The MPC8379E eTSECs support programmable CRC generation and checking, RMON statistics, and jumbo frames of up to 9.6 Kbytes. Frame headers and buffer descriptors can be forced into the L2 cache to speed classification or other frame processing.

Each eTSEC provides hardware support for accelerating TCP/IP packet transmission and reception. By default, TCP/IP acceleration is not enabled, and the eTSEC processes frames as pure Ethernet frames.

TCP/IP acceleration can be performed at a number of levels. The eTSEC can parse frames at layer 2 of the stack only (Ethernet headers and switching headers), layers 2 to 3 (including IP v4 or IP v6), or layers 2 to 4 (including TCP and UDP).

On receive, the eTSEC provides protocol header recognition, header verification (IP v4 header checksum verification), and TCP/UDP payload checksum verification including verification of associated pseudo-header checksums. On transmit, the eTSEC provides IP v4 and TCP/UDP header checksum generation. The eTSEC does not checksum transmitted packets with IP header options or IP fragments.

To provide for quality of service, transmission from up to eight queues is supported with priority-based queue selection. Arbitration is a modified weighted round-robin queue selection with fair bandwidth allocation.

On receive, packets may be distributed to any of the 64 virtual receive queues overlaid onto the 8 physical receive queues. A table-oriented queue filing strategy is provided based on 16 header fields or flags. Frame rejection is supported for filtering applications.

Filing can be based on Ethernet, IP, and TCP/UDP properties, including VLAN fields, Ether-type, IP protocol type, IP TOS or differentiated services, IP source and destination addresses, TCP/UDP port numbers, or user-defined bit fields.

1.3.5 Enhanced Secure Digital Host Controller (eSDHC)

The enhanced SD Host Controller (eSDHC) provides an interface between the host system and SD/SDIO/MMC cards. The eSDHC acts as a bridge, passing host bus transactions to SD/SDIO/MMC cards by sending commands and performing data accesses to or from the cards. It handles SD/SDIO/MMC protocol at the transmission level.

The eSDHC includes the following features:

- Supports single- and multi-block read and write
- Supports write protection switch for write operations
- Supports synchronous and asynchronous abort
- Supports pause during the data transfer at a block gap
- Supports SDIO read wait and suspend/resume operations
- Supports Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer commands while the data transfer is in progress
- Allows cards to interrupt the host in 1- and 4-bit SDIO modes
- Supports interrupt period, defined in the SDIO standard
- Embodies a fully configurable 128×32-bit FIFO for read/write data
- Supports internal DMA capabilities
- Full-speed mode (up to 25 MHz) or high-speed mode (up to 50 MHz)

1.3.6 SerDes PHY

The SerDes PHY block includes the SerDes PHY, the protocol converter per protocol, the protocol mux, and the control registers and control logic.

The SerDes PHY block has the following features:

- Support for one x2 PCI Express. two x1 PCI Express, two x1 SGMII, and two x1 SATA
  - Gen1i, Gen1m, Gen2i, and Gen2m electrical specifications are supported in SATA mode, compliant to *Serial ATA 2.5 Specification*
- Link-layer interfaces to IP controller
- Memory-mapped registers with 256-byte address region
- SerDes power-down/reset state machine for cold (power-on) or warm (software-initiated) reset of SerDes, PHY, and controllers
- Provides *reset_done* indication
- Interface to clock controls

The SerDes PHY block supports the following modes of operation:

- SerDes1
  - Two lanes running x1 SGMII at 1.25 Gbps
  - Two lanes running x1 SATA at 1.5 or 3.0 Gbps
- SerDes2
  - Two lanes running x1 PCI Express at 2.5 Gbps
  - One lane running x2 PCI Express at 2.5 Gbps

1.3.7 PCI Controller

The 32-bit PCI controller is compatible with the *PCI Local Bus Specification*, *Rev. 2.3*. The PCI interface can function as a host bridge interface. The PCI interface can optionally function as an agent device. The PCI controller supports 32-bit addressing and 32-bit data buses.

As a host, the device supports read and write operations to the PCI memory space, the PCI I/O space, and the PCI configuration space. Also, the device can generate PCI special-cycle and interrupt acknowledge commands. As an agent, the device supports read and write operations to system memory, as well as PCI configuration space and the on-chip memory mapped configuration space.

The device PCI controller includes the following distinctive features:

- Address stepping on configuration transactions
- Fast back-to-back transactions
- Data streaming
- Supports mapping from an external 32- or 64-bit address space to the internal 32-bit local space
- Supports dual address cycle (DAC) 64-bit addressing mode as target only
- When in host mode, the PCI controller supports external signal isolation, thus enabling power shut off to external devices

1.3.7.1 PCI Bus Arbitration Unit

The PCI controller contains a PCI bus arbitration unit, which eliminates the need for an external unit, thus lowering system complexity and cost. It has the following features:

- Supports five $\overline{\text{REQ}}/\overline{\text{GNT}}$ signal pairs, thus supporting five external masters. The device PCI controller is the sixth member of the arbitration pool.
- The bus arbitration unit allows fairness as well as a priority mechanism.
- A two-level round-robin scheme is used in which each device can be programmed within a pool of a high- or low-priority arbitration. One member of the low-priority pool is promoted to the high-priority pool. As soon as it is granted the bus, it returns to the low-priority pool.
- The unit can be disabled to allow a remote arbitration unit to be used.
- The unit can be isolated to allow power shut off of external devices.

The Serial ATA controller is a high-performance SATA solution incorporating some of the latest SATA-IO extensions. The SATA may also be referred to as a host bus adapter (HBA). The SATA controller is designed to operate in a system that supports command queuing and, in particular, a switching scheme based on a frame information structure (FIS) using port multipliers.

The SATA controller has the following features:

- Designed to be compliant to the *Serial ATA 2.5 Specification*
- Supports speeds: 1.5 Gbps (first-generation SATA), 3 Gbps (second-generation SATA and eSATA)
- Supports advanced technology attachment packet interface (ATAPI) devices
- Contains high-speed descriptor-based DMA controller
- Supports native command queuing (NCQ) commands
- Supports port multiplier operation
- Supports hot plug including asynchronous signal recovery

There are four layers in the SATA architecture: application, transport, link, and PHY. The application layer is responsible for overall ATA command execution, including controlling command block register accesses. The transport layer is responsible for placing control information and data to be transferred between the host and device in a packet/frame, known as a frame information structure (FIS). The link layer is responsible for taking data from the constructed frames, inserting control characters, and moving data to the PHY layer. The PHY layer is responsible for 8B/10B encoding/decoding, then transmitting and receiving the encoded information as a serial data stream on the wire.

1.3.8 Universal Serial Bus (USB) 2.0

The USB 2.0 controller offers operation as a host or device. The USB controller provides point-to-point connectivity, which complies with the *Universal Serial Bus Revision 2.0 Specification*. The USB controller can be configured to operate as a stand-alone hostor stand-alone device. See Figure 1-4 for more information.

The host and device functions are both configured to support the following four types of USB transfers:

- Bulk
- Control
- Interrupt
- Isochronous

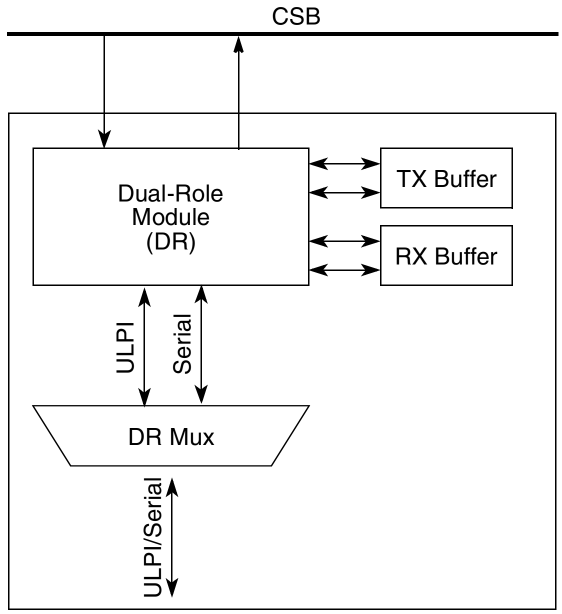


Figure 1-4. USB Controllers Port Configuration

1.3.8.1 USB Dual-Role Controller

- Designed to comply with *Universal Serial Bus Revision 2.0 Specification*
- Supports operation as a stand-alone USB host controller
  - Supports USB root hub with one downstream-facing port
  - Enhanced host controller interface (EHCI) compatible
- Supports operation as a stand-alone USB device
  - Supports one upstream-facing port
  - Supports six programmable bidirectional USB endpoints
- Supports high-speed (480-Mbps), full-speed (12-Mbps), and low-speed (1.5-Mbps) operations. Low speed is only supported in host mode.
- Host mode direct connect of full-speed and low-speed devices
- Supports USB on-the-go mode, which includes both device and host functionality
- Supports external PHY with serial and ULPI (UTMI+ low-pin interface)

1.3.9 Enhanced Local Bus Controller (eLBC)

The main component of the enhanced local bus controller (eLBC) is its memory controller, which provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling eight memory banks shared by a NAND Flash control machine (FCM), a general-purpose chip-select machine (GPCM),, and up to three user-programmable machines (UPMs). As such, it supports a minimal glue logic interface to SRAM, EPROM, NOR Flash EPROM, NAND Flash EPROM, burstable RAM, regular DRAM devices, extended data output DRAM devices, and other peripherals. The eLBC external address latch enable (LALE) signal allows multiplexing of addresses with data signals to reduce the device pin count.

The enhanced local bus controller also includes a number of data checking and protection features such as data parity generation and checking, write protection, and a bus monitor to ensure that each bus cycle is terminated within a user-specified period.

The main features of the enhanced local bus controller (eLBC) are as follows:

- Memory controller with eight memory banks (chip selects)
  - 32-bit address decoding with mask
  - Variable memory block sizes (32 Kbytes to 4 Gbytes in FCM mode)
  - Selection of control signal generation on a per-bank basis
  - Data buffer controls activated on a per-bank basis
  - Up to 256-byte bursts, arbitrarily aligned
  - Automatic segmentation of large transactions into memory accesses optimized for bus width and addressing capability
  - Odd/even parity checking including read-modify-write (RMW) parity for single accesses
  - Write-protection capability
  - Parity byte-select
  - Atomic operation
- General-purpose chip-select machine (GPCM)
  - Compatible with SRAM, EPROM, NOR Flash EEPROM, and peripherals
  - Global (boot) chip-select available at system reset
  - Boot chip-select support for 8-, 16-, 32-bit devices
  - Minimum three-clock access to external devices
  - Four byte-write-enable signals ($\overline{\text{LWE}}$[0:3])
  - Output enable signal ($\overline{\text{LOE}}$)
  - External access termination signal ($\overline{\text{LGTA}}$)
- NAND Flash control machine (FCM)
  - Compatible with small (512 + 16 bytes) and large (2048 + 64 bytes) page parallel NAND Flash EEPROM
  - Global (boot) chip-select available at system reset, with 4-Kbyte boot block buffer for execute-in-place boot loading
  - Boot chip-select support for 8-bit devices
  - Dual 2-Kbyte/eight 512-byte buffers allow simultaneous data transfer during Flash reads and programming
  - Interrupt-driven block transfer for reads and writes
  - Programmable command and data transfer sequences of up to eight steps supported
  - Generic command and address registers support proprietary Flash interfaces
  - Block write locking to ensure system security and integrity

- Three user-programmable machines (UPMs)
  - — Programmable-array-based machine controls external signal timing with a granularity of up to one quarter of an external bus clock period
  - — User-specified control-signal patterns run when an internal master requests a single-beat or burst read or write access
  - — UPM refresh timer runs a user-specified control signal pattern to support refresh
  - — User-specified control-signal patterns can be initiated by software
  - — Each UPM can be defined to support DRAM devices with depths of 64, 128, 256, and 512 Kbytes, and 1, 2, 4, 8, 16, 32, 64, 128, and 256 Mbytes
  - — Support for 8-, 16-, 32-bit devices
  - — Page mode support for successive transfers within a burst
  - — Internal address multiplexing supporting 64-, 128-, 256-, and 512-Kbyte, and 1-, 2-, 4-, 8-, 16-, 32-, 64-, 128-, and 256-Mbyte page banks
- Optional monitoring of transfers between local bus internal masters and local bus slaves (local bus error reporting)
- Support for phase-locked loop (PLL) with software-configurable bypass for low frequency bus clocks

1.3.10 Integrated Programmable Interrupt Controller (IPIC)

The IPIC implements the necessary functions to provide a flexible solution for general-purpose interrupt control. The IPIC includes the following features:

- Functional and programming models are compatible with the MPC8260 interrupt controller
- Support for external and internal discrete interrupt sources
- Support for one external (optional) and seven internal machine checkstop interrupt sources
- Programmable highest priority request
- Two programmable priority mixed groups of four on-chip and four external interrupt signals with two priority schemes for each group: grouped and spread
- Two programmable priority internal groups of eight on-chip interrupt signals with two priority schemes for each group: grouped and spread
- Priority interrupts can be programmed to support a critical ($\overline{cint}$) or system management ($\overline{smi}$) interrupt type
- External and internal interrupts directed to a host processor
- Unique vector number for each interrupt source
- Ability to redirect interrupts to external $\overline{\text{PCI\_INTA}}$ pin when in core disable mode

1.3.11 Dual I^{2}C Interfaces

The inter-IC (IIC or I^{2}C) bus is a two-wire—serial data (SDA) and serial clock (SCL)— bidirectional serial bus that provides a simple, efficient method of data exchange between the system and other devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The two-wire bus minimizes the interconnections between devices. The synchronous, multi-master bus of the I^{2}C allows the connection of additional devices to the bus for expansion and system development.

The I^{2}C controller is a true multi-master bus which includes collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously. This feature allows for complex applications with multiprocessor control. The I^{2}C controller consists of a transmitter/receiver unit, clocking unit, and control unit. The I^{2}C unit supports general broadcast mode and on-chip filtering rejects spikes on the bus.

The I^{2}C interfaces include the following features:

- Two-wire interface
- Multi-master operational
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Bus busy detection
- Software-programmable clock frequency
- Software-selectable acknowledge bit
- On-chip filtering for spikes on the bus
- Address broadcasting supported
- System initialization data is optionally loaded from I^{2}C EPROM by boot sequencer embedded hardware

1.3.12 DMA Controller

The DMA engine is capable of transferring blocks of data from any legal address range to any other legal address range. Therefore, it can perform a DMA transfer between any of its I/O or memory ports, or even between two devices or locations on the same port.

The DMA controller offers the following features:

- Four high-speed/high-bandwidth channels accessible by local and remote masters
- Handshaking (external control) signals supported for all channels: $\overline{\text{DREQ}}$[0:3], $\overline{\text{DACK}}$[0:3], $\overline{\text{DDONE}}$[0:3]
- Basic DMA operation modes (direct chaining)
- Data transfer between CSB, LBC, PCI Express
- Support for misaligned transfers
- Programmable bandwidth control between channels
- Interrupt on error and completed segment or chain
- Uses round-robin algorithm for channel arbitration

1.3.13 Dual Universal Asynchronous Receiver/Transmitter (DUART)

The device includes a DUART intended for use in maintenance, bring up, and debug systems. The device provides a standard four-wire handshake (TXD, RXD, $\overline{RTS}$, $\overline{CTS}$) for each port. The DUART is a slave interface. An interrupt is provided to the interrupt controller or optionally steered externally to allow device handshakes. Interrupts are generated for transmit, receive, and line status.

The DUART supports full-duplex operation. It is compatible with the PC16450 and PC16550 programming models. The transmitter and receiver both support 16-byte FIFOs.

Software programmable baud rate generators divide the system clock to generate a 16x clock. Serial interface data formats (data length, parity, 1/1.5/2 STOP bit, baud rate) are also software selectable.

The DUART includes the following features:

- Full-duplex operation
- Programming model compatible with the original PC16450 UART and the PC16550D (an improved version of the PC16450 that also operates in FIFO mode)
- PC16450 register reset values
- FIFO mode for both transmitter and receiver, providing 16-byte FIFOs
- Serial data encapsulation and decapsulation with standard asynchronous communication bits (START, STOP, and parity)
- Maskable transmit, receive, and line status interrupts
- Software-programmable baud rate generators that divide the system clock by 1 to (2^{16} – 1) and generate a 16x clock for the transmitter and receiver engines
- Clear to send ($\overline{CTS}$) and ready to send ($\overline{RTS}$) MODEM control functions
- Software-selectable serial-interface data format (data length, parity, 1/1.5/2 STOP bit, baud rate)
- Line status registers
- Line-break detection and generation
- Internal diagnostic support, local loopback, and break functions
- Prioritized interrupt reporting
- Overrun, parity, and framing error detection

1.3.14 Serial Peripheral Interface (SPI)

The serial peripheral interface (SPI) allows the device to exchange data between other PowerQUICC family chips, Ethernet PHYs for configuration, and peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.

The SPI is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface (receive, transmit, clock, and slave select). The SPI block consists of transmitter and receiver sections, an independent baud-rate generator, and a control unit. It has the ability to boot from an SPI serial flash device.

1.3.15 System Timers

The system includes the following timers:

- Periodic interrupt timer
- Real time clock
- Software watchdog timer
- Two general-purpose timer blocks, each supporting four 16-bit programmable timers, two cascaded 32-bit timers, or one cascaded 64-bit counter

1.4 Applications

The internal features of the MPC8379E make it suitable for a wide variety of network communication applications. It addresses the requirements of several networking applications, including routers, storage and printing and imaging applications. While addressing requirements related to time-to-market, price, power consumption, and board real estate, the MPC8379E extends offers higher CPU performance, new functionality, and high- speed interfaces.

Chapter 2
Memory Map

This chapter describes the MPC8379E memory map. The internal memory mapped registers are described, including a complete listing of all memory mapped registers with cross references to the sections detailing descriptions of each.

2.1 Internal Memory Mapped Registers

All of the memory mapped registers in the device are contained within a 1-Mbyte address region. To allow for flexibility, the base address of the memory mapped registers is relocatable in the local address space. The local address map location of this register block is controlled by the internal memory mapped registers base address register (IMMRBAR), see Appendix A 5.2.4.1, "Internal Memory Map Registers Base Address Register (IMMRBAR)," for more information. The default value for IMMRBAR is 0xFF40_0000.

2.2 Accessing IMMR Memory From the Local Processor

When the local e300 processor is used to configure IMMR space, the IMMR memory space should typically be marked as cache-inhibited and guarded.

In addition, many configuration registers affect accesses to other memory regions; therefore, writes to these registers must be guaranteed to have taken effect before accesses are made to the associated memory regions.

To guarantee that the results of any sequence of writes to configuration registers are in effect, the final configuration register write should be followed immediately by a read of the same register, and that should be followed by a sync instruction. Then accesses can safely be made to memory regions affected by the configuration register write.

2.3 IMMR Address Map

Table 2-1 lists the location of the functional block base addresses for the entire IMMRBAR space. Unless stated otherwise in a particular block, all accesses to and from the memory mapped registers must be made with 32-bit accesses. There is no support for accesses of sizes other than 32 bits.

Reading from address locations which appear as reserved in the memory map table is not guaranteed to return predictable data. Writing to address locations which appear as reserved in the memory map table is not allowed and could lead to unpredictable behavior of the device. Reserved bits in non-reserved registers will be read as zero unless the reset value of those bits is different due to internal logic considerations.

When writing to registers with reserved bits, those reserved bits should be cleared. By doing so, existing software would be able to run on a future modified device in which some reserved bits were allocated for enhanced modes. This would allow for maintaining the legacy functionality when set to zero.

In certain specific cases, reserved bits should not be cleared but should keep their reset value. Thus, the software should perform a ‘read-modify-write’ and make sure that it does not change the reset value of those bits. The description of the specific bits will indicate when this is needed.

Cross-references are provided to the IMMRBAR maps for each individual block. A complete listing of all registers is provided in Appendix A, “Complete List of Configuration, Control, and Status Registers.”

Table 2-1. IMMR Memory Map

Block Base Address	Block	Section/Page	Comments
0x0_0000	Local access window	5.2.3.1/5-5	—
	System configuration	5.3.1/5-20	—
0x0_0200	Watchdog timer (WDT)	5.4.4/5-37	—
0x0_0300	Real time clock (RTC)	5.5.5/5-44	—
0x0_0400	Periodic interval timer (PIT)	5.6.5/5-52	—
0x0_0500	General purpose (global) timers (GTMs)	5.7.5/5-61	Global timers module 1: 0x0_0500 Global timers module 2: 0x0_0600
0x0_0700	Integrated programable interrupt controller (IPIC)	8.5/8-6	—
0x0_0800	System arbiter	6.2/6-2	—
0x0_0900	Reset configuration	4.5.1/4-34	—
0x0_0A00	Clock configuration	4.5.2/4-39	—
0x0_0B00	Power management controller (PMC)	5.8.2/5-74	—
0x0_0C00	General purpose I/O (GPIO)	25.3/25-2	GPIO1: 0x0_0C00 GPIO2: 0x0_0D00
0x0_0E00	Reserved	—	—
0x0_2000	DDR memory controller	9.4/9-8	—
0x0_3000	I^2C controller	21.3/21-4	I^2C1: 0x0_3000 I^2C2: 0x0_3100
0x0_3200	Reserved	—	—
0x0_4500	Dual UART (DUART)	22.3/22-4	UART1: 0x0_4500 UART2: 0x0_4600
0x0_4700	Reserved	—	—
0x0_5000	Enhanced local bus controller (eLBC)	10.3/10-9	—
0x0_6000	Reserved	—	—
0x0_7000	Serial peripheral interface (SPI)	23.4/23-8	—
0x0_8000	DMA controller	13.3/13-3	—

Table 2-1. IMMR Memory Map (continued)

Block Base Address	Block	Section/Page	Comments
0x0_8300	PCI configuration access	14.3/14-11	—
0x0_8380	Reserved	—	—
0x0_8400	I/O sequencer (IOS)	12.3/12-2	—
0x0_8500	PCI controller	14.3/14-11	—
0x0_8600	Reserved	—	—
0x0_9000	PCI Express controller	15.3.1/15-5	PCI Express 1: 0x0_9000 PCI Express 2: 0x0_A000
0x0_B000	Reserved	—	—
0x1_8000	Serial ATA (SATA) controller	16.3.1/16-4	SATA1: 0x1_8000 SATA2: 0x1_9000 SATA3: 0x1_A000 SATA4: 0x1_B000
0x1_C000	Reserved	—	—
0x2_3000	USB dual-role (DR) controller	20.3/20-8	—
0x2_4000	Enhanced three-speed Ethernet controller (eTSEC)	18.5/18-11	eTSEC 1: 0x2_4000 eTSEC 2: 0x2_5000
0x2_6000	Reserved	—	—
0x2_E000	Enhanced secure digital host controller (eSDHC)	11.4/11-4	—
0x2_F000	Reserved	—	—
0x3_0000	Security engine controller (SEC)	17.2/17-11	—
0x4_0000	Reserved	—	—
0xE_3000	SerDes PHY	19.3/19-4	SerDes 1: 0xE_3000 SerDes 2: 0xE_3100
0xE_3200	Reserved	—	—
0xF_0000	On-Chip ROM	—	—

Chapter 3
Signal Descriptions

This chapter describes the external signals of the device. It is organized into the following sections:

- Overview of signals and cross references for signals that serve multiple functions, including two lists: one ordered by functional block and one alphabetical.
- List of reset configuration signals
- List of output signal states at reset

NOTE

A bar over a signal name indicates that the signal is active low, such as $\overline{\text{IRQ\_OUT}}$ (interrupt out). Active-low signals are referred to as asserted (active) when they are low and negated when they are high. Signals that are not active low, such as IRQ (interrupt input), are referred to as asserted when they are high and negated when they are low.

Internal signals throughout this document are shown as lower case and in italics. For example, *sys_logic_clk* is an internal signal. These are referenced only as necessary for understanding of the external functionality of the device.

3.1 Signals Overview

The signals are grouped as follows:

- DDR memory interface signals
- PCI interface signals
- DUART interface signals
- I^2C interface signals
- Serial peripheral interface signals
- Ethernet management interface signals
- eTSEC1 interface signals
- eTSEC2 interface signals
- SerDes interface signals
- Enhanced local bus interface signals
- USB signals
- Global timers interface signals
- PIC interface signals
- JTAG, PMC, system control signals
- SATA PHY signals
- Clock signals

Figure 3-1 and Figure 3-2 show the external signals of the device and how the signals are grouped. Refer to the *MPC8379E Integrated Processor Hardware Specifications* for a pinout diagram showing pin numbers and a listing of all the electrical and mechanical specifications.

Note that individual chapters of this document provide details for each signal, describing each signal's behavior when asserted and negated and when the signal is an input or an output.

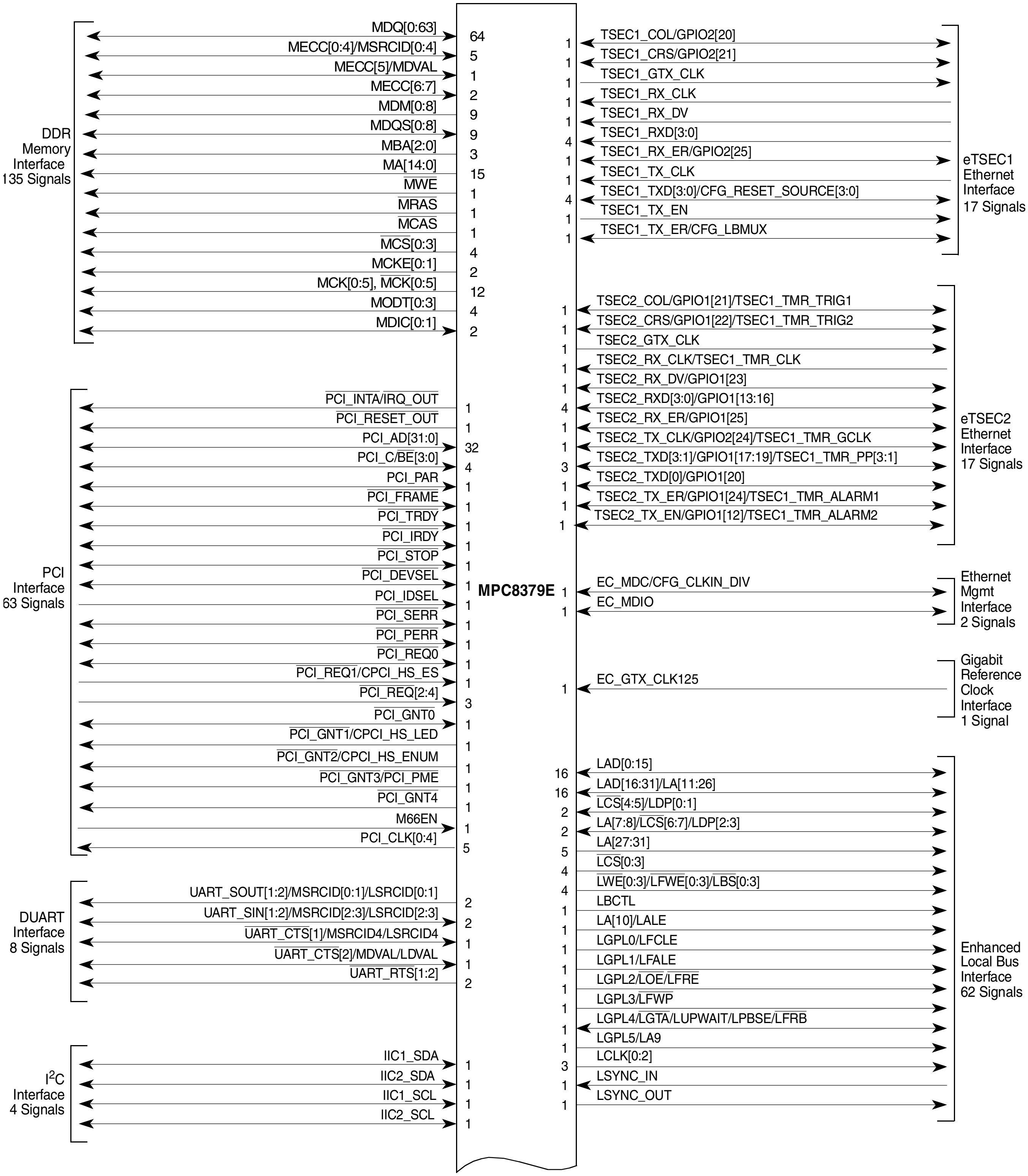


Figure 3-1. MPC8379E Signal Groupings (1 of 2)

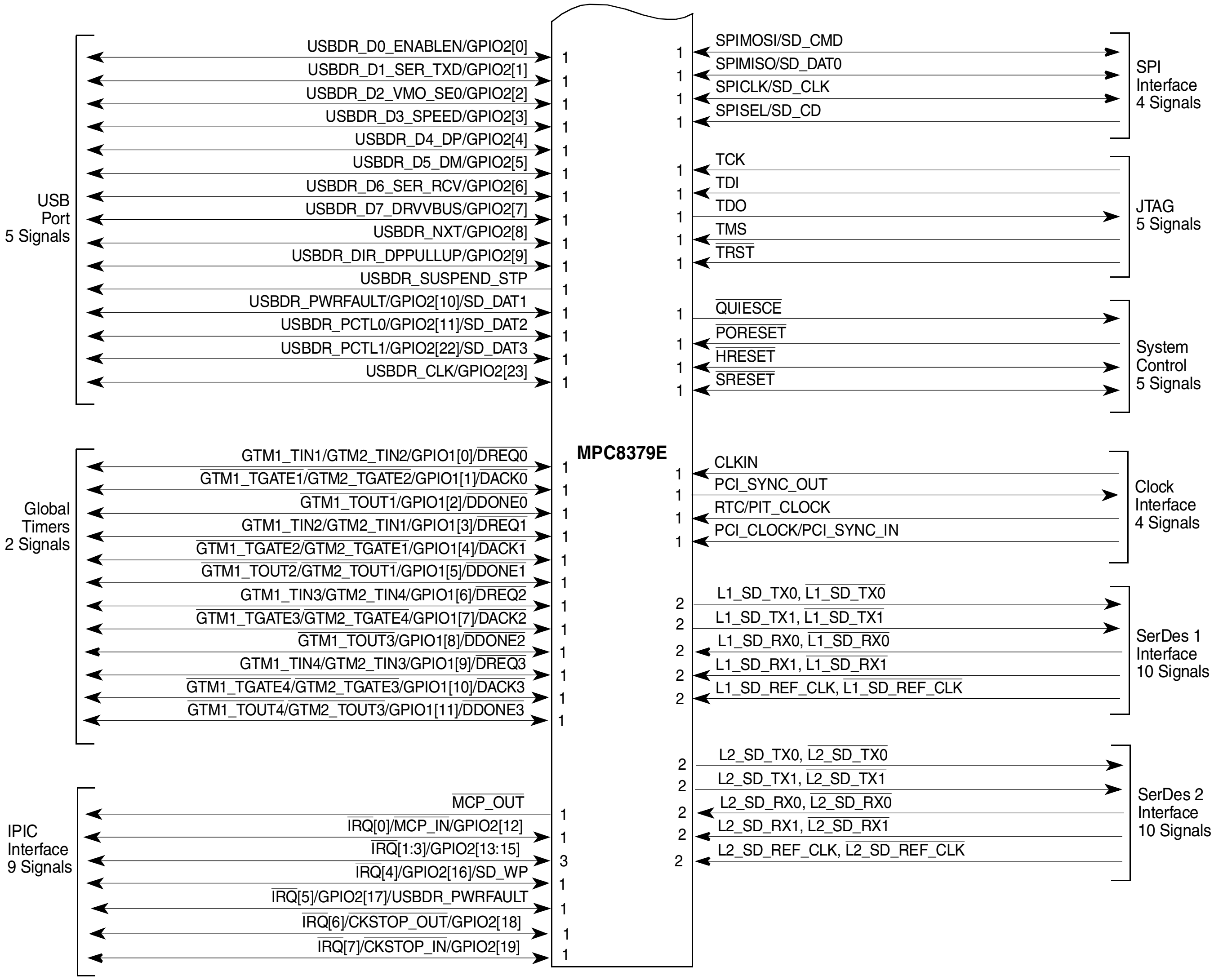


Figure 3-2. MPC8379E Signal Groupings (2 of 2)

The following tables provide summaries of signal functions. Table 3-1 provides a summary of the signals grouped by function, and Table 3-2 provides a summary of the signals grouped alphabetically. These tables detail the signal name, interface, alternate functions, number of signals, and whether the signal is an input, output, or bidirectional. Finally, the table provides a pointer to the table where the signal function is described.

Table 3-1. MPC8379E Signal Reference by Functional Block

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MDQ[0:63]	DDR data	DDR	64	I/O	9-3/9-5	—	—
MECC[0:4]	DDR error correcting code	DDR	5	I/O	9-3/9-5	MSRCID[0:4]	—
MECC[5]	DDR error correcting code	DDR	1	I/O	9-3/9-5	MDVAL	—
MECC[6:7]	DDR error correcting code	DDR	2	I/O	9-3/9-5	—	—
MDM[0:7]	DDR data mask	DDR	8	O	9-3/9-5	—	—
MDM[8]	DDR ECC data mask	DDR	1	O	9-3/9-5	—	—
MDQS[0:7]	DDR data strobe	DDR	8	I/O	9-3/9-5	—	—
MDQS[8]	DDR ECC data strobe	DDR	1	I/O	9-3/9-5	—	—
MBA[2:0]	DDR bank select	DDR	3	O	9-3/9-5	—	—
MA[14:0]	DDR address	DDR	15	O	9-3/9-5	—	—
$\overline{\text{MWE}}$	DDR write enable	DDR	1	O	9-3/9-5	—	—
$\overline{\text{MRAS}}$	DDR row address strobe	DDR	1	O	9-3/9-5	—	—
$\overline{\text{MCAS}}$	DDR column address strobe	DDR	1	O	9-3/9-5	—	—
$\overline{\text{MCS}}$[0:3]	DDR chip select (2/DIMM)	DDR	4	O	9-3/9-5	—	—
MCKE[0:1]	DDR clock enable	DDR	2	O	9-4/9-8	—	—
MCK[0:5], $\overline{\text{MCK}}$[0:5]	DDR differential clocks (3 pairs/DIMM)	DDR	12	O	9-4/9-8	—	—
MODT[0:3]	DRAM on-die termination	DDR	4	O	9-3/9-5	—	—
MDIC[0:1]	Driver impedance calibration	DDR	2	I/O	9-3/9-5	—	—
$\overline{\text{PCI\_INTA}}$	PCI interrupt output	PCI	1	O	14-3/14-5	$\overline{\text{IRQ\_OUT}}$	8-2/8-5
$\overline{\text{PCI\_RESET\_OUT}}$	PCI reset	PCI	1	O	14-3/14-5	—	—
PCI_AD[31:0]	PCI address/data	PCI	32	I/O	14-3/14-5	—	—

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
PCI_C/BE[3:0]	PCI command/byte enable	PCI	4	I/O	14-3/14-5	—	—
PCI_PAR	PCI parity	PCI	1	I/O	14-3/14-5	—	—
PCI_FRAME	PCI frame	PCI	1	I/O	14-3/14-5	—	—
PCI_TRDY	PCI target ready	PCI	1	I/O	14-3/14-5	—	—
PCI_IRDY	PCI initiator ready	PCI	1	I/O	14-3/14-5	—	—
PCI_STOP	PCI stop	PCI	1	I/O	14-3/14-5	—	—
PCI_DEVSEL	PCI device select	PCI	1	I/O	14-3/14-5	—	—
PCI_IDSEL	PCI initial device select	PCI	1	I	14-3/14-5	—	—
PCI_SERR	PCI system error	PCI	1	I/O	14-3/14-5	—	—
PCI_PERR	PCI parity error	PCI	1	I/O	14-3/14-5	—	—
PCI_REQ0	PCI request 0	PCI	1	I/O	14-3/14-5	—	—
PCI_REQ1	PCI request 1	PCI	1	I	14-3/14-5	CPCI_HS_ES	14-3/14-5
PCI_REQ[2:4]	PCI request 2–4	PCI	3	I	14-3/14-5	—	—
PCI_GNT0	PCI grant 0	PCI	1	I/O	14-3/14-5	—	—
PCI_GNT1	PCI grant 1	PCI	1	O	14-3/14-5	CPCI_HS_LED	14-3/14-5
PCI_GNT2	PCI grant 2	PCI	1	O	14-3/14-5	CPCI_HS_ENUM	14-3/14-5
PCI_GNT3	PCI grant 3	PCI	1	O	14-3/14-5	PCI_PME	14-3/14-5
PCI_GNT4	PCI grant 4	PCI	1	O	14-3/14-5	—	—
PCI_CLK[0:4]	PCI clock 0–4	PCI	5	O	4-2/4-3	—	—
PCI_PME	PCI PME assertion request	PCI	1	O	14-3/14-5	PCI_GNT3	14-3/14-5
M66EN	66-MHz system configuration	PCI	1	I	14-3/14-5	—	—
CPCI_HS_ENUM	CompactPCI hot swap enumerator	PCI	1	O	14-3/14-5	PCI_GNT2	14-3/14-5
CPCI_HS_ES	CompactPCI hot swap ejector switch	PCI	1	I	14-3/14-5	PCI_REQ1	14-3/14-5
CPCI_HS_LED	CompactPCI hot swap LED	PCI	1	O	14-3/14-5	PCI_GNT1	14-3/14-5
L1_REF_CLK	Reference clock for SerDes 1	SerDes 1	1	I	19-1/19-3	—	—
L1_REF_CLK	Reference clock for SerDes 1 (complement)	SerDes 1	1	I	19-1/19-3	—	—

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
L1_SD_RX0	SerDes 1, lane 0, receive data	SerDes 1	1	I	19-1/19-3	—	—
$\overline{\text{L1\_SD\_RX0}}$	SerDes 1, lane 0, receive data (complement)	SerDes 1	1	I	19-1/19-3	—	—
L1_SD_RX1	SerDes 1, lane 1, receive data	SerDes 1	1	I	19-1/19-3	—	—
$\overline{\text{L1\_SD\_RX1}}$	SerDes 1, lane 1, receive data (complement)	SerDes 1	1	I	19-1/19-3	—	—
L1_SD_TX0	SerDes 1, lane 0, transmit data	SerDes 1	1	O	19-1/19-3	—	—
$\overline{\text{L1\_SD\_TX0}}$	SerDes 1, lane 0, transmit data (complement)	SerDes 1	1	O	19-1/19-3	—	—
L1_SD_TX1	SerDes 1, lane 1, transmit data	SerDes 1	1	O	19-1/19-3	—	—
$\overline{\text{L1\_SD\_TX1}}$	SerDes 1, lane 1, transmit data (complement)	SerDes 1	1	O	19-1/19-3	—	—
L2_SD_REF_CLK	Reference clock for SerDes 2	SerDes 2	1	I	19-1/19-3	—	—
$\overline{\text{L2\_SD\_REF\_CLK}}$	Reference clock for SerDes 2 (complement)	SerDes 2	1	I	19-1/19-3	—	—
L2_SD_RX0	SerDes 2, lane 0, receive data	SerDes 2	1	I	19-1/19-3	—	—
$\overline{\text{L2\_SD\_RX0}}$	SerDes 2, lane 0, receive data (complement)	SerDes 2	1	I	19-1/19-3	—	—
L2_SD_RX1	SerDes 2, lane 1, receive data	SerDes 2	1	I	19-1/19-3	—	—
$\overline{\text{L2\_SD\_RX1}}$	SerDes 2, lane 1, receive data (complement)	SerDes 2	1	I	19-1/19-3	—	—
L2_SD_TX0	SerDes 2, lane 0, transmit data	SerDes 2	1	O	19-1/19-3	—	—
$\overline{\text{L2\_SD\_TX0}}$	SerDes 2, lane 0, transmit data (complement)	SerDes 2	1	O	19-1/19-3	—	—
L2_SD_TX1	SerDes 2, lane 1, transmit data	SerDes 2	1	O	19-1/19-3	—	—

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
$\overline{\text{L2\_SD\_TX1}}$	SerDes 2, lane 1, transmit data (complement)	SerDes 2	1	O	19-1/19-3	—	—
SD_DAT0	Dat0 line	eSDHC	1	I/O	11-1/11-3	SPIMISO	23-2/23-7
SD_DAT1	Dat1 line	eSDHC	1	I/O	11-1/11-3	GPIO2[10]/ USBDR_PWRFAULT	25-1/25-2, 20-2/20-5
SD_DAT2	Dat2 line	eSDHC	1	I/O	11-1/11-3	GPIO2[11]/ USBDR_PCTL0	25-1/25-2, 20-2/20-5
SD_DAT3	Dat3 line	eSDHC	1	I/O	11-1/11-3	GPIO2[22]/ USBDR_PCTL1	25-1/25-2, 20-2/20-5
SD_WP	Card write protect detect	eSDHC	1	I/O	11-1/11-3	GPIO2[16]/ IRQ[4]	25-1/25-2, 8-2/8-5
SD_CD	Card detection	eSDHC	1	I	11-1/11-3	SPISEL	23-2/23-7
SD_CLK	Clock for MMC/SD/SDIO card	eSDHC	1	I/O	11-1/11-3	SPICLK	23-2/23-7
SD_CMD	CMD line connect to card	eSDHC	1	I/O	11-1/11-3	SPIMOSI	23-2/23-7
UART_SOUT[1:2]	DUART serial data out	DUART	2	O	22-2/22-3	MSRCID[0:1]/ LSRCID[0:1]	—, 10-2/10-5
UART_SIN[1:2]	DUART serial data in	DUART	2	I/O	22-2/22-3	MSRCID[2:3]/ LSRCID[2:3]	—, 10-2/10-5
UART_CTS1	DUART clear to send	DUART	1	I/O	22-2/22-3	MSRCID4/ LSRCID4	—, 10-2/10-5
UART_CTS2	DUART clear to send	DUART	1	I/O	22-2/22-3	MDVAL/LDVAL	—, 10-2/10-5
$\overline{\text{UART\_RTS}}$[1:2]	DUART ready to send	DUART	2	O	22-2/22-3	—	—
IIC1_SDA	I^2C serial data	I^2C1	1	I/O	21-1/21-3	—	—
IIC1_SCL	I^2C serial clock	I^2C1	1	I/O	21-1/21-3	—	—
IIC2_SDA	I^2C serial data	I^2C2	1	I/O	21-1/21-3	—	—
IIC2_SCL	I^2C serial clock	I^2C2	1	I/O	21-1/21-3	—	—
SPIMOSI	SPI master-out slave-in	SPI	1	I/O	23-2/23-7	SD_CMD	11-1/11-3
SPIMISO	SPI master-in slave-out	SPI	1	I/O	23-2/23-7	SD_DAT0	11-1/11-3
SPICLK	SPI clock	SPI	1	I/O	23-2/23-7	SD_CLK	11-1/11-3
SPISEL	SPI slave select	SPI	1	I	23-2/23-7	SD_CD	11-1/11-3

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC1_COL	eTSEC1 collision detect	eTSEC1	1	I/O	18-2/18-8	GPIO2[20]	25-1/25-2
TSEC1_CRS	eTSEC11 carrier sense	eTSEC1	1	I/O	18-2/18-8	GPIO2[21]	25-1/25-2
TSEC1_GTX_CLK	eTSEC1 transmit clock out	eTSEC1	1	O	18-2/18-8	—	—
TSEC1_RX_CLK	eTSEC1 receive clock	eTSEC1	1	I	18-2/18-8	—	—
TSEC1_RX_DV	eTSEC1 receive data valid	eTSEC1	1	I	18-2/18-8	—	—
TSEC1_RXD[3:0]	eTSEC1 receive data 3–0	eTSEC1	4	I	18-2/18-8	—	—
TSEC1_RX_ER	eTSEC1 receiver error	eTSEC1	1	I/O	18-2/18-8	GPIO2[25]	25-1/25-2
TSEC1_TX_ER	eTSEC1 transmit error	eTSEC1	1	I/O	18-2/18-8	CFG_LBMUX	4-1/4-1
TSEC1_TX_CLK	eTSEC1 transmit clock in	eTSEC1	1	I	18-2/18-8	—	—
TSEC1_TXD[3:0]	eTSEC1 transmit data 3–0	eTSEC1	4	I/O	18-2/18-8	CFG_RESET_ SOURCE[3:0]	4-1/4-1
TSEC1_TX_EN	eTSEC1 transmit enable	eTSEC1	1	O	18-2/18-8	—	—
TSEC2_COL	eTSEC2 collision detect	eTSEC2	1	I/O	18-2/18-8	GPIO1[21]/ TSEC1_TMR_TRIG1	25-1/25-2, 18-2/18-8
TSEC2_CRS	eTSEC2 carrier sense	eTSEC2	1	I/O	18-2/18-8	GPIO1[22]/ TSEC1_TMR_TRIG2	25-1/25-2, 18-2/18-8
TSEC2_GTX_CLK	eTSEC2 transmit clock out	eTSEC2	1	O	18-2/18-8	—	—
TSEC2_RX_CLK	eTSEC2 receive clock	eTSEC2	1	I	18-2/18-8	TSEC1_TMR_CLK	18-2/18-8
TSEC2_RX_DV	eTSEC2 receive data valid	eTSEC2	1	I/O	18-2/18-8	GPIO1[23]	25-1/25-2
TSEC2_RXD[3:0]	eTSEC2 receive data 3–0	eTSEC2	4	I/O	18-2/18-8	GPIO1[13:16]	25-1/25-2
TSEC2_RX_ER	eTSEC2 receiver error	eTSEC2	1	I/O	18-2/18-8	GPIO1[25]	25-1/25-2
TSEC2_TXD[3:1]	eTSEC2 transmit data 3–1	eTSEC2	3	I/O	18-2/18-8	GPIO1[17:19]/ TSEC1_TMR_PP[3:1]	25-1/25-2, 18-2/18-8
TSEC2_TXD[0]	eTSEC2 transmit data 0	eTSEC2	1	I/O	18-2/18-8	GPIO1[20]	25-1/25-2

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC2_TX_ER	eTSEC2 transmit error	eTSEC2	1	I/O	18-2/18-8	GPIO1[24]/ TSEC1_TMR_ ALARM1	25-1/25-2, 18-2/18-8
TSEC2_TX_EN	eTSEC2 transmit enable	eTSEC2	1	I/O	18-2/18-8	GPIO1[12]/ TSEC1_TMR_ ALARM2	25-1/25-2
TSEC2_TX_CLK	eTSEC2 transmit clock in	eTSEC2	1	I/O	18-2/18-8	GPIO2[24]/ TSEC1_TMR_GCLK	25-1/25-2, 18-2/18-8
TSEC_TMR_ ALARM_OUT1	1588 timer alarm-out 1	eTSEC1/ eTSEC2	1	I/O	18-2/18-8	GPIO1[24]/ TSEC2_TX_ER	25-1/25-2, 18-2/18-8
TSEC_TMR_ ALARM_OUT2	1588 timer alarm-out 2	eTSEC1/ eTSEC2	1	I/O	18-2/18-8,	GPIO1[12]/ TSEC2_TX_EN	25-1/25-2, 18-2/18-8
TSEC_TMR_ PULSE_OUT[3:1]	1588 pulse-out [3:1]	eTSEC1/ eTSEC2	3	I/O	18-2/18-8	GPIO1[17:19]/ TSEC2_TXD[3:1]	25-1/25-2, 18-2/18-8
TSEC_TMR_ TRIG_IN1	1588 trigger-in 1	eTSEC1/ eTSEC2	1	I/O	18-2/18-8	GPIO1[21]/ TSEC2_COL	25-1/25-2, 18-2/18-8
TSEC_TMR_ TRIG_IN2	1588 trigger-in 2	eTSEC1/ eTSEC2	1	I/O	18-2/18-8	GPIO1[22]/ TSEC2_CRS	25-1/25-2, 18-2/18-8
TSEC_TMR_ CLK_OUT	1588 clock-out	eTSEC1/ eTSEC2	1	I/O	18-2/18-8	GPIO2[24]/ TSEC2_TX_CLK	25-1/25-2, 18-2/18-8
TSEC_TMR_CLK_IN	1588 clock-in	eTSEC1/ eTSEC2	1	I	18-2/18-8	TSEC2_RX_CLK	18-2/18-8
EC_MDC	Ethernet management data clock	Ethernet management	1	I/O	18-2/18-8	CFG_CLKIN_DIV	4-1/4-1
EC_MDIO	Ethernet management data in/out	Ethernet management	1	I/O	18-2/18-8	—	—
EC_GTX_CLK125	Gigabit reference clock	eTSEC1/ eTSEC2	1	I	18-2/18-8	—	—
LAD[0:15]	LBC address/data	eLBC	16	I/O	10-2/10-5	—	—
LAD[16:31]	LBC address/data	eLBC	16	I/O	10-2/10-5	LA[11:26]	10-2/10-5
LDP[0:1]	LBC data parity 0–1	eLBC	2	I/O	10-2/10-5	$\overline{\text{LCS}}$[4:5]	10-2/10-5
LDP[2:3]	LBC data parity 2–3	eLBC	2	I/O	10-2/10-5	$\overline{\text{LCS}}$[6:7]/LA[7:8]	10-2/10-5
LA[27:31]	LBC port address	eLBC	5	O	10-2/10-5	—	—
$\overline{\text{LCS}}$[0:3]	LBC chip select 0–3	eLBC	4	O	10-2/10-5	—	—
$\overline{\text{LWE}}$[0:3]/$\overline{\text{LFWE}}$[0:3]	LBC write enable/ byte select/FCM write enable	eLBC	4	O	10-2/10-5	$\overline{\text{LBS}}$[0:3]	10-2/10-5

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
LBCTL	LBC data buffer control	eLBC	1	O	10-2/10-5	—	—
LALE	LBC address latch enable	eLBC	1	O	10-2/10-5	LA10	10-2/10-5
LGPL0/LFCLE	LBC UPM general purpose line 0/Flash command latch enable	eLBC	1	O	10-2/10-5	—	—
LGPL1/LFALE	LBC GP line 1/Flash address latch enable	eLBC	1	O	10-2/10-5	—	—
LGPL2/LOE/LFRE	LBC output enable/GP line 2/FCM read enable	eLBC	1	O	10-2/10-5	—	—
LGPL3/LFWP	LBC GP line 3/Flash write project	eLBC	1	O	10-2/10-5	—	—
LGPL4/LGTA/ LUPWAIT/LPBSE/ LFRB	LBC GP line 4/GPCM terminate access/ UPM wait/parity byte select/Flash read/busy, open-drain shared pin	eLBC	1	I/O	10-2/10-5	—	—
LGPL5	LBC GP line 5	eLBC	1	O	10-2/10-5	LA9	10-2/10-5
LCLK[0:2]	LBC clocks 0–2	eLBC	3	O	10-2/10-5	—	—
LSYNC_OUT	LBC DLL synchronization output	eLBC	1	O	10-2/10-5	—	—
LSYNC_IN	LBC DLL synchronization input	eLBC	1	I	10-2/10-5	—	—
LA9	LBC port address 9	eLBC	1	O	10-2/10-5	LGPL5	10-2/10-5
LA10	LBC port address 10	eLBC	1	O	10-2/10-5	LALE	10-2/10-5
LA[11:26]	LBC port address 11–26	eLBC	16	I/O	10-2/10-5	LAD[16:31]	10-2/10-5
LBS[0:3]	Byte lane select	eLBC	4	O	10-2/10-5	LWE[0:3]/LFWE[0:3]	10-2/10-5
LCS[4:5]	LBC chip select 4–5	eLBC	2	I/O	10-2/10-5	LDP[0:1]	10-2/10-5
LCS[6:7]	LBC chip select 6–7	eLBC	2	I/O	10-2/10-5	LDP[2:3]/LA[7:8]	10-2/10-5
USBDR_D0_ ENABLEN	USB host 0 data bit 0	USB	1	I/O	20-2/20-5	GPIO2[0]	25-1/25-2
USBDR_D1_SER_ TXD	USB host 0 data bit 1	USB	1	I/O	20-2/20-5	GPIO2[1]	25-1/25-2

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
USBDR_D2_VMO_ SE0	USB host 0 data bit 2	USB	1	I/O	20-2/20-5	GPIO2[2]	25-1/25-2
USBDR_D3_SPEED	USB host 0 data bit 3	USB	1	I/O	20-2/20-5	GPIO2[3]	25-1/25-2
USBDR_D4_DP	USB host 0 data bit 4	USB	1	I/O	20-2/20-5	GPIO2[4]	25-1/25-2
USBDR_D5_DM	USB host 0 data bit 5	USB	1	I/O	20-2/20-5	GPIO2[5]	25-1/25-2
USBDR_D6_SER_ RCV	USB host 0 data bit 6	USB	1	I/O	20-2/20-5	GPIO2[6]	25-1/25-2
USBDR_D7_ DRVVBUS	USB host 0 data bit 7	USB	1	I/O	20-2/20-5	GPIO2[7]	25-1/25-2
USBDR_NXT	USB host 0 next data	USB	1	I/O	20-2/20-5	GPIO2[8]	25-1/25-2
USBDR_DIR_ DPPULLUP	USB host 0 data bus direction	USB	1	I/O	20-2/20-5	GPIO2[9]	25-1/25-2
USBDR_STP _SUSPEND	USB host 0 data stop/suspend	USB	1	I/O	20-2/20-5	—	—
USBDR_PWRFAULT	USB host 0 power fault	USB	1	I/O	20-2/20-5	GPIO2[10]/SD_DAT1	25-1/25-2, 11-1/11-3
USBDR_PWRFAULT	USB VBus power fault	USB	1	I/O	20-2/20-5	GPIO2[17]/ IRQ[5]	25-1/25-2, 8-2/8-5
USBDR_PCTL0	USB host 0 port control 0	USB	1	I/O	20-2/20-5	GPIO2[11]/SD_DAT2	25-1/25-2, 11-1/11-3
USBDR_PCTL1	USB host 0 port control 1	USB	1	I/O	20-2/20-5	GPIO2[22]/SD_DAT3	25-1/25-2, 11-1/11-3
USBDR_CLK	USB host 0 interface clock	USB	1	I/O	20-2/20-5	GPIO2[23]	25-1/25-2
GTM1_TIN1/ GTM2_TIN2	Timer in 1/2	Global Timers	1	I/O	5-59/5-60	GPIO1[0]/ DREQ0	25-1/25-2, 13-1/13-2
GTM1_TGATE1/ GTM2_TGATE2	Timer gate 1/2	Global Timers	1	I/O	5-59/5-60	GPIO1[1]/ DACK0	25-1/25-2, 13-1/13-2
GTM1_TOUT1	Timer out 1	Global Timers	1	I/O	5-59/5-60	GPIO1[2]/ DDONE0	25-1/25-2, 13-1/13-2
GTM1_TIN2/ GTM2_TIN1	Timer in 2/1	Global Timers	1	I/O	5-59/5-60	GPIO1[3]/ DREQ1	25-1/25-2, 13-1/13-2
GTM1_TGATE2/ GTM2_TGATE1	Timer gate 2/1	Global Timers	1	I/O	5-59/5-60	GPIO1[4]/ DACK1	25-1/25-2, 13-1/13-2
GTM1_TOUT2/ GTM2_TOUT1	Timer out 2/1	Global Timers	1	I/O	5-59/5-60	GPIO1[5]/ DDONE1	25-1/25-2, 13-1/13-2
GTM1_TIN3/ GTM2_TIN4	Timer in 3/4	Global Timers	1	I/O	5-59/5-60	GPIO1[6]/ DREQ2	25-1/25-2, 13-1/13-2

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GTM1_TGATE3/ GTM2_TGATE4	Timer gate 3/4	Global Timers	1	I/O	5-59/5-60	GPIO1[7]/ DACK2	25-1/25-2, 13-1/13-2
GTM1_TOUT3	Timer out 3	Global Timers	1	I/O	5-59/5-60	GPIO1[8]/ DDONE2	25-1/25-2, 13-1/13-2
GTM1_TIN4/ GTM2_TIN3	Timer in 4/3	Global Timers	1	I/O	5-59/5-60	GPIO1[9]/ DREQ3	25-1/25-2, 13-1/13-2
GTM1_TGATE4/ GTM2_TGATE3	Timer gate 4/3	Global Timers	1	I/O	5-59/5-60	GPIO1[10]/ DACK3	25-1/25-2 ,13-1/13-2
GTM1_TOUT4/ GTM2_TOUT3	Timer out 4/3	Global Timers	1	I/O	5-59/5-60	GPIO1[11]/ DDONE3	25-1/25-2, 13-1/13-2
MCP_OUT	Machine check interrupt output	IPIC	1	O	8-2/8-5	—	—
IRQ[0]/MCP_IN	External interrupt 0	IPIC	1	I/O	8-2/8-5	GPIO2[12]	25-1/25-2
IRQ[1:3]	External interrupt 1–3	IPIC	3	I/O	8-2/8-5	GPIO2[13:15]	25-1/25-2
IRQ[4]	External interrupt 4	IPIC	1	I/O	8-2/8-5	GPIO2[16]/SD_WP	25-1/25-2/ 11-1/11-3
IRQ[5]	External interrupt 5	IPIC	1	I/O	8-2/8-5	GPIO2[17]/ USBDR_PWRFAULT	25-1/25-2
IRQ[6]	External interrupt 6	IPIC	1	I/O	8-2/8-5	GPIO2[18]/ CKSTOP_OUT	25-1/25-2, 4-3/4-4
IRQ[7]	External interrupt 7	IPIC	1	I/O	8-2/8-5	GPIO2[19]/ CKSTOP_IN	25-1/25-2, 4-3/4-4
IRQ_OUT	Interrupt out	IPIC	1	O	8-2/8-5	PCI_INTA	14-3/14-5
TCK	Test clock	JTAG	1	I	24-2/24-2	—	—
TDI	Test data in	JTAG	1	I	24-2/24-2	—	—
TDO	Test data out	JTAG	1	O	24-2/24-2	—	—
TMS	Test mode select	JTAG	1	I	24-2/24-2	—	—
TRST	Test reset	JTAG	1	I	24-2/24-2	—	—
DREQ0	DMA request 0	DMA	1	I/O	13-1/13-2	GPIO1[0]/GTM1_TIN/ GTM2_TIN2	25-1/25-2, 5-59/5-60
DREQ1	DMA request 1	DMA	1	I/O	13-1/13-2	GPIO1[3]/ GTM1_TIN2/ GTM2_TIN1	25-1/25-2, 5-59/5-60
DREQ2	DMA request 3	DMA	1	I/O	13-1/13-2	GPIO1[6]/ GTM1_TIN3/ GTM2_TIN4	25-1/25-2, 5-59/5-60

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
DREQ3	DMA request 3	DMA	1	I/O	13-1/13-2	GPIO1[9]/ GTM1_TIN4/ GTM2_TIN3	25-1/25-2, 5-59/5-60
DACK0	DMA acknowledge 0	DMA	1	I/O	13-1/13-2	GPIO1[1]/ GTM1_TGATE1/ GTM2_TGATE2	25-1/25-2, 5-59/5-60
DACK1	DMA acknowledge 1	DMA	1	I/O	13-1/13-2	GPIO1[4]/ GTM1_TGATE2/ GTM2_TGATE1	25-1/25-2, 5-59/5-60
DACK2	DMA acknowledge 2	DMA	1	I/O	13-1/13-2	GPIO1[7]/ GTM1_TGATE3/ GTM2_TGATE4	25-1/25-2, 5-59/5-60
DACK3	DMA acknowledge 3	DMA	1	I/O	13-1/13-2	GPIO1[10]/ GTM1_TGATE4/ GTM2_TGATE3	25-1/25-2, 5-59/5-60
DDONE0	DMA done 0	DMA	1	I/O	13-1/13-2	GPIO1[2]/ GTM1_TOUT1	25-1/25-2, 5-59/5-60
DDONE1	DMA done 1	DMA	1	I/O	13-1/13-2	GPIO1[5]/ GTM1_TOUT2/ GTM2_TOUT1	25-1/25-2, 5-59/5-60
DDONE2	DMA done 2	DMA	1	I/O	13-1/13-2	GPIO1[8]/ GTM1_TOUT3	25-1/25-2, 5-59/5-60
DDONE3	DMA done 3	DMA	1	I/O	13-1/13-2	GPIO1[11]/ GTM1_TOUT4/ GTM2_TOUT3	25-1/25-2, 5-59/5-60
GPIO1[0]	General-purpose I/O 1 signal 0	GPIO1	1	I/O	25-1/25-2	GTM1_TIN1/ GTM2_TIN2/ DREQ0	5-59/5-60, 13-1/13-2
GPIO1[1]	General-purpose I/O 1 signal 1	GPIO1	1	I/O	25-1/25-2	GTM1_TGATE1/ GTM2_TGATE2/ DACK0	5-59/5-60, 13-1/13-2
GPIO1[2]	General-purpose I/O 1 signal 2	GPIO1	1	I/O	25-1/25-2	GTM1_TOUT1/ DDONE0	5-59/5-60, 13-1/13-2
GPIO1[3]	General-purpose I/O 1 signal 3	GPIO1	1	I/O	25-1/25-2	GTM1_TIN2/ GTM2_TIN1/ DREQ1	5-59/5-60, 13-1/13-2
GPIO1[4]	General-purpose I/O 1 signal 4	GPIO1	1	I/O	25-1/25-2	GTM1_TGATE2/ GTM2_TGATE1/ DACK1	5-59/5-60, 13-1/13-2
GPIO1[5]	General-purpose I/O 1 signal 5	GPIO1	1	I/O	25-1/25-2	GTM1_TOUT2/ GTM2_TOUT1/ DDONE1	5-59/5-60, 13-1/13-2

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GPIO1[6]	General-purpose I/O 1 signal 6	GPIO1	1	I/O	25-1/25-2	GTM1_TIN3/ GTM2_TIN4/ DREQ2	5-59/5-60, 13-1/13-2
GPIO1[7]	General-purpose I/O 1 signal 7	GPIO1	1	I/O	25-1/25-2	GTM1_TGATE3/ GTM2_TGATE4/ DACK2	5-59/5-60, 13-1/13-2
GPIO1[8]	General-purpose I/O 1 signal 8	GPIO1	1	I/O	25-1/25-2	GTM1_TOUT3/ DDONE2	5-59/5-60, 13-1/13-2
GPIO1[9]	General-purpose I/O 1 signal 9	GPIO1	1	I/O	25-1/25-2	GTM1_TIN4/ GTM2_TIN3/ DREQ3	5-59/5-60, 13-1/13-2
GPIO1[10]	General-purpose I/O 1 signal 10	GPIO1	1	I/O	25-1/25-2	GTM1_TGATE4/ GTM2_TGATE3/ DACK3	5-59/5-60, 13-1/13-2
GPIO1[11]	General-purpose I/O 1 signal 11	GPIO1	1	I/O	25-1/25-2	GTM1_TOUT4/ GTM2_TOUT3/ DDONE3	5-59/5-60, 13-1/13-2
GPIO1[12]	General-purpose I/O 1 signal 12	GPIO1	1	I/O	25-1/25-2, 18-2/18-8	TSEC2_TX_EN	18-2/18-8
GPIO1[13:16]	General-purpose I/O 1 signals 13–16	GPIO1	4	I/O	25-1/25-2	TSEC2_RXD[3:0]	18-2/18-8
GPIO1[17:19]	General-purpose I/O 1 signals 17–19	GPIO1	3	I/O	25-1/25-2	TSEC2_TXD[3:1]/ TSEC1_TMR_PP[3:1]	18-2/18-8
GPIO1[20]	General-purpose I/O 1 signal 20	GPIO1	1	I/O	25-1/25-2	TSEC2_TXD[0]	18-2/18-8
GPIO1[21]	General-purpose I/O 1 signal 21	GPIO1	1	I/O	25-1/25-2	TSEC2_COL/ TSEC1_TMR_TRIG1	18-2/18-8
GPIO1[22]	General-purpose I/O 1 signal 22	GPIO1	1	I/O	25-1/25-2	TSEC2_CRS/ TSEC1_TMR_TRIG2	18-2/18-8
GPIO1[23]	General-purpose I/O 1 signal 23	GPIO1	1	I/O	25-1/25-2	TSEC2_RX_DV	18-2/18-8
GPIO1[24]	General-purpose I/O 1 signal 24	GPIO1	1	I/O	25-1/25-2	TSEC2_TX_ER/ TSEC1_TMR_ ALARM1	18-2/18-8
GPIO1[25]	General-purpose I/O 1 signal 25	GPIO1	1	I/O	25-1/25-2	TSEC2_RX_ER	18-2/18-8
GPIO2[0]	General-purpose I/O 2 signal 0	GPIO2	1	I/O	25-1/25-2	USBDR_D0_ ENABLEN	20-2/20-5
GPIO2[1]	General-purpose I/O 2 signal 1	GPIO2	1	I/O	25-1/25-2	USBDR_D1_SER_ TXD	20-2/20-5

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GPIO2[2]	General-purpose I/O 2 signal 2	GPIO2	1	I/O	25-1/25-2	USBDR_D2_VMO_ SE0	20-2/20-5
GPIO2[3]	General-purpose I/O 2 signal 3	GPIO2	1	I/O	25-1/25-2	USBDR_D3_SPEED	20-2/20-5
GPIO2[4]	General-purpose I/O 2 signal 4	GPIO2	1	I/O	25-1/25-2	USBDR_D4_DP	20-2/20-5
GPIO2[5]	General-purpose I/O 2 signal 5	GPIO2	1	I/O	25-1/25-2	USBDR_D5_DM	20-2/20-5
GPIO2[6]	General-purpose I/O 2 signal 6	GPIO2	1	I/O	25-1/25-2	USBDR_D6_SER_ RCV	20-2/20-5
GPIO2[7]	General-purpose I/O 2 signal 7	GPIO2	1	I/O	25-1/25-2	USBDR_D7_ DRVVBUS	20-2/20-5
GPIO2[8]	General-purpose I/O 2 signal 8	GPIO2	1	I/O	25-1/25-2	USBDR_NXT	20-2/20-5
GPIO2[9]	General-purpose I/O 2 signal 9	GPIO2	1	I/O	25-1/25-2	USBDR_DIR_ DPPULLUP	20-2/20-5
GPIO2[10]	General-purpose I/O 2 signal 10	GPIO2	1	I/O	25-1/25-2	USBDR_PWRFAULT/ SD_DAT1	20-2/20-5, 11-1/11-3
GPIO2[11]	General-purpose I/O 2 signal 11	GPIO2	1	I/O	25-1/25-2	USBDR_PCTL0/ SD_DAT2	20-2/20-5, 11-1/11-3
GPIO2[12]	General-purpose I/O 2 signal 12	GPIO2	1	I/O	25-1/25-2	$\overline{\text{IRQ}}$[0]/$\overline{\text{MCP\_IN}}$	8-2/8-5
GPIO2[13:15]	General-purpose I/O 2 signal 13–15	GPIO2	3	I/O	25-1/25-2	$\overline{\text{IRQ}}$[1:3]	8-2/8-5
GPIO2[16]	General-purpose I/O 2 signal 16	GPIO2	1	I/O	25-1/25-2	IRQ[4]/SD_WP	8-2/8-5, 11-1/11-3
GPIO2[17]	General-purpose I/O 2 signal 17	GPIO2	1	I/O	25-1/25-2	IRQ[5]/ USBDR_PWRFAULT	8-2/8-5, 20-2/20-5
GPIO2[18]	General-purpose I/O 2 signal 18	GPIO2	1	I/O	25-1/25-2	$\overline{\text{IRQ}}$[6]/ $\overline{\text{CKSTOP\_OUT}}$	8-2/8-5, 4-3/4-4
GPIO2[19]	General-purpose I/O 2 signal 19	GPIO2	1	I/O	25-1/25-2	$\overline{\text{IRQ}}$[7]/ $\overline{\text{CKSTOP\_IN}}$	8-2/8-5, 4-3/4-4
GPIO2[20]	General-purpose I/O 2 signal 20	GPIO2	1	I/O	25-1/25-2	TSEC1_COL	18-2/18-8
GPIO2[21]	General-purpose I/O 2 signal 21	GPIO2	1	I/O	25-1/25-2	TSEC1_CRS	18-2/18-8
GPIO2[22]	General-purpose I/O 2 signal 22	GPIO2	1	I/O	25-1/25-2	USBDR_PCTL1/ SD_DAT3	20-2/20-5, 11-1/11-3
GPIO2[23]	General-purpose I/O 2 signal 23	GPIO2	1	I/O	25-1/25-2	USBDR_CLK	20-2/20-5

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GPIO2[24]	General-purpose I/O 2 signal 24	GPIO2	1	I/O	25-1/25-2	TSEC2_TX_CLK/ TSEC1_TMR_GCLK	18-2/18-8
GPIO2[25]	General-purpose I/O 2 signal 25	GPIO2	1	I/O	25-1/25-2	TSEC1_RX_ER	18-2/18-8
$\overline{\text{QUIESCE}}$	Quiesce state	PMC	1	O	5-70/5-74	—	—
$\overline{\text{PORESET}}$	Power on reset	System control	1	I	4-1/4-1	—	—
$\overline{\text{HRESET}}$	Hard reset	System control	1	I/O	4-1/4-1	—	—
$\overline{\text{SRESET}}$	Soft reset	System control	1	I/O	4-1/4-1	—	—
CFG_CLKIN_DIV	Configuration clock in division selection	System control	1	I/O	4-1/4-1	EC_MDC	18-2/18-8
CFG_RESET_ SOURCE[3:0]	Reset configuration word source selection	System control	4	I/O	4-1/4-1	TSEC1_TXD[3:0]	18-2/18-8
CFG_LBMUX	Local bus address/data pins usage selection	System control	1	I/O	4-1/4-1	TSEC1_TX_ER	18-2/18-8
$\overline{\text{CKSTOP\_OUT}}$	Clock stop out	Reset and clock	1	I/O	4-3/4-4	GPIO2[18]/ $\overline{\text{IRQ}}$[6]	25-1/25-2, 8-2/8-5
$\overline{\text{CKSTOP\_IN}}$	Clock stop in	Reset and clock	1	I/O	4-3/4-4	GPIO2[19]/ $\overline{\text{IRQ}}$[7]	25-1/25-2, 8-2/8-5
CLKIN	Clock input	Clocks	1	I	4-2/4-3	—	—
PCI_CLK/ PCI_SYNC_IN	PCI clock/ PCI clock sync input	Clocks	1	I	4-2/4-3	—	—
PCI_SYNC_OUT	PCI clock sync output	Clocks	1	O	4-2/4-3	—	—
RTC/PIT_CLOCK	Timer clock/ Real time clock	PIT/RTC	1	I	4-2/4-3	—	—
MDVAL	Memory debug data valid	Debug	1	I/O	—	MECC[5]	9-3/9-5
MDVAL/LDVAL	Memory debug data valid	Debug	1	I/O	—, 10-2/10-5	UART_CTS2	22-2/22-3
MSRCID[0:1]/ LSRCID[0:1]	Memory debug source ID 0–1	Debug	2	O	—, 10-2/10-5	UART_SOUT[1:2]	22-2/22-3
MSRCID[2:3]/ LSRCID[2:3]	Memory debug source ID 2–3	Debug	2	I/O	—, 10-2/10-5	UART_SIN[1:2]	22-2/22-3

Table 3-1. MPC8379E Signal Reference by Functional Block (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MSRCID4/ LSRCID4	Memory debug source ID 4	Debug	1	I/O	—, 10-2/10-5	UART_CTS1	22-2/22-3
MSRCID[0:4]	Memory debug source ID 0–4	Debug	5	I/O	—	MECC[0:4]	9-3/9-5

Table 3-2 lists the signals in alphabetical order.

Table 3-2. MPC8379E Alphabetical Signal Reference

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
CFG_CLKIN_DIV	Configuration clock in division selection	System control	1	I/O	4-1/4-1	EC_MDC	18-2/18-8
CFG_LBMUX	Local bus address/data pins usage selection	System control	1	I/O	4-1/4-1	TSEC1_TX_ER	18-2/18-8
CFG_RESET_ SOURCE[3:0]	Reset configuration word source selection	System control	4	I/O	4-1/4-1	TSEC1_TXD[3:0]	18-2/18-8
CKSTOP_IN	Clock stop in	Reset and clock	1	I/O	4-3/4-4	GPIO2[19]/ IRQ[7]	25-1/25-2, 8-2/8-5
CKSTOP_OUT	Clock stop out	Reset and clock	1	I/O	4-3/4-4	GPIO2[18]/ IRQ[6]	25-1/25-2, 8-2/8-5
CLKIN	Clock input	Clocks	1	I	4-2/4-3	—	—
CPCI_HS_ENUM	CompactPCI hot swap enumerator	PCI	1	O	14-3/14-5	PCI_GNT2	14-3/14-5
CPCI_HS_ES	CompactPCI hot swap ejector switch	PCI	1	I	14-3/14-5	PCI_REQ1	14-3/14-5
CPCI_HS_LED	CompactPCI hot swap LED	PCI	1	O	14-3/14-5	PCI_GNT1	14-3/14-5
DACK0	DMA acknowledge 0	DMA	1	I/O	13-1/13-2	GPIO1[1]/ GTM1_TGATE1/ GTM2_TGATE2	25-1/25-2, 5-59/5-60
DACK1	DMA acknowledge 1	DMA	1	I/O	13-1/13-2	GPIO1[4]/ GTM1_TGATE2/ GTM2_TGATE1	25-1/25-2, 5-59/5-60
DACK2	DMA acknowledge 2	DMA	1	I/O	13-1/13-2	GPIO1[7]/ GTM1_TGATE3/ GTM2_TGATE4	25-1/25-2, 5-59/5-60
DACK3	DMA acknowledge 3	DMA	1	I/O	13-1/13-2	GPIO1[10]/ GTM1_TGATE4/ GTM2_TGATE3	25-1/25-2, 5-59/5-60

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
DDONE0	DMA done 0	DMA	1	I/O	13-1/13-2	GPIO1[2]/ GTM1_TOUT1	25-1/25-2, 5-59/5-60
DDONE1	DMA done 1	DMA	1	I/O	13-1/13-2	GPIO1[5]/ GTM1_TOUT2/ GTM2_TOUT1	25-1/25-2, 5-59/5-60
DDONE2	DMA done 2	DMA	1	I/O	13-1/13-2	GPIO1[8]/ GTM1_TOUT3	25-1/25-2, 5-59/5-60
DDONE3	DMA done 3	DMA	1	I/O	13-1/13-2	GPIO1[11]/ GTM1_TOUT4/ GTM2_TOUT3	25-1/25-2, 5-59/5-60
DREQ0	DMA request 0	DMA	1	I/O	13-1/13-2	GPIO1[0]/GTM1_TIN/ GTM2_TIN2	25-1/25-2, 5-59/5-60
DREQ1	DMA request 1	DMA	1	I/O	13-1/13-2	GPIO1[3]/ GTM1_TIN2/ GTM2_TIN1	25-1/25-2, 5-59/5-60
DREQ2	DMA request 3	DMA	1	I/O	13-1/13-2	GPIO1[6]/ GTM1_TIN3/ GTM2_TIN4	25-1/25-2, 5-59/5-60
DREQ3	DMA request 3	DMA	1	I/O	13-1/13-2	GPIO1[9]/ GTM1_TIN4/ GTM2_TIN3	25-1/25-2, 5-59/5-60
EC_GTX_CLK125	Gigabit reference clock	eTSEC1/ eTSEC2	1	I	18-2/18-8	—	—
EC_MDC	Ethernet management data clock	Ethernet management	1	I/O	18-2/18-8	CFG_CLKIN_DIV	4-1/4-1
EC_MDIO	Ethernet management data in/out	Ethernet management	1	I/O	18-2/18-8	—	—
GPIO1[0]	General-purpose I/O 1 signal 0	GPIO1	1	I/O	25-1/25-2	GTM1_TIN1/ GTM2_TIN2/ DREQ0	5-59/5-60, 13-1/13-2
GPIO1[1]	General-purpose I/O 1 signal 1	GPIO1	1	I/O	25-1/25-2	GTM1_TGATE1/ GTM2_TGATE2/ DACK0	5-59/5-60, 13-1/13-2
GPIO1[10]	General-purpose I/O 1 signal 10	GPIO1	1	I/O	25-1/25-2	GTM1_TGATE4/ GTM2_TGATE3/ DACK3	5-59/5-60, 13-1/13-2
GPIO1[11]	General-purpose I/O 1 signal 11	GPIO1	1	I/O	25-1/25-2	GTM1_TOUT4/ GTM2_TOUT3/ DDONE3	5-59/5-60, 13-1/13-2
GPIO1[12]	General-purpose I/O 1 signal 12	GPIO1	1	I/O	25-1/25-2, 18-2/18-8	TSEC2_TX_EN	18-2/18-8

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GPIO1[13:16]	General-purpose I/O 1 signals 13–16	GPIO1	4	I/O	25-1/25-2	TSEC2_RXD[3:0]	18-2/18-8
GPIO1[17:19]	General-purpose I/O 1 signals 17–19	GPIO1	3	I/O	25-1/25-2	TSEC2_TXD[3:1]/ TSEC1_TMR_PP[3:1]	18-2/18-8
GPIO1[2]	General-purpose I/O 1 signal 2	GPIO1	1	I/O	25-1/25-2	GTM1_TOUT1/ DDONE0	5-59/5-60, 13-1/13-2
GPIO1[20]	General-purpose I/O 1 signal 20	GPIO1	1	I/O	25-1/25-2	TSEC2_TXD[0]	18-2/18-8
GPIO1[21]	General-purpose I/O 1 signal 21	GPIO1	1	I/O	25-1/25-2	TSEC2_COL/ TSEC1_TMR_TRIG1	18-2/18-8
GPIO1[22]	General-purpose I/O 1 signal 22	GPIO1	1	I/O	25-1/25-2	TSEC2_CRS/ TSEC1_TMR_TRIG2	18-2/18-8
GPIO1[23]	General-purpose I/O 1 signal 23	GPIO1	1	I/O	25-1/25-2	TSEC2_RX_DV	18-2/18-8
GPIO1[24]	General-purpose I/O 1 signal 24	GPIO1	1	I/O	25-1/25-2	TSEC2_TX_ER/ TSEC1_TMR_ ALARM1	18-2/18-8
GPIO1[25]	General-purpose I/O 1 signal 25	GPIO1	1	I/O	25-1/25-2	TSEC2_RX_ER	18-2/18-8
GPIO1[3]	General-purpose I/O 1 signal 3	GPIO1	1	I/O	25-1/25-2	GTM1_TIN2/ GTM2_TIN1/ DREQ1	5-59/5-60, 13-1/13-2
GPIO1[4]	General-purpose I/O 1 signal 4	GPIO1	1	I/O	25-1/25-2	GTM1_TGATE2/ GTM2_TGATE1/ DACK1	5-59/5-60, 13-1/13-2
GPIO1[5]	General-purpose I/O 1 signal 5	GPIO1	1	I/O	25-1/25-2	GTM1_TOUT2/ GTM2_TOUT1/ DDONE1	5-59/5-60, 13-1/13-2
GPIO1[6]	General-purpose I/O 1 signal 6	GPIO1	1	I/O	25-1/25-2	GTM1_TIN3/ GTM2_TIN4/ DREQ2	5-59/5-60, 13-1/13-2
GPIO1[7]	General-purpose I/O 1 signal 7	GPIO1	1	I/O	25-1/25-2	GTM1_TGATE3/ GTM2_TGATE4/ DACK2	5-59/5-60, 13-1/13-2
GPIO1[8]	General-purpose I/O 1 signal 8	GPIO1	1	I/O	25-1/25-2	GTM1_TOUT3/ DDONE2	5-59/5-60, 13-1/13-2
GPIO1[9]	General-purpose I/O 1 signal 9	GPIO1	1	I/O	25-1/25-2	GTM1_TIN4/ GTM2_TIN3/ DREQ3	5-59/5-60, 13-1/13-2
GPIO2[0]	General-purpose I/O 2 signal 0	GPIO2	1	I/O	25-1/25-2	USBDR_D0_ ENABLEN	20-2/20-5

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GPIO2[1]	General-purpose I/O 2 signal 1	GPIO2	1	I/O	25-1/25-2	USBDR_D1_SER_ TXD	20-2/20-5
GPIO2[10]	General-purpose I/O 2 signal 10	GPIO2	1	I/O	25-1/25-2	USBDR_PWRFAULT/ SD_DAT1	20-2/20-5, 11-1/11-3
GPIO2[11]	General-purpose I/O 2 signal 11	GPIO2	1	I/O	25-1/25-2	USBDR_PCTL0/ SD_DAT2	20-2/20-5, 11-1/11-3
GPIO2[12]	General-purpose I/O 2 signal 12	GPIO2	1	I/O	25-1/25-2	$\overline{\text{IRQ}}$[0]/$\overline{\text{MCP\_IN}}$	8-2/8-5
GPIO2[13:15]	General-purpose I/O 2 signal 13–15	GPIO2	3	I/O	25-1/25-2	$\overline{\text{IRQ}}$[1:3]	8-2/8-5
GPIO2[16]	General-purpose I/O 2 signal 16	GPIO2	1	I/O	25-1/25-2	IRQ[4]/SD_WP	8-2/8-5, 11-1/11-3
GPIO2[17]	General-purpose I/O 2 signal 17	GPIO2	1	I/O	25-1/25-2	IRQ[5]/ USBDR_PWRFAULT	8-2/8-5, 20-2/20-5
GPIO2[18]	General-purpose I/O 2 signal 18	GPIO2	1	I/O	25-1/25-2	$\overline{\text{IRQ}}$[6]/ $\overline{\text{CKSTOP\_OUT}}$	8-2/8-5, 4-3/4-4
GPIO2[19]	General-purpose I/O 2 signal 19	GPIO2	1	I/O	25-1/25-2	$\overline{\text{IRQ}}$[7]/ $\overline{\text{CKSTOP\_IN}}$	8-2/8-5, 4-3/4-4
GPIO2[2]	General-purpose I/O 2 signal 2	GPIO2	1	I/O	25-1/25-2	USBDR_D2_VMO_ SE0	20-2/20-5
GPIO2[20]	General-purpose I/O 2 signal 20	GPIO2	1	I/O	25-1/25-2	TSEC1_COL	18-2/18-8
GPIO2[21]	General-purpose I/O 2 signal 21	GPIO2	1	I/O	25-1/25-2	TSEC1_CRS	18-2/18-8
GPIO2[22]	General-purpose I/O 2 signal 22	GPIO2	1	I/O	25-1/25-2	USBDR_PCTL1/ SD_DAT3	20-2/20-5, 11-1/11-3
GPIO2[23]	General-purpose I/O 2 signal 23	GPIO2	1	I/O	25-1/25-2	USBDR_CLK	20-2/20-5
GPIO2[24]	General-purpose I/O 2 signal 24	GPIO2	1	I/O	25-1/25-2	TSEC2_TX_CLK/ TSEC1_TMR_GCLK	18-2/18-8
GPIO2[25]	General-purpose I/O 2 signal 25	GPIO2	1	I/O	25-1/25-2	TSEC1_RX_ER	18-2/18-8
GPIO2[3]	General-purpose I/O 2 signal 3	GPIO2	1	I/O	25-1/25-2	USBDR_D3_SPEED	20-2/20-5
GPIO2[4]	General-purpose I/O 2 signal 4	GPIO2	1	I/O	25-1/25-2	USBDR_D4_DP	20-2/20-5
GPIO2[5]	General-purpose I/O 2 signal 5	GPIO2	1	I/O	25-1/25-2	USBDR_D5_DM	20-2/20-5
GPIO2[6]	General-purpose I/O 2 signal 6	GPIO2	1	I/O	25-1/25-2	USBDR_D6_SER_ RCV	20-2/20-5

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
GPIO2[7]	General-purpose I/O 2 signal 7	GPIO2	1	I/O	25-1/25-2	USBDR_D7_ DRVVBUS	20-2/20-5
GPIO2[8]	General-purpose I/O 2 signal 8	GPIO2	1	I/O	25-1/25-2	USBDR_NXT	20-2/20-5
GPIO2[9]	General-purpose I/O 2 signal 9	GPIO2	1	I/O	25-1/25-2	USBDR_DIR_ DPPULLUP	20-2/20-5
GTM1_TGATE1/ GTM2_TGATE2	Timer gate 1/2	Global Timers	1	I/O	5-59/5-60	GPIO1[1]/ DACK0	25-1/25-2, 13-1/13-2
GTM1_TGATE2/ GTM2_TGATE1	Timer gate 2/1	Global Timers	1	I/O	5-59/5-60	GPIO1[4]/ DACK1	25-1/25-2, 13-1/13-2
GTM1_TGATE3/ GTM2_TGATE4	Timer gate 3/4	Global Timers	1	I/O	5-59/5-60	GPIO1[7]/ DACK2	25-1/25-2, 13-1/13-2
GTM1_TGATE4/ GTM2_TGATE3	Timer gate 4/3	Global Timers	1	I/O	5-59/5-60	GPIO1[10]/ DACK3	25-1/25-2 ,13-1/13-2
GTM1_TIN1/ GTM2_TIN2	Timer in 1/2	Global Timers	1	I/O	5-59/5-60	GPIO1[0]/ DREQ0	25-1/25-2, 13-1/13-2
GTM1_TIN2/ GTM2_TIN1	Timer in 2/1	Global Timers	1	I/O	5-59/5-60	GPIO1[3]/ DREQ1	25-1/25-2, 13-1/13-2
GTM1_TIN3/ GTM2_TIN4	Timer in 3/4	Global Timers	1	I/O	5-59/5-60	GPIO1[6]/ DREQ2	25-1/25-2, 13-1/13-2
GTM1_TIN4/ GTM2_TIN3	Timer in 4/3	Global Timers	1	I/O	5-59/5-60	GPIO1[9]/ DREQ3	25-1/25-2, 13-1/13-2
GTM1_TOUT1	Timer out 1	Global Timers	1	I/O	5-59/5-60	GPIO1[2]/ DDONE0	25-1/25-2, 13-1/13-2
GTM1_TOUT2/ GTM2_TOUT1	Timer out 2/1	Global Timers	1	I/O	5-59/5-60	GPIO1[5]/ DDONE1	25-1/25-2, 13-1/13-2
GTM1_TOUT3	Timer out 3	Global Timers	1	I/O	5-59/5-60	GPIO1[8]/ DDONE2	25-1/25-2, 13-1/13-2
GTM1_TOUT4/ GTM2_TOUT3	Timer out 4/3	Global Timers	1	I/O	5-59/5-60	GPIO1[11]/ DDONE3	25-1/25-2, 13-1/13-2
HRESET	Hard reset	System control	1	I/O	4-1/4-1	—	—
IIC1_SCL	I^2C serial clock	I^2C1	1	I/O	21-1/21-3	—	—
IIC1_SDA	I^2C serial data	I^2C1	1	I/O	21-1/21-3	—	—
IIC2_SCL	I^2C serial clock	I^2C2	1	I/O	21-1/21-3	—	—
IIC2_SDA	I^2C serial data	I^2C2	1	I/O	21-1/21-3	—	—
IRQ[0]/MCP_IN	External interrupt 0	IPIC	1	I/O	8-2/8-5	GPIO2[12]	25-1/25-2
IRQ[1:3]	External interrupt 1–3	IPIC	3	I/O	8-2/8-5	GPIO2[13:15]	25-1/25-2

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
IRQ[4]	External interrupt 4	IPIC	1	I/O	8-2/8-5	GPIO2[16]/SD_WP	25-1/25-2/ 11-1/11-3
IRQ[5]	External interrupt 5	IPIC	1	I/O	8-2/8-5	GPIO2[17]/ USBDR_PWRFAULT	25-1/25-2
IRQ[6]	External interrupt 6	IPIC	1	I/O	8-2/8-5	GPIO2[18]/ CKSTOP_OUT	25-1/25-2, 4-3/4-4
IRQ[7]	External interrupt 7	IPIC	1	I/O	8-2/8-5	GPIO2[19]/ CKSTOP_IN	25-1/25-2, 4-3/4-4
IRQ_OUT	Interrupt out	IPIC	1	O	8-2/8-5	PCI_INTA	14-3/14-5
L1_SD_REF_CLK	Reference clock for link 1	SerDes 1	1	I	19-1/19-3	—	—
L1_REF_CLK	Reference clock for link 1 (complement)	SerDes 1	1	I	19-1/19-3	—	—
L1_SD_RXA	Link 1, lane A, receive data	SerDes 1	1	I	19-1/19-3	—	—
L1_SD_RXA	Link 1, lane A, receive data (complement)	SerDes 1	1	I	19-1/19-3	—	—
L1_SD_RXE	Link 1, lane E, receive data	SerDes 1	1	I	19-1/19-3	—	—
L1_SD_RXE	Link 1, lane E, receive data (complement)	SerDes 1	1	I	19-1/19-3	—	—
L1_SD_TXA	Link 1, lane A, transmit data	SerDes 1	1	O	19-1/19-3	—	—
L1_SD_TXA	Link 1, lane A, transmit data (complement)	SerDes 1	1	O	19-1/19-3	—	—
L1_SD_TXE	Link 1, lane E, transmit data	SerDes 1	1	O	19-1/19-3	—	—
L1_SD_TXE	Link 1, lane E, transmit data (complement)	SerDes 1	1	O	19-1/19-3	—	—
L2_SD_REF_CLK	Reference clock for link 2	SerDes 2	1	I	19-1/19-3	—	—
L2_SD_REF_CLK	Reference clock for link 2 (complement)	SerDes 2	1	I	19-1/19-3	—	—
L2_SD_RXA	Link 2, lane A, receive data	SerDes 1	1	I	19-1/19-3	—	—

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
L2_SD_RXA	Link 2, lane A, receive data (complement)	SerDes 1	1	I	19-1/19-3	—	—
L2_SD_RXE	Link 2, lane E, receive data	SerDes 1	1	I	19-1/19-3	—	—
L2_SD_RXE	Link 2, lane E, receive data (complement)	SerDes 1	1	I	19-1/19-3	—	—
L2_SD_TXA	Link 2, lane A, transmit data	SerDes 1	1	O	19-1/19-3	—	—
L2_SD_TXA	Link 2, lane A, transmit data (complement)	SerDes 1	1	O	19-1/19-3	—	—
L2_SD_TXE	Link 2, lane E, transmit data	SerDes 1	1	O	19-1/19-3	—	—
L2_SD_TXE	Link 2, lane E, transmit data (complement)	SerDes 1	1	O	19-1/19-3	—	—
LA[11:26]	LBC port address 11–26	eLBC	16	I/O	10-2/10-5	LAD[16:31]	10-2/10-5
LA[27:31]	LBC port address	eLBC	5	O	10-2/10-5	—	—
LA10	LBC port address 10	eLBC	1	O	10-2/10-5	LALE	10-2/10-5
LA9	LBC port address 9	eLBC	1	O	10-2/10-5	LGPL5	10-2/10-5
LAD[0:15]	LBC address/data	eLBC	16	I/O	10-2/10-5	—	—
LAD[16:31]	LBC address/data	eLBC	16	I/O	10-2/10-5	LA[11:26]	10-2/10-5
LALE	LBC address latch enable	eLBC	1	O	10-2/10-5	LA10	10-2/10-5
LBCTL	LBC data buffer control	eLBC	1	O	10-2/10-5	—	—
LBS[0:3]	Byte lane select	eLBC	4	O	10-2/10-5	LWE[0:3]/LFWE[0:3]	10-2/10-5
LCLK[0:2]	LBC clocks 0–2	eLBC	3	O	10-2/10-5	—	—
LCS[0:3]	LBC chip select 0–3	eLBC	4	O	10-2/10-5	—	—
LCS[4:5]	LBC chip select 4–5	eLBC	2	I/O	10-2/10-5	LDP[0:1]	10-2/10-5
LCS[6:7]	LBC chip select 6–7	eLBC	2	I/O	10-2/10-5	LDP[2:3]/LA[7:8]	10-2/10-5
LDP[0:1]	LBC data parity 0–1	eLBC	2	I/O	10-2/10-5	LCS[4:5]	10-2/10-5
LDP[2:3]	LBC data parity 2–3	eLBC	2	I/O	10-2/10-5	LCS[6:7]/LA[7:8]	10-2/10-5

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
LGPL0/LFCLE	LBC UPM general purpose line 0/Flash command latch enable	eLBC	1	O	10-2/10-5	—	—
LGPL1/LFALE	LBC GP line 1/Flash address latch enable	eLBC	1	O	10-2/10-5	—	—
LGPL2/LOE/LFRE	LBC output enable/GP line 2/FCM read enable	eLBC	1	O	10-2/10-5	—	—
LGPL3/LFWP	LBC GP line 3/Flash write project	eLBC	1	O	10-2/10-5	—	—
LGPL4/LGTA/ LUPWAIT/LPBSE/ LFRB	LBC GP line 4/GPCM terminate access/ UPM wait/parity byte select/Flash read/busy, open-drain shared pin	eLBC	1	I/O	10-2/10-5	—	—
LGPL5	LBC GP line 5	eLBC	1	O	10-2/10-5	LA9	10-2/10-5
LSYNC_IN	LBC DLL synchronization input	eLBC	1	I	10-2/10-5	—	—
LSYNC_OUT	LBC DLL synchronization output	eLBC	1	O	10-2/10-5	—	—
LWE[0:3]/LFWE[0:3]	LBC write enable/ byte select/FCM write enable	eLBC	4	O	10-2/10-5	LBS[0:3]	10-2/10-5
M66EN	66-MHz system configuration	PCI	1	I	14-3/14-5	—	—
MA[14:0]	DDR address	DDR	15	O	9-3/9-5	—	—
MBA[2:0]	DDR bank select	DDR	3	O	9-3/9-5	—	—
MCAS	DDR column address strobe	DDR	1	O	9-3/9-5	—	—
MCK[0:5], MCK[0:5]	DDR differential clocks (3 pairs/DIMM)	DDR	12	O	9-4/9-8	—	—
MCKE[0:1]	DDR clock enable	DDR	2	O	9-4/9-8	—	—
MCP_OUT	Machine check interrupt output	IPIC	1	O	8-2/8-5	—	—

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
MCS[0:3]	DDR chip select (2/DIMM)	DDR	4	O	9-3/9-5	—	—
MDIC[0:1]	Driver impedance calibration	DDR	2	I/O	9-3/9-5	—	—
MDM[0:7]	DDR data mask	DDR	8	O	9-3/9-5	—	—
MDM[8]	DDR ECC data mask	DDR	1	O	9-3/9-5	—	—
MDQ[0:63]	DDR data	DDR	64	I/O	9-3/9-5	—	—
MDQS[0:7]	DDR data strobe	DDR	8	I/O	9-3/9-5	—	—
MDQS[8]	DDR ECC data strobe	DDR	1	I/O	9-3/9-5	—	—
MDVAL	Memory debug data valid	Debug	1	I/O	—	MECC[5]	9-3/9-5
MDVAL/LDVAL	Memory debug data valid	Debug	1	I/O	—, 10-2/10-5	UART_CTS2	22-2/22-3
MECC[0:4]	DDR error correcting code	DDR	5	I/O	9-3/9-5	MSRCID[0:4]	—
MECC[5]	DDR error correcting code	DDR	1	I/O	9-3/9-5	MDVAL	—
MECC[6:7]	DDR error correcting code	DDR	2	I/O	9-3/9-5	—	—
MODT[0:3]	DRAM on-die termination	DDR	4	O	9-3/9-5	—	—
MRAS	DDR row address strobe	DDR	1	O	9-3/9-5	—	—
MSRCID[0:1]/ LSRCID[0:1]	Memory debug source ID 0–1	Debug	2	O	—, 10-2/10-5	UART_SOUT[1:2]	22-2/22-3
MSRCID[0:4]	Memory debug source ID 0–4	Debug	5	I/O	—	MECC[0:4]	9-3/9-5
MSRCID[2:3]/ LSRCID[2:3]	Memory debug source ID 2–3	Debug	2	I/O	—, 10-2/10-5	UART_SIN[1:2]	22-2/22-3
MSRCID4/ LSRCID4	Memory debug source ID 4	Debug	1	I/O	—, 10-2/10-5	UART_CTS1	22-2/22-3
MWE	DDR write enable	DDR	1	O	9-3/9-5	—	—
PCI_AD[31:0]	PCI address/data	PCI	32	I/O	14-3/14-5	—	—
PCI_C/BE[3:0]	PCI command/byte enable	PCI	4	I/O	14-3/14-5	—	—
PCI_CLK/ PCI_SYNC_IN	PCI clock/PCI clock sync input	Clocks	1	I	4-2/4-3	—	—

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
PCI_CLK[0:4]	PCI clock 0–4	PCI	5	O	4-2/4-3	—	—
PCI_DEVSEL	PCI device select	PCI	1	I/O	14-3/14-5	—	—
PCI_FRAME	PCI frame	PCI	1	I/O	14-3/14-5	—	—
PCI_GNT0	PCI grant 0	PCI	1	I/O	14-3/14-5	—	—
PCI_GNT1	PCI grant 1	PCI	1	O	14-3/14-5	CPCI_HS_LED	14-3/14-5
PCI_GNT2	PCI grant 2	PCI	1	O	14-3/14-5	CPCI_HS_ENUM	14-3/14-5
PCI_GNT3	PCI grant 3	PCI	1	O	14-3/14-5	PCI_PME	14-3/14-5
PCI_GNT4	PCI grant 4	PCI	1	O	14-3/14-5	—	—
PCI_IDSEL	PCI initial device select	PCI	1	I	14-3/14-5	—	—
PCI_INTA	PCI interrupt output	PCI	1	O	14-3/14-5	IRQ_OUT	8-2/8-5
PCI_IRDY	PCI initiator ready	PCI	1	I/O	14-3/14-5	—	—
PCI_PAR	PCI parity	PCI	1	I/O	14-3/14-5	—	—
PCI_PERR	PCI parity error	PCI	1	I/O	14-3/14-5	—	—
PCI_PME	PCI PME assertion request	PCI	1	O	14-3/14-5	PCI_GNT3	14-3/14-5
PCI_REQ[2:4]	PCI request 2–4	PCI	3	I	14-3/14-5	—	—
PCI_REQ0	PCI request 0	PCI	1	I/O	14-3/14-5	—	—
PCI_REQ1	PCI request 1	PCI	1	I	14-3/14-5	CPCI_HS_ES	14-3/14-5
PCI_RESET_OUT	PCI reset	PCI	1	O	14-3/14-5	—	—
PCI_SERR	PCI system error	PCI	1	I/O	14-3/14-5	—	—
PCI_STOP	PCI stop	PCI	1	I/O	14-3/14-5	—	—
PCI_SYNC_OUT	PCI clock sync output	Clocks	1	O	4-2/4-3	—	—
PCI_TRDY	PCI target ready	PCI	1	I/O	14-3/14-5	—	—
PORESET	Power on reset	System control	1	I	4-1/4-1	—	—
QUIESCE	Quiesce state	PMC	1	O	5-70/5-74	—	—
RTC/PIT_CLOCK	Timer clock/ Real time clock	PIT/RTC	1	I	4-2/4-3	—	—
SD_CD	Card detection	eSDHC	1	I	11-1/11-3	SPISEL	23-2/23-7
SD_CLK	Clock for MMC/SD/SDIO card	eSDHC	1	I/O	11-1/11-3	SPICLK	23-2/23-7
SD_CMD	CMD line connect to card	eSDHC	1	I/O	11-1/11-3	SPIMOSI	23-2/23-7

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
SD_DAT0	Dat0 line	eSDHC	1	I/O	11-1/11-3	SPIMISO	23-2/23-7
SD_DAT1	Dat1 line	eSDHC	1	I/O	11-1/11-3	GPIO2[10]/ USBDR_PWRFAULT	25-1/25-2, 20-2/20-5
SD_DAT2	Dat2 line	eSDHC	1	I/O	11-1/11-3	GPIO2[11]/ USBDR_PCTL0	25-1/25-2, 20-2/20-5
SD_DAT3	Dat3 line	eSDHC	1	I/O	11-1/11-3	GPIO2[22]/ USBDR_PCTL1	25-1/25-2, 20-2/20-5
SD_WP	Card write protect detect	eSDHC	1	I/O	11-1/11-3	GPIO2[16]/ IRQ[4]	25-1/25-2, 8-2/8-5
SPICLK	SPI clock	SPI	1	I/O	23-2/23-7	SD_CLK	11-1/11-3
SPIMISO	SPI master-in slave-out	SPI	1	I/O	23-2/23-7	SD_DAT0	11-1/11-3
SPIMOSI	SPI master-out slave-in	SPI	1	I/O	23-2/23-7	SD_CMD	11-1/11-3
SPISEL	SPI slave select	SPI	1	I	23-2/23-7	SD_CD	11-1/11-3
$\overline{\text{SRESET}}$	Soft reset	System control	1	I/O	4-1/4-1	—	—
TCK	Test clock	JTAG	1	I	24-2/24-2	—	—
TDI	Test data in	JTAG	1	I	24-2/24-2	—	—
TDO	Test data out	JTAG	1	O	24-2/24-2	—	—
TMS	Test mode select	JTAG	1	I	24-2/24-2	—	—
$\overline{\text{TRST}}$	Test reset	JTAG	1	I	24-2/24-2	—	—
TSEC_TMR_ ALARM_OUT1	1588 timer alarm-out 1	eTSEC1/ eTSEC2	1	I/O	18-2/18-8	GPIO1[24]/ TSEC2_TX_ER	25-1/25-2, 18-2/18-8
TSEC_TMR_ ALARM_OUT2	1588 timer alarm-out 2	eTSEC1/ eTSEC2	1	I/O	18-2/18-8,	GPIO1[12]/ TSEC2_TX_EN	25-1/25-2, 18-2/18-8
TSEC_TMR_CLK_IN	1588 clock-in	eTSEC1/ eTSEC2	1	I	18-2/18-8	TSEC2_RX_CLK	18-2/18-8
TSEC_TMR_ CLK_OUT	1588 clock-out	eTSEC1/ eTSEC2	1	I/O	18-2/18-8	GPIO2[24]/ TSEC2_TX_CLK	25-1/25-2, 18-2/18-8
TSEC_TMR_ PULSE_OUT[3:1]	1588 pulse-out [3:1]	eTSEC1/ eTSEC2	3	I/O	18-2/18-8	GPIO1[17:19]/ TSEC2_TXD[3:1]	25-1/25-2, 18-2/18-8
TSEC_TMR_ TRIG_IN1	1588 trigger-in 1	eTSEC1/ eTSEC2	1	I/O	18-2/18-8	GPIO1[21]/ TSEC2_COL	25-1/25-2, 18-2/18-8
TSEC_TMR_ TRIG_IN2	1588 trigger-in 2	eTSEC1/ eTSEC2	1	I/O	18-2/18-8	GPIO1[22]/ TSEC2_CRS	25-1/25-2, 18-2/18-8
TSEC1_COL	eTSEC1 collision detect	eTSEC1	1	I/O	18-2/18-8	GPIO2[20]	25-1/25-2

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC1_CRS	eTSEC11 carrier sense	eTSEC1	1	I/O	18-2/18-8	GPIO2[21]	25-1/25-2
TSEC1_GTX_CLK	eTSEC1 transmit clock out	eTSEC1	1	O	18-2/18-8	—	—
TSEC1_RX_CLK	eTSEC1 receive clock	eTSEC1	1	I	18-2/18-8	—	—
TSEC1_RX_DV	eTSEC1 receive data valid	eTSEC1	1	I	18-2/18-8	—	—
TSEC1_RX_ER	eTSEC1 receiver error	eTSEC1	1	I/O	18-2/18-8	GPIO2[25]	25-1/25-2
TSEC1_RXD[3:0]	eTSEC1 receive data 3–0	eTSEC1	4	I	18-2/18-8	—	—
TSEC1_TX_CLK	eTSEC1 transmit clock in	eTSEC1	1	I	18-2/18-8	—	—
TSEC1_TX_EN	eTSEC1 transmit enable	eTSEC1	1	O	18-2/18-8	—	—
TSEC1_TX_ER	eTSEC1 transmit error	eTSEC1	1	I/O	18-2/18-8	CFG_LBMUX	4-1/4-1
TSEC1_TXD[3:0]	eTSEC1 transmit data 3–0	eTSEC1	4	I/O	18-2/18-8	CFG_RESET_ SOURCE[3:0]	4-1/4-1
TSEC2_COL	eTSEC2 collision detect	eTSEC2	1	I/O	18-2/18-8	GPIO1[21]/ TSEC1_TMR_TRIG1	25-1/25-2, 18-2/18-8
TSEC2_CRS	eTSEC2 carrier sense	eTSEC2	1	I/O	18-2/18-8	GPIO1[22]/ TSEC1_TMR_TRIG2	25-1/25-2, 18-2/18-8
TSEC2_GTX_CLK	eTSEC2 transmit clock out	eTSEC2	1	O	18-2/18-8	—	—
TSEC2_RX_CLK	eTSEC2 receive clock	eTSEC2	1	I	18-2/18-8	TSEC1_TMR_CLK	18-2/18-8
TSEC2_RX_DV	eTSEC2 receive data valid	eTSEC2	1	I/O	18-2/18-8	GPIO1[23]	25-1/25-2
TSEC2_RX_ER	eTSEC2 receiver error	eTSEC2	1	I/O	18-2/18-8	GPIO1[25]	25-1/25-2
TSEC2_RXD[3:0]	eTSEC2 receive data 3–0	eTSEC2	4	I/O	18-2/18-8	GPIO1[13:16]	25-1/25-2
TSEC2_TX_CLK	eTSEC2 transmit clock in	eTSEC2	1	I/O	18-2/18-8	GPIO2[24]/ TSEC1_TMR_GCLK	25-1/25-2, 18-2/18-8
TSEC2_TX_EN	eTSEC2 transmit enable	eTSEC2	1	I/O	18-2/18-8	GPIO1[12]/ TSEC1_TMR_ ALARM2	25-1/25-2

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
TSEC2_TX_ER	eTSEC2 transmit error	eTSEC2	1	I/O	18-2/18-8	GPIO1[24]/ TSEC1_TMR_ ALARM1	25-1/25-2, 18-2/18-8
TSEC2_TXD[0]	eTSEC2 transmit data 0	eTSEC2	1	I/O	18-2/18-8	GPIO1[20]	25-1/25-2
TSEC2_TXD[3:1]	eTSEC2 transmit data 3–1	eTSEC2	3	I/O	18-2/18-8	GPIO1[17:19]/ TSEC1_TMR_PP[3:1]	25-1/25-2, 18-2/18-8
UART_CTS1	DUART clear to send	DUART	1	I/O	22-2/22-3	MSRCID4/ LSRCID4	—, 10-2/10-5
UART_CTS2	DUART clear to send	DUART	1	I/O	22-2/22-3	MDVAL/LDVAL	—, 10-2/10-5
$\overline{\text{UART\_RTS}}$[1:2]	DUART ready to send	DUART	2	O	22-2/22-3	—	—
UART_SIN[1:2]	DUART serial data in	DUART	2	I/O	22-2/22-3	MSRCID[2:3]/ LSRCID[2:3]	—, 10-2/10-5
UART_SOUT[1:2]	DUART serial data out	DUART	2	O	22-2/22-3	MSRCID[0:1]/ LSRCID[0:1]	—, 10-2/10-5
USBDR_CLK	USB host 0 interface clock	USB	1	I/O	20-2/20-5	GPIO2[23]	25-1/25-2
USBDR_D0_ ENABLEN	USB host 0 data bit 0	USB	1	I/O	20-2/20-5	GPIO2[0]	25-1/25-2
USBDR_D1_SER_ TXD	USB host 0 data bit 1	USB	1	I/O	20-2/20-5	GPIO2[1]	25-1/25-2
USBDR_D2_VMO_ SE0	USB host 0 data bit 2	USB	1	I/O	20-2/20-5	GPIO2[2]	25-1/25-2
USBDR_D3_SPEED	USB host 0 data bit 3	USB	1	I/O	20-2/20-5	GPIO2[3]	25-1/25-2
USBDR_D4_DP	USB host 0 data bit 4	USB	1	I/O	20-2/20-5	GPIO2[4]	25-1/25-2
USBDR_D5_DM	USB host 0 data bit 5	USB	1	I/O	20-2/20-5	GPIO2[5]	25-1/25-2
USBDR_D6_SER_ RCV	USB host 0 data bit 6	USB	1	I/O	20-2/20-5	GPIO2[6]	25-1/25-2
USBDR_D7_ DRVVBUS	USB host 0 data bit 7	USB	1	I/O	20-2/20-5	GPIO2[7]	25-1/25-2
USBDR_DIR_ DPPULLUP	USB host 0 data bus direction	USB	1	I/O	20-2/20-5	GPIO2[9]	25-1/25-2
USBDR_NXT	USB host 0 next data	USB	1	I/O	20-2/20-5	GPIO2[8]	25-1/25-2

Table 3-2. MPC8379E Alphabetical Signal Reference (continued)

Name	Description	Functional Block	No. of Signals	I/O	Table/ Page	Alternate Function(s)	Table/ Page
USBDR_PCTL0	USB host 0 port control 0	USB	1	I/O	20-2/20-5	GPIO2[11]/SD_DAT2	25-1/25-2, 11-1/11-3
USBDR_PCTL1	USB host 0 port control 1	USB	1	I/O	20-2/20-5	GPIO2[22]/SD_DAT3	25-1/25-2, 11-1/11-3
USBDR_PWRFAULT	USB host 0 power fault	USB	1	I/O	20-2/20-5	GPIO2[10]/SD_DAT1	25-1/25-2, 11-1/11-3
USBDR_PWRFAULT	USB VBus power fault	USB	1	I/O	20-2/20-5	GPIO2[17]/ IRQ[5]	25-1/25-2, 8-2/8-5
USBDR_STP _SUSPEND	USB host 0 data stop/suspend	USB	1	I/O	20-2/20-5	—	—

3.2 Configuration Signals Sampled at Reset

The signals that serve alternate functions as configuration input signals during system reset are summarized in Table 3-3. The detailed interpretation of their voltage levels during reset is described in Chapter 4, "Reset, Clocking, and Initialization."

Table 3-3. Reset Configuration Signals

Functional Interface	Functional Signal Name	Reset Configuration Name
eTSEC1	TSEC1_TXD[3:0]	CFG_RESET_SOURCE[3:0]
	TSEC_MDC	CFG_CLKIN_DIV
	TSEC1_TX_ER	CFG_LBMUX

3.3 Output Signal States During Reset

When a system reset is recognized ($\overline{\text{PORESET}}$ or $\overline{\text{HRESET}}$ are asserted), the device aborts all current internal and external transactions and releases all bidirectional I/O signals to a high-impedance state. See Chapter 4, "Reset, Clocking, and Initialization," for a complete description of the reset functionality.

During reset, the device ignores most input signals (except for the reset configuration signals) and drives most of the output-only signals to an inactive state. Table 3-4 shows the states of the output-only signals.

Table 3-4. Output Signal States During System Reset

Interface	Signal	State During Reset
MDM[0:8]	DDR data mask	All 'Z'
MBA[2]	DDR bank select	All 'Z'
MA[14:0]	DDR address	All 'Z'
$\overline{\text{MWE}}$	DDR write enable	'Z'
$\overline{\text{MRAS}}$	DDR row address strobe	'Z

Table 3-4. Output Signal States During System Reset (continued)

Interface	Signal	State During Reset
MCAS	DDR column address strobe	'Z'
MCS[0:3]	DDR chip select (2/DIMM)	All 'Z'
MCKE[0:1]	DDR clock enable	All '0'
MCK[0:5]	DDR differential clocks	All '0'
MCK[0:5]	DDR differential clocks	All '1'
MODT[0:3]	DRAM on-die termination	All 'Z'
PCI_INTA/ IRQ_OUT	PCI interrupt output	'Z'
PCI_RESET_OUT	PCI reset output	'0'
PCI_GNT[1:4]	PCI grant 1–4	All 'Z'
UART_SOUT[1:2]	DUART serial data out	All 'Z'
UART_RTS[1:2]	DUART ready to send	'11'
TSEC1_GTX_CLK	eTSEC1 transmit clock out	'0'
TSEC1_TX_EN	eTSEC1 transmit enable	'0'
TSEC2_GTX_CLK	eTSEC2 transmit clock out	'Z'
LA[27:31]	eLBC port address	Active—used to load reset configuration word
LCS[0]	eLBC chip select 0	Active—used to load reset configuration word
LCS[1:3]	eLBC chip select 1–3	All '1'
LWE[0:3]/ LBS[0:3]/LFWE[0:1]	eLBC write enable/byte select/FCM write enable	All '1'
LBCTL	eLBC data buffer control	Active—used to load reset configuration word
LALE	eLBC address latch enable	Active—used to load reset configuration word
LGPL0	GP line 0	Active—used to load reset configuration word
LGPL1	GP line 1	Active—used to load reset configuration word
LOE/LGPL2/LFRE	eLBC output enable/GP line 2/FCM read enable	Active—used to load reset configuration word
LGPL3	GP line 3	Active—used to load reset configuration word
LGPL5	GP line 5	Active—used to load reset configuration word
LCLK[0:2]	eLBC clock	All '0'

Table 3-4. Output Signal States During System Reset (continued)

Interface	Signal	State During Reset
LSYNC_OUT	eLBC clock synchronization output	'0'
MCP_OUT	Machine check interrupt output	'Z'
TDO	Test data out	'Z'
QUIESCE	Quiesce state	'1'
PCI_CLK[0:4]	PCI clock 0–4	All '0'
PCI_SYNC_OUT	PCI sync output	Active clock
USBDR_STP_SUSPEND[1]	USB host 0 data stop/suspend	Z

[1] This pin should be pulled high during a hard reset for proper functionality of the device since it has a weak internal pull up. No external pull-down resistors are allowed to be mounted on this net.

Table 3-5 provides the list of registers that control the device's signal multiplexing.

Table 3-5. Signals for Multiplexing

Signal Group	Multiplexing is Controlled By	Table/Page
PCI/CPCI and PCI/PME	RCWH[PCIARB]	4.3.2.2/4-16
LA[7:26]	CFG_LBMUX state during reset	4.3.1.3/4-11
SerDes1/SerDes2	SRDSCR4 of SerDes1/SerDes2	19.3.5/19-16
All others	SICRL/SICRH	5.3.2.5/5-24/5.3.2.6/5-28

Chapter 4
Reset, Clocking, and Initialization

The reset, clocking, and control signals offer many options for operating the device. Various modes and features can be configured during hard reset or power-on reset. Most configurable features are loaded to the device through a reset configuration word, and a few device signals are used as reset configuration inputs during the reset sequence.

4.1 External Signals

The following sections describe the reset and clock signals in detail.

4.1.1 Reset Signals

Table 4-1 describes the reset signals of the device. Section 4.3.2, “Reset Configuration Words,” describes the signals that also function as reset configuration signals.

Table 4-1. System Control Signals

Signal	I/O	Description	
$\overline{\text{PORESET}}$	I	Power-on reset. Initiates the power-on reset flow that resets the device and configures various attributes of the device, including its clock modes.	
		State Meaning	Asserted—An external agent has triggered a power-on reset sequence. Negated—No power-on reset.
		Timing	See the hardware specifications for timing information.
		Reset State	Always input.
$\overline{\text{HRESET}}$	I/O	Hard reset. Causes the device to abort all current internal and external transactions and set most registers to their default values. $\overline{\text{HRESET}}$ can be asserted completely asynchronously with respect to all other signals. The device can detect an external assertion of $\overline{\text{HRESET}}$ while the device is not asserting hard reset. During $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$ is asserted. $\overline{\text{HRESET}}$ is an open-drain signal.	
		State Meaning	Asserted—An external agent or internal hardware has triggered a hard reset. The internal hardware drives $\overline{\text{HRESET}}$ until the sequence completes. Negated—No hard reset.
		Timing	Assertion—Occur at any time, asynchronously to any clock. Negation—Must be asserted for at least 32 CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) cycles.
		Requirements	An open-drain signal. An external pull-up is required.
		Reset State	Output, driven low during power-on and hard reset flows. High impedance after reset flow completes.

Table 4-1. System Control Signals (continued)

Signal	I/O	Description	
$\overline{\text{SRESET}}$	I/O	Soft reset. Causes the device to abort all current internal transactions, set most registers to their default values, and cause the core to enter its reset state. The I/O signal functionality and direction as well as the memory controller operation are unaffected by $\overline{\text{SRESET}}$. $\overline{\text{SRESET}}$ can be asserted completely asynchronously with respect to all other signals. The device can detect an external assertion of $\overline{\text{SRESET}}$ while the device is not asserting hard or soft reset. $\overline{\text{SRESET}}$ is an open-drain signal.	
		State Meaning	Asserted—An external agent or internal hardware has triggered a soft reset sequence. The internal hardware drives $\overline{\text{SRESET}}$ until the sequence completes.
		Timing	Assertion—Occurs at any time, asynchronously to any clock. Negation—Must be asserted for at least 32 CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) cycles.
		Requirements	An open-drain signal. An external pull-up is required.
		Reset State	Output, driven low during power-on, hard reset, and soft reset flows. High impedance after reset flow completes.
CFG_RESET_ SOURCE[0:3]	I	Reset configuration word source selection. These signals are on device pins that have other functions when the device is not in reset. They are sampled during the assertion of $\overline{\text{PORESET}}$ to determine the interface from which the device loads the reset configuration words.	
		State Meaning	See Section 4.3.1.1, "Reset Configuration Word Source."
		Timing	These signals are sampled during the assertion of $\overline{\text{PORESET}}$ after a stable clock is supplied ($\overline{\text{PORESET}}$ flow) and must be pulled high or low by external resistors as long as $\overline{\text{HRESET}}$ is asserted.
		Requirements	During $\overline{\text{PORESET}}$ and $\overline{\text{HRESET}}$ flows, all other signal drivers connected to these signals must be in the high-impedance state. Refer to the hardware specifications for proper resistor values to pull reset configuration signals high or low.
		Reset State	Input during power-on and hard reset flows. Functional signal after reset flow completes.
CFG_CLKIN_DIV	I	Clock in division selection. This signal is sampled during the assertion of $\overline{\text{PORESET}}$ to determine whether CLKIN is divided by two.	
		State Meaning	See Section 4.3.1.2, "CLKIN Division."
		Timing	This signal is sampled during the assertion of $\overline{\text{PORESET}}$ after a stable clock is supplied ($\overline{\text{PORESET}}$ flow), and it must be pulled high or low by external resistors as long as $\overline{\text{HRESET}}$ is asserted.
		Requirements	During $\overline{\text{PORESET}}$ and $\overline{\text{HRESET}}$ flows, all other signal drivers connected to this signal must be in the high-impedance state. Refer to the hardware specifications for proper resistors values to pull reset configuration signals high or low.
		Reset State	Input during power-on and hard reset flows. Functional signal after reset flow completes.

Table 4-1. System Control Signals (continued)

Signal	I/O	Description	
CFG_LBMUX	I	Local bus address/data pins usage selection. This reset configuration signal is located on a device pin that has another function when the device is not in reset. This input signal is sampled during the assertion of $\overline{\text{PORESET}}$ to determine whether local bus address signals will be multiplexed together with data signals or not.	
		State Meaning	See Section 4.3.1.3, “LBMUX Configuration.”
		Timing	This signal is sampled during the assertion of $\overline{\text{PORESET}}$ after a stable clock is supplied ($\overline{\text{PORESET}}$ flow) and must be pulled high or low by external resistors as long as $\overline{\text{HRESET}}$ is asserted.
		Requirements	During $\overline{\text{PORESET}}$ and $\overline{\text{HRESET}}$ flows, all other signal drivers connected to this signal must be in the high-impedance state. Refer to the *MPC8379E Hardware Specifications* for proper resistors values to pull reset configuration signals high or low.
		Reset State	Input during power-on and hard reset flows. Functional signal after reset flow completes.

4.1.2 Clock Signals

In Table 4-2, some clock signals are specific to blocks within the device. Although some of their functionality is described in Section 4.4, “Clocking,” they are defined in detail in their respective chapters. See Figure 4-8 for the internal distribution of clocks in the device.

Table 4-2. External Clock Signals

Signal	I/O	Description	
CLKIN	I	System clock. In PCI host mode, CLKIN is the primary input clock. CLKIN directly feeds the PCI output clock dividers and is driven out on the PCI_SYNC_OUT signal for de-skewing external PCI clocks routing. In PCI agent mode, this signal should be tied to GND.	
		Timing	Assertion/Negation—See the hardware specifications for timing information.
		Requirements	Should be tied low in PCI agent mode.
		Reset State	Always input.
PCI_CLK/ PCI_SYNC_IN	I	PCI clock/ PCI synchronization clock (PCI_CLK/PCI_SYNC_IN). In PCI agent mode, PCI_CLK is the primary clock input to the device. In PCI host mode, PCI_SYNC_IN is connected externally to PCI_SYNC_OUT	
		Timing	Assertion/Negation—See the hardware specifications for timing information
		Reset State	Always input.
PCI_SYNC_OUT	O	Reference PCI output synchronization clock (PCI_SYNC_OUT). In PCI host mode, PCI_SYNC_OUT is connected externally to PCI_SYNC_IN signal for de-skewing external PCI clocks routing. PCI_SYNC_OUT has the same frequency as CLKIN or CLKIN/2 depending on the state of CFG_CLKIN_DIV at reset. See Section 4.3.1.2, “CLKIN Division.” In PCI agent mode, this signal is typically not used.	
		Timing	Assertion/Negation—See the hardware specifications for timing information.
		Reset State	Always output, toggling in PCI host mode.

Table 4-2. External Clock Signals (continued)

Signal	I/O	Description	
PCI_CLK_OUT[0:4]	O	PCI output clocks bank. In PCI host mode, the device provides five separate clock output signals for feeding PCI agent devices.	
		Timing	Assertion/Negation—See the hardware specifications for timing information.
		Reset State	Always output. Drive ‘0’ and after power-on reset flow. Enabled by a memory-mapped register.

4.2 Functional Description

This section describes the various ways to reset the device, the power-on reset configurations, and clocking.

4.2.1 Reset Operations

The device has several inputs to the reset logic:

- Power-on reset ($\overline{\text{PORESET}}$)
- External hard reset ($\overline{\text{HRESET}}$)
- External soft reset ($\overline{\text{SRESET}}$)
- Software watchdog reset
- System bus monitor reset
- Checkstop reset
- JTAG reset
- Software hard reset
- Software soft reset

All of these reset sources are fed into the reset controller and, depending on the source of the reset, different actions are taken. The reset status register, described in Section 4.5.1.3, “Reset Status Register (RSR),” indicates the last sources to cause a reset.

4.2.1.1 Reset Causes

Table 4-3 describes reset causes.

Table 4-3. Reset Causes

Name	Description
Power-on reset ($\overline{\text{PORESET}}$)	Input signal. Asserting this signal initiates the power-on reset flow that resets the entire device and configures various attributes of the device including its clock modes.
Hard reset ($\overline{\text{HRESET}}$)	A bidirectional I/O signal. The device can detect an external assertion of $\overline{\text{HRESET}}$ only while it is not asserting hard reset. During $\overline{\text{HRESET}}$, $\overline{\text{SRESET}}$ is asserted. $\overline{\text{HRESET}}$ is an open-drain signal.
Soft reset ($\overline{\text{SRESET}}$)	Bidirectional I/O signal. The device can detect an external assertion of $\overline{\text{SRESET}}$ only while it is not asserting hard or soft reset. $\overline{\text{SRESET}}$ is an open-drain signal.

Table 4-3. Reset Causes (continued)

Name	Description
Software watchdog reset	After the device watchdog counts to zero, a software watchdog reset is signaled. The enabled software watchdog event then generates an internal hard reset sequence.
System bus monitor reset	After the device CSB bus monitor reaches a timeout condition, a bus monitor reset is asserted. The enabled bus monitor event then generates an internal hard reset sequence.
Checkstop reset	If the core enters checkstop state and the checkstop reset is enabled (RMR[CSRE] = 1), the checkstop reset is asserted. The enabled checkstop event then generates an internal hard reset sequence.
JTAG reset	When JTAG logic asserts the JTAG soft reset signal, an internal soft reset sequence is generated.
Software hard reset	A hard reset sequence can be initialized by writing to a memory-mapped register (RCR).
Software soft reset	A soft reset sequence can be initialized by writing to a memory-mapped register (RCR).

4.2.1.2 Reset Actions

The reset control logic determines the cause of reset, synchronizes it if necessary, and resets the appropriate internal hardware. Each reset flow has a different impact on the device logic:

- Power-on reset has the greatest impact, resetting the entire device, including clock logic and error capture registers.
- Hard reset resets the entire device excluding clock logic and error capture registers.
- Soft reset initializes the internal logic while maintaining the system configuration.

All reset types generate a reset to the core, and the impact on the application is that the core resets the MSR[IP] to the value in the BMS field of the reset configuration word high, see Section 4.3.2.2, “Reset Configuration Word High Register (RCWHR).”

The memory controller, system protection logic, interrupt controller, and I/O signals are initialized only on hard reset. Soft reset initializes the internal logic while maintaining the system configuration. Asserting external $\overline{\text{SRESET}}$ generates a hard reset to the core and to the rest of the device.Table 4-4 identifies the reset actions for each reset source.

Table 4-4. Reset Actions

Action	Reset Source		
	Power-On Reset	External Hard Reset Software Watchdog Bus Monitor Checkstop Software Hard Reset	JTAG Reset External Soft Reset
Resets: PLLs, clocks, RTC unit, and error capture registers	**Yes**	No	No
Resets: DDR, LBC, I/O multiplexors, GTM, PIT, GPIO, system configuration, and local access windows	**Yes**	Yes	No

Table 4-4. Reset Actions (continued)

Action	Reset Source		
	Power-On Reset	External Hard Reset Software Watchdog Bus Monitor Checkstop Software Hard Reset	JTAG Reset External Soft Reset
Resets other internal logic	**Yes**	Yes	Yes
Reset configuration words loaded	**Yes**	Yes	No
$\overline{\text{HRESET}}$ driven	**Yes**	Yes	No
$\overline{\text{SRESET}}$ driven	**Yes**	Yes	Yes
Hard reset to e300 core	**Yes**	Yes	Yes

4.2.2 Power-On Reset Flow

Assertion of the $\overline{\text{PORESET}}$ external signal initiates the power-on reset flow. $\overline{\text{PORESET}}$ should be asserted externally for at least 32 input clock cycles after stable external power to the device is applied.

Directly after the negation of $\overline{\text{PORESET}}$, the device starts the configuration process. The device asserts $\overline{\text{HRESET}}$ and $\overline{\text{SRESET}}$ throughout the power-on reset process, including configuration. Configuration time varies according to the configuration source and CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) frequency. Initially, the reset configuration inputs are sampled to determine the configuration source and the input clock division mode. Next, the device starts loading the reset configuration words. The system PLL begins to lock according to the clock mode values in the reset configuration word low. When the system PLL is locked, the clock unit starts distributing clock signals in the device. At this stage, the core PLL begins to lock. When it is locked and the reset configuration words are loaded, $\overline{\text{HRESET}}$ is released; $\overline{\text{SRESET}}$ is released 16 clocks later.

The detailed power-on reset (POR) flow for the device is as follows:

1. Power is applied to meet the specifications in the device data sheet.
2. The system asserts $\overline{\text{PORESET}}$ and $\overline{\text{TRST}}$, causing all registers to be initialized to their default states and most I/O drivers to be released to high-impedance.

   Some clock, clock enabled, and system control signals remain active.
3. The system applies a stable CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) signal and stable reset configuration inputs (CFG_RESET_SOURCE, CFG_CLKIN_DIV).
4. The system negates $\overline{\text{PORESET}}$ after at least 32 stable CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) clock cycles.
5. The device samples the reset configuration input signals to determine the clock division and the reset configuration words source.
6. The device starts loading the reset configuration words.

   Loading time depends on the reset configuration word source.
7. When the reset configuration word low is loaded, the system PLL begins to lock.

When the system PLL is locked, *csb_clk* is supplied to the core PLL.

8. The core PLL begins to lock.
9. The device drives HRESET asserted until the e300 PLL is locked and the reset configuration words are loaded.
10. The user optionally negates HRESET if it was not negated earlier.

    JTAG logic must always be initialized by asserting TRST. If the JTAG signals are not used, TRST should be connected directly to PORESET. TRST must not remain asserted after the negation of PORESET. There is no need to assert the SRESET signal when HRESET is asserted.
11. The internal reset to the core and the rest of the logic is negated. I/O drivers are enabled. The eLBC PLL begins to lock. The PCI interface can assert DEVSEL in response to configuration cycles.
12. The device stops driving SRESET and SRESET is negated. The reset to the e300 core is negated and the core is enabled.The boot sequencer, if enabled, is released, causing it to load configuration data from serial ROMs, as described in Section 21.4.5, "Boot Sequencer Mode."
13. Before the boot sequencer finishes, it can enable the PCI interface to accept external requests, if required, by clearing the CFG_LOCK bit in the PCI function configuration register as described in Table 14-40. If the e300 core is required to proceed, the boot sequencer should enable boot vector fetch by clearing ACR[COREDIS] as described in Section 6.2.1, "Arbiter Configuration Register (ACR)."
14. The PCI interface can now accept external requests, if enabled, and the boot vector fetch by the core can proceed, if enabled.

The device is now in its ready state.

Figure 4-1 shows a timing diagram of the power-on reset flow.

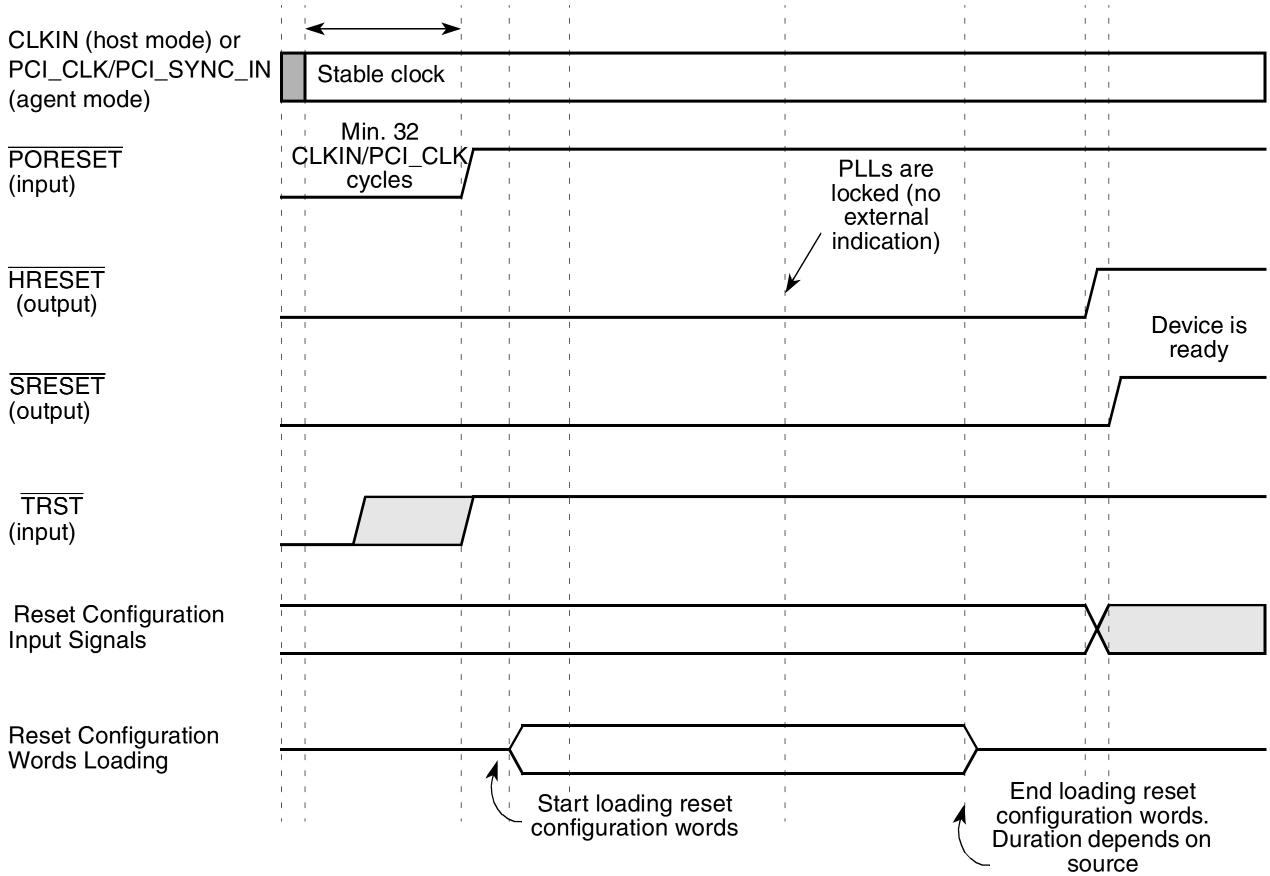


Figure 4-1. Power-On Reset Flow

4.2.3 Hard Reset Flow

The HRESET signal is initiated externally by asserting HRESET or internally when the device detects a reason to generate an internal hard reset sequence. In both cases, the device continues asserting HRESET and SRESET throughout the HRESET state. The hard reset sequence time varies according to the configuration source and CLKIN (PCI host mode) or PCI_CLK (PCI agent mode) frequency. The reset configuration input signals (CFG_RESET_SOURCE and CFG_CLKIN_DIV) are not sampled by hard reset (only by power-on reset), so the device immediately starts loading the reset configuration words and configures the device as explained in Section 4.3.3, "Loading the Reset Configuration Words." After the configuration sequence completes, the device releasesboth the HRESET and SRESET signals and exits the HRESET state. An external pull-up resistor should negate the signals. After negation is detected, a 16-cycle period is taken before testing for the presence of an external (hard/soft) reset.

NOTE

Because the device does not sample the reset configuration input signals (CFG_RESET_SOURCE, CFG_CLKIN_DIV) during a hard reset flow, setting a new value on those signals (other than that set during power-on reset) has no effect.

Figure 4-2 shows a timing diagram of the hard reset flow.

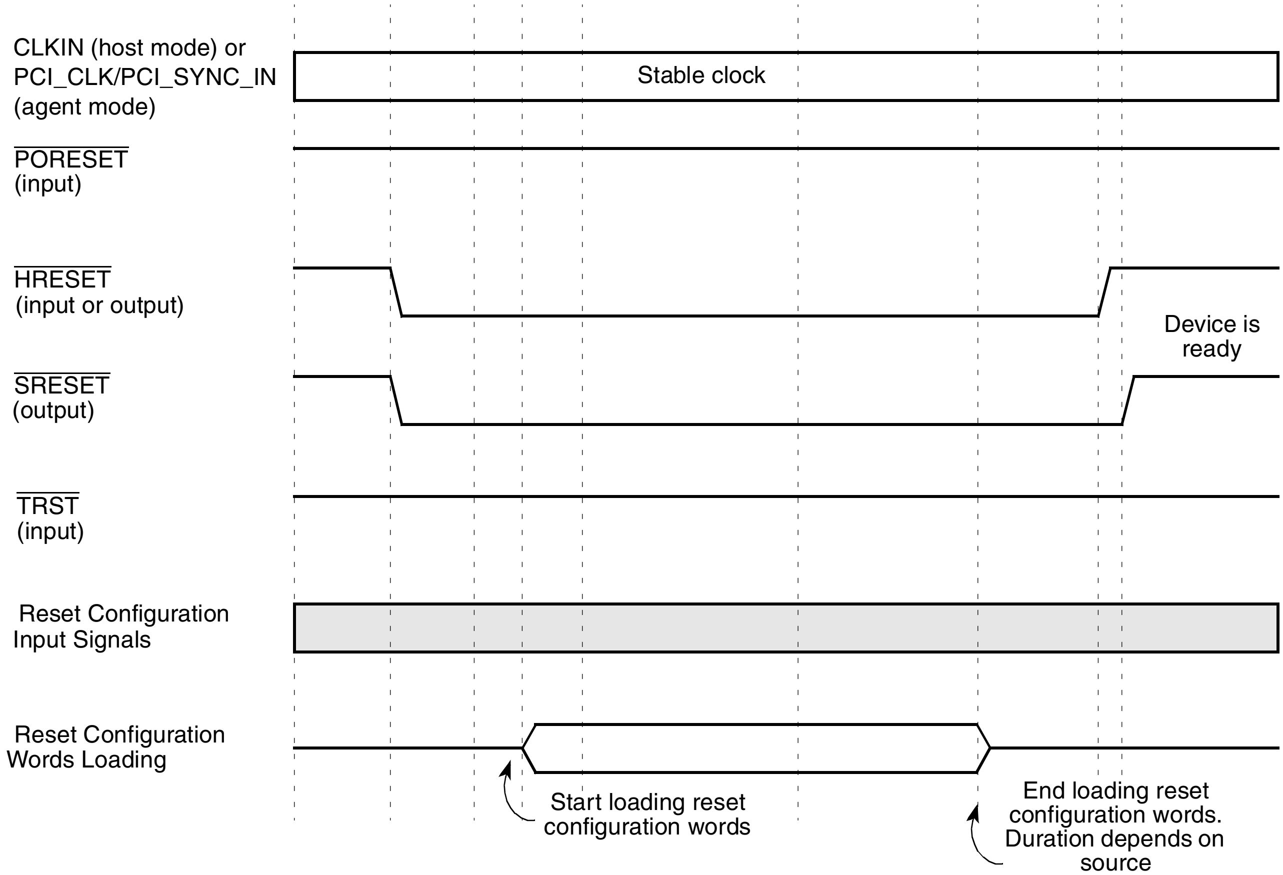


Figure 4-2. Hard Reset Flow

4.2.4 Soft Reset Flow

The SRESET signal can be initiated externally by asserting SRESET or internally when the device detects a cause to assert SRESET. In both cases, the device asserts SRESET for 512 PCI_CLK/PCI_SYNC_IN/ SYNC_IN clock cycles and then releases SRESET and exits the SRESET signal. An external pull-up resistor should negate SRESET; after negation is detected, a 16-cycle period is taken before testing for the presence of an external (hard/soft) reset. While SRESET is asserted, internal hardware is reset but hard reset configuration does not change.

4.3 Reset Configuration

The device is initialized using two complementary methods, latching CFG_RESET_SOURCE and loading the reset configuration words. Initially, a few input signals are sampled during the assertion of the PORESET signal. These signals determine whether a reset configuration word is required and the device source interface from which it is loaded. According to the value on these signals, the device continues loading the reset configuration word.

4.3.1 Reset Configuration Signals

Reset configuration input signals are on device pins that have other functions when the device is not in reset state. These input signals are sampled into registers during the assertion of PORESET, after a stable clock is supplied (PORESET), and must be pulled high or low by external resistors as long as HRESET is asserted. While the PORESET and HRESET signals are asserted, all other signal drivers connected to these signals must be in the high-impedance state. Refer to the hardware specifications for proper resistor values for pulling reset configuration signals high or low.

This section describes the modes configured by the reset configuration signals. Note that the reset configuration inputs sampled values are accessible to software through memory-mapped registers described in Section 4.5.1.3, "Reset Status Register (RSR)," and Section 4.5.2.1, "System PLL Mode Register (SPMR)."

NOTE

Implement one of the following methods to control the selection between the reset and non-reset function of these pins.

- Resistors. Use pullup or pulldown resistors to set the desired value on the reset configuration input signals. During the power-on and hard reset sequences, these signals are inputs to the device.
- Active driving device. Use HRESET to control the driving device. When HRESET is asserted, drive reset configuration values on the pins; when HRESET is negated, stop driving the reset configuration input signals.

4.3.1.1 Reset Configuration Word Source

The reset configuration word source options, shown in Table 4-5, select whether the device loads a reset configuration word from NOR Flash, NAND Flash, or an I^2C EEPROM or uses hard-coded default options. The value of these signals also affects the duration of power-on and hard reset sequences.

Table 4-5. Reset Configuration Words Source

CFG_RESET_SOURCE[0:3]	Meaning
0000	Reset configuration word is loaded from NOR Flash
0001	Reset configuration word is loaded from NAND Flash memory (8-bit small page).
0010	Reserved
0011	Reserved
0100	Reset configuration word is loaded from an I^2C EEPROM. PCI_CLK/PCI_SYNC_IN is valid for any PCI frequency up to 66.666 MHz (range of 25–66.666 MHz).
0101	Reset configuration word is loaded from NAND Flash memory (8-bit large page).
0110	Reserved
0111	Reserved
1000	Hard-coded option 0. Reset configuration word is not loaded.
1001	Hard-coded option 1. Reset configuration word is not loaded.

Table 4-5. Reset Configuration Words Source (continued)

CFG_RESET_SOURCE[0:3]	Meaning
1010	Hard-coded option 2. Reset configuration word is not loaded.
1011	Hard-coded option 3. Reset configuration word is not loaded.
1100	Hard-coded option 4. Reset configuration word is not loaded.
1101	Hard-coded option 5. Reset configuration word is not loaded.
1110	Hard-coded option 6. Reset configuration word is not loaded.
1111	Hard-coded option 7. Reset configuration word is not loaded.

4.3.1.2 CLKIN Division

When the device is configured as a PCI host, the CFG_CLKIN_DIV configuration input selects the relationship between CLKIN and PCI_SYNC_OUT as shown in Table 4-7. As a PCI host, the device supports five PCI_CLK output signals. The frequency of the output clocks will be equal to the PCI_SYNC_OUT frequency.

When the device is configured as a PCI agent, the CFG_CLKIN_DIV configuration input can be used to double the internal clock frequencies, if sampled as ‘1’ during power-on reset assertion. This feature is useful if a fixed internal frequency is desired regardless of whether the PCI clock is running at 33 or 66 MHz. PCI specifications require the PCI clock frequency information to be provided by the M66EN signal.

4.3.1.3 LBMUX Configuration

Local bus address/data pins usage is selected by the state of CFG_LBMUX during the period in which $\overline{\text{HRESET}}$ is asserted. This configuration pin needs to reflect the board usage of the local bus. The pin’s state should be stable from the beginning of $\overline{\text{HRESET}}$ assertion in order to allow proper loading of the reset configuration words in case that those words reside in a local bus slave device.

Table 4-6. CFG_LBMUX

CFG_LBMUX	Description
0	Local bus is used in a non-multiplexed address/data mode, comprised of LA[7:31] and LAD[0:15] pins.
1	Local bus is used in a multiplexed address/data mode. Address and data are multiplexed on LAD[0:31] pins.

4.3.1.4 Selecting Reset Configuration Input Signals

The example described in Table 4-7 shows how the user should pull down or pull up the reset configuration input signals (CFG_RESET_SOURCE, CFG_CLKIN_DIV). The reset sequence duration is measured from the negation of $\overline{\text{PORESET}}$ to the negation of $\overline{\text{SRESET}}$. Note that the duration mentioned in this table is typical and does not represent cases in which the process of loading the reset configuration word had to be retried due to errors.

Table 4-7. Selecting Reset Configuration Input Signals

I²C EEPROM Configuration Words	CLKIN Frequency (Host Mode)	CFG_CLKIN_DIV (Host Mode)	PCI_CLK Frequency (Agent Mode)	CFG_RESET_ SOURCE[0:3]	Reset Sequence Duration in CLKIN/ PCI_CLK Cycles	Duration
No	33 MHz	0	33 MHz	0000 (RCW loaded from NOR Flash)	15210	456 µs
No	66 MHz	0	66 MHz	1000–1111 (use hard coded RCW)	15380	231 µs
No	66 MHz	1	33 MHz	0000 (RCW loaded from NOR Flash)	30420/15210	456 µs
Yes	33 MHz	0	33 MHz	0100 (I²C EEPROM)	106534	3196 µs
Yes	66 MHz	0	66 MHz	0100 (I²C EEPROM)	106534	1598 µs
Yes	66 MHz	1	33 MHz	0100 (I²C EEPROM)	213068/106534	3196 µs
No	66 MHz	0	66 MHz	0001 (RCW loaded from 8-bit small page NAND Flash)	23024	345 µs
No	66 MHz	0	66 MHz	0010 (RCW loaded from 16-bit small page NAND FLASH)	19328	289 µs
No	66 MHz	0	66 MHz	0101 (RCW loaded from 8-bit large page NAND Flash)	45284	679 µs
No	66 MHz	0	66 MHz	0110 (RCW loaded from 16-bit large page NAND FLASH)	30500	457 µs

4.3.2 Reset Configuration Words

The reset configuration words control the clock ratios and other basic device functions such as PCI host or agent mode, boot location, and endian mode. The reset configuration words are loaded from NOR Flash, NAND Flash, or the I²C interfaces or from hard-coded values during the power-on or hard reset flows. See Section 4.3.1, “Reset Configuration Signals,” for information on the reset configuration word source. Although the configuration reset words are loaded during hard reset flows, the clocks and PLL modes are reset only when $\overline{\text{PORESET}}$ is asserted during a power-on reset flow. See Section 4.2.1.2, “Reset Actions.” The values of fields in the reset configuration words registers (RCWLR and RCWHR) reflect only their state during the reset flow. Some of these parameters and modes can be modified by changing their values in the memory-mapped registers of other units, which does not affect RCWLR and RCWHR.

The reset configuration settings are accessible to software through the following read-only memory-mapped registers:

- Reset configuration word low register (RCWLR)
- Reset configuration word high register (RCWHR)
- Reset status register (RSR)
- System PLL mode register (SPMR)

See Section 4.5, "Memory Map/Register Definitions."

4.3.2.1 Reset Configuration Word Low Register (RCWLR)

RCWLR is shown in Figure 4-3. This read-only register gets its values according to the reset configuration word low loaded during the reset flow.

	0	1	2–3	4–7	8	9–15
Field	LBCM	DDRCM	SVCOD	SPMF	—	COREPLL

	16–31
Field	—

Figure 4-3. Reset Configuration Word Low Register (RCWLR)

Table 4-8 defines the RCWLR bit fields.

Table 4-8. RCWLR Bit Settings

Bits	Name	Description
0	LBCM	Local bus memory controller clock mode. Selects the local bus controller clock ratio. If this bit is set, the local bus controller operate at twice the frequency of the *csb_clk*. If this bit is cleared, the local bus controller operate at the *csb_clk* frequency. The 2:1 mode is useful when the csb_clk operates at low frequency. 0 *csb_clk* ratio is 1:1 1 *csb_clk* ratio is 2:1
1	DDRCM	DDR SDRAM memory controller clock mode. Selects the DDR SDRAM memory controller clock ratio. If this bit is set, the DDR SDRAM memory controller operates at twice the frequency of the *csb_clk*. If this bit is cleared, the DDR SDRAM memory controller operates at the *csb_clk* frequency. The 2:1 mode is useful mostly for a 32-bit data bus width. 0 *csb_clk* ratio is 1:1 1 *csb_clk* ratio is 2:1
2–3	SVCOD	System PLL VCO division. See Section 4.3.2.1.1, "System PLL VCO Division."
4–7	SPMF	System PLL multiplication factor. See Table 4-11," for more information.
8	—	Reserved, should be cleared
9–15	COREPLL	Core PLL configuration. COREPLL sets the ratio between the e300 core clock and the internal *csb_clk* of the device. The encodings for COREPLL are given in the hardware specifications for this device.
16–31	—	Reserved, should be cleared.

4.3.2.1.1 System PLL VCO Division

The RCWLR field SVCOD (system PLL VCO division), shown in Table 4-9, establishes the internal ratio between the system PLL VCO frequency and the PLL output clock frequency. The PLL output clock frequency equals *csb_clk* frequency if RCWLR[LBCM] and RCWLR[DDRCM] are both cleared or twice the csb_clk frequency if RCWLR[LBCM] or RCWLR[DDRCM] or both of them are set.

The RCWLR[SVCOD] value should be set such that the PLL VCO frequency will be in the range of 400 MHz ≤ system PLL VCO frequency ≤ 800 MHz.

Table 4-9 describes the setting of SVCOD bits.

Table 4-9. System PLL VCO Division

Reset Configuration Word Low Register (RCWLR) Bits	Field Name	Value (Binary)	VCO Division Factor
2–3	SVCOD	00	4
		01	8
		10	2
		11	1

4.3.2.1.2 System PLL Configuration

The system PLL ratio reset, shown in Table 4-10, establishes the clock ratio between the CLKIN signal and the internal *csb_clk* of the device. *csb_clk* drives internal units and feeds the e300 core PLL.

Table 4-10. System PLL Ratio

RCWLR Bits	Field Name	Value (Binary)	*csb_clk* : CLKIN (PCI Host Mode) *csb_clk* : (PCI_CLK x (1+~sampled_cfg_clkin_div)) (PCI Agent Mode)
4–7	SPMF	0000	Reserved
		0001	Reserved
		0010	2 : 1
		0011	3 : 1
		0100	4 : 1
		0101	5 : 1
		0110	6 : 1
		0111	7 : 1
		1000	8 : 1
		1001	9 : 1
		1010	10 : 1
		1011	11 : 1
		1100	12 : 1
		1101	13 : 1
		1110	14 : 1
		1111	15: 1

NOTE

In PCI host mode, the SPMF field described in Table 4-10 always selects the *csb_clk*:CLKIN ratio regardless of the CFG_CLKIN_DIV reset configuration input value during reset flow.

The SPMF field maximum allowed value is dependent on the value sampled on CFG_CLKIN_DIV during power-on reset. Table 4-11 defines the upper limit of SPMF with respect to these values. Values for SPMF are as follows:

Table 4-11. SPMF Maximum Values

CFG_CLKIN_DIV	LBCM	DDRCM	Maximum SPMF Value (decimal)
0	0	0	16
0	0	1	8
0	1	0	8
0	1	1	8
1	0	0	8

Table 4-11. SPMF Maximum Values (continued)

CFG_CLKIN_DIV	LBCM	DDRCM	Maximum SPMF Value (decimal)
1	0	1	4
1	1	0	4
1	1	1	4

4.3.2.2 Reset Configuration Word High Register (RCWHR)

RCWHR is shown in Figure 4-4. This read-only register gets its values according to the reset configuration word high loaded during the reset flow.

Offset 0x0_0904 | Access: Read/Write

	0	1	2	3	4	5	6–7	8	9–11	12–13	14	15
Field	PCIHOST	—	PCIARB	—	CORE DIS	BMS	BOOTSEQ	SWEN	ROMLOC	RLEXT	—	—

	16	21	22–27	28	29	30	31
Field	TSEC1M	TSEC2M	—	TLE	—	LDP	—

Figure 4-4. Reset Configuration Word High Register (RCWHR)

Table 4-12 defines the reset configuration word high bit fields.

Table 4-12. Reset Configuration Word High Bit Settings

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>PCIHOST</td><td>PCI host mode.
See Section 4.3.2.2.1, “PCI Host/Agent Configuration,” for more information.</td></tr>
<tr><td>1</td><td>—</td><td>Reserved, should be cleared.</td></tr>
<tr><td>2</td><td>PCIARB</td><td>PCI internal arbiter mode. Enables the on-chip PCI arbiter.
0 On-chip PCI arbiter is disabled. External arbitration is required.
1 On-chip PCI arbiter is enabled.
<table>
<tr><th>Pin Function When PCIARB = 0</th><th>Pin Function When PCIARB = 1</th></tr>
<tr><td>CPCI_HS_ES</td><td>PCI_REQ[1]</td></tr>
<tr><td>CPCI_HS_LED</td><td>PCI_GNT[1]</td></tr>
<tr><td>CPCI_HS_ENUM</td><td>PCI_GNT[2]</td></tr>
<tr><td>PCI_PME</td><td>PCI_GNT[3]</td></tr>
</table></td></tr>
<tr><td>3</td><td>—</td><td>Reserved, should be cleared.</td></tr>
</table>

Table 4-12. Reset Configuration Word High Bit Settings (continued)

Bits	Name	Description
4	COREDIS	Core disable mode. Specifies the e300 core mode out of reset. If COREDIS is set, the core cannot fetch boot code until it is configured by an external master. The external master frees the core to boot by clearing the COREDIS bit in the arbiter configuration register as described in Section 6.2.1, "Arbiter Configuration Register (ACR)." This bit must be set when the boot sequencer is enabled to initiate the device (BOOTSEQ is not 0b00). Otherwise, unpredictable behavior occurs. 0 The core can boot without waiting for configuration by an external master. 1 Core boot holdoff mode. The core is prevented from booting until it is configured by an external master.
5	BMS	Boot memory space. See Section 4.3.2.2.2, "Boot Memory Space (BMS)," for more information.
6–7	BOOTSEQ	Boot sequencer configuration. See Section 4.3.2.2.3, "Boot Sequencer Configuration," for more information.
8	SWEN	Software watchdog enable. Selects whether the software watchdog is enabled to start counting down immediately when coming out of reset. The user can override this value by writing to the system watchdog control register (SWCRR[SWEN]) during system initialization. 0 Disabled 1 Enabled
9–11	ROMLOC	Boot ROM interface location. This bit combined with bit RLEXT determines where the device boots from. See Section 4.3.2.2.4, "Boot ROM Location," for more information.
12–13	RLEXT	Boot ROM location extension. This bit combined with bit ROMLOC determines where the device boots from. See Section 4.3.2.2.4, "Boot ROM Location," for more information. 00 Legacy mode—allows for booting from on-chip peripherals. Refer to Table 4-16 for more information. 01 NAND Flash mode—allows for booting from NAND flash devices. Refer to Table 4-16 for more information. 10 Reserved 11 Reserved
14–15	—	Reserved, should be cleared.
16–18	TSEC1M	TSEC1 mode. See Section 4.3.2.2.6, "eTSEC1 Mode," for more information.
19–21	TSEC2M	TSEC2 mode. See Section 4.3.2.2.7, "eTSEC2 Mode," for more information.
22–27	—	Reserved, should be cleared.
28	TLE	True little-endian. See Section 4.3.2.2.8, "e300 Core True Little-Endian," for more information.
29	—	Reserved, should be cleared.
30	LDP	LDP pin mux state after reset. See Section 4.3.2.2.9, "LDP Configuration," for more information.
31	—	Reserved, should be cleared.

4.3.2.2.1 PCI Host/Agent Configuration

The PCIHOST configuration parameter, shown in Table 4-13, configures the device to act as a PCI host or as a PCI agent device. In host mode, the device can immediately master transactions to the PCI interface. If the device is a PCI agent device, the device is disabled from mastering PCI transactions until the external host enables it to do so. The external host does this by setting the control registers of the device's interfaces appropriately. See details in the PCI programming model described in Section 14.3, "Memory Map/Register Definitions."

Table 4-13. PCI Host/Agent Configuration

RCWHR Bit	Field Name	Value (Binary)	Meaning
0	PCIHOST	0	The device acts as a PCI agent device.
		1	The device acts as the host processor (default).

NOTE

If the device is a PCI agent, and the e300 core is not in holdoff mode (as described in Section 4.3.2.2, "Reset Configuration Word High Register (RCWHR)"), the boot ROM should not be located on the PCI interface because the device is not enabled to master reads onto the PCI bus.

4.3.2.2.2 Boot Memory Space (BMS)

BMS defines the initial value of the e300 core MSR[IP] bit, which specifies the location of the interrupt vectors (including the hard reset exception vector). The device defines the default boot ROM memory space to be 8 Mbytes at addresses 0x0000_0000 to 0x007F_FFFF or 0xFF80_0000 to 0xFFFF_FFFF. When the core comes out of reset, if it is enabled to boot, it begins fetching boot code from one of two addresses: 0x0000_0100 or 0xFFF0_0100, and exceptions are vectored to physical addresses 0x000*n_nnnn* or 0xFFF*n_nnnn* appropriately. This bit specifies whether an interrupt vector offset is prepended with 0xFFF or 0x000. In the description below, *n_nnnn* is the offset of the exception vector.

The boot memory space reset configuration word field, shown in Table 4-14, specifies both the device boot ROM address window and the initial e300 core boot address.

Table 4-14. Boot Memory Space

RCWHR Bit	Field Name	Value (Binary)	Meaning
5	BMS	0	Boot memory space is 8 Mbytes at 0x0000_0000 to 0x007F_FFFF. e300 core register MSR[IP] initial value is 0b0. The core, if enabled to boot, begins fetching boot code from address 0x0000_0100 and exceptions are vectored to the physical address of 0x000*n_nnnn*.
		1	Boot memory space is 8 Mbytes at 0xFF80_0000 to 0xFFFF_FFFF. e300 core register MSR[IP] initial value is 0b1. The core, if enabled to boot, begins fetching boot code from address 0xFFF0_0100 and exceptions are vectored to the physical address of 0xFFF*n_nnnn*.

4.3.2.2.3 Boot Sequencer Configuration

The boot sequencer configuration options, shown in Table 4-15, allow the boot sequencer to load configuration data from the serial ROM located on the I^2C port before the host tries to configure the device. These options also specify normal or extended I^2C addressing modes. See Section 21.4.5, “Boot Sequencer Mode.”

Table 4-15. Boot Sequencer Configuration

RCWHR Bits	Field Name	Value (Binary)	Meaning
6–7	BOOTSEQ	00	Boot sequencer is disabled. No I^2C ROM is accessed.
		01	Normal I^2C addressing mode is used. Boot sequencer is enabled and loads configuration information from a ROM on the I^2C interface. A valid ROM must be present.
		10	Extended I^2C addressing mode is used. Boot sequencer is enabled and loads configuration information from a ROM on the I^2C interface. A valid ROM must be present.
		11	Reserved, should be cleared.

NOTE

When the boot sequencer is enabled, the e300 core must be prevented from fetching boot code, by setting the core disable reset configuration word field (COREDIS) as described in Section 4.3.2.2, “Reset Configuration Word High Register (RCWHR).” If the e300 core is required to proceed, the boot sequencer should enable boot vector fetch by clearing ACR[COREDIS] as described in Section 6.2.1, “Arbiter Configuration Register (ACR).”

4.3.2.2.4 Boot ROM Location

The device defines the default boot ROM address range to be 8 Mbytes at addresses 0x0000_0000 to 0x007F_FFFF or 0xFF80_0000 to 0xFFFF_FFFF (selected by the BMS reset configuration word field). However, the on-chip peripheral that manages these boot ROM accesses can be selected at power up.

The boot ROM location reset configuration word field, shown in Table 4-16, establishes the location of boot ROM. The exact boot ROM location table to be used is defined by the setting of RCWHR[RLEXT] bits, as shown in Table 4-12. Accesses to the boot vector and the default boot ROM region of the local address map are directed to the interface specified by this field.

Table 4-16. Boot ROM Location

RCWHR Bits	Field Name	Value (Binary)	Meaning	
			Legacy Mode (RLEXT = 00)	NAND Flash Mode (RLEXT = 01)
9–11	ROMLOC	000	DDR SDRAM	Reserved
		001	PCI	Local bus NAND Flash—8-bit small page ROM
		010	Reserved	Reserved
		011	On-chip boot ROM	Reserved
		100	Reserved	Reserved
		101	Local bus GPCM—8-bit ROM	Local bus NAND Flash—8-bit large page ROM
		110	Local bus GPCM—16-bit ROM	Reserved
		111	Local Bus GPCM—32-bit ROM	Reserved

The local access window of the selected boot ROM interface is enabled and initialized with the proper base address and size, as described in Section 5.2, "Local Memory Map Overview and Example."

4.3.2.2.5 Boot from SPI

Boot from SPI is supported by the MPC8379E using an on-chip ROM which contains the basic SPI device driver and the code to perform block copy from SPI Eprom to DDR memory. Selecting on-chip ROM in boot ROM location (see Table 4-16) will cause the e300 CPU to fetch data from the on-chip ROM. See Section 23.5, "SPI Boot ROM."

4.3.2.2.6 eTSEC1 Mode

The TSEC1 mode reset configuration word field, shown in Table 4-17, selects the protocol used by the eTSEC1 controller (enhanced three-speed Ethernet controller interface).

Table 4-17. eTSEC1 Mode Configuration

Reset Configuration Word High Register (RCWHR) Bits	Field Name	Value (Binary)	Meaning
16–18	TSEC1M	000	The eTSEC1 controller operates in the MII protocol, using only four transmit data signals and four receive data signals.
		001	The eTSEC1 controller operates in the RMII protocol, using only two transmit data signals and two receive data signals.
		010	Reserved
		011	The eTSEC1 controller operates in the RGMII protocol, using four transmit data signals and four receive data signals.
		100	Reserved
		101	The eTSEC1 controller operates in the RTBI protocol, using only four transmit data signals and four receive data signals.
		110	The eTSEC1 controller operates in the SGMII protocol, using the on-chip PHY.
		111	Reserved

4.3.2.2.7 eTSEC2 Mode

The eTSEC2 mode reset configuration word field, shown in Table 4-18, selects the protocol used by the eTSEC2 controller (enhanced three-speed Ethernet controller interface).

Table 4-18. eTSEC2 Mode Configuration

Reset Configuration Word High Register (RCWHR) Bits	Field Name	Value (Binary)	Meaning
19–21	TSEC2M	000	The eTSEC2 controller operates in the MII protocol, using only four transmit data signals and four receive data signals.
		001	The eTSEC2 controller operates in the RMII protocol, using only two transmit data signals and two receive data signals.
		010	Reserved
		011	The eTSEC2 controller operates in the RGMII protocol, using four transmit data signals and four receive data signals.
		100	Reserved
		101	The eTSEC2 controller operates in the RTBI protocol, using only four transmit data signals and four receive data signals.
		110	The eTSEC2 controller operates in the SGMII protocol, using the on-chip PHY.
		111	Reserved

4.3.2.2.8 e300 Core True Little-Endian

The true little endian reset configuration word field, shown in Table 4-19, selects whether the e300 core operates in big-endian mode or true little-endian mode at reset.

Table 4-19. e300 Core True Little-Endian

Reset Configuration Word High Register (RCWHR) Bit	Field Name	Value (Binary)	Meaning
28	TLE	0	Big-endian mode
		1	True little-endian mode

4.3.2.2.9 LDP Configuration

The LDP reset configuration word field configures the initial state of SICRL[LDP_A] and SICRL[LDP_B], which controls the functionality of the LPD[0:3] pins. Table 4-20 shows the LDP configuration.

Table 4-20. LDP Configuration

Reset Configuration Word High Register (RCWHR) Bit	Field Name	Meaning
30	LDP	0 Initial value of SICRL[LDP_A] and SICRL[LDP_B] is 1, meaning that LDP0–LDP3 are used for local data parity. 1 Initial value of SICRL[LDP_A] is 0; the meaning is as follows: • LDP0 is used as $\overline{\text{LCS4}}$ • LDP1 is used as $\overline{\text{LCS5}}$ • LDP2 is used as $\overline{\text{LCS6}}$ • LDP3 is used as $\overline{\text{LCS7}}$

4.3.3 Loading the Reset Configuration Words

The device loads the reset configuration words from a local bus EEPROM, or an I^2C serial EEPROM, or uses hard-coded configuration, as selected by the reset configuration inputs described in Section 4.3.1, "Reset Configuration Signals." The following sections describe each of these options.

4.3.3.1 Loading from Local Bus

The reset configuration words are assumed to reside in an EEPROM or NOR Flash or NAND Flash device connected to $\overline{\text{LCS0}}$ of the device local bus. Because the port size of this EEPROM is unknown, the device reads all configuration words byte-by-byte only from locations that are independent of port size. + LCS0 is the default for GPCM, so GPCM controlled is used to read the reset configuration word from EEPROM. /LGTA should be high to avoid unintended early termination of the read cycle.

Table 4-21 shows addresses that should be used to contain the reset configuration words. Byte addresses that do not appear in this table have no effect on the configuration of the device. The values of the bytes in Table 4-21 are always read on byte lane LAD[0:7] regardless of the port size.

Table 4-21. Local Bus Configuration EEPROM Addresses

Reset Configuration Word	Bits [0:7] Address	Bits [8:15] Address	Bits [16:23] Address	Bits [24:31] Address
Low	0x00	0x08	0x10	0x18
High	0x20	0x28	0x30	0x38

Table 4-22 shows the data structure of the local bus device containing the reset configuration words (RCWL and RCWH).

Table 4-22. Local Bus Reset Configuration Words Data Structure

EEPROM Address	EEPROM Data Bits			
	[0:7]	[8:15]	[16:23]	[24:31]
0x00	RCWL[0:7]			
0x04				
0x08	RCWL[8:15]			
0x0C				
0x10	RCWL[16:23]			
0x14				
0x18	RCWL[24:31]			
0x1C				
0x20	RCWH[0:7]			
0x24				
0x28	RCWH[8:15]			
0x2C				
0x30	RCWH[16:23]			
0x34				
0x38	RCWH[24:31]			
0x3C				

4.3.3.1.1 Local Bus Controller Setting

The device will use GPCM to load the reset configuration from EEPROM or NOR Flash. The device will read 64 bytes in this case. The local bus controller’s registers setting will be set according to Table 4-23.

The device will use FCM to load the reset configuration from NAND Flash. The device will read 512 bytes if small size NAND Flash is used, or 2048 bytes if large page NAND Flash is used. The local bus controller’s registers setting will be set according to Table 4-23.

The device will use PCI_SYNC_IN clock to generate the internal LCLK, which will run at half the frequency of PCI_SYNC_IN.

Table 4-23. Local Bus Controller Setting when Loading RCW

CFG_RESET_SOURCE	Meaning	BR0[PS]	BR0[MSEL]	OR0[SCY]	OR0[PGS]
0000	NOR Flash	11	000	1111	NA
0001	NAND Flash, 8 bit, small page	01	001	0010	0
0101	NAND Flash, 8 bit, large page	01	001	0010	1

Figure 4-5 depicts an example of loading RCW from local bus. Here, the RCW source is the NOR flash and the CFG_LBMUX is configured as multiplexed address/data. The figure shows the beginning of the loading process. The process is repeated eight times and ends when the last byte is loaded at address 0x3C. Note that in each pair of readings only the first read carries meaningful information.

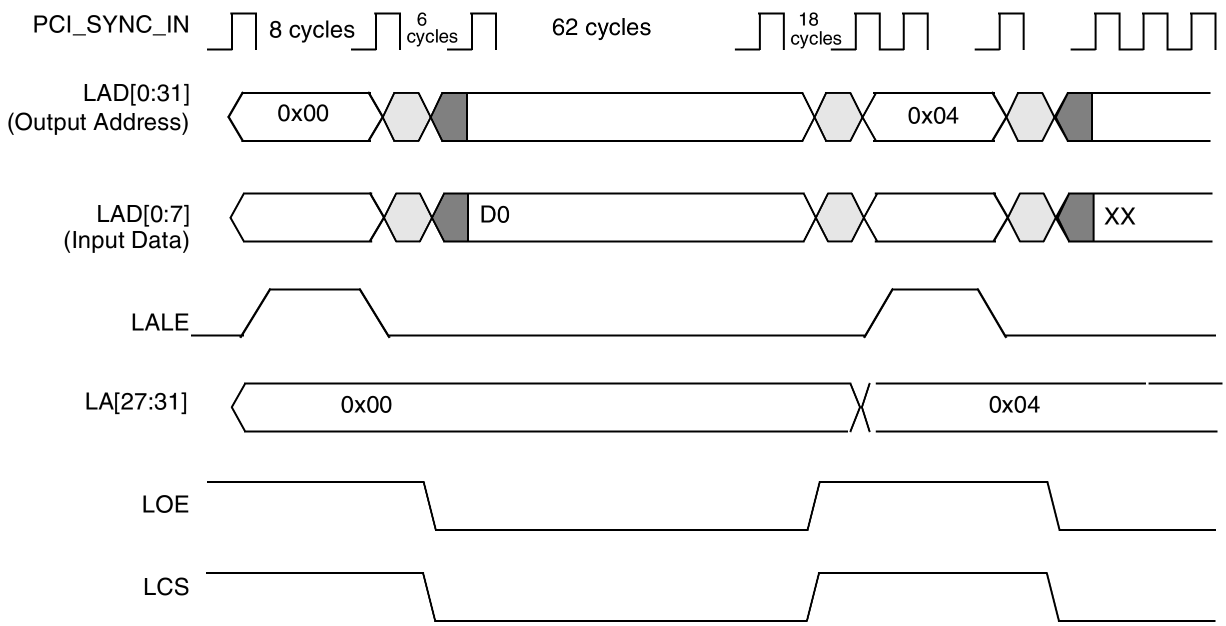


Figure 4-5. Loading RCW from Local bus, NOR Flash, Multiplexed Address/Data

4.3.3.2 Loading from I²C EEPROM

The device is capable of loading the reset configuration word from the I²C interface. If the device is configured to load the reset configuration word from the I²C interface, according to the reset configuration input signals, it uses the I²C unit boot sequencer in a special mode. In this mode, the I²C boot sequencer is activated while the rest of the device is still in reset state ($\overline{\text{HRESET}}$ asserted) to load the reset configuration words from an I²C serial EEPROM.

Note that this does not prevent using the I²C boot sequencer to initiate the device in the normal functional mode after reset state has completed. The only restriction is that the first two EEPROM data structures contain dedicated reset information.

4.3.3.2.1 Using the Boot Sequencer Reset Configuration

For a detailed description about the I^{2}C interface and the boot sequencer refer to Section 21.4.5, "Boot Sequencer Mode."

NOTE

When reset configuration words are loaded from an I^{2}C EEPROM, an I^{2}C serial EEPROM of extended addressing type must be used.

If the I^{2}C interface is used for loading the reset configuration words, the I^{2}C module addresses the EEPROM and reads the first two data structures (after reading the preamble). Upon being read, the reset configuration words are latched inside the device and the I^{2}C module enters its reset state until $\overline{\text{HRESET}}$ is negated. There should be no other I^{2}C traffic when the boot sequencer is active.

After $\overline{\text{HRESET}}$ is negated, the functional boot sequencer, in extended I^{2}C addressing mode, may be activated if the BOOTSEQ field of the reset configuration word high is set to 0b10.

4.3.3.2.2 EEPROM Calling Address

The device uses 0b101_0000 for the EEPROM calling address. The EEPROM to be addressed must contain the reset configuration information and be programmed to respond to this address. No additional EEPROMs are accessed by the boot sequencer in reset configuration mode.

4.3.3.2.3 EEPROM Data Format in Reset Configuration Mode

The I^{2}C module expects that a particular data format be used for data in the EEPROM. A preamble should be the first 3 bytes programmed into the EEPROM. It should have a value of 0xAA55AA. The I^{2}C module checks to ensure that this preamble is correctly detected before proceeding further. Following the preamble, there should be the two reset configuration words, programmed according to a particular format, as shown in Figure 4-6.

The first 3 bytes hold the attributes and address offset. The addresses of the two reset configuration words must be programmed to the offset of the reset configuration word low register (RCWLR) and reset configuration word high register (RCWHR) respectively (see Section 4.5.1.1, "Reset Configuration Word Low Register (RCWLR)," and Section 4.5.1.2, "Reset Configuration Word High Register (RCWHR)"). The attributes should be programmed as follows: alternate configuration space (ACS) should be cleared (0b0), byte enables should be all ones, and continue (CONT) should be set.

After the first 3 bytes, 4 bytes of data should hold the desired value of the reset configuration word. The boot sequencer assumes that a big-endian address is stored in the EEPROM.

IMMRBAR value is prepended to the EEPROM address to generate the complete memory-mapped register's address.

When the I^{2}C operates in reset configuration mode, the cyclic redundancy check (CRC) is ignored, as well as any registers following the first two reset configuration words.

0	1–4	5	6–7
ACS (0)	BYTE_EN (1111)	CONT (1)	RCWLR ADDR[12–13]
RCWLR ADDR[14:21]			
RCWLR ADDR[22:29]			
Reset configuration word low [0–7]			
Reset configuration word low [8–15]			
Reset configuration word low [16–23]			
Reset configuration word low [24–31]			
ACS (0)	BYTE_EN (1111)	CONT (1)	RCWHR ADDR[12–13]
RCWHR ADDR[14–21]			
RCWHR ADDR[22–29]			
Reset configuration word high [0–7]			
Reset configuration word high [8–15]			
Reset configuration word high [16–23]			
Reset configuration word high [24–31]			

Figure 4-6. EEPROM Data Format for Reset Configuration Words Preload Command

Figure 4-7 shows an example of the EEPROM contents, including the preamble, reset configuration words and additional initialization data, and CRC. In this example, it is assumed that the EEPROM contains information additional to the reset configuration words, which should be loaded in the functional state after the device completes its reset flow.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td></td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td rowspan="3">Preamble</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="2">RCWLR ADDR[12:13]</td><td rowspan="7">Reset configuration word low preload command</td></tr>
<tr><td colspan="8">RCWLR ADDR[14–21]</td></tr>
<tr><td colspan="8">RCWLR ADDR[22–29]</td></tr>
<tr><td colspan="8">Reset configuration word low [0–7]</td></tr>
<tr><td colspan="8">Reset configuration word low [8–15]</td></tr>
<tr><td colspan="8">Reset configuration word low [16–23]</td></tr>
<tr><td colspan="8">Reset configuration word low [24–31]</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="2">RCWHR ADDR[12:13]</td><td rowspan="7">Reset configuration word high preload command</td></tr>
<tr><td colspan="8">RCWHR ADDR[14–21]</td></tr>
<tr><td colspan="8">RCWHR ADDR[22–29]</td></tr>
<tr><td colspan="8">Reset configuration word high [0–7]</td></tr>
<tr><td colspan="8">Reset configuration word high [8–15]</td></tr>
<tr><td colspan="8">Reset configuration word high [16–23]</td></tr>
<tr><td colspan="8">Reset configuration word high [24–31]</td></tr>
<tr><td colspan="8">*</td><td></td></tr>
<tr><td>ACS</td><td colspan="4">BYTE_EN</td><td>1</td><td colspan="2">ADDR[12–13]</td><td rowspan="7">Last configuration preload command</td></tr>
<tr><td colspan="8">ADDR[14–21]</td></tr>
<tr><td colspan="8">ADDR[22–29]</td></tr>
<tr><td colspan="8">DATA[0–7]</td></tr>
<tr><td colspan="8">DATA[8–15]</td></tr>
<tr><td colspan="8">DATA[16–23]</td></tr>
<tr><td colspan="8">DATA[24–31]</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td rowspan="3">End command</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="8">CRC[0–7]</td><td rowspan="4">Cyclic redundancy check</td></tr>
<tr><td colspan="8">CRC[8–15]</td></tr>
<tr><td colspan="8">CRC[16–23]</td></tr>
<tr><td colspan="8">CRC[24–31]</td></tr>
</table>

Figure 4-7. EEPROM Contents

4.3.3.2.4 Reset Configuration Load Fail

Failure of reset configuration load by the I^{2}C boot sequencer can be caused by an incorrect EEPROM data structure or I^{2}C bus problem. If a reset configuration load failure occurs, due to preamble fail or any other I^{2}C bus error detection, the device will continuously attempt to reload the hard reset configuration words from the I^{2}C bus. The device does not negate $\overline{\text{HRESET}}$ and remains in hard reset state until the HRCWs are successfully loaded or the PORESET flow is restarted.

4.3.3.3 Default Reset Configuration Words

If the device is configured not to load the reset configuration words from NOR Flash, NAND Flash, or an I^{2}C EEPROM, it can also be initialized with one of eight hard-coded default options, selected by the reset configuration input signals, CFG_RESET_SOURCE[0:3].

Default reset configuration words are used for two main reasons:

- Boot from SPI. The first six hard-coded default options are intended to be used when boot from SPI is intended. In those options the PCI will be in host mode, the core will be enabled to boot and the ROM location will be set to on-chip ROM, in which the SPI boot code will be running. See Section 4.3.2.2.5, "Boot from SPI."
- PCI agent mode. The remaining two hard-coded default options are intended to be used when no boot device is present locally for the device. The device will be used in PCI agent mode and the core will be disabled when reset negates. The remote PCI host is expected to configure the device and prepare the boot code for the local e300 CPU, before it is enabled to boot. Alternatively the application can select to use the device in a core disabled mode.

The reset configuration words are driven internally with the values shown in Table 4-24 and Table 4-25.

NOTE

In this modeand in those cases where RCWH[PCIHOST] is cleared, the device is also configured to accept PCI configuration cycles when completing its reset sequence (In PCI function configuration register, the CFG_LOCK bit is cleared). In addition, the inbound window size of the PCI inbound window attribute registers (PIWAR*n*[IWS]) is set to 0b010011, defining 1-Mbyte ($2^{(19+1)}$) memory windows. See Section 14.3.3.24, "PCI Function Configuration Register."

Table 4-24. Hard Coded Reset Configuration Word Low Fields Values

RCWL Bits:	0	1	2–3	4–7	8	9–15	16–31
Field:	LBCM	DDRCM	SVCOD	SPMF	Res	COREPLL	Res
Meaning:	LBC controller clock: *csb_clk*	DDR controller clock: *csb_clk*	VCO Division:	*csb_clk*: PCI_CLK ratio	—	Core clock: *csb_clk* ratio	—
CFG_RESET_SOURCE Value	0 1:1 1 2:1	0 1:1 1 2:1	00 4 01 8 10 2 11 1	SPMF:1			

Table 4-24. Hard Coded Reset Configuration Word Low Fields Values (continued)

RCWL Bits:	0	1	2–3	4–7	8	9–15	16–31
Field:	LBCM	DDRCM	SVCOD	SPMF	Res	COREPLL	Res
1000	0	0	10	0101	0	0000100	16’b0
1001	1	1	10	0100	0	0001000	16’b0
1010	0	0	10	1000	0	0000011	16’b0
1011	1	1	10	0100	0	0000100	16’b0
1100	1	1	10	0101	0	0000100	16’b0
1101	0	0	10	0100	0	0000100	16’b0
1110	1	1	10	0100	0	0000110	16’b0
1111	1	1	10	0010	0	0000110	16’b0

Table 4-25 defines the hard-coded reset configuration word high fields values. These values select hard-coded reset configuration words options, as described in Section 4.3.1.1, “Reset Configuration Word Source.”

Table 4-25. Hard-Coded Reset Configuration Word High Field Values

Bits	Name	Field Value when CFG_RESET_SOURCE[0:3] = 1000–1111								Meaning
		1000	1001	1010	1011	1100	1101	1110	1111	
0	PCIHOST	1	1	1	1	1	1	0	0	1 = PCI host mode 0 = PCI agent mode
1	Reserved, should be cleared.	0	0	0	0	0	0	0	0	—
2	PCIARB	1	1	1	1	1	1	0	0	1 = Internal arbiter is used 0 = External arbiter is used
3	Reserved, should be cleared.	0	0	0	0	0	0	0	0	—
4	COREDIS	0	0	0	0	0	0	1	1	1 = e300 core is disabled (boot holdoff)
5	BMS	1	1	1	1	1	1	1	1	Boot memory space is 0xFF80_0000–0xFFFF_FFFF. MSR[IP] initial value is 0b1
6–7	BOOTSEQ	00	00	00	00	00	00	00	00	Boot sequencer is disabled.
8	SWEN	0	0	0	0	0	0	0	0	Software watchdog disabled.

Table 4-25. Hard-Coded Reset Configuration Word High Field Values (continued)

Bits	Name	Field Value when CFG_RESET_SOURCE[0:3] = 1000–1111								Meaning
		1000	1001	1010	1011	1100	1101	1110	1111	
9–11	ROMLOC	011	011	011	011	011	011	000	000	Boot ROM interface location 011 = On-chip ROM 000 = DDR
12–13	RLEXT	00	00	00	00	00	00	00	00	Rom Location Extension. Select legacy mode.
14–15	Reserved, should be cleared.	00	00	00	00	00	00	00	00	—
16–18	TSEC1M	011	011	011	011	011	011	011	011	RGMIII mode
19–21	TSEC2M	011	011	011	011	011	011	011	011	RGMII mode
22–27	Reserved, should be cleared.	000000	000000	000000	000000	000000	000000	000000	000000	—
28	TLE	0	0	0	0	0	0	0	0	Big-endian mode
29	Reserved, should be cleared.	0	0	0	0	0	0	0	0	—
30	LDP	1	1	1	1	1	1	1	1	LPD pins used for local chip selects
31	Reserved, should be cleared.	0	0	0	0	0	0	0	0	—

4.3.3.3.1 Examples for Hard-Coded Reset Configuration Words Usage

Examples for various clock modes are listed in Table 4-26.

Table 4-26. Examples For Hard-Coded Reset Configuration Words Usage

CFG_RESET_SOURCE[0:3]	1000	1001	1010	1011	1100	1101	1110	1111
PCI_CLK (MHz)	66	33	33	33	33	66	33	66
csb_clk (MHz)	333	133	266	133	166	266	133	133
Core clock (MHz)	667	533	400	266	333	533	400	400
DDR Controller clock	333	266	266	266	333	266	266	266
eLBC Controller clock	333	266	266	266	333	266	266	266

4.4 Clocking

Figure 4-8 shows the internal distribution of clocks within the device.

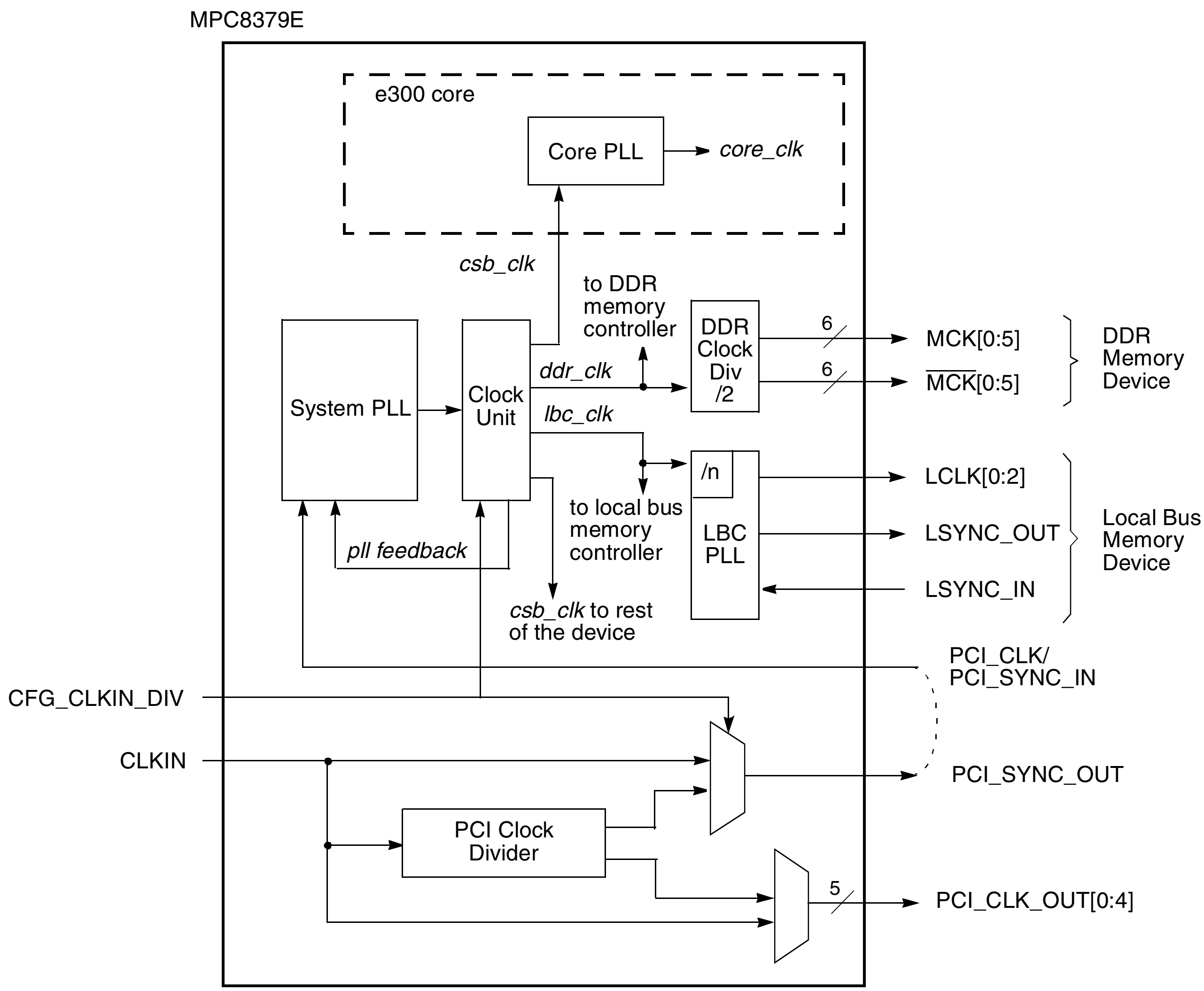


Figure 4-8. Clock Subsystem Block Diagram

The primary clock source for the device can be one of two inputs, CLKIN or PCI_CLK, depending on whether the device is configured in PCI host or PCI agent mode, respectively.

4.4.1 Clocking in PCI Host Mode

When the device is configured as a PCI host device (RCWH[PCIHOST] = 1), CLKIN is the primary input clock. CLKIN feeds the PCI clock divider (÷ 2) and the PCI_SYNC_OUT and PCI_CLK_OUT multiplexors. The CFG_CLKIN_DIV configuration input selects whether CLKIN or CLKIN/2 is driven out on the PCI_SYNC_OUT signal.

PCI_SYNC_OUT is connected externally to PCI_SYNC_IN to allow the internal clock subsystem to synchronize to the system PCI clocks. PCI_SYNC_OUT must be connected properly to PCI_SYNC_IN, with equal delay to all PCI agent devices in the system.

4.4.1.1 PCI Clock Outputs (PCI_CLK_OUT[0:4])

When the device is configured as a PCI host, it provides five clock output signals, PCI_CLK_OUT[0:4], for external PCI agents.

When the device comes out of reset, the PCI clock outputs are disabled and are actively driven to a steady low state. Each of the individual clock outputs can be enabled (enable toggling of the clock) by setting its corresponding OCCR[PCICOE*n*] bit. All output clocks are phase aligned to each other and to PCI_SYNC_OUT.

4.4.2 Clocking In PCI Agent Mode

When the device is configured as a PCI agent, PCI_CLK is the primary input clock. In agent mode, the CLKIN signal should be tied to GND, and the clock output signals, PCI_CLK_OUT*n* and PCI_SYNC_OUT, are not used.

In agent mode, the CFG_CLKIN_DIV configuration input can be used to double the internal clock frequencies, if sampled as 1 during PORESET assertion. This feature is useful if a fixed internal frequency is desired regardless of whether the PCI clock is running at 33 or 66 MHz. PCI specifications require that the signal M66EN provides the PCI clock frequency information.

4.4.3 System Clock Domains

As shown in Figure 4-8, the primary clock input (PCI_CLK/PCI_SYNC_IN) frequency is multiplied up by the system phase-locked loop (PLL) and the clock unit to create three major clock domains:

- The coherent system bus clock (*csb_clk*)
- The internal clock for the DDR controller (*ddr_clk*)
- The internal clock for the local bus interface unit (*lbc_clk*)

The *csb_clk* frequency is derived from a complex set of factors that can be simplified into the following equation:

$$csb\_clk = [\text{PCI\_SYNC\_IN} \times (1 + \overline{\text{CFG\_CLKIN\_DIV}})] \times \text{SPMF}$$

In PCI host mode, PCI_SYNC_IN × (1 + $\overline{\text{CFG\_CLKIN\_DIV}}$) is the CLKIN frequency.

The *csb_clk* serves as the clock input to the e300 core. A second PLL inside the core multiplies up the *csb_clk* frequency to create the internal clock for the core (*core_clk*). The system and core PLL multipliers are selected by the SPMF and COREPLL fields in the reset configuration word low (RCWL), which is loaded at power-on reset or by one of the hard-coded reset options. See Section 4.3, “Reset Configuration.”

The internal *ddr_clk* frequency is determined by RCWL[DDRCM].See Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR). Note that *ddr_clk* is not the external memory bus frequency;

ddr_clk passes through the DDR clock divider (÷2) to create the differential DDR memory bus clock outputs (MCK and $\overline{\text{MCK}}$). However, the data rate is the same frequency as *ddr_clk*.

The internal *lbc_clk* frequency is determined by RCWL[LBCM]. See Section 4.3.2.1, "Reset Configuration Word Low Register (RCWLR). Note that *lbc_clk* is not the external local bus frequency; *lbc_clk* passes through the LBC clock divider to create the external local bus clock outputs (LSYNC_OUT and LCLK[0:2]). The LBC clock divider ratio is controlled by LCCR[CLKDIV]. See Section 10.1.3.1, "eLBC Bus Clock and Clock Ratios," for more information.

In addition, some of the internal units may be required to be shut off or operate at lower frequency than the *csb_clk* frequency. These units have a default clock ratio that can be configured by a memory mapped register after the device comes out of reset. Table 4-27 specifies which units have a configurable clock frequency. Refer to Section 4.5.2.3, "System Clock Control Register (SCCR)."

Table 4-27. Configurable Clock Units

Unit	Default Frequency	Options
eTSEC1 and eTSEC2	*csb_clk/3*	Off, *csb_clk, csb_clk/2, csb_clk/3*
Security core	*csb_clk*/3	Off, *csb_clk, csb_clk*/2, *csb_clk*/3
USB DR	*csb_clk*/3	Off, *csb_clk, csb_clk*/2, *csb_clk*/3
PCI and DMA complex	*csb_clk*	Off, *csb_clk*
PCIEXP1 and PCIEXP2	*csb_clk*/3	Off, *csb_clk*, *csb_clk*/2, *csb_clk*/3
eSDHC	*csb_clk*/3	Off, *csb_clk*, *csb_clk*/2, *csb_clk*/3
SATA1/SATA2/SATA3/SATA4	*csb_clk*/3	Off, *csb_clk*, *csb_clk*/2, *csb_clk*/3

NOTE

The clock ratios of these units must be set before they are accessed.

4.5 Memory Map/Register Definitions

This section presents the memory maps and register descriptions for both reset and clocking.

4.5.1 Reset Configuration Register Descriptions

The reset configuration and status registers are shown in Table 4-28.

Table 4-28. Reset Configuration and Status Registers Memory Map

Address	Register	Access	Reset	Section/Page
0x0_0900	Reset configuration word low register (RCWLR)	R	0x0000_0000	4.5.1.1/4-35
0x0_0904	Reset configuration word high register (RCWHR)	R	0x0000_0000	4.5.1.2/4-35
0x0_0908	Reserved, should be cleared	—	—	—
0x0_090C	Reserved, should be cleared	—	—	—
0x0_0910	Reset status register (RSR)	R/W	0x0000_0000	4.5.1.3/4-35

Table 4-28. Reset Configuration and Status Registers Memory Map (continued)

Address	Register	Access	Reset	Section/Page
0x0_0914	Reset mode register (RMR)	R/W	0x0000_0000	4.5.1.4/4-37
0x0_0918	Reset protection register (RPR)	R/W	0x0000_0000	4.5.1.5/4-37
0x0_091C	Reset control register (RCR)	R/W	0x0000_0000	4.5.1.6/4-38
0x0_0920	Reset control enable register (RCER)	R/W	0x0000_0000	4.5.1.7/4-39
0x0_0924– 0x0_09FC	Reserved, should be cleared.	—	—	—

4.5.1.1 Reset Configuration Word Low Register (RCWLR)

The reset configuration word low register (RCWLR) is shown in Figure 4-3 and described in Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR).”

4.5.1.2 Reset Configuration Word High Register (RCWHR)

The reset configuration word high register (RCWHR) is shown in Figure 4-4 and described in Section 4.3.2.2, “Reset Configuration Word High Register (RCWHR).”

4.5.1.3 Reset Status Register (RSR)

RSR, shown in Figure 4-9, captures various reset events in the device.The RSR accumulates reset events. For example, because software watchdog expiration results in a hard reset, SWRS and HRS are all set after a software watchdog reset. This register returns to its reset value only when power-on reset occurs.

Address 0x0_0910 Access: User read/write

Bits	0–3	4–14											15
R/W	RSTSRC	—											BSF
Reset	n[1]	0	0	0	0	0	0	0	0	0	0	0	0

Bits	16–17	18	19	20–22	23	24–26	27	28	29	30	31
R/W	—	SWSR	SWHR	—	JSRS	—	CSHR	SWRS	BMRS	SRS	HRS
Reset	All zeros										

[1] The reset value of this field is determined according to the reset configuration input signals CFG_RESET_SOURCE[0:3] sampled during the reset flow.

Figure 4-9. Reset Status Register (RSR)

Table 4-29 defines the reset status register bit fields.

Table 4-29. Reset Status Register Field Descriptions

Bits	Name	Description
0–3	RSTSRC	Reset configuration word source. Reflects the value of CFG_RESET_SOURCE input signal during the reset flow. See Section 4.3.1.1, "Reset Configuration Word Source," on page 4-10. Changing this field has no effect.
4–14	—	Reserved, should be cleared.
15	BSF	Boot sequencer fail. If set, indicates that the I^2C boot sequencer has failed while loading the reset configuration words. Cleared by writing a 1 to it (writing zero has no effect).
16–17	—	Reserved, should be cleared.
18	SWSR	Software soft reset. If set, indicates that a software soft reset has occurred. Cleared by writing a 1 to it (writing zero has no effect).
19	SWHR	Software hard reset. If set, indicates a software hard reset. SWHR is cleared by writing a 1 to it (writing zero has no effect).
20–22	—	Reserved, should be cleared.
23	JSRS	JTAG soft reset status. Set when the JTAG reset request is set and remains set until software clears it. JSRS is cleared by writing a 1 to it (writing zero has no effect). 0 No JTAG reset event. 1 JTAG reset event.
24–26	—	Reserved, should be cleared.
27	CSHR	Check stop reset status. When the core enters a checkstop state and the checkstop reset is enabled by the RMR[CSRE], CSRS is set and it remains set until software clears it. CSRS is cleared by writing a 1 to it (writing zero has no effect). 0 No enabled check stop reset event. 1 Enabled check stop reset event.
28	SWRS	Software watchdog reset status. When a software watchdog expire event (which causes a reset) is detected, SWRS is set and remains that way until the software clears it. SWRS is cleared by writing a 1 to it (writing zero has no effect). 0 No software watchdog reset event. 1 Software watchdog reset event.
29	BMRS	Bus monitor reset status. When a bus monitor expire event (which causes a reset) is detected, BMRS is set and remains set until the software clears it. BMRS can be cleared by writing a 1 to it (writing zero has no effect). 0 No bus monitor reset event. 1 Bus monitor reset event.
30	SRS	Soft reset status. When an external or internal soft reset event is detected, SRS is set and remains set until software clears it. SRS is cleared by writing a 1 to it (writing zero has no effect). 0 No soft reset event. 1 Soft reset event.
31	HRS	Hard reset status. When an external or internal hard reset event is detected, HRS is set and remains set until software clears it. HRS is cleared by writing a 1 (writing zero has no effect). 0 No hard reset event. 1 Hard reset event.

4.5.1.4 Reset Mode Register (RMR)

RMR, shown in Figure 4-10, enables a hard reset sequence on the device when the e300 core enters checkstop state.

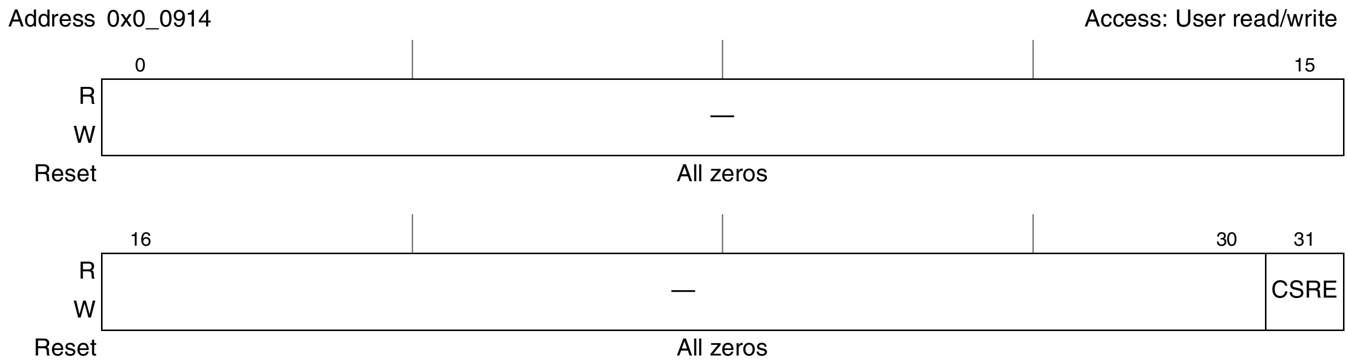


Figure 4-10. Reset Mode Register (RMR)

Table 4-30 describes the RMR fields.

Table 4-30. RMR Field Descriptions

Bits	Name	Function
0–30	—	Reserved, should be cleared.
31	CSRE	Checkstop reset enable. The core can enter checkstop mode as the result of several exception conditions. Setting CSRE configures the device to perform a hard reset sequence when the core enters checkstop state. 0 Reset not generated when core enters checkstop state. 1 Reset generated when core enters checkstop state.

4.5.1.5 Reset Protection Register (RPR)

RPR, shown in Figure 4-11, prevents unintended software reset requests caused by writes to the reset control register (RCR). To disable a write to the reset control register (RCR), the user should write a 1 to RCER[CRE].

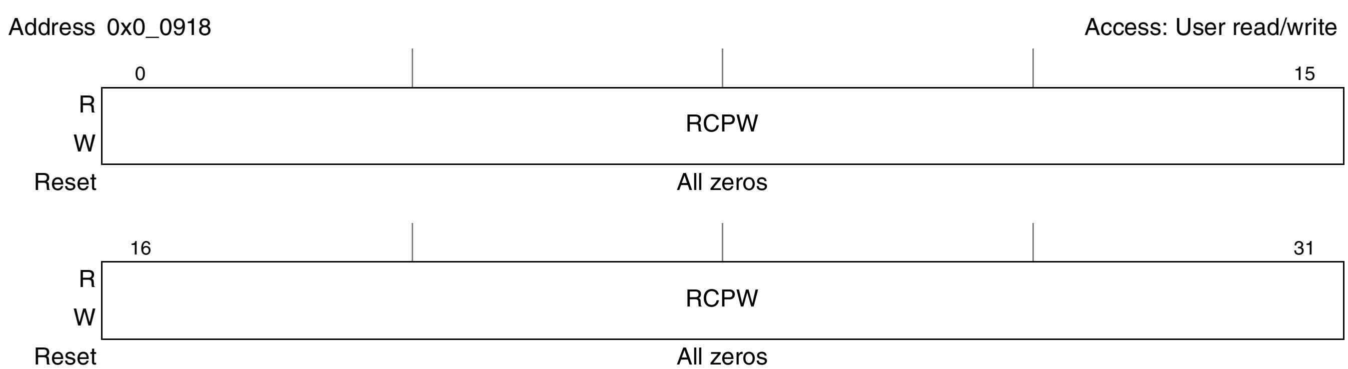


Figure 4-11. Reset Protection Register (RPR)

Table 4-31 defines the bit fields of RPR.

Table 4-31. RPR Bit Descriptions

Bits	Name	Description
0–31	RCPW	Reset control protection word. Prevents unintended software reset requests because of a write to the RCR. The user should write the value 0x5253_5445 (RSTE in ASCII) to enable. Enable indication appears in the reset control enable register (RCER[CRE]). Reading this register always returns all zeros.

4.5.1.6 Reset Control Register (RCR)

RCR, shown in Figure 4-12, can be used by software to initiate a soft or hard reset sequence. To allow writing to this register, the user must enable it by writing the value 0x5253_5445 to the RPR.

Address 0x0_091C — Access: User read/write

	0–15
R	—
W	
Reset	All zeros

	16–29	30	31
R	—	SWHR	SWSR
W			
Reset	All zeros		

Figure 4-12. Reset Control Register (RCR)

Table 4-32 defines the bit fields of RCR.

Table 4-32. RCR Bit Settings

Bits	Name	Description
0–29	—	Reserved, should be cleared.
30	SWHR	Software hard reset. Setting this bit causes the device to begin a hard reset flow. This bit returns to its reset state during the reset sequence, so reading it always returns all zeros.
31	SWSR	Software soft reset. Setting this bit causes the device to begin a soft reset flow. This bit returns to its reset state during the reset sequence, so reading it always returns 0. H

4.5.1.7 Reset Control Enable Register (RCER)

RCER, shown in Figure 4-13, indicates by the CRE field that the RPR is accessed with a value that enables RCR.

Address 0x0_0920 Access: User read/write

	0–15
R W	—
Reset	All zeros

	16–30	31
R W	—	CRE
Reset	All zeros	

Figure 4-13. Reset Control Enable Register (RCER)

Table 4-33 defines the bit fields of RCER.

Table 4-33. RCER Bit Settings

Bits	Name	Description
0–30	—	Reserved, should be cleared.
31	CRE	Control register enabled. When set, indicates that the RPR was accessed with a value that enables the RCR. Writing 1 to this bit disables the RCR and clears this bit. Writing zero has no effect.

4.5.2 Clock Configuration Registers

The clock configuration and status registers are shown in Table 4-34.

Table 4-34. Clock Configuration Registers Memory Map

Address	Register	Access	Reset	Section/Page
0x0_0A00	System PLL mode register (SPMR)	R	0x*nnnn_nnnn*	4.5.2.1/4-39
0x0_0A04	Output clock control register (OCCR)	R/W	0x0000_FFF8	4.5.2.2/4-41
0x0_0A08	System clock control register (SCCR)	R/W	0xFFFF_FFFF	4.5.2.3/4-42
0x0_0A0C–0x0_0AFC	Reserved, should be cleared	—	—	—

4.5.2.1 System PLL Mode Register (SPMR)

SPMR is shown in Figure 4-14, gets its values according to the CFG_CLKIN_DIV reset configuration input signal and the reset configuration word low loaded during the reset flow. Note that this register is updated only during a power-on reset sequence and not by a hard reset sequence. It may hold values different than those in the RCWLR after a a hard reset sequence.

Address 0x0_0A00 Access: Read only

	0	1	2	3	4			7	8	9						15
R	LBCM	DDRCM	—		SPMF				CKID	COREPLL						
W																
Reset[1]	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

	16															31
R	—															
W																
Reset	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n	n

Figure 4-14. System PLL Mode Register

[1] See Table 4-35 for reset values.

Table 4-35 defines the system PLL mode register bit fields.

Table 4-35. System PLL Mode Register Bit Settings

Bits	Name	Meaning	Description
0	LBCM	Local bus memory controller clock mode.	Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR)”
1	DDRCM	DDR SDRAM memory controller clock mode.	Section 4.3.2.1, “Reset Configuration Word Low Register (RCWLR)”
2–3	—	Reserved, should be cleared.	—
4–7	SPMF	System PLL multiplication factor	Section 4.3.2.1.2, “System PLL Configuration”
8	CKID	CLKIN division factor. Reflects the value of CFG_CLKIN_DIV input signal during the reset flow.	Section 4.3.1.2, “CLKIN Division”
9–15	COREPLL	Core PLL configuration.	See the hardware specifications for this device
16–31	—	Reserved, should be cleared.	—

4.5.2.2 Output Clock Control Register (OCCR)

The OCCR shown in Figure 4-15, controls the device output clocks. It is possible to control some output clock modes by writing to this memory mapped register as described below.

Address 0x0_0A04 Access: Read/Write

	0	1	2	3	4	5–15
R/W	PCICOE0	PCICOE1	PCICOE2	PCICOE3	PCICOE4	—
Reset	All zeros					

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	MCK0E, MCK_B0E	MCK1E, MCK_B1E	MCK2E, MCK_B2E	MCK3E, MCK_B3E	MCK4E, MCK_B4E	MCK5E, MCK_5E	—		LCLK0OE	LCLK1OE	LCLK2OE	LSYNC_OUTE	—			
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

Figure 4-15. Output Clock Control Register (OCCR)

Table 4-36 defines the bit fields of OCCR.

Table 4-36. OCCR Bit Settings

Bits	Name	Description
0	PCICOE0	PCI_CLK_OUT0 enable. 0 PCI_CLK_OUT0 signal is disabled (drive constant zero). 1 PCI_CLK_OUT0 signal is enabled to toggle.
1	PCICOE1	PCI_CLK_OUT1 enable. 0 PCI_CLK_OUT1 signal is disabled (drive constant zero). 1 PCI_CLK_OUT1 signal is enabled to toggle.
2	PCICOE2	PCI_CLK_OUT2 enable. 0 PCI_CLK_OUT2 signal is disabled (drive constant zero). 1 PCI_CLK_OUT2 signal is enabled to toggle.
3	PCICOE3	PCI_CLK_OUT3 enable. 0 PCI_CLK_OUT3 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT3 signal is enabled to toggle.
4	PCICOE4	PCI_CLK_OUT4 enable. 0 PCI_CLK_OUT4 signal is disabled (drive constant ‘0’). 1 PCI_CLK_OUT4 signal is enabled to toggle.
5–15	—	Reserved, should be cleared
16	MCK0E, MCK_B0E	Enable/Disable MCK[0] pins clock out 0 Disable MCK[0] and $\overline{\text{MCK}}$[0] 1 Enable MCK[0] and $\overline{\text{MCK}}$[0]
17	MCK1E, MCK_B1E	Enable/Disable MCK[1] pins clock out 0 Disable MCK[1] and $\overline{\text{MCK}}$[1] 1 Enable MCK[1] and $\overline{\text{MCK}}$[1]
18	MCK2E, MCK_B2E	Enable/Disable MCK[2] pins clock out 0 Disable MCK[2] and $\overline{\text{MCK}}$[2] 1 Enable MCK[2] and $\overline{\text{MCK}}$[2]

Table 4-36. OCCR Bit Settings (continued)

Bits	Name	Description
19	MCK3E, MCK_B3E	Enable/Disable MCK[3] pins clock out 0 Disable MCK[3] and $\overline{\text{MCK}}$[3] 1 Enable MCK[3] and $\overline{\text{MCK}}$[3]
20	MCK4E, MCK_B4E	Enable/Disable MCK[4] pins clock out 0 Disable MCK[4] and $\overline{\text{MCK}}$[4] 1 Enable MCK[4] and $\overline{\text{MCK}}$[4]
21	MCK5E, MCK_B5E	Enable/Disable MCK[5] pins clock out 0 Disable MCK[5] and $\overline{\text{MCK}}$[5] 1 Enable MCK[5] and $\overline{\text{MCK}}$[5]
22–23	—	Reserved
24	LCLK0E	Enable/Disable LCLK[0] pin clock out 0 Disable LCLK[0] 1 Enable LCLK[0]
25	LCLK1E	Enable/Disable LCLK[1] pin clock out 0 Disable LCLK[1] 1 Enable LCLK[1]
26	LCLK2E	Enable/Disable LCLK[2] pin clock out 0 Disable LCLK[2] 1 Enable LCLK[2]
27	LSYNC_OUTE	Enable/Disable LSYNC_OUT pin clock out 0 Disable LSYNC_OUT 1 Enable LSYNC_OUT
28–31	—	Reserved

4.5.2.3 System Clock Control Register (SCCR)

SCCR, shown in Figure 4-16, controls device units that have a configurable clock ratio.

Address 0x0_0A08 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	TSEC1CM		TSEC2CM		SDHCCM		ENCCM		USBDRCM		PCIEXP1CM		PCIEXP2CM		—	PCICM
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	—								SATA1CM		SATA2CM		SATA3CM		SATA4CM	
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 4-16. System Clock Control Register (SCCR)

Table 4-37 defines the bit fields of SCCR.

Table 4-37. SCCR Bit Descriptions

Bits	Name	Description
0–1	TSEC1CM	TSEC1 clock mode. 00 TSEC1 clock is disabled 01 TSEC1 clock/*csb_clk* ratio is 1:1. 10 TSEC1 clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than TSEC1). 11 TSEC1 clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than TSEC1).
2–3	TSEC2CM	TSEC2 clock mode. 00 TSEC2 clock is disabled. 01 TSEC2 clock/*csb_clk* ratio is 1:1. 10 TSEC2 clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than TSEC2). 11 TSEC2 clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than TSEC2). **Note:** The TSEC2 unit must have the same clock ratio as the TSEC1 unit, unless one of them has its clock disabled.
4–5	SDHCCM	SDHC clock mode. 00 SDHC and I^2C1 clock is disabled. 01 SDHC and I^2C1 clock/*csb_clk* ratio is 1:1. 10 SDHC and I^2C1 clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than SDHC and I^2C1). 11 SDHC and I^2C1 clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than SDHC and I^2C1).
6–7	ENCCM	Encryption core and I^2C1 clock mode. 00 Encryption core clock is disabled. 01 Encryption core clock/*csb_clk* ratio is 1:1. 10 Encryption core clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than the encryption core). 11 Encryption core clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than the encryption core).
8–9	USB DRCM	USB DR clock mode. 00 USB DR clock is disabled. 01 USB DR clock/*csb_clk* ratio is 1:1. 10 USB DR clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than the USB DR). 11 USB DR clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than the USB DR). **Note:** The USB DR unit must have the same clock ratio as the encryption core unit, unless one of them has its clock disabled.
10–11	PCIEXP1 CM	PCIEXP1 clock mode. Define the clock mode for the PCI Express 1 controller. 00 PCIEXP1 clock is disabled. 01 PCIEXP1 clock/*csb_clk* ratio is 1:1. 10 PCIEXP1 clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than the PCIEXP1 clock). 11 PCIEXP1 clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than the PCIEXP1 clock).
12–13	PCIEXP2 CM	PCIEXP2 clock mode. Define the clock mode for the PCI Express 2 controller. 00 PCIEXP2 clock is disabled. 01 PCIEXP2 clock/*csb_clk* ratio is 1:1. 10 PCIEXP2 clock/*csb_clk* ratio is 1:2 (*csb_clk* has higher frequency than the PCIEXP2 clock). 11 PCIEXP2 clock/*csb_clk* ratio is 1:3 (*csb_clk* has higher frequency than the PCIEXP2 clock).
14	—	Reserved
15	PCICM	PCI clock mode. Define the clock mode for all of the PCI complex - PCI and DMA. 0 PCI complex clocks are disabled. 1 PCI complex clocks are enabled.
16–23	—	Reserved

Table 4-37. SCCR Bit Descriptions (continued)

Bits	Name	Description
24 –25	SATA1CM	SATA1 Clock mode. Defines the clock mode for SATA1 controller 00 SATA1 clock is disabled 01 SATA1 clock: csb_clk ratio is 1:1 10 SATA1 clock: csb_clk ratio is 1:2 (csb_clk has higher frequency than SATA1 clock) 11 SATA1 clock: csb_clk ratio is 1:3 (csb_clk has higher frequency than SATA1 clock)
26–27	SATA2CM	SATA2 Clock mode. Defines the clock mode for SATA2 controller 00 SATA2 clock is disabled 01 SATA2 clock: csb_clk ratio is 1:1 10 SATA2 clock: csb_clk ratio is 1:2 (csb_clk has higher frequency than SATA2 clock) 11 SATA2 clock: csb_clk ratio is 1:3 (csb_clk has higher frequency than SATA2 clock) **Note:** All SATA controllers must have the same clock ratio, unless one or two or three of them have their clock disabled.
28–29	SATA3CM	SATA3 Clock mode. Defines the clock mode for SATA3 controller 00 SATA3 clock is disabled 01 SATA3 clock: csb_clk ratio is 1:1 10 SATA3 clock: csb_clk ratio is 1:2 (csb_clk has higher frequency than SATA3 clock) 11 SATA3 clock: csb_clk ratio is 1:3 (csb_clk has higher frequency than SATA3 clock) **Note:** All SATA controllers must have the same clock ratio, unless one or two or three of them have their clock disabled.
30–31	SATA4CM	SATA4 Clock mode. Defines the clock mode for SATA4 controller 00 SATA4 clock is disabled 01 SATA4 clock: csb_clk ratio is 1:1 10 SATA4 clock: csb_clk ratio is 1:2 (csb_clk has higher frequency than SATA4 clock) 11 SATA4 clock: csb_clk ratio is 1:3 (csb_clk has higher frequency than SATA4 clock) **Note:** All SATA controllers must have the same clock ratio, unless one or two or three of them have their clock disabled.

Chapter 5
System Configuration

5.1 Introduction

This chapter describes several functions that control the local access windows, system configuration, protection, and general utilities. These functions are discussed in the following sections:

- Section 5.2, “Local Memory Map Overview and Example”
- Section 5.3, “System Configuration”
- Section 5.4, “Software Watchdog Timer (WDT)”
- Section 5.5, “Real Time Clock Module (RTC)”
- Section 5.6, “Periodic Interval Timer (PIT)”
- Section 5.7, “General-Purpose Timers (GTMs)”
- Section 5.8, “Power Management Control (PMC)”

5.2 Local Memory Map Overview and Example

The device provides a flexible local memory map. The local memory map refers to the 32-bit address space seen by the processor as it accesses memory and I/O space. Internal DMA engines also see this same local memory map. All memory accessed by the DDR SDRAM and local bus memory controllers exists in this memory map, as do all memory-mapped configuration, control, and status registers.

The local memory map is defined by a set of eleven local access windows. Each of these windows maps a region of memory to a particular target interface, such as the DDR SDRAM controller or the PCI controller. The DSP subsystem is not operational in the MSC7104. Note that the local access windows do not perform any address translation. The size of each window can be configured from 4 Kbytes to 2 Gbytes. Each local access window is assigned to a specific target interface as specified in Table 5-1.

Table 5-1. Local Access Windows Target Interface

Window Number	Target Interface	Comments
0	Configuration registers (IMMR)	Fixed 1-Mbyte window size
1	Local bus	—
2	Local bus	—
3	Local bus	—
4	Local bus	—
5	PCI	—
6	PCI	—

Table 5-1. Local Access Windows Target Interface (continued)

Window Number	Target Interface	Comments
7	DDR SDRAM	—
8	DDR SDRAM	—
9	PCI Express 1	—
10	PCI Express 2	—

Figure 5-1 shows an example memory map.

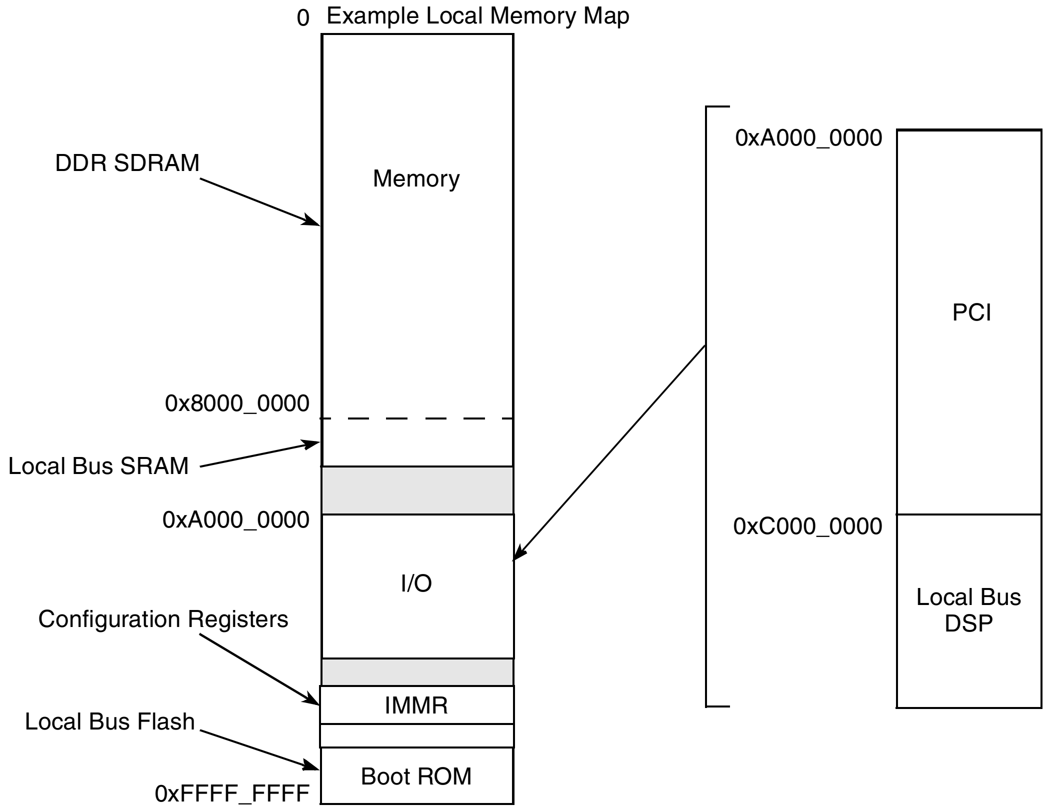


Figure 5-1. Local Memory Map Example

Table 5-2 shows one example of local access window settings.

Table 5-2. Local Access Windows Example

Window	Base Address	Size	Target Interface
7	0x0000_0000	2 Gbytes	DDR SDRAM
2	0x8000_0000	1 Mbyte	Local bus
5	0xA000_0000	512 Mbytes	PCI
9	0x8100_0000	1 Mbyte	PCI Express 1
10	0x8200_0000	1 Mbyte	PCI Express 2
3	0xC000_0000	256 Mbytes	Local bus

Table 5-2. Local Access Windows Example (continued)

Window	Base Address	Size	Target Interface
0	0xFF40_0000	1 Mbyte	Configuration registers (IMMR)
1	0xFF80_0000	8 Mbytes	Local bus boot ROM Flash
4, 6, 8	Unused		

In this example, the local access window of the boot ROM is defined as window number 1, on a local bus device, in the highest 8 Mbytes of memory as set by the reset configuration word high during the reset sequence (see Section 4.3.2.2.4, "Boot ROM Location") and Section 5.2.4.3.1, "LBLAWBAR0[BASE_ADDR] Reset Value." The local access window, which describes the range of memory used for memory-mapped registers (IMMR), is a fixed 1-Mbyte space pointed to by the IMMRBAR register, using its default value (0xFF40_0000). See Section 5.2.4.1, "Internal Memory Map Registers Base Address Register (IMMRBAR)."

5.2.1 Address Translation and Mapping

In addition to any address translation performed by the e300c4s core MMU, three distinct types of translation and mapping operations are performed on transactions at the integrated device level. These are as follows:

- Mapping a local address to a target interface
- Translating the local 32-bit address to an external address space
- Translating external addresses to the local 32-bit address space

The local access windows perform target mapping for transactions within the local address space. The local access windows do not perform any address translation.

Outbound windows perform the mapping from the local 32-bit address space to the address space of PCI, which may be much larger than the local space.

Inbound windows perform address translation from the external address spaces of PCI to the local address space.

The target mappings created by an inbound window must be consistent with those of the local access windows. That is, if an inbound window maps a transaction to a given local address, a valid local access window for that address must be set independently.

All of the configuration registers that define mapping of local access windows follow the same register format. Table 5-3 summarizes the general format of these window definitions.

Table 5-3. Format of Window Definitions

Register	Function
Base address	High-order address bits defining location of the window in the initial address space
Window size/attributes	Window enable, window size[1]

[1] An exception is the IMMR window, which is always enabled and has a fixed 1-Mbyte size.

Windows must be a power-of-two size. To perform a mapping function, the address of the transaction is compared with the base address register of each window. The number of bits used in the comparison is dictated by each window's size attribute. When an address hits within a window, the transaction is directed to the appropriate target.

5.2.2 Window into Configuration Space

The internal memory map registers' base address register (IMMRBAR) defines a window that is used to access all memory-mapped configuration, control, and status registers, referred to as internal memory map registers or IMMR. This window is always enabled with a fixed size of 1 Mbyte, and no other attributes are attached so there is no associated size/attribute register. This window always takes precedence over all local access windows. The IMMRBAR always come out of reset with a default base address value of 0xFF40_0000, and this base address value can be modified by writing to this register. The only exception is the case when ROMLOC is defined as boot from on-chip ROM. For more information, see Section 5.2.4.1, "Internal Memory Map Registers Base Address Register (IMMRBAR)."

NOTE

Although it is legal to use the 3-Mbyte space consecutive to the 1 Mbyte of the IMMR (for example, if IMMRBAR is 0xFF40_0000, the 3-Mbyte address space consecutive to it is 0xFF50_0000–0xFF7F_FFFF), it is not recommended. This space may be used in future derivatives of the device that require a larger internal memory space.

5.2.3 Local Access Windows

As demonstrated in the address map overview in Section 5.2, "Local Memory Map Overview and Example," local access windows associate a range of the local 32-bit address space with a particular target interface. This allows the internal interconnections of the device to route a transaction from its source to the proper target. No address translation is performed. The base address defines the high order address bits that give the location of the window in the local address space. The window attributes enable the window and define its size, while the window number specifies the target interface.

With the exception of configuration space (mapped by IMMRBAR), all addresses used by the system must be mapped by a local access window. This includes addresses that are mapped by PCI inbound windows.

The local access window registers exist as part of the local access block in the system configuration registers. See Section 5.3.2, "System Configuration Registers." A detailed description of the local access window registers is given in the following sections. Note that the minimum size of a window is 4 Kbytes, so the low order 12 bits of the base address cannot be specified.

5.2.3.1 Local Access Register Memory Map

Table 5-4 shows the memory map for the local access registers.

Table 5-4. Local Access Register Memory Map

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_0000	Internal memory map base address register (IMMRBAR)	R/W	0xFF40_0000[1]	5.2.4.1/5-6
0x0_0004	Reserved	—	—	—
0x0_0008	Alternate configuration base address register (ALTCBAR)	R/W	0x0000_0000	5.2.4.2/5-8
0x0_000C–0x0_001C	Reserved	—	—	—
0x0_0020	eLBC local access window 0 base address register (LBLAWBAR0)	R/W	0x0000_0000[2]	5.2.4.3/5-9
0x0_0024	eLBC local access window 0 attribute register (LBLAWAR0)	R/W	0x0000_0000[3]	5.2.4.4/5-10
0x0_0028	eLBC local access window 1 base address register (LBLAWBAR1)	R/W	0x0000_0000	5.2.4.3/5-9
0x0_002C	eLBC local access window 1 attribute register (LBLAWAR1)	R/W	0x0000_0000	5.2.4.4/5-10
0x0_0030	eLBC local access window 2 base address register (LBLAWBAR2)	R/W	0x0000_0000	5.2.4.3/5-9
0x0_0034	eLBC local access window 2 attribute register (LBLAWAR2)	R/W	0x0000_0000	5.2.4.4/5-10
0x0_0038	eLBC local access window 3 base address register (LBLAWBAR3)	R/W	0x0000_0000	5.2.4.3/5-9
0x0_003C	eLBC local access window 3 attribute register (LBLAWAR3)	R/W	0x0000_0000	5.2.4.4/5-10
0x0_0040–0x0_005C	Reserved	—	—	—
0x0_0060	PCI local access window 0 base address register (PCILAWBAR0)	R/W	0x0000_0000[4]	5.2.4.5/5-11
0x0_0064	PCI local access window 0 attribute register (PCILAWAR0)	R/W	0x0000_0000[5]	5.2.4.6/5-12
0x0_0068	PCI local access window 1 base address register (PCILAWBAR1)	R/W	0x0000_0000[6]	5.2.4.5/5-11
0x0_006C	PCI local access window 1 attribute register (PCILAWAR1)	R/W	0x0000_0000	5.2.4.6/5-12
0x0_0070–0x0_007C	Reserved	—	—	—
0x0_0080	PCI Express 1 local access window base address register (PCIEXP1LAWBAR)	R/W	0x0000_0000	5.2.4.7/5-13
0x0_0084	PCI Express 1 local access window attribute register (PCIEXP1LAWAR)	R/W	0x0000_0000	5.2.4.8/5-13
0x0_0088	PCI Express 2 local access window base address register (PCIEXP2LAWBAR)	R/W	0x0000_0000	5.2.4.9/5-14
0x0_008C	PCI Express 2 local access window attribute register (PCIEXP2LAWAR)	R/W	0x0000_0000	5.2.4.10/5-15
0x0_0090–0x0_009C	Reserved	—	—	—
0x0_00A0	DDR local access window 0 base address register (DDRLAWBAR0)	R/W	0x0000_0000[7]	5.2.4.11/5-15
0x0_00A4	DDR local access window 0 attribute register (DDRLAWAR0)	R/W	0x0000_0000[8]	5.2.4.12/5-16
0x0_00A8	DDR local access window 1 base address register (DDRLAWBAR1)	R/W	0x0000_0000	5.2.4.11/5-15

Table 5-4. Local Access Register Memory Map (continued)

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x0_00AC	DDR local access window 1 attribute register (DDRLAWAR1)	R/W	0x0000_0000	5.2.4.12/5-16
0x0_00B0–0x0_00FC	Reserved	—	—	—

1 Depends on reset configuration word high values. See Section 5.2.4.1.1, "Updating IMMRBAR," for details.
2 Depends on reset configuration word high values. See Section 5.2.4.3.1, "LBLAWBAR0[BASE_ADDR] Reset Value," for details.
3 Depends on reset configuration word high values. See Section 5.2.4.4.1, "LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value," for details.
4 Depends on reset configuration word high values. See Section 5.2.4.5.1, "PCILAWBAR0[BASE_ADDR] Reset Value," for details.
5 Depends on reset configuration word high values. See Section 5.2.4.11.1, "DDRLAWBAR0[BASE_ADDR] Reset Value," for details.
6 Depends on reset configuration word high values. See Section 5.2.4.6.1, "PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value," for details.
7 Depends on reset configuration word high values. See Section 5.2.4.11.1, "DDRLAWBAR0[BASE_ADDR] Reset Value," for details.
8 Depends on reset configuration word high values. See Section 5.2.4.12.1, "DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value," for details.

5.2.4 Local Access Register Descriptions

This section describes the local access registers.

5.2.4.1 Internal Memory Map Registers Base Address Register (IMMRBAR)

The IMMR window contains configuration, control, and status registers, as well as internal device memory arrays. The internal memory map occupies a 1-Mbyte region of memory space. Its location is programmable using the internal memory map register (IMMR). The default base address for the internal memory map register is 0xFF40_0000.The only exception is the case when ROMLOC is defined as boot from on-chip ROM. See Section 5.2.4.1.2, "IMMRBAR[BASE_ADDR] Reset Value," for a detailed description. Because IMMRBAR is at offset 0x0 from the beginning of the local access registers, IMMRBAR always points to itself.

5.2.4.1.1 Updating IMMRBAR

Updates to IMMRBAR that relocate the entire 1-Mbyte region of the internal memory block require special treatment. The effect of the update must be guaranteed to be visible by the mapping logic before an access to the new location is seen. To make sure this happens, the following guidelines should be followed:

- IMMRBAR should be updated during initial configuration of the device when only one host or controller has access to the device as follows:
  - If an external host on PCI is configuring the device, it should set IMMRBAR to the desired final location before the e300c4s core is released to boot.
  - If the core is initializing the device, it should set IMMRBAR to the desired final location before enabling other I/O devices to access the device.

- When the e300 core is writing to IMMRBAR, it should use the following sequence:
  - Read the current value of IMMRBAR using a load word instruction followed by an **isync**. This forces all accesses to configuration space to complete.
  - Write the new value to IMMRBAR.
  - Perform a load of an address that does not access configuration space or the on-chip SRAM, but has an address mapping already in effect (for example, boot ROM). Follow this load with an **isync**.
  - Read the contents of IMMRBAR from its new location, followed by another **isync**.

The IMMRBAR is shown in Figure 5-2.

Offset 0x00 Access: User Read/Write

	0											11	12			15
R/W	BASE_ADDR												—			
Reset	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0

	16–31
R/W	—
Reset	All zeros

1 Reset value of BASE_ADDR depends on the setting of RCWH[RLEXT] and RCWH[ROMLOC]. See Section 5.2.4.1.2, "IMMRBAR[BASE_ADDR] Reset Value for a detailed description.

Figure 5-2. Internal Memory Map Registers' Base Address Register (IMMRBAR)

Table 5-5 defines the bit fields of IMMRBAR.

Table 5-5. IMMRBAR Bit Settings

Bits	Name	Description
0–11	BASE_ADDR	Identifies the 12 most-significant address bits of the base of the 1-Mbyte internal memory window.
12–31	—	Reserved. Software must write all zeros.

5.2.4.1.2 IMMRBAR[BASE_ADDR] Reset Value

In most cases, the IMMRBAR[BASE_ADDR] reset value is set to 0xFF4. However, if RCWH[RLEXT] and RCWH[ROMLOC] are set such that the on-chip ROM is the boot target, the IMMRBAR[BASE_ADDR] reset value will be set to 0xFFF. By doing so, and together with the reset value of the SGPRL register, the e300 CPU is able to fetch code directly from the on-chip ROM. The SGPRL is mapped to 0xFFF0_0100, which is the initial boot code execution of the e300 CPU when

RCWH[BMS] is set, and the value of SGPRL is coded with a branch instruction to the on-chip ROM starting address.

Table 5-6 defines the reset value of LBLAWBAR0[BASE_ADDR].

Table 5-6. IMMRBAR[BASE_ADDR] Reset Value

RCWH[ROMLOC]/RCWH[RLEXT]	BASE_ADDR Reset Value
011/00	0xFFF
else	0xFF4

5.2.4.2 Alternate Configuration Base Address Register (ALTCBAR)

The alternate configuration base address register (ALTCBAR) is used to define the base address for an alternate 1-Mbyte region of configuration space to be used by the boot sequencer. By loading the proper boot sequencer command in the serial ROM, the base address in the ALTCBAR can be combined with the 20 bits of address offset supplied from the serial ROM to generate a 32-bit address. Thus, by configuring this register, the boot sequencer has access to the entire memory map, one 1-Mbyte block at a time. See Section 21.4.5, “Boot Sequencer Mode,” for more information.

NOTE

ALTCBAR is not considered a local access window on its own, so the boot sequencer must configure one of the other eight local access windows properly to reach the desired target peripherals.

The alternate configuration base address register is shown in Figure 5-3.

Offset 0x08 Access: Read/Write

	0–11	12–31
R W	BASE_ADDR	—
Reset	All zeros	

Figure 5-3. Alternate Configuration Base Address Register (ALTCBAR)

Table 5-7 defines the bit fields of ALTCBAR.

Table 5-7. ALTCBAR Bit Settings

Bits	Name	Description
0–11	BASE_ADDR	Identifies the12 most-significant address bits of an alternate base address used for boot sequencer configuration accesses.
12–31	—	Reserved. Write has no effect, read returns 0.

5.2.4.3 LBC Local Access Window *n* Base Address Registers (LBLAWBAR0–LBLAWBAR3)

The LBC local access window *n* base address registers (LBLAWBAR0–LBLAWBAR3) are shown in Figure 5-4.

Offset 0x20, 0x28, 0x30, 0x38 — Access: Read/Write

	0–19	20–31
R/W	BASE_ADDR	—
Reset	All zeros	

1. The LBLAWBAR0[BASE_ADDR] reset value depends on the reset configuration word high values. See Section 5.2.4.3.1, "LBLAWBAR0[BASE_ADDR] Reset Value," for a detailed description.

Figure 5-4. LBC Local Access Window *n* Base Address Registers (LBLAWBAR0–LBLAWBAR3)

Table 5-8 defines the bit fields of LBLAWBAR0–LBLAWBAR3.

Table 5-8. LBLAWBAR0–LBLAWBAR3 Bit Settings

Bits	Name	Description
0–19	BASE_ADDR	Identifies the 20 most-significant address bits of the base of local access window *n*. The specified base address should be aligned to the window size, as defined by LBLAWARn[SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

5.2.4.3.1 LBLAWBAR0[BASE_ADDR] Reset Value

The core may also use a local bus peripheral device to fetch its boot vector. For this purpose, the LBLAWBAR0[BASE_ADDR] reset value is set according to the value set in the reset configuration word high BMS field.

Table 5-9 defines the reset value of LBLAWBAR0[BASE_ADDR].

Table 5-9. LBLAWBAR0[BASE_ADDR] Reset Value

RCWHR[BMS]	BASE_ADDR Reset Value
0	0x00000
1	0xFF800

5.2.4.4 LBC Local Access Window *n* Attributes Registers (LBLAWAR0–LBLAWAR3)

The LBC local access window *n* attributes registers (LBLAWAR0–LBLAWAR3) are shown in Figure 5-5.

Offset 0x24, 0x2C, 0x34, 0x3C | Access: Read/Write

	0	1–25	26–31
R / W	EN	—	SIZE
Reset	0[1]	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0[2]

1 The LBLAWAR0[EN] reset value depends on the reset configuration word high values. See Section 5.2.4.4.1, "LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value," for a detailed description.

2 The LBLAWAR0[SIZE] reset value is always 0b010110, meaning an 8-Mbyte local access window. See Section 5.2.4.4.1, "LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value," for a detailed description.

Figure 5-5. LBC Local Access Window *n* Attributes Registers (LBLAWAR0–LBLAWAR3)

Table 5-10 defines the bit fields of LBLAWAR0–LBLAWAR3.

Table 5-10. LBLAWAR0–LBLAWAR3 Bit Settings

Bits	Name	Description
0	EN	0 Local bus local access window *n* is disabled. 1 Local bus local access window *n* is enabled and other LBLAWAR0 and LBLAWBAR0 fields combine to identify an address range for this window.
1–25	—	Reserved. Write has no effect, read returns 0.
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes . . . . $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

5.2.4.4.1 LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value

The core may use a local bus peripheral device to fetch its boot vector. For this purpose an 8-Mbyte ($2^{(22+1)}$) local access window is defined by the LBLAWBAR0[SIZE] reset value, and LBLAWAR0 is enabled according to the value set in the reset configuration word high ROMLOCand RLEXT fields.

Table 5-11 defines the reset value for LBLAWAR0[EN].

Table 5-11. LBLAWAR0[EN] Reset Value

RCWHR[RLEXT]/RCWHR[ROMLOC]	LBLAWAR0[EN] Reset Value	Description
00/000–100	0	e300c4s core boot not performed from a local bus device.
10,11/000–111	0	e300c4s core boot not performed from a local bus device.

Table 5-11. LBLAWAR0[EN] Reset Value (continued)

RCWHR[RLEXT]/RCWHR[ROMLOC]	LBLAWAR0[EN] Reset Value	Description
00/101–111	1	e300c4s core boot performed from a local bus device. Local bus 8-Mbyte ($2^{(22+1)}$) local access window is enabled.
01/000–111	1	e300c4s core boot performed from a local bus device. Local bus 8-Mbyte ($2^{(22+1)}$) local access window is enabled.

5.2.4.5 PCI Local Access Window *n* Base Address Register (PCILAWBAR0–PCILAWBAR1)

The PCI local access window *n* base address registers (PCILAWBAR0–PCILAWBAR1) are shown in Figure 5-6.

Offset 0x60
0x68

Access: Read/Write

	0 … 19	20 … 31
R / W	BASE_ADDR	—
Reset	All zeros[1]	

[1] The reset value of PCILAWBAR0[BASE_ADDR] depends on the reset configuration word high values. See Section 5.2.4.5.1, "PCILAWBAR0[BASE_ADDR] Reset Value," for a detailed description.

Figure 5-6. PCI Local Access Window *n* Base Address Registers (PCILAWBAR0–PCILAWBAR1)

Table 5-12 defines the bit fields of PCILAWBAR0–PCILAWBAR1.

Table 5-12. PCILAWBAR0–PCILAWBAR1 Bit Settings

Bits	Name	Description
0–19	BASE_ADDR	Identifies the 20 most-significant address bits of the base of local access window *n*. The specified base address should be aligned to the window size, as defined by PCILAWAR*n*[SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

5.2.4.5.1 PCILAWBAR0[BASE_ADDR] Reset Value

The core may use a PCI peripheral device to fetch its boot vector. For this purpose, the PCILAWBAR0[BASE_ADDR] reset value is set according to the value set in the reset configuration word high BMS field.

Table 5-13 defines the reset value of PCILAWBAR0[BASE_ADDR].

Table 5-13. PCILAWBAR0[BASE_ADDR] Reset Value

RCWHR[BMS]	PCILAWBAR0[BASE_ADDR] Reset Value
0	0x00000
1	0xFF800

5.2.4.6 PCI Local Access Window *n* Attributes Registers (PCILAWAR0–PCILAWAR1)

The PCI local access window *n* attributes registers (PCILAWAR0–PCILAWAR1) are shown in Figure 5-7.

Offset 0x64
0x6C

Access: Read/Write

	0	1–25	26–31
R/W	EN	—	SIZE
Reset	All zeros[1,2]		

[1] The reset value of PCILAWAR0[EN] depends on the reset configuration word high values. See Section 5.2.4.6.1, "PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value," for a detailed description.

[2] The reset value of PCILAWAR0[SIZE] is always 0b010110, meaning an 8-Mbyte local access window. See Section 5.2.4.6.1, "PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value," for a detailed description

Figure 5-7. PCI Local Access Window *n* Attributes Registers (PCILAWAR0–PCILAWAR1)

Table 5-14 defines the bit fields of PCILAWAR0–PCILAWAR1.

Table 5-14. PCILAWAR0–PCILAWAR1 Bit Settings

Bits	Name	Description
0	EN	0 The PCI local access window *n* is disabled. 1 The PCI local access window *n* is enabled and other PCILAWAR*n* and PCILAWBAR*n* fields combine to identify an address range for this window.
1–25	—	Reserved. Write has no effect, read returns 0.
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes . . . . $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

5.2.4.6.1 PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value

The core may use a PCI peripheral device to fetch its boot vector. For this purpose an 8-Mbyte ($2^{(22+1)}$) local access window is defined by the PCILAWBAR0[SIZE] reset value, and PCILAWAR0 is enabled according to the value set in the reset configuration word high ROMLOC and RLEXT fields.

Table 5-15 defines the reset value of PCILAWAR0[EN].

Table 5-15. PCILAWAR0[EN] Reset Value

RCWHR[RLEXT]/RCWHR [ROMLOC]	PCILAWR0[EN] Reset Value	Description
00/000, 010–111	0	e300c4s core boot not performed from a PCI device.

Table 5-15. PCILAWAR0[EN] Reset Value (continued)

RCWHR[RLEXT]/RCWHR [ROMLOC]	PCILAWR0[EN] Reset Value	Description
01-11 / 000-111	0	e300c4s core boot not performed from a PCI device.
00 / 001	1	e300c4s core boot performed from a PCI device. PCI 8-Mbyte ($2^{(22+1)}$) local access window is enabled.

5.2.4.7 PCI Express 1 Local Access Window Base Address Register (PCIEXP1LAWBAR)

The PCI Express 1 local access window base address register is shown in Figure 5-8.

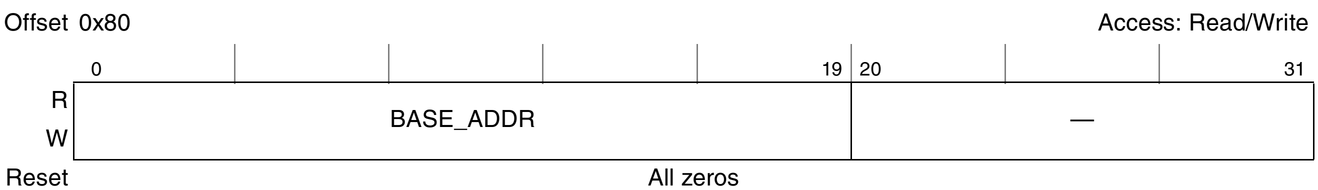


Figure 5-8. PCI Express 1 Local Access Window Base Address Register (PCIEXP1LAWBAR)

Table 5-16 defines the bit fields of PCIEXP1LAWBAR.

Table 5-16. PCIEXP1LAWBAR Bit Settings

Bits	Name	Description
0–19	BASE_ADDR	Identifies the 20 most-significant address bits of the base of local access window. The specified base address should be aligned to the window size, as defined by PCIEXP1LAWAR[SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

5.2.4.8 PCI Express 1 Local Access Window Attributes Registers (PCIEXP1LAWAR)

The PCI Express 1 local access window attributes register is shown in Figure 5-9.

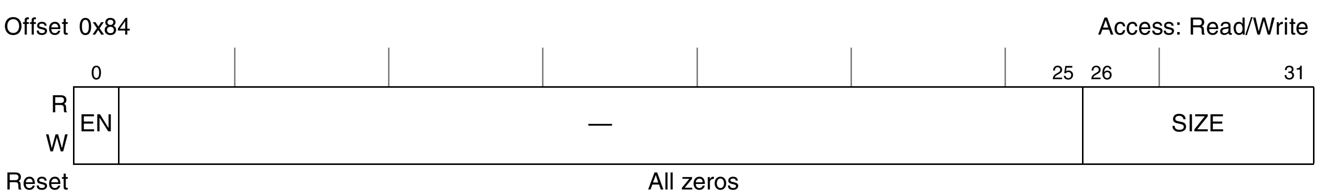


Figure 5-9. PCI Express 1 Local Access Window Attributes Register (PCIEXP1LAWAR)

Table 5-17 defines the bit fields of PCIEXP1LAWAR.

Table 5-17. PCIEXP1LAWAR Bit Settings

Bits	Name	Description
0	EN	0 The PCI Express 1 local access window is disabled. 1 The PCI Express 1 local access window is enabled and other PCIEXP1LAWAR fields combine to identify an address range for this window.
1–25	—	Reserved. Write has no effect, read returns 0.
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes . . . . $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

5.2.4.9 PCI Express 2 Local Access Window Base Address Register (PCIEXP2LAWBAR)

The PCI Express 2 local access window base address register is shown in Figure 5-10.

Offset 0x88 Access: Read/Write

	0–19	20–31
R W	BASE_ADDR	—
Reset	All zeros	

Figure 5-10. PCI Express 2Local Access Window Base Address Register (PCIEXP2LAWBAR)

Table 5-18 defines the bit fields of PCIEXP2LAWBAR.

Table 5-18. PCIEXP2LAWBAR Bit Settings

Bits	Name	Description
0–19	BASE_ADDR	Identifies the 20 most-significant address bits of the base of local access window. The specified base address should be aligned to the window size, as defined by PCIEXP2LAWAR[SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

5.2.4.10 PCI Express 2 Local Access Window Attributes Registers (PCIEXP2LAWAR)

The PCI Express 2 local access window attributes register is shown in Figure 5-11.

Offset 0x8C | Access: Read/Write

	0	1–25	26–31
R/W	EN	—	SIZE
Reset	All zeros		

Figure 5-11. PCI Express 2 Local Access Window Attributes Register (PCIEXP2LAWAR)

Table 5-19 defines the bit fields of PCIEXP2LAWAR.

Table 5-19. PCIEXP2LAWAR Bit Settings

Bits	Name	Description
0	EN	0 The PCI Express 2 local access window is disabled. 1 The PCI Express 2 local access window is enabled and other PCIEXP2LAWAR fields combine to identify an address range for this window.
1–25	—	Reserved. Write has no effect, read returns 0.
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes . . . . $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

5.2.4.11 DDR Local Access Window *n* Base Address Registers (DDRLAWBAR0–DDRLAWBAR1)

The DDR local access window *n* base address registers (DDRLAWBAR0–DDRLAWBAR1) are shown in Figure 5-12.

Offset 0xA0
0xA8 | Access: Read/Write

	0–19	20–31
R/W	BASE_ADDR	—
Reset	All zeros[1]	

[1] The reset value of DDRLAWBAR0[BASE_ADDR] depends on the reset configuration word high values. See Section 5.2.4.11.1, "DDRLAWBAR0[BASE_ADDR] Reset Value," for a detailed description

Figure 5-12. DDR Local Access Window *n* Base Address Registers (DDRLAWBAR0–DDRLAWBAR1)

Table 5-20 defines the bit fields of DDRLAWBAR0–DDRLAWBAR1.

Table 5-20. DDRLAWBAR0–DDRLAWBAR1 Bit Settings

Bits	Name	Description
0–19	BASE_ADDR	Identifies the 20 most-significant address bits of the base of local access window *n*. The specified base address should be aligned to the window size, as defined by DDRLAWAR*n*[SIZE].
20–31	—	Reserved. Write has no effect, read returns 0.

5.2.4.11.1 DDRLAWBAR0[BASE_ADDR] Reset Value

The core may use a DDR SDRAM device to fetch its boot vector. For this purpose, the DDRLAWBAR0[BASE_ADDR] reset value is set according to the value set in the reset configuration word high BMS field.

Table 5-21 defines the reset value DDRLAWBAR0.

Table 5-21. DDRLAWBAR0[BASE_ADDR] Reset Value

RCWHR[BMS]	DDRLAWBAR0[BASE_ADDR] Reset Value
0	0x00000
1	0xFF800

5.2.4.12 DDR Local Access Window *n* Attributes Registers (DDRLAWAR0–DDRLAWAR1)

The DDR local access window *n* attributes registers (DDRLAWAR0–DDRLAWAR1) are shown in Figure 5-13.

Offset 0xA4
0xAC

Access: Read/Write

	0	1–25	26–31
R W	EN	—	SIZE
Reset	All zeros[1,2]		

1 The reset value of DDRLAWAR0[EN] depends on the reset configuration word high values. See Section 5.2.4.12.1, "DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value," for a detailed description.

2 The reset value of DDRLAWAR0[SIZE] is always 0b010110, meaning an 8-Mbyte local access window. See Section 5.2.4.12.1, "DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value," for a detailed description.

Figure 5-13. DDR Local Access Window *n* Attributes Registers (DDRLAWAR0–DDRLAWAR1)

Table 5-22 defines the bit fields of DDRLAWAR0–DDRLAWAR1.

Table 5-22. DDRLAWAR0–DDRLAWAR1 Bit Settings

Bits	Name	Description
0	EN	0 The DDR local access window *n* is disabled. 1 The DDR local access window *n* is enabled and other DDRLAWAR*n* and DDRLAWBAR*n* fields combine to identify an address range for this window.
1–25	—	Reserved. Write has no effect, read returns 0.
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved. Window is undefined. 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes . . . . $2^{(SIZE+1)}$ bytes 011110 2 Gbytes 011111–111111 Reserved. Window is undefined.

5.2.4.12.1 DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value

The core may use a DDR SDRAM device to fetch its boot vector. For this purpose an 8-Mbyte ($2^{(22+1)}$) local access window is defined by DDRLAWBAR0[SIZE] reset value, and DDRLAWAR0 is enabled according to the value set in the reset configuration word high ROMLOC and RLEXT fields.

Table 5-23 defines the reset value DDRLAWAR0[EN] and DDRLAWAR0[SIZE].

Table 5-23. DDRLAWAR0[EN] Reset Value

RCWHR[RLEXT]/RCWHR [ROMLOC]	DDRLAWAR0[EN] Reset Value	Description
00 / 000	1	e300c4s core boot performed from a DDR SDRAM device. DDR 8-Mbyte ($2^{(22+1)}$) local access window is enabled.
Else	0	e300c4s core boot not performed from a DDR SDRAM device.

5.2.5 Precedence of Local Access Windows

If two local access windows overlap, the lower numbered window takes precedence (see Table 5-1 for window numbers). For instance, if two windows are set up as shown in Table 5-24, local access window 1 governs the mapping of the 1-Mbyte region from 0x7FF0_0000 to 0x7FFF_FFF, even though the window described in local access window 7 also encompasses that memory region.

Table 5-24. Overlapping Local Access Windows

Window	Base Address	Size	Target Interface
1	0x7FF0_0000	1 Mbyte	Local bus
7	0x0000_0000	2 Gbytes	DDR SDRAM

5.2.6 Configuring Local Access Windows

After a local access window is enabled, it should not be modified while any device in the system may be using the window. Accordingly, a new window should not be used until the effect of the write to the window is visible to all blocks that use the window. This can be guaranteed by completing a read of the last local access window configuration register before enabling any other devices to use the window. For instance, if local bus local access windows 1–3 are being configured in order during the initialization process, the last write (to LBLAWAR3) should be followed by a read of LBLAWAR3 before any devices try to use any of these windows. If the configuration is being done by the local e300c4s core, the read of LBLAWAR3 should be followed by an **isync** instruction.

5.2.7 Distinguishing Local Access Windows from Other Mapping Functions

It is important to distinguish between the mapping function performed by the local access windows and the additional mapping functions that happen at the target interface. The local access windows define how a transaction is routed through the device internal interconnects from the transaction's source to its target. Once the transaction has arrived at its target interface, that interface controller may perform additional mapping. For instance, the DDR SDRAM controller has chip select registers that map a memory request to a particular external device. The local bus controller has base registers that perform a similar function. The PCI interface has outbound address translation units that map the local address into an external address space.

These other mapping functions are configured by programming the configuration, control, and status registers of the individual interfaces. Note that there is no need to have a one-to-one correspondence between local access windows and chip select regions or outbound windows. A single local access window can be further decoded to any number of chip selects or to any number or outbound windows at the target interface.

5.2.8 Outbound Address Translation and Mapping Windows

Outbound address translation and mapping refers to the translation of addresses from the local 32-bit address space to the external address space and attributes of a particular I/O interface. On this device, the PCI block has an outbound address translation unit.

The PCI controller has six outbound windows plus a default window. See Section 4.5, "Memory Map/Register Definitions," for a detailed description of the PCI outbound windows.

5.2.9 Inbound Address Translation and Mapping Windows

Inbound address translation and mapping refers to the translation of an address from the external address space of an I/O interface (such as PCI address space) to the local address space understood by the internal interfaces of this processor. It also refers to the mapping of transactions to a particular target interface and the assignment of transaction attributes. The PCI controller has inbound address translation unit.

5.2.9.1 PCI Inbound Windows

The PCI controller has three general inbound windows plus a dedicated window for memory mapped configuration accesses (PIMMR). These windows have a one-to-one correspondence with the base address registers in the PCI programming model. Updating one automatically updates the other. There is no default inbound window; if a PCI address does not match one of the inbound windows, this processor does not respond with an assertion of $\overline{\text{PCI\_DEVSEL}}$. See Section 14.4.6, "PCI Inbound Address Translation," for a detailed description of the PCI inbound windows.

5.2.10 Internal Memory Map

All of the memory mapped configuration, control, and status registers in the device are contained within a 1-Mbyte address region, referred as the IMMR. To allow for flexibility, the internal memory map block can be relocated in the local address space. The local address map location of this register block is controlled by the internal memory map registers' base address register (IMMRBAR); see Section 5.2.4.1, "Internal Memory Map Registers Base Address Register (IMMRBAR)." The default value for the IMMRBAR is 0xFF40_0000.

NOTE

The internal memory map window is always the highest priority local access window.

5.2.11 Accessing Internal Memory from External Masters

In addition to being accessible by the e300 processor, the IMMR memory window is accessible from external interfaces. This allows external masters on the I/O ports to configure the device.

External masters do not need to know the location of the IMMR memory in the local address map. Rather, they access this region of the local memory map through a window defined by a register in the interface's programming model that is accessible to the external master from its external memory map.

The PCI base address for accessing the local IMMR memory is selectable through the PCI internal memory map register (PIMMR), at offset 0x10, described in Section 14.3.2.12, "PCI Inbound Base Address Registers (PIBARn)." When the device is a PCI agent, an external PCI master sets this register by running a PCI configuration cycle. Subsequent memory accesses by a PCI master to the PCI address range indicated by PIMMR are translated to the local address indicated by the current setting of IMMRBAR.

5.3 System Configuration

The following sections describe some general information and configuration options that affect system behavior and performance.

5.3.1 System Configuration Register Memory Map

Table 5-25 shows the memory map for the system configuration registers.

Table 5-25. System Configuration Register Memory Map

Local Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00100	System general purpose register low (SGPRL)	R/W	0x480E_FF20	5.3.2.1/5-20
0x00104	System general purpose register high (SGPRH)	R/W	0x0000_0000	5.3.2.2/5-21
0x00108	System part and revision ID register (SPRIDR)	R	0x80C0_0010	5.3.2.3/5-21
0x0010C	Reserved	—	—	—
0x00110	System priority configuration register (SPCR)	R/W	0x0000_0000	5.3.2.4/5-22
0x00114	System I/O configuration register low (SICRL)	R/W	0x0000_0000[1]	5.3.2.5/5-24
0x00118	System I/O configuration register high (SICRH)	R/W	0x1FFC_0000	5.3.2.6/5-28
0x0011C–0x00124	Reserved	—	—	—
0x00128	DDR control driver register (DDRCDR)	R/W	0x7304_0001	5.3.2.8/5-31
0x0012C	DDR debug status register (DDRDSR)	R	0x3300_0000	5.3.2.9/5-33
0x00130	Output buffer impedance register (OBIR)	R/W	0x3111_0000	5.3.2.10/55-34
0x00134–0x0013C	Reserved	—	—	—
0x00140	PCI Express control register 1 (PECR1)	R/W	0x0000_0000	5.3.2.11/55-34
0x00144	PCI Express control register 2 (PECR2)	R/W	0x0000_0000	5.3.2.11/55-34
0x00148–0x001FC	Reserved	—	—	—

[1] Depends on the reset configuration word high configuration values.

5.3.2 System Configuration Registers

This section describes the system configuration registers.

5.3.2.1 System General Purpose Register Low (SGPRL)

The system general purpose register low (SGPRL), shown in Figure 5-14, can be used by software for any purpose. The values set in this register have no effect on the internal hardware.

Offset 0x00100 — Access: Read/Write

	0–31
R/W	GP
Reset	0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0

Figure 5-14. System General Purpose Register Low (SGPRL)

Table 5-26 defines the bit fields of SGPRL.

Table 5-26. SGPRL Bit Settings

Bits	Name	Description
0–31	GP	General purpose

5.3.2.2 System General Purpose Register High (SGPRH)

The system general purpose register high (SGPRH), shown in Figure 5-15, can be used by software for any purpose. The values set in this register have no effect on the internal hardware.

Offset 0x00104 Access: Read/Write

	0–31
R	GP
W	
Reset	All zeros

Figure 5-15. System General Purpose Register High (SGPRH)

Table 5-27 defines the bit fields of SGPRH.

Table 5-27. SGPRH Bit Settings

Bits	Name	Description
0–31	GP	General purpose

5.3.2.3 System Part and Revision ID Register (SPRIDR)

SPRIDR, shown in Figure 5-16, provides information about the device and revision numbers.

Offset 0x00108 Access: Read only

	0–15	16–31
R	PARTID	REVID
W		
Reset	See Table 5-29 and Table 5-30 for reset values of this register.	

Figure 5-16. System Part and Revision ID Register (SPRIDR)

Table 5-28 defines the bit fields of SPRIDR.

Table 5-28. SPRIDR Bit Settings

Bits	Name	Description
0–15	PARTID	Part identification. This read-only field is mask-programmed with a code corresponding to the device number. It is intended to help factory test and user code that is sensitive to device changes. The device number changes according to manufacturing considerations. See Table 5-29 for values of this field.
16–31	REVID	Revision identification. This read-only field is mask-programmed with a code corresponding to the revision number of the part defined in PARTID field. It is intended to help factory test and user code that is sensitive to device changes. The mask number is programmed in a commonly changed layer, and changes with each mask set change. See Table 5-30 for values of this field.

5.3.2.3.1 SPRIDR[PARTID] Coding

Table 5-29 defines the reset values of SPRIDR[PARTID].

Table 5-29. PARTID Coding

PARTID	Device Name	Package Type
0x80C2	MPC8379E	PBGA
0x80C3	MPC8379	PBGA
0x80C4	MPC8378E	PBGA
0x80C5	MPC8378	PBGA
0x80C6	MPC8377E	PBGA
0x80C7	MPC8377	PBGA

Table 5-30 defines the reset values of SPRIDR[REVID].

Table 5-30. REVID Coding

REVID	Device Revision
0x0010	1.0

5.3.2.4 System Priority and Configuration Register (SPCR)

The system priority and configuration register (SPCR), shown in Figure 5-17, controls the priority of requests for transactions on the internal system bus. This priority is considered by the system arbiter whenever an internal unit requests mastership of the coherent system bus (CSB). The SPCR also includes some other control functions.

Offset 0x00110 Access: Read/Write

Bits	0–2	3	4–5	6–7	8	9	10–11	12–15
R/W	—	PCIHPE	—	PCIPR	OPT	TBEN	COREPR	—
Reset	All zeros							

Bits	16–17	18–19	20–21	22–23	24–31
R/W	—	TSECDP	TSECBDP	TSECEP	—
Reset	All zeros				

Figure 5-17. System Priority Configuration Register (SPCR)

Table 5-31 defines the bit fields of SPCR.

Table 5-31. SPCR Bit Settings

Bits	Name	Description
0–2	—	Reserved. Should be cleared.
3	PCIHPE	PCI highest priority enable. If this bit is set, the PCI bridge is permitted to request the coherent system bus (CSB) with highest priority, regardless of SPCR[PCIPR] value, when it needs to complete a posted write transaction from an external PCI master. To follow PCI ordering rules specifications, the PCI bridge must flush any outstanding write transactions before it can start a new read transaction. Setting this bit allows faster flushing of the outstanding write transactions coming from the PCI bus onto the CSB and to the device targets, such as DDR SDRAM and local bus memories.
4–5	—	Reserved. Should be cleared.
6–7	PCIPR	PCI bridge CSB request priority. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority) **Note:** DMA has the same priority as PCI.
8	OPT	Optimize. Setting this bit may enhance the performance of transactions issued to the internal coherent system bus (CSB) by the security engine (SEC) and the USB controller. Performance is enhanced by reading more bytes on the bus than actually needed by the master in the case that this is more efficient. The user may set this bit only if it is known that USB transactions sent to the internal CSB are not accessing devices in which speculative reads may change the state of the device (for example, FIFOs in which reading a byte may advance some internal counter). 0 No performance enhancement. 1 Performance enhancement by speculative reading is enabled.
9	TBEN	e300c4s core time base unit enable 0 Time base unit is disabled. 1 Time base unit is enabled.

Table 5-31. SPCR Bit Settings (continued)

Bits	Name	Description
10–11	COREPR	e300c4s core CSB request priority. The priority level for the core in accessing the CSB can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
12–17	—	Reserved. Should be cleared.
18–19	TSECDP	eTSEC data priority. Selects the CSB request priority driven by eTSEC1 and eTSEC2 when they require to transfer data on this bus. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
20–21	TSECBDP	eTSEC buffer descriptor priority. Selects the CSB request priority driven by eTSEC1 and eTSEC2 when they require to transfer a buffer descriptor (BD) on this bus. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
22–23	TSECEP	eTSEC emergency priority. Selects the CSB request priority driven by eTSEC1 and eTSEC2 when an emergency condition occurs. The level of priority can be chosen from 4 possible levels. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
24-31	—	Reserved, should be cleared.

5.3.2.5 System I/O Configuration Register Low (SICRL)

The system I/O configuration register low (SICRL) controls the multiplexing of some of the device I/O pins. Each bit or set of bits in the SICRL selects which function is used by a certain group of the device pins.

Figure 5-18 shows SICRL.

Offset 0x00114 Access: Read/Write

	0	1	2	3	4	5	6	7
R/W	USB_A		USB_B		UART		GPIO1_A	GPIO1_B
Reset	All zeros							

	8	9	10	11	12	13	14	15
R/W	GPIO_C	GPIO_D	GPIO1_E	GPIO1_F		GPIO1_G	GPIO1_H	GPIO1_I
Reset	All zeros							

	16	17	18	19	20	21	22	23
R/W	GPIO1_J	GPIO1_K	GPIO1_L		DMA_A	DMA_B	DMA_C	DMA_D
Reset	All zeros							

	24	25	26	27	28	29	30	31
R/W	DMA_E	DMA_F	DMA_G	DMA_H	DMA_I	DMA_J	LDP_A[1]	LDP_B[1]
Reset	All zeros							

1 The SICRL[LDP_A] and SICRL[LDP_B] reset value depends on the value of RCWH[30] which is loaded into the reset configuration word.

Figure 5-18. System I/O Configuration Register Low (SICRL)

Table 5-32 defines the bit fields of SICRL. Each Pin Function column lists the name of the multi-function pin used in this option. Some groups have only two options (shown as Pin Function 0 and Pin Function 1) and therefore, only one control bit. In this case they can only have a value of 0b0 or 0b1. Other groups may have four options (shown as Pin Function 0, Pin Function 1, Pin Function 2, and Pin Function 3) and therefore, two control bits. In this case they can have a value of 0b00, 0b01, 0b10, or 0b11. Use the notations ‘0b*N*’ or ‘0b*NN*’ according to whether a group has one or two control bits, respectively.

Table 5-32. SICRL Bit Settings

SICRL[Bits] Value		0b0/0b00	0b1/0b01	0b10	0b11
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	Pin Function 3
0–1	USB_A	GPIO2[0]	DR_D0_ENABLEN	—	—
		GPIO2[1]	DR_D1_SER_TXD	—	—
		GPIO2[2]	DR_D2_VMO_SE0	—	—
		GPIO2[3]	DR_D3_SPEED	—	—
		GPIO2[4]	DR_D4_DP	—	—
		GPIO2[5]	DR_D5_DM	—	—
		GPIO2[6]	DR_D6_SER_RCV	—	—
		GPIO2[7]	DR_D7_DRVVBUS	—	—
		GPIO2[8]	DR_SESS_VLD_NXT	—	—
		GPIO2[9]	DR_DIR_DPPULLUP	—	—
		GPIO2[23]	DR_CLK	—	—
2–3	USB_B	GPIO2[10]	DR_PWRFAULT	SD_DAT1	—
		GPIO2[11]	DR_PCTL0	SD_DAT2	—
		GPIO2[22]	DR_PCTL1	SD_DAT3	—
4–5	UART	UART_SIN1	MSRCID[2]	LSRCID[2]	—
		UART_SIN2	MSRCID[3]	LSRCID[3]	—
		UART_SOUT1	MSRCID[0]	LSRCID[0]	—
		UART_SOUT2	MSRCID[1]	LSRCID[1]	—
		UART_CTS[1]	MSRCID[4]	LSRCID[4]	—
		UART_CTS[2]	MDVAL	LDVAL	—
6[1]	GPIO1_A	GPIO1[0]	GTM1_TIN1/GTM2_TIN2	—	—
7[1]	GPIO1_B	GPIO1[1]	GTM1_TGATE1/GTM2_TGATE2	—	—
8[1]	GPIO1_C	GPIO1[2]	GTM1_TOUT1	—	—
9[1]	GPIO1_D	GPIO1[3]	GTM1_TIN2/GTM2_TIN1	—	—
10[1]	GPIO1_E	GPIO1[4]	GTM1_TGATE2/GTM2_TGATE1	—	—
11–12	GPIO1_F	GPIO1[5]	GTM1_TOUT2	GTM2_TOUT1	DMA_DDONE1
13[2]	GPIO1_G	GPIO1[6]	GTM1_TIN3/GTM2_TIN4	—	—
14[2]	GPIO1_H	GPIO1[7]	GTM1_TGATE3/GTM2_TGATE4	—	—
15[2]	GPIO1_I	GPIO1[8]	GTM1_TOUT3	—	—
16[2]	GPIO1_J	GPIO1[10]	GTM1_TGATE4/GTM2_TGATE3	—	—

Table 5-32. SICRL Bit Settings (continued)

SICRL[Bits] Value		0b0/0b00	0b1/0b01	0b10	0b11
Bits	**Group**	**Pin Function 0**	**Pin Function 1**	**Pin Function 2**	**Pin Function 3**
17[2]	GPIO1_K	GPIO1[9]	GTM1_TIN4/GTM2_TIN3	—	—
18–19	GPIO1_L	GPIO1[11]	GTM1_TOUT4	GTM2_TOUT3	DMA_DDONE3
20	DMA_A	Controlled by SICRL[6]	DMA_DREQ0	—	—
21	DMA_B	Controlled by SICRL[7]	DMA_DACK0	—	—
22	DMA_C	Controlled by SICRL[8]	DMA_DDONE0	—	—
23	DMA_D	Controlled by SICRL[9]	DMA_DREQ1	—	—
24	DMA_E	Controlled by SICRL[10]	DMA_DACK1	—	—
25	DMA_F	Controlled by SICRL[13]	DMA_DREQ2	—	—
26	DMA_G	Controlled by SICRL[14]	DMA_DACK2	—	—
27	DMA_H	Controlled by SICRL[15]	DMA_DDONE2	—	—
28	DMA_I	Controlled by SICRL[17]	DMA_DACK3	—	—
29	DMA_J	Controlled by SICRL[16]	DMA_DREQ3	—	—
30	LDP_A	LCS[4]	LDP[0]	—	—
		LCS[5]	LDP[1]	—	—
31	LDP_B	LCS[6]	LDP[2]	—	—
		LCS[7]	LDP[3]	—	—

[1] The setting of the signals controlled by bits 6–10 also depends on bits 20–24.

[2] The setting of the signals controlled by bits 13–17 also depends on bits 25–29.

5.3.2.6 System I/O Configuration Register High (SICRH)

The system I/O configuration register high, shown in Figure 5-19, controls the multiplexing of the rest of the device I/O pins. Each bit or set of bits in this register select which function is used by a certain group of the device pins.

Offset 0x00118 Access: Read/Write

	0	1	2	3	4	5	6	7
R/W	DDR	—		TSEC1_A	TSEC1_B	—		
Reset	0	0	0	1	1	1	1	1

	8	9	10	11	12	13	14	15
R/W	—	TSEC2_A	TSEC2_B	TSEC2_C	TSEC2_D	TSEC2_E	—	TMR
Reset	1	1	1	1	1	1	0	0

	16	17	18	19	20	21	22	23
R/W	GPIO2_A	GPIO2_B	GPIO2_C	GPIO2_D	GPIO2_E		GPIO2_F	
Reset	All zeros							

	24	25	26	27	28	29	30	31
R/W	GPIO2_G		GPIO2_H		—		SPI	
Reset	All zeros							

Figure 5-19. System I/O Configuration Register High (SICRH)

Table 5-33 defines the bit fields of SICRH. Each Pin Function column lists the name of the multi-function pin used in this option. Some groups have only two options (shown as Pin Function 0 and Pin Function 1) and therefore, only one control bit. In this case they can have the value of 0b0 or 0b1 only. Other groups may have three options (shown as Pin Function 0, Pin Function 1, and Pin Function 2) and therefore, two control bits. In this case they can have the value of 0b00, 0b01, or 0b10. A value of 0b11 is illegal for all groups. Use the 0b*N* or 0b*NN* notations according to a group having one or two control bits, respectively.

Table 5-33. SICRH Bit Settings

SICRH[Bits] Value:		0b0/0b00	0b1/0b01	0b10	Reset Value
Bits	**Group**	**Pin Function 0**	**Pin Function 1**	**Pin Function 2**	
0	DDR	MECC[0]	MSRCID[0]	—	0
		MECC[1]	MSRCID[1]	—	
		MECC[2]	MSRCID[2]	—	
		MECC[3]	MSRCID[3]	—	
		MECC[4]	MSRCID[4]	—	
		MECC[5]	MDVAL	—	

Table 5-33. SICRH Bit Settings (continued)

SICRH[Bits] Value:		0b0/0b00	0b1/0b01	0b10	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	
1–2	Reserved	—	—	—	00
3	TSEC1_A	TSEC1_COL	GPIO2[20]	—	1
		TSEC1_RX_ER	GPIO2[25]	—	
4	TSEC1_B	TSEC1_CRS	GPIO2[21]	—	1
5–8	Reserved	—	—	—	0000
9[1]	TSEC2_A	TSEC2_COL	GPIO1[21]	—	1
		TSEC2_RX_ER	GPIO1[25]	—	
		TSEC2_TX_ER	GPIO1[24]	—	
10[1]	TSEC2_B	TSEC2_CRS	GPIO1[22]	—	1
11[1]	TSEC2_C	TSEC2_RXD1	GPIO1[15]	—	1
		TSEC2_RXD0	GPIO1[16]	—	
		TSEC2_TXD1	GPIO1[19]	—	
		TSEC2_TXD0	GPIO1[20]	—	
		TSEC2_RX_DV	GPIO1[23]	—	
		TSEC2_TX_EN	GPIO1[12]	—	
12[1]	TSEC2_D	TSEC2_RXD3	GPIO1[13]	—	1
		TSEC2_RXD2	GPIO1[14]	—	
		TSEC2_TXD3	GPIO1[17]	—	
		TSEC2_TXD2	GPIO1[18]	—	
13[1]	TSEC2_E	TSEC2_TX_CLK	GPIO2[24]	—	1
14	Reserved	—			0
15[2]	TMR	controlled by SICRH[9]	TSEC1_1588_TRIG1	—	0
		controlled by SICRH[10]	TSEC1_1588_TRIG2	—	
		TSEC2_RX_CLK	TSEC1_1588_CLK_IN	—	
		controlled by SICRH[12]	TSEC1_1588_PP3	—	
		controlled by SICRH[12]	TSEC1_1588_PP2	—	
		controlled by SICRH[11]	TSEC1_1588_PP1	—	
		controlled by SICRH[9]	TSEC1_1588_ALARM1	—	
		controlled by SICRH[11]	TSEC1_1588_ALARM2	—	
		controlled by SICRH[13]	TSEC1_1588_GCLK	—	
16	GPIO2_A	$\overline{\text{IRQ}}$[0]/$\overline{\text{MCP\_IN}}$	GPIO2[12]	—	0

Table 5-33. SICRH Bit Settings (continued)

SICRH[Bits] Value:		0b0/0b00	0b1/0b01	0b10	Reset Value
Bits	Group	Pin Function 0	Pin Function 1	Pin Function 2	
17	GPIO2_B	$\overline{\text{IRQ}}$[1]	GPIO2[13]	—	0
18	GPIO2_C	$\overline{\text{IRQ}}$[2]	GPIO2[14]	—	0
19	GPIO2_D	$\overline{\text{IRQ}}$[3]	GPIO2[15]	—	0
20–21	GPIO2_E	$\overline{\text{IRQ}}$[4]	GPIO2[16]	SD_WP	00
22–23	GPIO2_F	$\overline{\text{IRQ}}$[5]	GPIO2[17]	USB_PWRFAULT	00
24–25	GPIO2_G	$\overline{\text{IRQ}}$[6]	GPIO2[18]	$\overline{\text{CKSTOP\_OUT}}$	00
26–27	GPIO2_H	$\overline{\text{IRQ}}$[7]	GPIO2[19]	$\overline{\text{CKSTOP\_IN}}$	00
28–29	Reserved	—			00
30–31	SPI	SPIMOSI	SD_CMD	—	00
		SPIMISO	SD_DAT0	—	
		SPICLK	SD_CLK	—	
		SPISEL	SD_CD	—	

1 The setting of the signals controlled by bits 9–13 also depends on bit 15.
2 If TMR pins are selected, SICRH[9:13] must be cleared.

5.3.2.7 Debug Configuration

Debug information may be driven on the device pins. This information can identify the internal source of a transaction that reached the DDR SDRAM or local bus interfaces. The device can be configured to drive the MSRCID[0:4] and MDVAL, or LSRCID[0:4] and LDVAL signals, respectively on other device pins. The coding of the source ID debug information is the same as the coding of the MSTR_ID field in the AEATR register of the arbiter (See Section 6.2.6, “Arbiter Event Attributes Register (AEATR)”).

5.3.2.7.1 DDR Debug Configuration

The DDR debug configuration enables a DDR memory controller to enter debug mode in which the DDR SDRAM source ID field and data valid strobe are driven onto one of two optional sets of pins:

- ECC pins. ECC checking and generation are disabled in this case. ECC signals driven from the SDRAMs must be electrically disconnected from the ECC I/O pins of the processor in this mode. Set SICRH[0] to select this mode.
- UART pins. UART operation is disabled, and any signals driven by UART devices must be electrically disconnected from the UART I/O pins. Set SICRL[4–5] to 0b01 to select this mode.

5.3.2.7.2 Local Bus Debug Configuration

The local bus debug configuration enables a LBC debug mode in which the SDRAM source ID field and data valid strobe for LBC memory accesses are driven onto UART pins. UART operation must be disabled, and any signals driven by UART devices must be electrically disconnected from the UARTI/O pins in this case. Set SICRL[4–5] to 0b10 to select this mode.

5.3.2.8 DDR Control Driver Register (DDRCDR)

The DDR control driver register (DDRCDR) contains bits that allow control over the driver of the DDR SDRAM controller. Note that the MDIC signals require the use of precision 18.2-Ω resistors; MDIC0 should be pulled to GND, while MDIC1 should be pulled to GV_{DD}.

The fields in DDRCDR (other than ODT) are used to enable driver calibration with the MDIC[0:1] signals. This can be used to calibrate the DDR drivers to 18.2 Ω. However, this should only be used for full-strength driver applications. If half strength is desired, then this calibration should remain disabled.

Hardware DDR driver calibration is enabled using DDRCDR[DHC_EN]. If hardware calibration is used, then it should be set before DDR_SDRAM_CFG[MEM_EN] is set.

Software can be used to calibrate the drivers instead of the automatic hardware calibration. If software calibration is used, the following steps should be taken:

1. Set DDRCDR[DSO_EN] and ensure that DDRCDR[DHC_EN] is cleared.
2. Set the highest impedance (value 0000) for DDRCDR[DSO_PZ].
3. Set DDRCDR[MDIC0_OE] to enable the output enable for MDIC[0].
4. After at least 4 cycles, read DDRDSR[0]. If the value is 0, then use the next lower impedance, and read DDRDSR[0] again. Once a value of 1 is detected, then leave DDRCDR[DSO_PZ] at the calibrated value.
5. Clear DDRCDR[MDIC0_OE].
6. After DDRCDR[DSO_PZ] is calibrated, set a value of 0000 for DDRCDR[DSO_NZ]
7. Set DDRCDR[MDIC1_OE] to enable the output enable for MDIC[1]
8. After at least 4 cycles, read DDRDSR[1]. If the value is 1, then use the next lower impedance, and read DDRDSR[1] again. Once a value of 0 is detected, then leave DDRCDR[DSO_NZ] at the calibrated value
9. Clear DDRCDR[MDIC1_OE]

Note that the legal impedance values, from highest impedance to lowest impedance, are as follows:

- 0000
- 1000
- 1100
- 1110
- 1111

DDRCDR is shown in Figure 5-20.

Offset 0x00128 Access: Read/Write

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	DHC_EN	DSO_EN	DSO_PZ				DSO_NZ				MDIC0_OE	MDIC0_OE	ODT	DDR_TYPE	—	
Reset	0	1	1	1	0	0	1	1	0	0	0	0	0	1	0	0

Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	—															DRQ
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 5-20. DDR Control Driver Register (DDRCDR)

Table 5-34 shows the bit definition of the DDRCDR.

Table 5-34. DDRCDR Field Descriptions

Bits	Name	Description
0	DHC_EN	DDR driver hardware compensation enable
1	DSO_EN	0 DDR driver software override disable 1 DDR driver software override enable
2–5	DSO_PZ	DDR driver software p-impedance override 0000 Half strength—Highest Z 1000 Much higher Z than nominal 1100 Higher Z than nominal 1110 Nominal impedance setting 1111 Lower Z than nominal
6–9	DSO_NZ	DDR driver software n-impedance override 0000 Half strength—Highest Z 1000 Much higher Z than nominal 1100 Higher Z than nominal 1110 Nominal impedance setting 1111 Lower Z than nominal
10	MDIC0_OE	Output enable control for MDIC0 pin when software impedance calibration is performed. 0 MDIC0 is disabled 1 MDIC0 is enabled
11	MDIC1_OE	Output enable control for MDIC1 pin when software impedance calibration is performed. 0 MDIC1 is disabled 1 MDIC1 is enabled
12	ODT	ODT termination value for I/Os 0 75 Ω 1 150 Ω
13	DDR_TYPE	Selects voltage level for DDR pads 0 DDR2 (1.8 V mode) nominal impedance—18 Ω 1 DDR1 (2.5 V mode) nominal impedance—18 Ω **Note:** DDR_TYPE must be set according to the logical type of the DDR memory devices, as it effects logic behavior of the DDR controller as well as the physical parameters of the DDR I/O pads.

Table 5-34. DDRCDR Field Descriptions (continued)

Bits	Name	Description
14–30	—	Reserved
31	DRQ	0 Drain queue before sleep disable 1 Drain queue before sleep enable

5.3.2.9 DDR Debug Status Register (DDRDSR)

Figure 5-21 contains the debug status bits from the DDR SDRAM controller.

Offset 0x0012C — Access: Read

	0	1	2–5	6–9	10–31
R	MDIC0	MDIC1	PZ	NZ	—
W					
Reset[1]	All zeros				

Figure 5-21. DDR Debug Status Register (DDRDSR)

[1] Reset value of bits 0-9 depends on the actual state of the signals being monitored.

Table 5-35 shows the bit settings of the DDRDSR.

Table 5-35. DDRDSR Field Descriptions

Bits	Name	Description
0	MDIC0	DDR driver compensation input value read back for MDIC0. Note that MDIC0 is the PFET driver impedance calibration pin.
1	MDIC1	DDR driver compensation input value read back for MDIC1. Note that MDIC1 is the NFET driver impedance calibration pin.
2–5	PZ	Current setting of PFET driver impedance 0000 Half strength—highest Z 1000 Higher Z than nominal 1100 Nominal impedance setting 1110 Lower Z than nominal 1111 Much lower Z than nominal
6–9	NZ	Current setting of NFET driver impedance 0000 Half strength—highest Z 1000 Higher Z than nominal 1100 Nominal impedance setting 1110 Lower Z than nominal 1111 Much lower Z than nominal
10–31	—	Reserved

5.3.2.10 Output Buffer Impedance Register (OBIR)

The output buffer impedance register (OBIR), shown in Figure 5-22, controls the drive strength (impedance) of certain groups of I/O and provides the ability to configure the output impedance according to the application type.

Offset 0x00130 Access: Read/Write

Bits	0–1	2–3	4–5	6–7	8–9	10–11	12–13	14–15	16–31
R/W	—	—	—	TSEC1OBI	—	TSEC2OBI	—	LBOBI	—
Reset	0 0	1 1	0 0	0 1	0 0	0 1	0 0	0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 5-22. Output Buffer Impedance Register (OBIR)

Table 5-36 shows the bit settings of the OBIR.

Table 5-36. OBIR Field Description

Bits	Name	Description
0–1	—	Reserved
2–3	—	Reserved **Note:** A write operation must preserve reset value of bits 2–3.
4–5	—	Reserved
6–7	TSEC1OBI	TSEC1 output buffer impedance 00 3.3 V 45 ohm 01 2.5 V 45 ohm 10 Reserved 11 Reserved
8–9	—	Reserved
10–11	TSEC2OBI	TSEC2 output buffer impedance 00 3.3 V 45 ohm 01 2.5 V 45 ohm 10 Reserved 11 Reserved
12–13	—	Reserved
14–15	LBOBI	Local bus output buffer impedance 00 3.3 V 45 ohm 01 2.5 V 45 ohm 10 1.8 V 40 ohm 11 Reserved
16–31	—	Reserved

5.3.2.11 PCI Express Control Registers (PECR1 and PECR2)

The PCI Express control registers can be used to control various settings that affect the system response to PCI Express controller DMA operations or PIO inbound operation. Those registers also control soft reset assertion and negation to various logic parts of the PCI Express controllers. Note that PECR1 programming controls the behavior of PCI Express controller for port 1, and PECR2 controls the behavior of PCI Express controller for port 2. PECR1 is located at offset 0x140 and PECR2 is located at offset 0x144.

Figure 5-23 shows the PECR bit settings.

Offset 0x00140
0x00144

Access: Read/Write

	0	1	2	3–15
R W	LINK_RST	CBRST	CSR_RST	—
Reset	All zeros			

	16	17–25	26–27	28–29	30–31
R W	DEV_TYPE	—	PRI_DATA	PRI_DES	PRI_PIO
Reset	All zeros				

Figure 5-23. PCI Express Controller Registers (PECR1 and PECR2)

Table 5-37 describes the bits of the PECR register.

Table 5-37. PECR Field Description

Bits	Name	Description
0	LINK_RST	Link soft reset (active low). Assert soft reset to PCI Express controller's logic that is related to the link, the configuration registers, and the core logic. Should be negated after device type is programmed and the SerDes initialization is completed. 0 Link soft reset is asserted 1 Link soft reset is negated
1	CBRST	CSB bridge soft reset (active low). Assert soft reset to PCI Express controller's logic related to the CSB bridge. Should be negated after device type is programmed and the SerDes initialization is completed. The CSB bridge soft reset should be asserted also when hot reset is detected, in order to clean the queues of the CSB interface. 0 CSB bridge soft reset is asserted 1 CSB bridge soft reset is negated
2	CSR_RST	CSR soft reset (active low). Assert soft reset to PCI Express controller's logic that is related to the CSR (control and status) registers. Should be negated after device type is programmed and the SerDes initialization is completed. 0 CSR soft reset is asserted 1 CSR soft reset is negated
3–15	—	Reserved
16	DEV_TYPE	Device type. Program device type. 0 EP (end point) 1 RC (root complex)
17–25	—	Reserved
26–27	PRI_DATA	Data priority. This field will be used to present priority level for CSB arbitration for PCI Express controller's DMA requests. Bits 26–27 will be used when the request belongs to data transfer. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)

Table 5-37. PECR Field Description (continued)

Bits	Name	Description
28–29	PRI_DES	DMA descriptor priority. This field will be used to present priority level for CSB arbitration for PCI Express controller's DMA requests. Bits 28–29 will be used when the request belongs to descriptor fetch or update. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
30–31	PRI_PIO	PIO priority. This field will be used to present priority level for CSB arbitration for PCI Express controller's PIO inbound requests. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)

5.4 Software Watchdog Timer (WDT)

The following sections describe the theory of operation of the software watchdog timer (WDT) in the device, including a definition of the external signals and the functions they serve. Additionally, the configuration, control, and status registers are also described. Note that individual chapters in this book describe specific initialization aspects for each individual block.

5.4.1 WDT Overview

The device provides a software watchdog timer (WDT) feature to prevent system lock in case the software becomes trapped in loops with no controlled exit. Watchdog timer operations are configured in the system watchdog control register (SWCRR).

The watchdog counter is a free-running down-counter that generates a reset or a non-maskable interrupt on underflow. To prevent a reset, software must periodically restart the countdown. The WDT is responsible for asserting a hardware reset or machine-check interrupt (*mcp*) if the software fails to service the software watchdog timer for a certain period of time (for example, because software is lost or trapped in a loop with no controlled exit).

Figure 5-24 shows a high-level block diagram of the WDT.

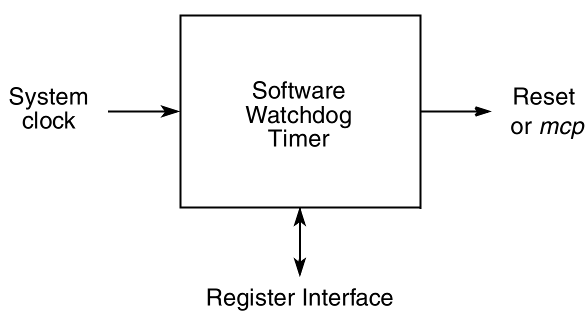


Figure 5-24. Software Watchdog Timer High-Level Block Diagram

The software watchdog timer is enabled after reset to cause a hardware reset if it times out. The user has the option of disabling the software watchdog if it is not needed. If used, the software watchdog timer requires a special service sequence to be executed periodically. Without this periodic servicing, the software watchdog timer times out and issues a reset or a non-maskable interrupt.

5.4.2 WDT Features

The WDT includes the following key features:

- Based on 16-bit prescaler and 16-bit down-counter
- Provides a selectable range for the time-out period
- Provides ~12.8-sec maximum software time-out delay for 333-MHz input clock
- Functional and programming compatibility with MPC8260 watchdog timer

5.4.3 WDT Modes of Operation

The WDT unit can operate in the following modes:

- WDT enable/disable mode:

  If the software watchdog timer is not needed, the user can disable it with software after a system reset. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state.
- WDT output reset/interrupt mode:

  Without software periodic servicing, the software watchdog timer times out and issues a reset or a nonmaskable interrupt (*mcp*)
- WDT prescaled/non-prescaled clock mode:

  The WDT counter clock can be prescaled by programming the SWCRR[SWPR] bit, which controls the divide-by-65,536 of the WDT counter.

5.4.4 WDT Memory Map/Register Definition

The WDT programmable register map occupies 16 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All WDT registers are 16- or 32-bits wide, located on 16-bit address boundaries, and should be accessed as 16- or 32-bit quantities. All addresses used in this chapter are offsets from the WDT base, as defined in Chapter 2, “Memory Map.”

Table 5-38 shows the WDT memory map.

Table 5-38. WDT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x0–0x3	Reserved	—	—	—
0x4	System watchdog control register (SWCRR)	R/W	0xFFFF_0003 or 0xFFFF_0007[1]	5.4.4.1/5-38

Table 5-38. WDT Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x8	System watchdog count register (SWCNR)	R	0x0000_FFFF	5.4.4.2/5-39
0xC–0xD	Reserved	—	—	—
0xE	System watchdog service register (SWSRR)	R/W	0x0000	5.4.4.3/5-39

[1] SWCRR[SWEN] reset value directly depends on RCWHR[SWEN] (reset configuration word high).

5.4.4.1 System Watchdog Control Register (SWCRR)

The system watchdog control register (SWCRR), shown in Figure 5-25, controls the software watchdog period and configures watchdog timer operation. SWCRR can be read at any time but can be written only once after system reset.

Offset 0x4 Access: Read/Write

	0–15	16–28	29	30	31
R/W	SWTC	—	SWEN	SWRI	SWPR
Reset	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0	0[1]	1	1

[1] SWCRR[SWEN] reset value directly depends on RCWHR[SWEN] (reset configuration word high).

Figure 5-25. System Watchdog Control Register (SWCRR)

Table 5-39 defines the bit fields of SWCRR.

Table 5-39. SWCRR Bit Settings

Bits	Name	Description
0–15	SWTC	Software watchdog time count The SWTC field contains the modulus that is reloaded into the watchdog counter by a service sequence. When a new value is loaded into SWCRR[SWTC], the software watchdog timer is not updated until the servicing sequence is written to the SWSRR. If SWCRR[SWEN] is loaded with 0, the modulus counter does not count. The new value is also used at the next and all subsequent reloads. Reading the SWCRR register returns the value in the system watchdog control register. Reset initializes the SWCRR[SWTC] field to 0xFFFF. **Note:** The prescaler counter is reset any time a new value is loaded into the watchdog counter and also during reset.
16–28	—	Write reserved, read = 0
29	SWEN	Watchdog enable bit Enables the watchdog timer. The reset value directly depends on the value of the RCWHR[SWEN] bit. It should be cleared by software after a system reset to disable the software watchdog timer. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state. 0 Watchdog timer disabled 1 Watchdog timer enabled **Note:** After software writes the SWRI bit, the state of SWEN cannot be changed.

Table 5-39. SWCRR Bit Settings (continued)

Bits	Name	Description
30	SWRI	Software watchdog reset/interrupt select bit A WDT timer out causes either a hard reset or machine check interrupt to the core. 0 Software watchdog timer causes a machine check interrupt to the core 1 Software watchdog timer causes a hard reset
31	SWPR	Software watchdog counter prescale bit Controls the divide-by-65,536 WDT counter prescaler 0 The WDT counter is not prescaled. 1 The WDT counter clock is prescaled.

5.4.4.2 System Watchdog Count Register (SWCNR)

The system watchdog count register (SWCNR), shown in Figure 5-26, provides visibility to the watchdog counter value. SWCNR is a read-only register. Writes to SWCNR have no effect and terminate without transfer error exception.

Offset 0x8 Access: Read only

	0–15	16–31
R	—	SWCN
W		
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 5-26. System Watchdog Count Register (SWCNR)

Table 5-40 defines the bit fields of SWCNR.

Table 5-40. SWCNR Bit Settings

Bits	Name	Description
0–15	—	Write reserved, read = 0
16–31	SWCN	Software watchdog count field. The read-only SWCNR[SWCN] field reflects the current value in the watchdog counter. Writing to the SWCNR register has no effect, and write cycles are terminated normally. Reset initializes the SWCNR[SWCN] field to 0xFFFF. **Note:** Reading the 16 least-significant bits of 32-bit SWCNR register with two 8-bit reads is not guaranteed to return a coherent value.

5.4.4.3 System Watchdog Service Register (SWSRR)

The system watchdog service register (SWSRR) is shown in Figure 5-27. When the watchdog timer is enabled, a write of 0x556C followed by a write 0xAA39 to the SWSRR register before the watchdog counter times out prevents a device reset. If the SWSRR register is not serviced before the timeout, a signal from the watchdog timer to the reset or interrupt controller module asserts a system reset or interrupt (depending on the setting of SWCRR[SWRI]).

Both writes must occur before the timeout in the order listed, but any number of instructions can be executed between the two writes. However, writing any value other than 0x556C or 0xAA39 to the SWSRR register resets the servicing sequence, requiring both values to be written to keep the watchdog timer from causing a reset. Reset initializes the SWSRR[WS] field to 0x0000. SWSRR can be written at any time, but returns all zeros when read.

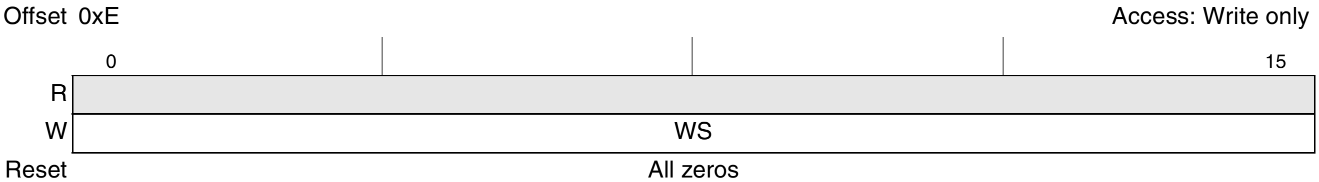


Figure 5-27. System Watchdog Service Register (SWSRR)

Table 5-41 defines the bit fields of SWCNR.

Table 5-41. SWSRR Bit Settings

Bits	Name	Description
0–15	WS	Software watchdog service field. The user should periodically write 0x556C followed by 0xAA39 to this register to prevent a software watchdog timer timeout. SWSRR[WS] can be written at any time, but returns all zeros when read.

5.4.5 Functional Description

5.4.5.1 Software Watchdog Timer Unit

The device provides a software watchdog timer (WDT) feature to prevent system lock in case the software becomes trapped in loops with no controlled exit. Watchdog timer operations are configured in the system watchdog control register (SWCRR).

The software watchdog timer is enabled after reset to cause a soft reset or non-maskable interrupt (MCP) if it times out. If the software watchdog timer is not needed, the user must clear SWCRR[SWEN] to disable it. If used, the software watchdog timer requires a special service sequence to be executed periodically. Without this periodic servicing, the software watchdog timer times out and issues a reset or a nonmaskable interrupt, as programmed in SWCRR[SWRI]. Once software writes SWRI, the state of SWEN cannot be changed.

The software watchdog timer service sequence consists of the following two steps:

- Write 0x556C to the system watchdog service register (SWSRR)
- Write 0xAA39 to SWSRR

The service sequence reloads the watchdog timer and the timing process begins again. If a value other than 0x556C or 0xAA39 is written to the SWSRR, the entire sequence must start over. Although the writes must occur in the correct order before a time-out, any number of instructions can be executed between the writes. This allows interrupts and exceptions to occur between the two writes when necessary. Figure 5-28 shows a state diagram for the watchdog timer.

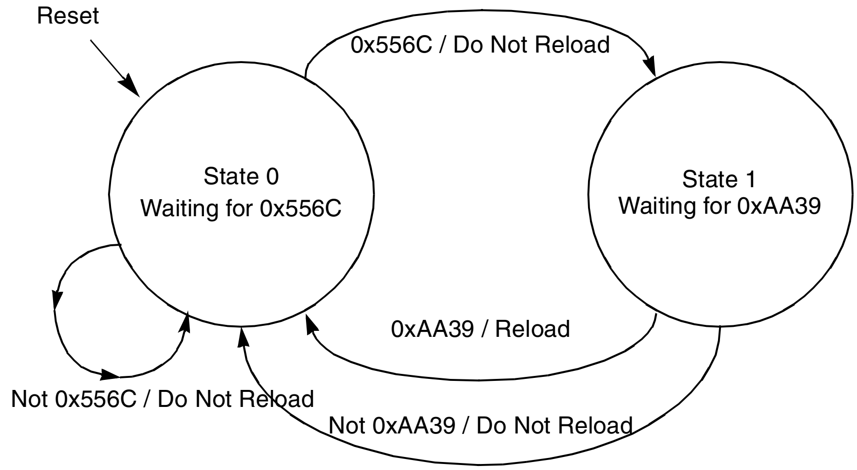


Figure 5-28. Software Watchdog Timer Service State Diagram

Although most software disciplines permit or even encourage the watchdog concept, some systems require a selection of time-out periods. For this reason, the software watchdog timer must provide a selectable range for the time-out period. Figure 5-29 shows how to handle this need.

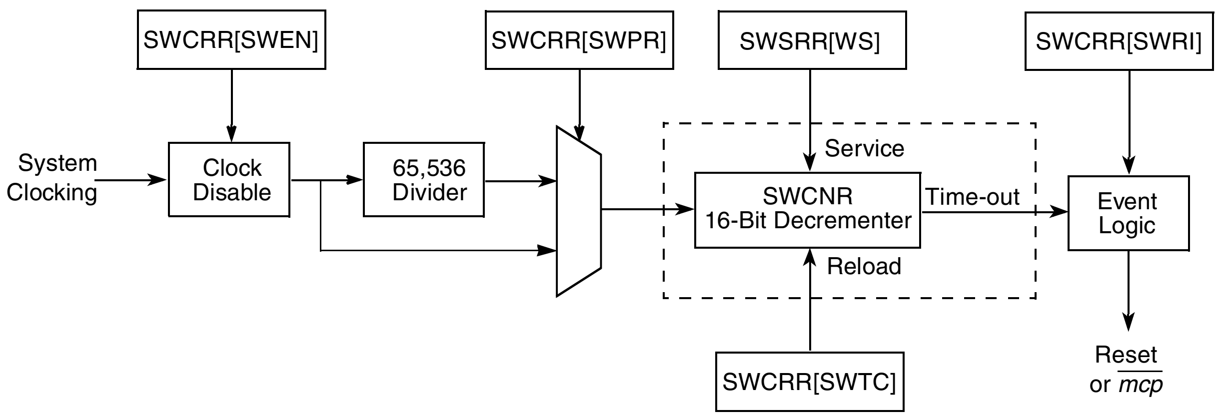


Figure 5-29. Software Watchdog Timer Functional Block Diagram

Figure 5-29 shows that the range is determined by SWCRR[SWTC]. The value in SWTC is then loaded into a 16-bit decrementer clocked by the system clock. An additional divide-by-65,536 prescaler value is used when needed.

The decrementer begins counting when loaded with a value from SWTC. After the timer reaches 0x0, a software watchdog expiration request is issued to the reset or *mcp* (machine check) control logic. Upon reset, SWTC is set to the maximum value and is again loaded into the system watchdog service register (SWSRR), starting the process over. When a new value is loaded into SWTC, the software watchdog timer is not updated until the servicing sequence is written to the SWSRR. If SWCRR[SWEN] is loaded with 0, the modulus counter does not count.

5.4.5.2 Modes of Operation

The WDT unit can operate in the following modes:

- WDT enable/disable mode:

  If the software watchdog timer is not needed, the user can disable it. The SWCRR[SWEN] bit enables the watchdog timer. It should be cleared by software after a system reset to disable the software watchdog timer. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state.

  — WDT enable mode (SWCRR[SWEN] = 1)

    This is the default value after soft reset.

  — WDT disable mode (SWCRR[SWEN] = 0)

- WDT reset/interrupt output mode

  Without software periodic servicing, the software watchdog timer times out and issues a reset or a nonmaskable interrupt (*mcp*), programmed in SWCRR[SWRI].

  According to the value of SWCRR[SWRI], the WDT timer causes a hard reset or machine check interrupt to the core.

  — Reset mode (SWCRR[SWRI] = 1).

    Software watchdog timer causes a hard reset (this is the default value after hard reset).

  — Interrupt mode (SWCRR[SWRI] = 0).

    Software watchdog timer causes a machine check interrupt to the core.

- WDT prescaled/non-prescaled clock mode

  The WDT counter clock can be prescaled by programming the SWCRR[SWPR] bit that controls the divide-by-65,536 of the WDT counter.

  — Prescale mode (SWCRR[SWPR] = 1)

    The WDT clock is prescaled.

  — Non-prescale mode (SWCRR[SWPR] = 0)

    The WDT clock is not prescaled.

5.4.6 Initialization/Application Information

5.4.6.1 WDT Programming Guidelines

The software watchdog timer is enabled (by the default value of SWCRR[SWEN]) after reset. The following initialization sequence of WDT is required:

- WDT disabling

  If the software watchdog timer is not needed, the user must clear SWCRR[SWEN] bit to disable the WDT not later than its timer times out (~12.8 sec. for a 333-MHz system clock).

- WDT initial servicing

  If the software watchdog timer is to be used, the special service sequence, described in Section 5.4.5.1, “Software Watchdog Timer Unit,” must be executed after system reset and not later than the first WDT time-out (~12.8 sec. for a 333-MHz system clock).

  Subsequently, periodical WDT servicing should be performed according to the programming guidelines given in Section 5.4.5.1, “Software Watchdog Timer Unit.”

5.5 Real Time Clock Module (RTC)

The following sections describe the theory of operation of the real time clock module (RTC) including a definition of the external signals and the functions it serves. Additionally, the configuration, control, and status registers are described. Note that individual chapters in this reference manual describe additional specific initialization aspects for each individual block.

5.5.1 RTC Overview

The device platform provides a real time clock (RTC) timer suitable for timestamping or time and calendar generation. It can maintain a one-second count which is unique over a period of approximately 136 years.

The RTC can be initialized by software with an initial count value using the real time counter load register (RTLDR). It can also be programmed to generate an interrupt every second. The real time counter control register (RTCTR) is used to enable or disable the various timer functions. The real time counter event register (RTEVR) is used to report the interrupt source. The RTC counter is initialized by software and can be disabled if needed.

Figure 5-30 shows the high level RTC block diagram.

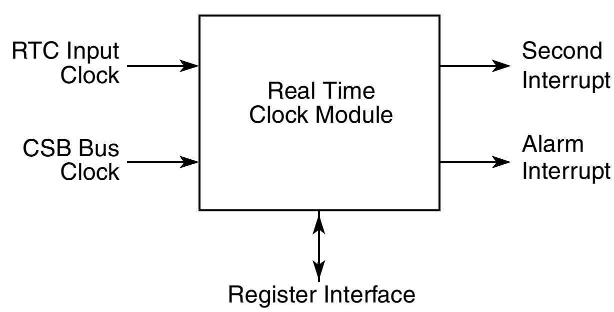


Figure 5-30. Real Time Clock Module High Level Block Diagram

5.5.2 RTC Features

The key features of the RTC include the following:

- Maintains a one-second count, unique over a period of thousand of years
- 32-bit RTC counter can be initialized by software to specific initial count value
- Provides an alarm function with programmable and maskable alarm interrupt
- Provides programmable and maskable every second interrupt

- Uses two possible clock sources: the CSB bus clock or an external RTC clock
- RTC function can be disabled if needed

5.5.3 RTC Modes of Operation

The RTC unit can operate in the following modes:

- RTC enable/disable mode
- RTC every-second interrupt enable/disable mode
- RTC alarm interrupt enable/disable mode
- RTC internal/external input clock mode

5.5.4 RTC External Signal Description

This section provides an overview and detailed descriptions of the RTC signals.

There is one distinct external RTC clock input signal, defined in Table 5-42.

Table 5-42. RTC Signal Properties

Name	Port	Function	I/O	Reset	Pull Up
RTC_CLK	RTC_CLK	Real time clock input.	I	N/A	—

Table 5-43 provides a detailed description of the external RTC signal.

Table 5-43. RTC External Signal—Detailed Signal Description

Signal	I/O	Description	
RTC_CLK	I	This signal is used as the timebase for the real time clock module.	
		State Meaning	—
		Timing	—

5.5.5 RTC Memory Map/Register Definition

The RTC programmable register map occupies 32 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All RTC registers are 32 bits wide that are located on 32-bit address boundaries and should only be accessed as a 32-bit quantities.

All addresses used in this section are offsets from the RTC base, as defined in Chapter 2, “Memory Map.”

Table 5-38 shows the memory map of the RTC.

Table 5-44. RTC Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00	Real time counter control register (RTCNR)	R/W	0x0000_0000	5.5.5.1/5-45
0x04	Real time counter load register (RTLDR)	R/W	0x0000_0000	5.5.5.2/5-46
0x08	Real time counter prescale register (RTPSR)	R/W	0x0000_0000	5.5.5.3/5-46
0x0C	Real time counter register (RTCTR)	R	0x0000_0000	5.5.5.4/5-47
0x10	Real time counter event register (RTEVR)	w1c	0x0000_0000	5.5.5.5/5-47
0x14	Real time counter alarm register (RTALR)	R/W	0xFFFF_FFFF	5.5.5.6/5-48
0x18–0x1F	Reserved	—	—	

5.5.5.1 Real Time Counter Control Register (RTCNR)

The real time counter control register (RTCNR), shown in Figure 5-31, is used to enable RTC functions. The register can be read at any time.

Offset 0x00 Access: Read/Write

	0–23	24	25	26–29	30	31
R W	—	CLEN	CLIN	—	AIM	SIM
Reset	All zeros					

Figure 5-31. Real Time Counter Control Register (RTCNR)

Table 5-45 defines the bit fields of RTCNR.

Table 5-45. RTCNR Bit Settings

Bits	Name	Description
0–23	—	Write reserved, read = 0
24	CLEN	Clock enable control bit. This bit controls the counting of the RTC. When the RTC's clock is disabled, the counter maintains its old value. When the counter's clock is enabled, it continues counting using the previous value. 0 Disable counter. 1 Enable counter.
25	CLIN	Input clock control bit. The input clock to the RTC may be either the CSB clock or an external RTC clock. 0 The input clock to the periodic interrupt timer is CSB input clock. 1 The input clock to the periodic interrupt timer is the external RTC clock.
26–29	—	Write reserved, read = 0

Table 5-45. RTCNR Bit Settings (continued)

Bits	Name	Description
30	AIM	Alarm interrupt mask bit. Used to enable or disable (mask) the RTC alarm interrupt when the RTC's 32-bit counter reaches RTALR[ALR] value. 0 Alarm interrupt generation disabled. 1 Alarm interrupt generation enabled.
31	SIM	Second interrupt mask bit. Used to enable or disable (mask) the RTC periodic interrupt. 0 Periodic interrupt generation disabled. 1 Periodic interrupt generation enabled.

5.5.5.2 Real Time Counter Load Register (RTLDR)

The real time counter load register (RTLDR), shown in Figure 5-32, contains the 32-bit value to be loaded in the 32-bit RTC counter.

Offset 0x04 Access: Read/Write

	0 … 31
R W	CLDV
Reset	All zeros

Figure 5-32. Real Time Counter Load Register (RTLDR)

Table 5-46 defines the bit fields of RTLDR.

Table 5-46. RTLDR Bit Settings

Bits	Name	Description
0–31	CLDV	Contains the 32-bit value to be loaded in the 32-bit RTC counter.

5.5.5.3 Real Time Counter Prescale Register (RTPSR)

The real time counter prescale register (RTPSR), shown in Figure 5-33, is a read/write register used to configure the RTC prescaler's value.

Offset 0x08 Access: Read/Write

	0 … 31
R W	PRSC
Reset	All zeros

Figure 5-33. Real Time Counter Prescale Register (RTPSR)

Table 5-47 defines the bit fields of RTPSR.

Table 5-47. RTPSR Bit Settings

Bits	Name	Description
0–31	PRSC	RTC prescaler bits. Select the input clock divider for the RTC counter clock. The prescaler is programmed to divide the RTC clock input by values from 1 to 4,294,967,296. The value 0x0000 divides the clock by 1 and 0xFFFF_FFFF divides the clock by 4,294,967,296. To accurately predict the timing of the next count, change the RTPSR[PRSC] field only when the enable bit RTCNR[CLE] is clear. Changing the RTPSR[PRSC] bits resets the prescaler counter. System reset and the loading of a new value into the counter both reset the prescaler counter. Clearing RTCNR[CLE] stops the prescaler counter.

5.5.5.4 Real Time Counter Register (RTCTR)

The real time counter register (RTCTR), shown in Figure 5-34, is a read-only register that shows the current value in the RTC counter.

The CNTV value is not affected by reads or writes to RTCTR.

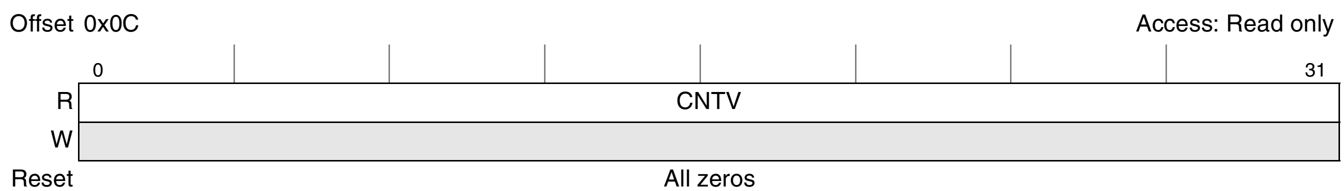


Figure 5-34. Real Time Counter Register (RTCTR)

Table 5-48 defines the bit fields of RTCTR.

Table 5-48. RTCTR Bit Settings

Bits	Name	Description
0–31	CNTV	RTC counter value field. RTCTR[CNTV] contains the current value of the time counter. This is a read-only field. Writes have no effect on RTCTR[CNTV].

5.5.5.5 Real Time Counter Event Register (RTEVR)

The real time counter event register (RTEVR), shown in Figure 5-35, is used to report the source of the interrupts. The register can be read at any time.

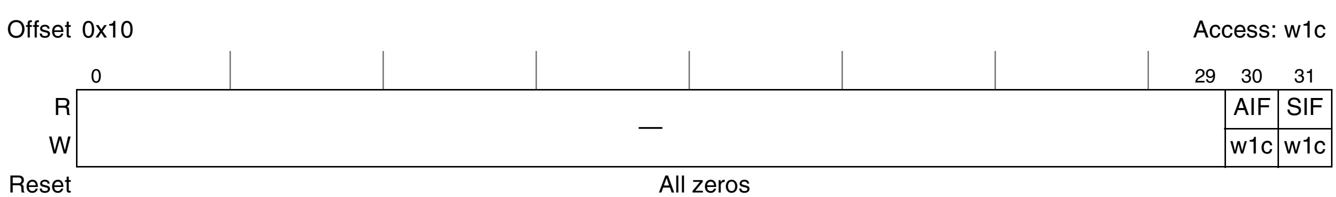


Figure 5-35. Real Time Counter Event Register (RTEVR)

RTEVR bits are cleared by writing ones. Writing zeros does not affect the value of the status bits.

Table 5-49 defines the bit fields of RTEVR.

Table 5-49. RTEVR Bit Settings

Bits	Name	Description
0–29	—	Write reserved, read = 0
30	AIF	Alarm interrupt flag bit. Used to indicate the alarm interrupt. It is set if the RTC issues an interrupt after the RTC counter counts to zero.
31	SIF	Second interrupt flag bit. Used to indicate the every-second interrupt. This status bit is set each time that the prescaler count reaches zero and should be cleared by software.

5.5.5.6 Real Time Counter Alarm Register (RTALR)

The real time counter alarm register (RTALR), shown in Figure 5-36, contains the 32-bit alarm (ALRM) value. When the value of the RTC counter equals the RTALR[ALRM] value, a maskable interrupt is generated.

Offset 0x14 Access: Read/Write

	0–31
R W	ALRM
Reset	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 5-36. Real Time Counter Alarm Register (RTALR)

Table 5-50 defines the bit fields of RTALR.

Table 5-50. RTALR Bit Settings

Bits	Name	Description
0–31	ALRM	RTC alarm value. The alarm interrupt is generated when the value of the RTC counter equals RTALR[ALRM].

5.5.6 Functional Description

5.5.6.1 Real Time Counter Unit

The real time clock (RTC) timer is suitable for time stamping or time and calendar generation. It can maintain a one-second count which is unique over a period of approximately 136 years. Software can convert this count into time-of-day or calendar information if required. An alarm function is also provided. The RTC can be clocked by the internal system bus clock or by an external clock source. The RTC consists of 32-bit up-counter which is incremented by an one-second count clock derived from the RTC input clock. The RTC can be programmed to generate a maskable interrupt when the time value matches the value in its associated alarm register.

The RTC can be initialized by software with an initial count value in the real time counter load register (RTLDR). It can also be programmed to generate an interrupt every second. The real time counter control register (RTCTR) is used to enable or disable the various timer functions. The real time counter event register (RTEVR) is used to report the interrupt source. The RTC counter is reset to zero on hard reset but is not affected by soft reset. It is initialized by the software. The RTC function can be disabled.

Figure 5-37 shows the functional RTC block diagram.

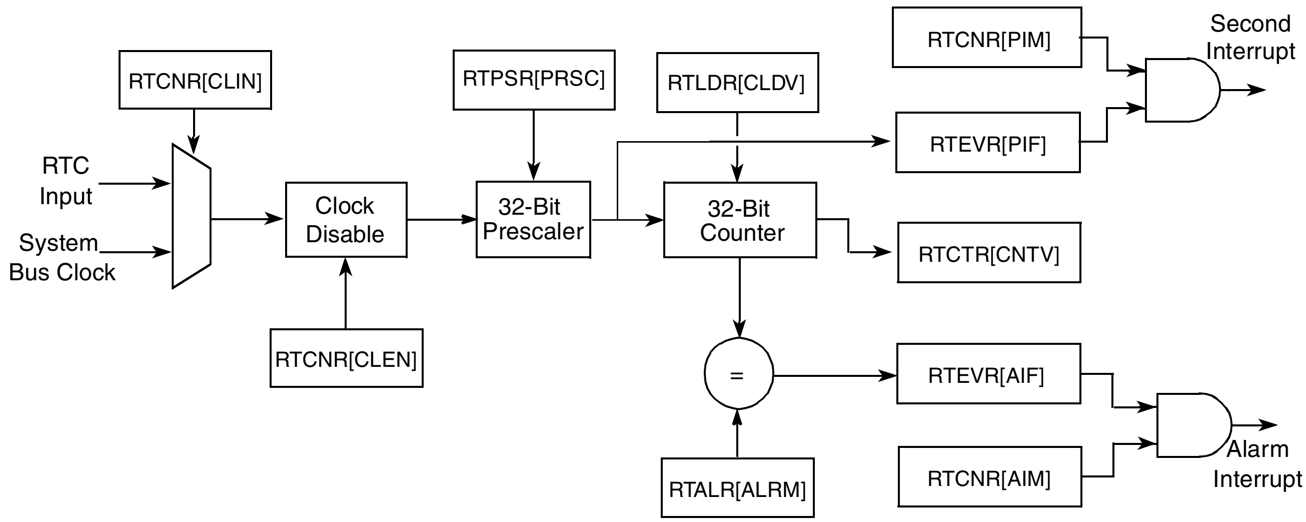


Figure 5-37. Real Time Clock Module Functional Block Diagram

5.5.6.2 RTC Operational Modes

The RTC unit can operate in the following modes:

- RTC enable/disable mode:

  RTCNR[CLEN] enables the RTC timer. It should be set by software after a system reset to enable the RTC timer.

  — RTC disable mode (RTCNR[CLEN] = 0)

  When the RTC's clock is disabled, counter maintains its old value (default).

  — RTC enable mode (RTCNR[CLEN] = 1)

  When the counter's clock is enabled, it continues counting using the previous value.

- RTC every-second interrupt enable/disable mode:

  — RTC every-second interrupt enable mode (RTCNR[SIM] = 1)

  In this mode the RTC set the RTEVR[SIF] flag and generate an interrupt after the RTC's 32-bit counter reaches zero.

  — RTC every-second interrupt disable mode (RTCNR[SIM] = 0)

  In this mode the RTC sets the RTEVR[SIF] flag but does not generate an interrupt after the RTC's 32-bit counter reaches zero.

- RTC alarm interrupt enable/disable mode:

  — RTC alarm interrupt enable mode (RTCNR[AIM] = 1)

  In this mode, the RTC sets the RTEVR[AIF] flag and generates an interrupt each time when the RTC's 32-bit counter reaches the RTALR[ALR] value.

— RTC alarm interrupt disable mode (RTCNR[AIM] = 0)

In this mode the RTC sets the RTEVR[AIF] flag but does not generate an interrupt when the RTC's 32-bit counter reaches the RTALR[ALR] value.

- RTC internal/external input clock mode:

  The input clock to the RTC may be the CSB clock or an external 32.768-kHz crystal.

  — RTC uses the internal input clock mode (RTCNR[CLIN] = 0)

  — RTC uses the external 32.768-kHz crystal clock (RTCNR[CLIN] = 1)

5.5.7 RTC Programming Guidelines

The following initialization sequence for the RTC is recommended:

1. Write to RTPSR to set the RTC prescaler to the desired value
2. Write to RTLDR to initialize the RTC initial value
3. Write to RTALR to program the RTC alarm value, if needed
4. Write to RTCNR to configure and start the RTC operation: RTC input clock source, second/alarm interrupt mask, RTC clock enable.

5.6 Periodic Interval Timer (PIT)

The following sections describe theory of operation of the periodic interval timer (PIT) including a definition of the external signals and the functions it serves. Additionally, the configuration, control, and status registers are described. Note that individual chapters in this reference manual describe additional specific initialization aspects for each individual block.

5.6.1 PIT Overview

The periodic interval timer (PIT) that generates periodic interrupts for a real-time operating system or an application software.

The PIT consists of a 32-bit down-counter which is decremented by a clock derived from a CSB clock or from an external 32.768-kHz crystal. The 32-bit counter decrements to zero when loaded with a initial value from the periodic interval timer load register (PTLDR). The periodic interval timer control register (PTCTR) is used to enable or disable the various timer functions. The periodic interval timer event register (PTEVR) is used to report the interrupt source. The PIT function can be disabled if needed.

Figure 5-38 shows the functional PIT block diagram.

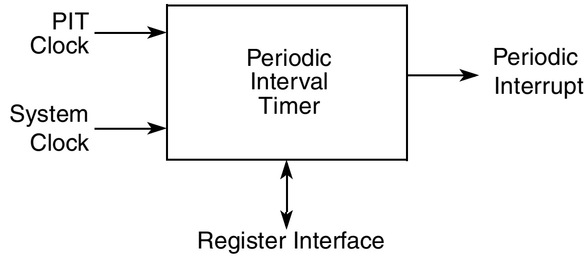


Figure 5-38. Periodic Interval Timer High Level Block Diagram

5.6.2 PIT Features

The key features of the PIT include the following:

- Maintains a 32-bit down-counter, clocked by a 32-bit prescaled input clock
- 32-bit PIT counter can be initialized by software to specific initial count value
- Provides programmable and maskable periodic interrupt
- Uses two possible clock sources: the CSB clock or an external PIT clock
- PIT function can be disabled

5.6.3 PIT Modes of Operation

The PIT unit can operate in the following modes:

- PIT enable/disable mode
- PIT periodic interrupt enable/disable mode
- PIT internal/external input clock mode

5.6.4 PIT External Signal Description

This section provides an overview and detailed descriptions of the PIT signals.

There is one distinct external input signal (PIT clock), defined in Table 5-51.

Table 5-51. PIT Signal Properties

Name	Port	Function	I/O	Reset	Pull Up
PIT_CLK	PIT_CLK	Periodic interval timer.	I	N/A	—

Table 5-52 describes of the external PIT signal.

Table 5-52. PIT External Signal—Detailed Signal Descriptions

Signal	I/O	Description	
PIT_CLK	I	This signal is used as the timebase for the periodic interval timer module.	
		State Meaning	—
		Timing	—

5.6.5 PIT Memory Map/Register Definition

The PIT programmable register map occupies 32 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All PIT registers are 32 bits wide and reside on 32-bit address boundaries and should only be accessed as 32-bit quantities.

All addresses used in this chapter are offsets from PIT base, as defined in Chapter 2, “Memory Map.”

Table 5-53 shows the PIT memory map.

Table 5-53. PIT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00	Periodic interval timer control register (PTCNR)	R/W	0x0000_0000	5.6.5.1/5-52
0x04	Periodic interval timer load register (PTLDR)	R/W	0x0000_0000	5.6.5.2/5-53
0x08	Periodic interval timer prescale register (PTPSR)	R/W	0x0000_0000	5.6.5.3/5-54
0x0C	Periodic interval timer counter register (PTCTR)	R	0x0000_0000	5.6.5.4/5-54
0x10	Periodic interval timer event register (PTEVR)	w1c	0x0000_0000	5.6.5.5/5-55
0x14–0x1F	Reserved	—	—	

5.6.5.1 Periodic Interval Timer Control Register (PTCNR)

The periodic interval timer control register (PTCNR), shown in Figure 5-39, is used to enable the different PIT functions. The register can be read at any time.

Offset 0x00 Access: Read/Write

	0–23	24	25	26–30	31
R/W	—	CLEN	CLIN	—	PIM
Reset	All zeros				

Figure 5-39. Periodic Interval Timer Control Register (PTCNR)

Table 5-54 defines the bit fields of PTCNR.

Table 5-54. PTCNR Bit Settings

Bits	Name	Description
0–23	—	Write reserved, read = 0
24	CLEN	Clock enable control bit. Controls the counting of the PIT. When the PIT's clock is disabled, the counter maintains its old value. When the counter's clock is enabled, it continues counting using the previous value. 0 Disable counter. 1 Enable counter.
25	CLIN	Input clock control bit. The input clock to the PIT can be either an internal system clock or an external PIT clock. 0 The input clock to the periodic interrupt timer is internal system clock. 1 The input clock to the periodic interrupt timer is external PIT clock.
26–30	—	Write reserved, read = 0
31	PIM	Periodic interrupt mask bit. Used to enable or disable (mask) the PIT periodic interrupt. 0 Periodic interrupt generation disabled. 1 Periodic interrupt generation enabled.

5.6.5.2 Periodic Interval Timer Load Register (PTLDR)

The periodic interval timer load register (PTLDR), shown in Figure 5-40, contains the 32-bit value to be loaded in a 32-bit PIT counter.

Offset 0x04 — Access: Read/Write

	0–31
R / W	CLDV
Reset	All zeros

Figure 5-40. Periodic Interval Timer Load Register (PTLDR)

Table 5-55 defines the bit fields of PTLDR.

Table 5-55. PTLDR Bit Settings

Bits	Name	Description
0–31	CLDV	Contains the 32-bit value to be loaded in a 32-bit PIT counter.

5.6.5.3 Periodic Interval Timer Prescale Register (PTPSR)

The periodic interval timer prescale register (PTPSR), shown in Figure 5-41, is a read/write register that used to configure the PIT prescaler's value.

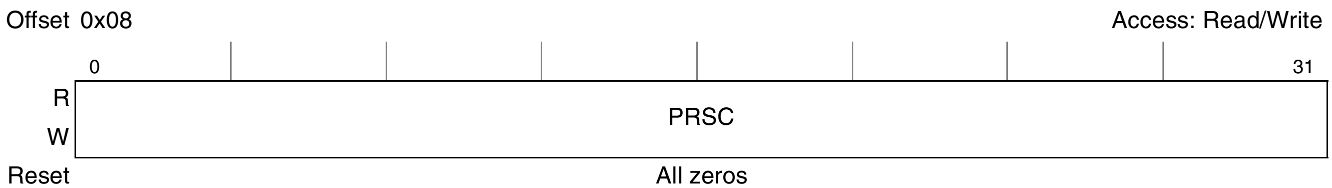


Figure 5-41. Periodic Interval Timer Prescale Register (PTPSR)

Table 5-56 defines the bit fields of PTPSR.

Table 5-56. PTPSR Bit Settings

Bits	Name	Description
0–31	PRSC	PIT prescaler bits. Selects the input clock divider to generate the PIT counter clock. The prescaler is programmed to divide the PIT clock input by values from 1 to 4,294,967,296. The value 0x0000 divides the clock by 1 and 0xFFFF_FFFF divides the clock by 4,294,967,296. To accurately predict the timing of the next count, change the PRSC bit only when the enable bit PTCNR[CLE] is clear. Changing PRSC resets the prescaler counter. System reset and the loading of a new value into the counter also reset the prescaler counter. Clearing the PTCNR[CLE] bit stops the prescaler counter.

5.6.5.4 Periodic Interval Timer Counter Register (PTCTR)

The periodic interval timer counter register (PTCTR), shown in Figure 5-42, is a read-only register that shows the current value in the PIT counter. The PTCTR counter is not affected by reads or writes.

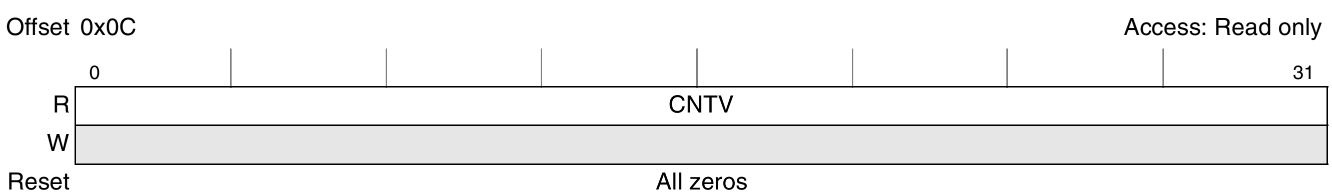


Figure 5-42. Periodic Interval Timer Counter Register (PTCTR)

Table 5-57 defines the bit fields of PTCTR.

Table 5-57. PTCTR Bit Settings

Bits	Name	Description
0–31	CNTV	PIT counter value field. Contains the current value of the time counter. This is a read-only field. Writes have no effect on PTCTR[CNTV].

5.6.5.5 Periodic Interval Timer Event Register (PTEVR)

The periodic interval timer event register (PTEVR), shown in Figure 5-43, is used to report the source of the interrupts. The register can be read at any time.

PTEVR bits are cleared by writing ones. Writing zeros does not affect the value of the status bits.

Offset 0x10 Access: w1c

	0–30	31
R	—	PIF
W		w1c
Reset	All zeros	

Figure 5-43. Periodic Interval Timer Event Register (PTEVR)

Table 5-58 defines the bit fields of PTEVR.

Table 5-58. PTEVR Bit Settings

Bits	Name	Description
0–30	—	Write reserved, read = 0
31	PIF	Periodic interrupt flag bit. Used to indicate the periodic interrupt. Its asserted if the PIT issues an interrupt after the SPMPIT counter counts to zero. This status bit should be cleared by software.

5.6.6 Functional Description

5.6.6.1 Periodic Interval Timer Unit

The PIT generates periodic interrupts for use with a real-time operating system or the application software. It consists of a 32-bit down-counter which is decremented by a clock derived from the CSB clock or from the PIT clock. The 32-bit counter decrements to zero when loaded with a initial value from the periodic interval timer load register (PTLDR). After the timer reaches zero, PTEVR[PIF] is set and an interrupt is generated if PTCNR[PIM] = 1. At the next count cycle, the value in the PTLDR[CLDV] is loaded into the counter and the process repeats. When a new value is loaded into the PTLDR[CLDV], the PIT is updated, the prescaler counter is reset, and the counter begins counting. Setting of PTEVR[PIF] generates an interrupt, that remains pending until PTEVR[PIF] is cleared. If PTEVR[PIF] is set again before being cleared, the interrupt remains pending until PTEVR[PIF] is cleared. Any write to the PTLDR[CLDV] stops the current countdown and the count resumes with the new value in PTLDR[CLDV]. If PTCNR[CLEN] = 0, the PIT cannot count and retains the old count value. PTCTR contain the PIT current value. The PIT function can be disabled if needed.

Figure 5-44 shows the functional PIT block diagram.

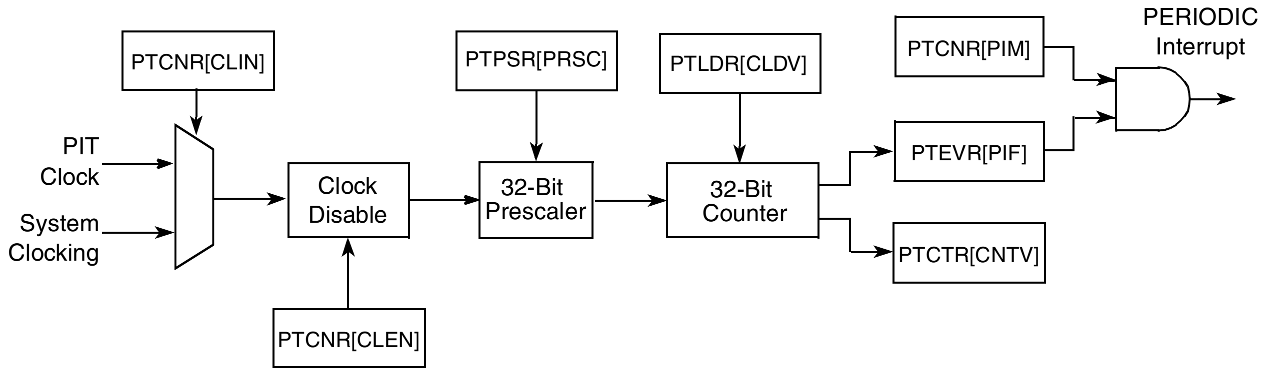


Figure 5-44. Periodic Interval Timer Functional Block Diagram

5.6.6.2 PIT Operational Modes

The PIT unit can operate in the following modes:

- PIT enable/disable mode:

  The PTCNR[CLEN] bit enables the PIT timer. It should be set by software after a system reset to enable the PIT timer.

  — PIT disable mode (PTCNR[CLEN] = 0). When the PIT's clock is disabled, counter maintains its old value.

  — PIT enable mode (PTCNR[CLEN] = 1). When the counter's clock is enabled, it continues counting using the previous value.
- PIT periodic interrupt enable/disable mode:

  — PIT periodic interrupt enable mode (PTCNR[PIM] = 1). After the PIT's 32-bit counter reaches zero, the PIT sets the PTEVR[PIF] flag and generates an interrupt.

  — PIT periodic interrupt disable mode (PTCNR[PIM] = 0). After the PIT's 32-bit counter reaches zero, the PIT sets the PTEVR[PIF] flag but does not generate an interrupt.
- PIT internal/external input clock mode:

  The input clock to the PIT may be an internal system clock or the PIT clock.

  — PIT use the internal input clock mode (PTCNR[CLIN] = 0)

  — PIT use the PIT clock (PTCNR[CLIN] = 1)

5.6.7 PIT Programming Guidelines

The following initialization sequence of PIT is recommended:

1. Write to PTPSR to set the PIT prescaler to the desired value
2. Write to PTLDR to initialize the PIT initial value
3. Write to PTCNR to configure and start the PIT operation: PIT input clock source, periodic interrupt mask, PIT clock enable.

See Section 5.5.7, "RTC Programming Guidelines," for real-time clock programming guidelines.

5.7 General-Purpose Timers (GTMs)

The following sections describe theory of operation of the general purpose (global) timer module, including a definition of the external signals and the functionality. Additionally, the configuration, control, and status registers are described. Note that individual chapters in this book describe additional specific initialization aspects for each individual block.

5.7.1 GTM Overview

Each global timer module (GTM) includes four identical 16-bit general-purpose timers, two 32-bit timers or one 64-bit timer. Each GTM timer consists of a timer prescale register (GTPSR), a timer mode register (GTMDR) a timer capture register (GTCPR), a timer counter register (GTCNR), a timer reference register (GTRFR), a timer event register (GTEVR), and a timer global configuration register (GTCFR). The GTPSRs and the GTMDRs contain the primary and secondary prescalers, programmed by the user.

Figure 5-45 shows the functional GTM block diagram.

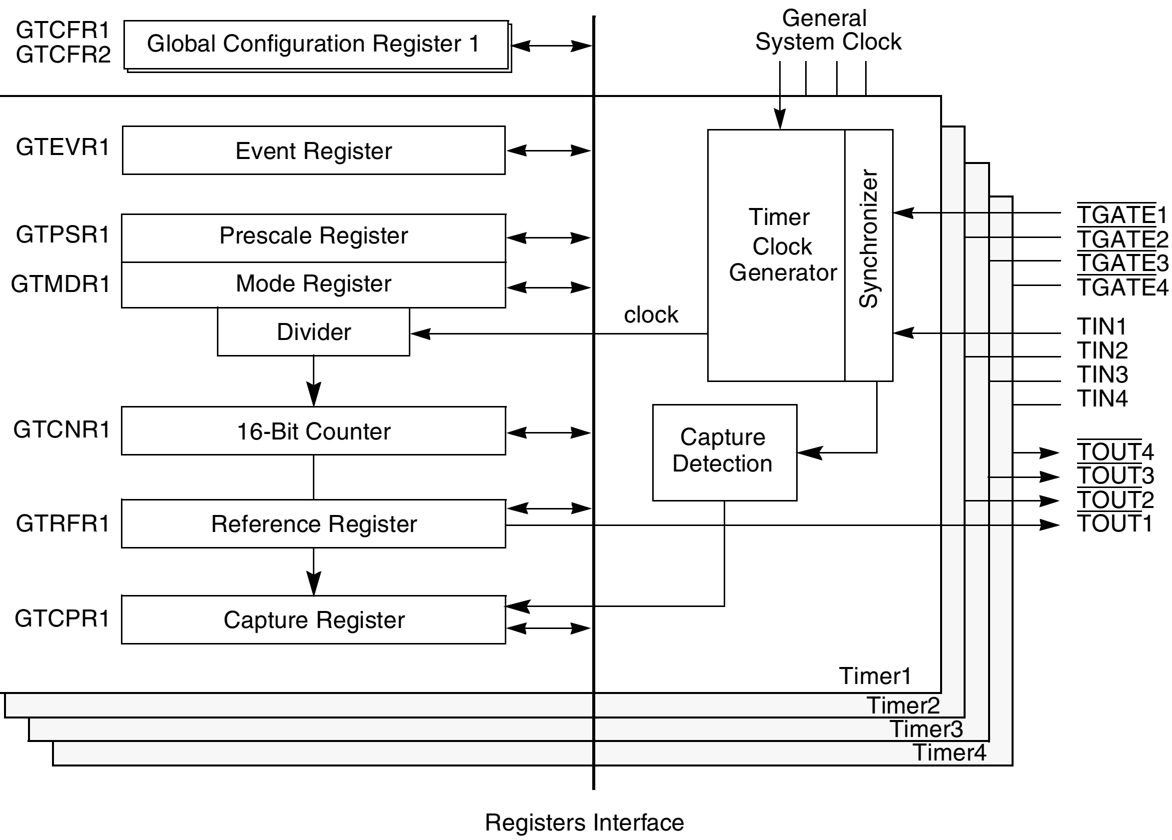


Figure 5-45. Global Timers Block Diagram

5.7.2 GTM Features

The key features of the timer include the following:

- The maximum input clock is the system bus clock
- Four 16-bit programmable timers
- Two timers cascaded internally or externally to form a 32-bit timer
- One timer cascaded internally or externally to form a 64-bit timer
- Maximum period of ~50 msecond (at 333-MHz bus clock and prescaler = 256) for 16-bit timer
- Maximum period of ~12.8 seconds (at 333-MHz bus clock and prescaler = 256) for 32-bit timer
- Maximum period of thousands of years (at 333-MHz bus clock and prescaler = 256) for 64-bit timer
- 3-nanosecond timer resolution (at 333-MHz bus clock and no prescaler)
- Resolution and maximum period can be traded off by selecting prescaler divisor
- Three programmable input clock sources for the timer prescalers
- Input capture capability
- Output compare with programmable mode for the output pin
- Free run and restart modes
- Functional and programming compatibility with MPC8260 timers

5.7.3 GTM Modes of Operation

The GTM unit can operate in the following modes:

- Cascaded modes
- Clock source modes
- Reference modes
- Capture modes

5.7.3.1 Cascaded Modes

GTCFR*n*[PCAS] and GTCFR2[SCAS] are used to put the timers into different cascaded modes:

- Non-cascaded mode: Each timer (timer 1, timer 2, timer 3, and timer 4), function as a independent 16-bit timer with a 16-bit GTRFR, GTCPR, GTMDR, and GTCNR. In this mode, the non-cascaded GTRFR, GTCPR, and GTCNR should be referenced with corresponding 16-bit bus cycles.
- Pair-cascaded mode: In this mode, two 16-bit timers can be internally cascaded to form a 32-bit counter: timer 1 can be internally cascaded to timer 2 and timer 3 may be internally cascaded to timer 4. Because the decision to cascade timers is made independently, the user has the option of selecting two 16-bit timers and one 32-bit timer, or two 32-bit timers. When working in the pair-cascaded mode, the cascaded GTRFR, GTCPR, and GTCNR should be referenced with 32-bit bus cycles.

- Super-cascaded mode: In this mode, all four 16-bit timers can be internally cascaded to form a 64-bit counter. When working in the super-cascaded mode, the cascaded GTRFR, GTCPR, and GTCNR should be referenced with two 32-bit bus cycles.

5.7.3.2 Clock Source Modes

The clock input to the timer's prescaler can be selected from three sources:

- The system clock
- The system slow go clock (system bus clock internally divided by 16)
- The corresponding TINx pin

5.7.3.3 Reference Modes

Each timer can be configured to count until a reference is reached and then either begin a new time count immediately or continue to run. The FRR bit of the corresponding GTMRR selects each mode.

- Free run reference mode. The corresponding timer count continues to increment after the reference value is reached.
- Reset reference mode. The corresponding timer count is reset immediately after the reference value is reached.

5.7.3.4 Capture Modes

Each timer has a 16-bit field in GTCPR, used to latch the value of the counter when a defined transition of TINx is sensed by the corresponding input capture edge detector.

- Normal gate mode enables the count on a falling edge of the $\overline{\text{TGATE}}$ pin and disables the count on the rising edge of $\overline{\text{TGATE}}$. This mode allows the timer to count conditionally, based on the state of $\overline{\text{TGATE}}$.
- The restart gate mode performs the same function as normal mode, except it also resets the counter on the falling edge of the $\overline{\text{TGATE}}$ pin. This mode has applications in pulse interval measurement and bus monitoring.

5.7.4 GTM External Signal Description

This section provides an overview and detailed descriptions of the GTM signals.

There are four distinct external input timer capture signals (TIN1, TIN2, TIN3, and TIN4), four distinct external input timer get signals ($\overline{\text{TGATE1}}$, $\overline{\text{TGATE2}}$, $\overline{\text{TGATE3}}$, and $\overline{\text{TGATE4}}$), and four distinct external timer output signals ($\overline{\text{TOUT1}}$, $\overline{\text{TOUT2}}$, $\overline{\text{TOUT3}}$, and $\overline{\text{TOUT4}}$). The GTM interface signals are defined in Table 5-59.

Table 5-59. GTM Signal Properties

Name	Port	Function	I/O	Reset	Require Pull Up
TIN1	TIN1	Global timer 1 capture control signal	I	0	No
TIN2	TIN2	Global timer 2 capture control signal	I	0	No
TIN3	TIN3	Global timer 3 capture control signal	I	0	No
TIN4	TIN4	Global timer 4 capture control signal	I	0	No
$\overline{\text{TGATE1}}$	$\overline{\text{TGATE1}}$	Global timer 1 counter gate control signal	I	0	No
$\overline{\text{TGATE2}}$	$\overline{\text{TGATE2}}$	Global timer 2 counter gate control signal	I	0	No
$\overline{\text{TGATE3}}$	$\overline{\text{TGATE3}}$	Global timer 3 counter gate control signal	I	0	No
$\overline{\text{TGATE4}}$	$\overline{\text{TGATE4}}$	Global timer 4 counter gate control signal	I	0	No
$\overline{\text{TOUT1}}$	$\overline{\text{TOUT1}}$	Global timer 1 counter output signal	O	1	No
$\overline{\text{TOUT2}}$	$\overline{\text{TOUT2}}$	Global timer 2 counter output signal	O	1	No
$\overline{\text{TOUT3}}$	$\overline{\text{TOUT3}}$	Global timer 3 counter output signal	O	1	No
$\overline{\text{TOUT4}}$	$\overline{\text{TOUT4}}$	Global timer 4 counter output signal	O	1	No

Table 5-60 provides detailed descriptions of the external GTM signals.

Table 5-60. GTM External Signals—Detailed Signal Descriptions

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">TINn</td><td rowspan="3">I</td><td colspan="2">Global timer capture control signal. Used to latch the value of the counter when a defined transition of TINn is sensed by the corresponding input capture edge detector.</td></tr>
<tr><td>State Meaning</td><td>Asserted/Negated —According to the programmed polarity by the corresponding GTMDRn[CE]). Each timer has a 16-bit GTCPR used to latch the value of the counter when a defined transition of TINn is sensed by the corresponding input capture edge detector. Upon a capture or reference event, the corresponding GTEVR bit is set and a maskable interrupt request is issued to the interrupt controller.</td></tr>
<tr><td>Timing</td><td>Assertion/Negation—Asynchronous to internal bus clock. TINn is internally synchronized to the system bus clock. If TINn meets the asynchronous input setup time, the value of counter is captured after one system bus clock when working with the internal clock.</td></tr>
</table>

Table 5-60. GTM External Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
TGATE*n*	I	Global timer counter gate control signal. Used to gate/restart the counter when a defined transition of TGATE*n* is sensed by the corresponding input capture edge detector.	
		State Meaning	Asserted/Negated—According to the programmed polarity by the corresponding GTCFR[GMx] bits. In a reset gate mode (GTCFR[GM*n*] = 0), the TGATE*n* pin is used to enable/disable count. A falling TGATE*n* pin enables and restarts the count and a rising edge of TGATE*n* disables the count. In a normal gate mode (GTCFR[GM*n*] = 1), the TGATE*n* have similar functionality, except the falling edge of TGATE*n* does not restart the appropriate count value in GTCNR*n*[CNV*n*].
		Timing	Assertion/Negation—Asynchronous to internal bus clock. TGATE*n* is internally synchronized to the system bus clock. If TGATE*n* meets the asynchronous input setup time, the counter begins counting after one system bus clock when working with the internal clock.
TOUT*n*	O	Global timer counter output signal. The GTM output a signal on the timer output pin TOUT*n* when the reference value is reached.	
		State Meaning	Asserted/Negated—According to the programmed polarity by the corresponding GTMDR*n*[OM*n*]. 1. Active-low pulse on TOUT*n* for one timer input clock cycle as defined by the GTMDR*n*[ICLK*n*] bits (GTMDR*n*[OM*n*] = 1). Thus, TOUT*n* may be low for one general system clock period, one general system slow go clock period, or one TIN*n* pin clock cycle period. 2. Toggle the TOUT*n* pin (GTMDR*n*[OM*n*] = 0). TOUT*n* changes occur on the rising edge of the system clock.
		Timing	Assertion/Negation—TOUT*n* changes occur on the rising edge of the system clock.

5.7.5 GTM Memory Map/Register Definition

The GTM programmable register map occupies 64 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All GTM registers are 8 or 16 bits wide, located on 8-bit or 16-bit address boundaries, and should only be accessed as 8-bit or 16-bit quantities. All addresses used in this chapter are offsets from GPT Base, as defined in Chapter 2, “Memory Map.”

Table 5-61 shows the memory map of the GTM.

Table 5-61. GTM Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
General Purpose (Global) Timer Module 1—Block Base Address 0x0_0500				
0x00	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x00	5.7.5.1/5-62
0x01–0x03	Reserved	—	—	—
0x04	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x00	5.7.5.1/5-62
0x05–0x0F	Reserved	—	—	—

Table 5-61. GTM Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x10	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	5.7.5.2/5-66
0x12	Timer 2 global timers mode register (GTMDR2)			
0x14	Timer 1 global timers reference register (GTRFR1)	R/W	0xFFFF	5.7.5.3/5-67
0x16	Timer 2 global timers reference register (GTRFR2)			
0x18	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	5.7.5.4/5-67
0x1A	Timer 2 global timers capture register (GTCPR2)			
0x1C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	5.7.5.5/5-68
0x1E	Timer 2 global timers counter register (GTCNR2)			
0x20	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	5.7.5.2/5-66
0x22	Timer 4 global timers mode register (GTMDR4)			
0x24	Timer 3 global timers reference register (GTRFR3)	R/W	0xFFFF	5.7.5.3/5-67
0x26	Timer 4 global timers reference register (GTRFR4)			
0x28	Timer 3 global timers capture register (GTCPR3)	R	0x0000	5.7.5.4/5-67
0x2A	Timer 4 global timers capture register (GTCPR4)			
0x2C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	5.7.5.5/5-68
0x2E	Timer 4 global timers counter register (GTCNR4)			
0x30	Timer 1 global timers event register (GTEVR1)	w1c	0x0000	5.7.5.6/5-68
0x32	Timer 2 global timers event register (GTEVR2)			
0x34	Timer 3 global timers event register (GTEVR3)			
0x36	Timer 4 global timers event register (GTEVR4)			
0x38	Timer 1 global timers prescale register (GTPSR1)	R/W	0x0003	5.7.5.7/5-69
0x3A	Timer 2 global timers prescale register (GTPSR2)			
0x3C	Timer 3 global timers prescale register (GTPSR3)			
0x3E	Timer 4 global timers prescale register (GTPSR4)			
General Purpose (Global) Timer Module 2: All registers defined for GTM1 are also defined for GTM2; the base address of GTM2 registers is 0x0_06*nn*.				

5.7.5.1 Global Timers Configuration Registers (GTCFR*n*)

The global timers configuration registers (GTCFR1 and GTCFR2), shown in Figure 5-46 and Figure 5-47, contain configuration parameters used by the timers. These registers allow simultaneous starting, stopping and resetting of a pair of timers (1 and 2 or 3 and 4) or of a groups of timers (1, 2, 3, and 4) if one bus cycle is used. GTCFR is cleared by reset.

NOTE

For proper operation of the timers, do not change the modes of operation and enable the timer in the same register write operation. The modes can be changed when GTCFR*n*[RST*n*] is cleared. However, when GTCFR*n*[RST*n*] are set, they are the only bits that can be changed.

Offset 0x00 | Access: Read/Write

	0	1	2	3	4	5	6	7
R W	PCAS	BCM	STP2	RST2	GM2	GM1	STP1	RST1
Reset	All zeros							

Figure 5-46. Global Timers Configuration Register 1 (GTCFR1)

Table 5-62 defines the bit fields of GTCFR1.

Table 5-62. GTCFR1 Bit Settings

Bits	Name	Description
0	PCAS	Pair-cascade mode 0 Normal operation 1 Timers 1 and 2 cascade to form a 32-bit timer. **Note:** This bit is ignored in super-cascade mode (GTCFR2[SCAS] = 1). **Note:** It is allowed to change the value of this bit only when the corresponding timers are in reset mode. Thus, the user should first clear the RST1 and RST2 bits (without changing PCAS) and then, in a separate write to the register, change the value of PCAS.
1	BCM	Backward compatible mode 0 Provide backward compatibility to PowerQUICC II family timers. In this mode GTCFR1[GM2] bit will control the gate mode for timers 1 and 2 and GTCFR2[GM4] bit will control the gate mode for timers 3 and 4. GTCFR1[GM1] and GTCFR2[GM3] bits are ignored. 1 Normal operational mode
2	STP2	Stop timer 2 0 Normal operation 1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 2, except the Register Interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.
3	RST2	Reset timer 2 0 Reset the timer 2, including GTMDR2, GTRFR2, GTCNR2, GTCPR2, and GTEVR2 (a software reset is identical to an external reset). 1 Enable the corresponding timer if the STP2 bit is cleared.
4	GM2	Gate mode for $\overline{\text{TGATE2}}$ 0 Restart gate mode. The $\overline{\text{TGATE2}}$ pin is used to enable/disable count. A low level of $\overline{\text{TGATE2}}$ enables and a falling edge of $\overline{\text{TGATE2}}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{\text{TGATE2}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE2}}$ does not restart the appropriate count value in GTCNR2[CNV2].

Table 5-62. GTCFR1 Bit Settings (continued)

Bits	Name	Description
5	GM1	Gate mode for $\overline{\text{TGATE1}}$ 0 Restart gate mode. The $\overline{\text{TGATE1}}$ is used to enable/disable count. A low level of $\overline{\text{TGATE1}}$ enables and a falling edge of $\overline{\text{TGATE1}}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{\text{TGATE1}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE1}}$ does not restart the appropriate count value in GTCNR1[CNV1]. **Note:** In backward compatible mode (GTCFR1[BCM] = 0) this bit is ignored. GTCFR1[GM2] bit will control the gate mode for timers 1 and 2.
6	STP1	Stop timer 1 0 Normal operation 1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 1, except the register interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.
7	RST1	Reset timer 1 0 Reset the timer 1, including GTMDR1, GTRFR1, GTCNR1, GTCPR1, and GTEVR1 (a software reset is identical to an external reset). 1 Enable the corresponding timer if the STP1 bit is cleared.

The GTCFR2 register is shown in Figure 5-47.

Offset 0x04 Access: Read/Write

	0	1	2	3	4	5	6	7
R W	PCAS	SCAS	STP4	RST4	GM4	GM3	STP3	RST3
Reset	All zeros							

Figure 5-47. Global Timers Configuration Register 2 (GTCFR2)

Table 5-63 defines the bit fields of GTCFR2.

Table 5-63. GTCFR2 Bit Settings

Bits	Name	Description
0	PCAS	Pair-cascade mode 0 Normal operation. 1 Timers 3 and 4 cascade to form a 32-bit timer. **Note:** This bit is ignored in super-cascade mode (GTCFR2[SCAS] = 1). **Note:** It is allowed to change the value of this bit only when the corresponding timers are in reset mode. Thus, the user should first clear the RST3 and RST4 bits (without changing PCAS) and then, in a separate write to the register, change the value of PCAS.
1	SCAS	Super cascade mode 0 Normal operation 1 Timers 1, 2, 3 and 4 cascade to form a 64-bit timer. **Note:** In super-cascade mode (GTCFR2[SCAS] = 1) the pair-cascade mode bits are ignored, (GTCFR1/2[PCAS] = Don't Care). **Note:** It is allowed to change the value of this bit only when the corresponding timers are in reset mode. Thus, the user should first clear the RST1, RST2, RST3, and RST4 bits (without changing SCAS) and then, in a separate write to the register, change the value of SCAS.
2	STP4	Stop timer 4 0 Normal operation 1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 4, except the register interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.
3	RST4	Reset timer 4 0 Reset the timer 4, including GTMDR4, GTRFR4, GTCNR4, GTCPR4, and GTEVR4 (a software reset is identical to an external reset). 1 Enable the corresponding timer if the STP4 bit is cleared.
4	GM4	Gate mode for $\overline{\text{TGATE4}}$ 0 Restart gate mode. The $\overline{\text{TGATE4}}$ is used to enable/disable count. A low level of $\overline{\text{TGATE4}}$ enables and a falling edge of $\overline{\text{TGATE4}}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{\text{TGATE4}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE4}}$ does not restart the appropriate count value in GTCNR4[CNV4].
5	GM3	Gate mode for $\overline{\text{TGATE3}}$ 0 Restart gate mode. The $\overline{\text{TGATE3}}$ is used to enable/disable count. A low level of $\overline{\text{TGATE3}}$ enables and a falling edge of $\overline{\text{TGATE3}}$ restarts the count (reset the dynamic counter's count value to 0) and a high level of $\overline{\text{TGATE3}}$ disables the count. 1 Normal gate mode. This mode is the same as 0, except the falling edge of $\overline{\text{TGATE3}}$ does not restart the appropriate count value in GTCNR3[CNV3]. **Note:** In backward compatible mode (GTCFR1[BCM] = 0) this bit is ignored. The GTCFR2[GM4] bit controls the gate mode for timers 3 and 4.
6	STP3	Stop timer 3 0 Normal operation 1 Reduce power consumption of the corresponding timer. This bit stops all clocks to the timer 3, except the register interface clock, which allows to read and write timer registers. The clocks to the timer remain stopped until the user clears this bit or a hardware reset occurs.
7	RST3	Reset timer 3 0 Reset the timer 3, including GTMDR3, GTRFR3, GTCNR3, GTCPR3, and GTEVR3 (a software reset is identical to an external reset). 1 Enable the corresponding timer if the STP3 bit is cleared.

5.7.5.2 Global Timers Mode Registers (GTMDR1–GTMDR4)

The global timers mode registers (GTMDR1, GTMDR2, GTMDR3, and GTMDR4) are shown in Figure 5-48.

Erratic behavior may occur if GTCFR1 and GTCFR2 are not initialized before the GTMDR*n*. Only GTCFR*n*[RST*n*] and GTCFR*n*[STP*n*] can be modified at any time.

Offset 0x10(GTMDR1) 0x12(GTMDR2) 0x20(GTMDR3) 0x22(GTMDR4) — Access: Read/Write

	0–7	8–9	10	11	12	13–14	15
R/W	SPS	CE	OM	ORI	FRR	ICLK	GE
Reset	All zeros						

Figure 5-48. Global Timers Mode Registers (GTMDR1–GTMDR4)

Table 5-64 defines the bit fields of GTMDR.

Table 5-64. GTMDR Bit Settings

Bits	Name	Description
0–7	SPS	Secondary prescaler value The secondary prescaler is programmed to divide the clock input to corresponding timer by values from 1 to 256. The value 0x00 divides the clock by 1 and 0xFF divides the clock by 256.
8–9	CE	Capture edge and enable interrupt 00 Disable interrupt on capture event; capture function is disabled 01 Capture on rising TIN*n* edge only and enable interrupt on capture event. 10 Capture on falling TIN*n* edge only and enable interrupt on capture event. 11 Capture on any TIN*n* edge and enable interrupt on capture event. **Note:** The frequency of TIN*n* should be slower than system clock (TIN*n* is sampled internally by system clock to detect TIN*n*'s rising/falling edge before updating the counter)
10	OM	Output mode 0 Toggle $\overline{\text{TOUT}n}$ every time when the corresponding timer matches its reference value. 1 Active-low pulse on $\overline{\text{TOUT}n}$ for one timer input clock cycle (4 input clock cycles for the system clock) as defined by the ICLK*n* bits. Thus, $\overline{\text{TOUT}n}$ may be low for four general system clocks, one general system slow go clock period, or one TIN*n* pin clock cycle period. **Note:** $\overline{\text{TOUT}n}$ changes are internally synchronized to the rising edge of the system clock
11	ORI	Output reference interrupt enable 0 Disable interrupt for reference reached (does not affect interrupt on capture function). 1 Enable interrupt on reaching the reference value.
12	FRR	Free run/restart mode 0 Free run. The timer count continues to increment after the reference value is reached. 1 Restart. The timer count is reset immediately after the reference value is reached.

Table 5-64. GTMDR Bit Settings (continued)

Bits	Name	Description
13–14	ICLK	Input clock source for the timer. 00 Internally cascaded input. This selection means: For ICLK1, the timer 1 input is the output of timer 2. For ICLK2, the timer 1 input is the output of timer 2, the timer 2 input is the output of timer 3, the timer 3 input is the output of timer 4. For ICLK3, the timer 3 input is the output of timer 4. For ICLK4 this selection means no input clock is provided to the timer. 01 Internal general system bus clock. 10 Internal slow go clock (divided by 16 system bus clock). 11 TIN*n*: corresponding TIN1, TIN2, TIN3, or TIN4 pin (falling edge).
15	GE	Gate enable 0 The $\overline{\text{TGATE}n}$ signal is ignored. 1 The $\overline{\text{TGATE}n}$ signal is used to control the timer.

5.7.5.3 Global Timers Reference Registers (GTRFR1–GTRFR4)

Global timers reference registers, shown in Figure 5-49, are 16-bit memory-mapped, read/write registers containing the 16-bit reference values for each timer's timeout. The reference value is not reached until GTCNR*n*[CNV] increments to the value in GTRFR*n*[TRV].

Offset 0x14(GTRFR1) 0x24(GTRFR3) Access: Read/Write
0x16(GTRFR2) 0x26(GTRFR4)

	0															15
R / W	TRV															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 5-49. Global Timers Reference Registers (GTRFR1–GTRFR4)

Table 5-65 defines the bit fields of GTRFR.

Table 5-65. GTRFR Bit Settings

Bits	Name	Description
0–15	TRV	Timeout reference value. 16-bit timeout reference value for the corresponding timer. Set to all ones by reset.

5.7.5.4 Global Timers Capture Registers (GTCPR1–GTCPR4)

Global timers capture registers (GTCPR1, GTCPR2, GTCPR3, and GTCPR4), shown in Figure 5-50, are used to latch the value of the counters according to GTMDR*n*[CE].

Offset 0x18(GTCPR1) 0x28(GTCPR3) Access: Read only
0x1A(GTCPR2) 0x2A(GTCPR4)

	0 … 15
R	LCV
W	
Reset	All zeros

Figure 5-50. Global Timers Capture Registers (GTCPR1–GTCPR4)

Table 5-66 defines the bit fields of GTCPR*n*.

Table 5-66. GTCPR*n* Bit Settings

Bits	Name	Description
0–15	LCV	Latched counter value. Corresponding timer's 16-bit latched value.

5.7.5.5 Global Timers Counter Registers (GTCNR1–GTCNR4)

Global timers counter registers (GTCNR1, GTCNR2, GTCNR3, and GTCNR4), shown in Figure 5-51, are four 16-bit, memory-mapped, read/write up-counters. A read cycle to a GTCNR*n*[CNV] fields yields the current value of the appropriate timer but does not affect the counting operation. A write cycle to a GTCNR*n*[CNV] field sets the register to the written value, causing its corresponding primary and secondary prescaler counters to be reset.

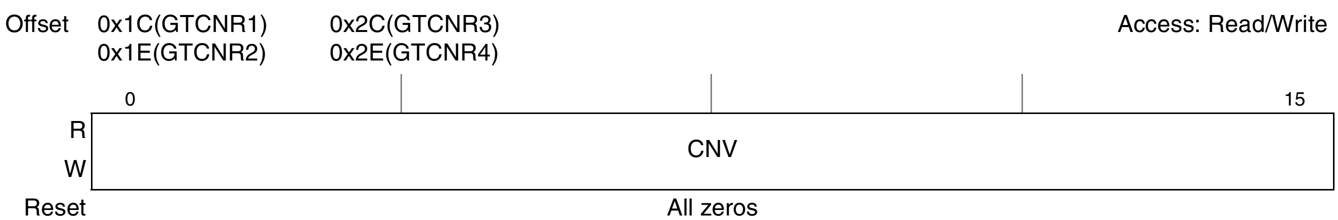


Figure 5-51. Global Timers Counter Registers (GTCNR1—GTCNR4)

Table 5-67 defines the bit fields of GTCNR.

Table 5-67. GTCNR Bit Settings

Bits	Name	Description
0–15	CNV	Counter value. Corresponding timer's 16-bit read/write up-counter value.

5.7.5.6 Global Timers Event Registers (GTEVR1–GTEVR4)

Global timers event registers (GTEVR1, GTEVR2, GTEVR3, and GTEVR4), shown in Figure 5-52, are used to report events recognized by any of the timers. On recognition of an output reference event, the appropriate timer sets GTEVR*n*[REF], regardless of the corresponding GTMDR*n*[ORI]. The capture event is only set if it is enabled by GTMDR*n*[CE]. GTEVRs appear as memory-mapped registers to users, which can be read at any time.

GTEVR*n* bits are cleared by writing ones to them (writing zeros does not affect bit values). Both bits must be reset before the timer negates the interrupt to the interrupt controller.

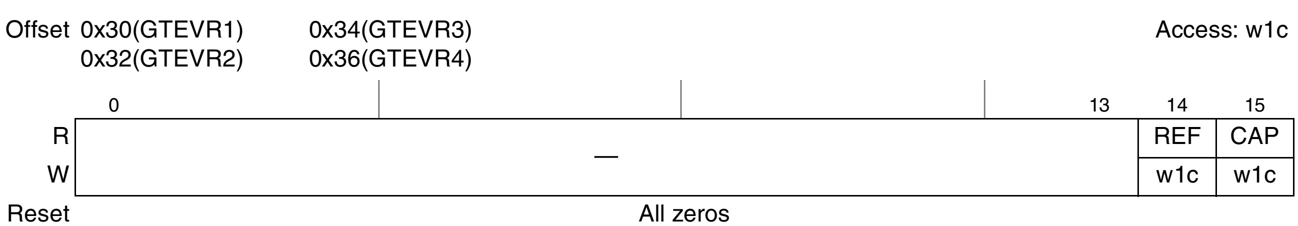


Figure 5-52. Global Timers Event Registers (GTEVR1—GTEVR4)

Table 5-68 defines the bit fields of GTEVR*n*.

Table 5-68. GTEVR*n* Bit Settings

Bits	Name	Description
0–13	—	Reserved, should be cleared.
14	REF	Output reference event 0 No event 1 The counter reached the GTRFR*n*[TRV] value. GTMDR*n*[ORI] is used to enable the interrupt request caused by this event.
15	CAP	Counter capture event Corresponding timer's 16-bit read/write up-counter value. 0 No event 1 The counter value has been latched into the GTCPR*n*[LCV]. GTMDR*n*[CE] is used to enable generation of this event.

5.7.5.7 Global Timers Prescale Registers (GTPSR1–GTPSR4)

The global timers prescale registers (GTPSR1, GTPSR2, GTPSR3, and GTPSR4) are shown in Figure 5-53.

Erratic behavior may occur if GTPSR*n* is not initialized before the corresponding GTMDR*n*.

Offset 0x38(GTPSR1) 0x3C(GTPSR3) 0x3A(GTPSR2) 0x3E(GTPSR4) — Access: Read/Write

	0–7	8–15
R/W	—	PPS
Reset	0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 1

Figure 5-53. Global Timers Prescale Registers (GTPSR1–GTPSR4)

Table 5-69 defines the bit fields of GTPSR*n*.

Table 5-69. GTPSR*n* Bit Settings

Bits	Name	Description
0–7	—	Reserved, should be cleared.
8–15	PPS	Primary prescaler bits The primary prescaler is programmed to divide the clock input to corresponding timer by values from 1 to 256. The value 0x00 divides the clock by 1 and 0xFF divides the clock by 256.

NOTE

The total timer prescale value is calculated as follows:

$$GTMn_{prescaler} = (GTPSRn[PPS] + 1) \cdot (GTMDRn[SPS] + 1)$$

This gives a total prescale range from 1 (GTPSR*n*[PPS] = 0x00, GTMDR*n*[SPS] = 0x00) to 65,536 (GTPSR*n*[PPS] = 0xFF, GTMDR[SPS] = 0xFF).

5.7.6 Functional Description

5.7.6.1 General-Purpose Timer Units

The clock input to the timer's prescaler can be selected from the following sources:

- The system clock
- The system slow go clock (internally divided by 16)

The general system clock is generated in the clock synthesizer and defaults to the system frequency. However, the general system clock has the option to be divided before it leaves the clock synthesizer. This mode, called slow go, is used to save power. Whatever the resulting frequency of the general system clock, the user can either choose that frequency or the frequency divided by 16 as the input to the prescaler of each timer. Alternatively, the user may prefer TIN*n* to be the clock source. TIN*n* is internally synchronized to the internal clock. If the user has chosen to internally cascade two 16-bit timers to a 32-bit timer, then a timer can use the clock generated by the output of another timer.

The clock input source is selected by the corresponding GTMDR*n*[ICLK] bits. The prescalers (GTMDR*n*[SPS] and GTPSR*n*[PPS]) can be programmed to divide the clock input by values from 1 to 65,537 and the output of the prescaler is used as an input to the 16-bit counters. The best resolution of the timer is one clock cycle (3 ns at a 333-MHz system clock, for example). The maximum period (when the reference value is all ones and the prescaler divides by 256) for one 16-bit timer is ~50 ms at 333 MHz.

5.7.6.2 Reference Modes

Each timer can be configured to count until a reference is reached and then either begin a new time count immediately or continue to run. The FRR bit of the corresponding GTMRR selects each mode.

- Free run reference mode (GTMDR*n*[FRR] = 0)

  The corresponding timer count continues to increment after the reference value is reached.
- Reset reference mode (GTMDR*n*[FRR] = 1)

  The corresponding timer count is reset immediately after the reference value is reached.

Upon reaching the reference value, the corresponding GTEVR*n*[REF] bit is set and an interrupt is issued if GTMDR*n*[ORI] = 1. The timers can output a signal on the timer output pin $\overline{\text{TOUT}n}$ if the reference value is reached (selected by the corresponding GTMDR*n*[OM]). This signal can be an active-low pulse or a toggle of the current output. The output can also be connected internally to the input of another timer, resulting in a 32- or 64-bit timer.

5.7.6.3 Capture Modes

In addition, each timer has a 16-bit field in GTCPR, used to latch the value of the counter when a defined transition of TIN*n* is sensed by the corresponding input capture edge detector. The timers may be gated/restarted by an external gate signals ($\overline{\text{TGATE}n}$) that controls the timers. The type of transition triggering the capture is selected by the corresponding GTMDR*n*[CE] bits. Upon a capture or reference event, corresponding GTEVR*n*[REF] or GTEVR*n*[CAP] is set and a maskable interrupt request is issued to the interrupt controller.

- Normal gate mode enables the count on a falling edge of $\overline{\text{TGATE}}$ and disables the count on the rising edge of $\overline{\text{TGATE}}$. This mode allows the timer to count conditionally, based on the state of $\overline{\text{TGATE}}$.
- The restart gate mode performs the same function as normal mode, except it also resets the counter on the falling edge of $\overline{\text{TGATE}}$.

  This mode has applications in pulse interval measurement and bus monitoring as follows:

  — Pulse measurement—The restart gate mode can measure a low pulse on $\overline{\text{TGATE}}$. The rising edge of $\overline{\text{TGATE}}$ completes the measurement and if $\overline{\text{TGATE}n}$ is connected externally to TIN*n*, it causes the timer to capture the count value and generate a rising-edge interrupt.

  — Bus monitoring—The restart gate mode can detect a signal that is stuck abnormally low. The bus signal should be connected to $\overline{\text{TGATE}}$. The timer count is reset on the falling edge of the bus signal and if the bus signal does not go high again within the number of user-defined clocks, an interrupt can be generated.

The gate function is enabled in the GTMDR; the gate operating mode is selected in the GTCFR*n*.

NOTE

$\overline{\text{TGATE}}$ is internally synchronized to the system clock. If $\overline{\text{TGATE}}$ meets the asynchronous input setup time, the counter begins counting after one system clock when working with the internal clock.

5.7.6.4 Cascaded Modes

GTCFR*n*[PCAS] and GTCFR2[SCAS] are used to put the timers into different cascaded modes:

- Non-cascaded mode (GTCFR*n*[PCAS] = 0 and GTGCF2[SCAS] = 0)

  If GTCFR*n*[PCAS] = 0 and GTCFR2[SCAS] = 0, the each timer (timer 1, timer 2, timer 3, and timer 4), function as a independent 16-bit timer with a 16-bit GTRFR, GTCPR, GTMDR, and GTCNR for each one (Figure 5-54). When working in the none-cascaded mode, the non-cascaded GTRFR, GTCPR, and GTCNR should be referenced with appropriate 16-bit bus cycles.

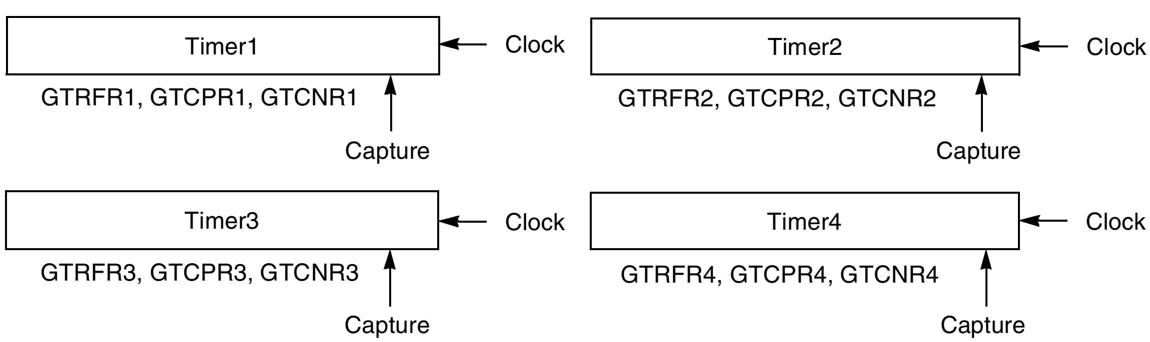


Figure 5-54. Timers Non-Cascaded Mode Block Diagram

- Pair-cascaded mode (GTCFR1[PCAS] = 1 and/or GTCFR2[PCAS] = 1, GTCFR2[SCAS] = 0)

  In this mode, two 16-bit timers can be internally cascaded to form a 32-bit counter: timer 1 may be internally cascaded to timer 2 and timer 3 may be internally cascaded to timer 4, as shown in Figure 5-55. Since the decision to cascade timers is made independently, the user has the option of selecting two 16-bit timers and one 32-bit timer (GTCFR1[PCAS] = 1, GTCFR2[PCAS] = 0 or GTCFR1[PCAS1] = 0, GTCFR2[PCAS] = 1), or two 32-bit timers (GTCFR1[PCAS] = 1 and GTCFR2[PCAS] = 1).

  If GTCFR1[PCAS] = 1 and/or GTCFR2[SCAS] = 1, the two 16-bit timers (timer 1 and timer 2 or timer 3 and timer 4) function as a 32-bit timer with a 32-bit GTRFR, GTCPR, and GTCNR. In this case, GTMDR1/GTMDR3 is ignored, and the modes and functions are defined using GTMDR2/GTMDR4, and GTCFR1/GTCFR2. The capture are controlled from TIN2, and the interrupts are generated from GTEVR2. When working in the pair-cascaded mode, the cascaded GTRFR, GTCPR, and GTCNR should be referenced with 32-bit bus cycles.

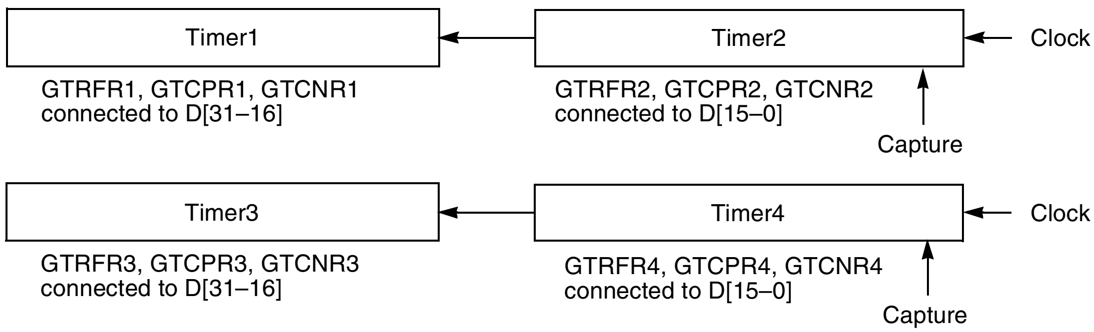


Figure 5-55. Timer Pair-Cascaded Mode Block Diagram

- Super-cascaded mode (GTCFR2[SCAS] = 1)

  In this mode, all four 16-bit timers can be internally cascaded to form a 64-bit counter, as shown in Figure 5-56.

  If GTCFR2[SCAS] = 1, the all four 16-bit timers function as a 64-bit timer with a cascaded 32-bit GTRFR, GTCPR, and GTCNR. In this case, registers GTMDR1, GTMDR2, GTMDR3, and GTCFR1 are ignored, and the modes and functions are defined using GTMDR4 and GTCFR2 only. The capture are controlled from TIN4, and the interrupts are generated from GTEVR4. When working in the super-cascaded mode, the cascaded GTRFR, GTCPR, and GTCNR should be referenced with two 32-bit bus cycles.

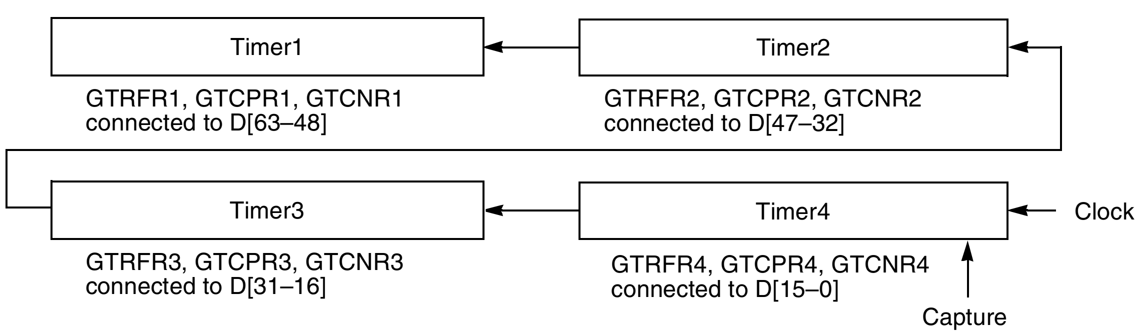


Figure 5-56. Timers Super-Cascaded Mode Block Diagram

5.7.7 Initialization/Application Information

5.7.7.1 GTM Register Programming Guidelines

The following initialization sequence of GTM is recommended:

- Write to GTCFR*n* in order to reset, to stop or to configure the appropriate timer's operation: cascaded timers configuration, gate mode configuration.
- Write to GTPSR*n*[PPS] fields in order to program the appropriate timer's clock primary prescaler.
- Write to GTMDR*n* in order to choose an input clock, to program the secondary prescaler and to set a desirable appropriate timer's operational mode.

NOTE

Erratic behavior may occur if GTCFR*n* and GTPSR are not initialized before the GTMDR. Only GTCFR*n*[RST*n*] can be modified at any time

- Clear GTEVR*n*[REF] and GTEVR*n*[CAP] by writing 1s in order to clear the previous events.
- Write to GTRFR and to GTCNR*n* according to appropriate timer's GTMDR*n* programming.

NOTE

A write cycle to a GTCNR*n*[CNV] fields sets the register to the written value, causing its corresponding primary and secondary prescalers, (GTPSR*n*[PPS] and GTMDR*n*[SPS]), to be reset.

- Write to GTCFR*n*[STP*n*] and to GTCFR*n*[RST*n*] in order to initialize the appropriate timer's operation.

5.8 Power Management Control (PMC)

The device provides a power management control (PMC) unit, which enables the device to smoothly enter and exit low power modes. Low power modes may be used when internal units in the device temporarily or permanently do not perform any action.

The device uses one or more of the following methods for power saving:

- Dynamic power management
- Shutting down unused blocks
- Software-controlled power-down states
- Support for the PCI Power Management Interface Specification in both host and agent modes. When the device is in either mode, the PMC is capable of placing the device into one of the supported low-power states and supporting the power management event (PME) signaling protocol.

5.8.1 External Signal Description

Table 5-70 describes the power management signals.

Table 5-70. System Control Signals—Detailed Signal Descriptions

Signal	I/O	Description	
$\overline{QUIESCE}$	O	Quiesce state. Indicates that the processor system and PowerPC core are in low power state.	
		State Meaning	Asserted—The system and PowerPC core are in low power state. Negated—The system and PowerPC core are not in low power state.
		Timing	The timing between a quiesce request from the PowerPC core and the assertion of the external indication or between negation of the core's quiesce request and negation of the external indication depends on the current state of the internal system units and may vary accordingly.

5.8.2 PMC Memory Map/Register Definition

Table 5-71 shows the memory map for the power management controller registers.

Table 5-71. Power Management Controller Registers Memory Map

Offset	Register	Access	Reset	Section/Page
0x00B00	Power management controller configuration register (PMCCR)	R/W	0x0000_0000	5.8.2.1/5-74
0x00B04	Power management controller event register (PMCER)	R/W	0x0000_0000	5.8.2.2/5-75
0x00B08	Power management controller mask register (PMCMR)	R/W	0x0000_0000	5.8.2.3/5-77
0x00B0C	Power management controller configuration register 1 (PMCCR1)	R/W	0x0000_0000	5.8.2.4/5-78
0x00B10–0x00BFC	Reserved	—	—	—

5.8.2.1 Power Management Controller Configuration Register (PMCCR)

The power management controller configuration register (PMCCR), shown in Figure 5-57, controls whether only the PowerPC core will enter low power state upon quiesce request or additional parts of the device will also enter low power state.

Offset 0x00B00 Access: Read/Write

	0–29	30	31
R/W	—	DLPEN	SLPEN
Reset	All zeros		

Figure 5-57. Power Management Controller Configuration Register

Table 5-5 defines the bit fields of PMCCR.

Table 5-72. PMCCR Bit Settings

Bits	Name	Description
0–29	—	Reserved. Write has no effect, read returns 0.
30	DLPEN	DDR SDRAM low power enable 0 The DDR SDRAM memory controller is prevented from entering low power state. 1 The DDR SDRAM memory controller will enter low power state when the rest of the system enters low power, according to SLPEN setting. DDR SDRAM will enter self-refresh mode (if enabled by DDR_SDRAM_CFG[SREN] memory controller register) and DDR clocks (MCK*n*) are shut off. This bit is cleared when the device exits from low power state. Note that setting this bit without setting SLPEN has no effect.
31	SLPEN	System low power enable 0 The system is prevented from entering low power state. 1 The system will enter low power state when a quiesce request from the PowerPC core arrives. This bit is cleared when the device exits from low power state.

5.8.2.2 Power Management Controller Event Register (PMCER)

The power management controller event register (PMCER), shown in Figure 5-58, indicates with the PMCI bit that the power management controller has detected a wake-up event, that the system is not in idle state anymore, and that the device should exit low power state. If PMCMR[PMCIE] is set, the PMC interrupt request to the PowerPC core is driven. When set, bits 23–30 indicate the sources of various wake-up events.

Offset 0x00B04 Access: Read/Write

	0–15
R/W	—
Reset	All zeros

	16–22	23	24	25	26	27	28	29	30	31
R/W	—	GPIO1	GPIO2	—	eTSEC1	eTSEC2	Timer	Int1	Int2	PMCI
Reset	All zeros									

Figure 5-58. Power Management Controller Event Register

Table 5-73 defines the bit fields of PMCER.

Table 5-73. PMCER Bit Settings

Bits	Name	Description
0–22	—	Reserved. Write has no effect, read returns 0.
23	GPIO1	Wake-up event detected. 0 A wake-up event did not occur from this wake-up source. 1 A wake-up event occurred on GPIO1. This wake-up event was caused by an unmasked event of GPIO1 module. See Chapter 25, "General Purpose I/O (GPIO)," for more details. If the corresponding PMCMR bit is set, the PMC will assert interrupt request to the PowerPC core or external PME to the remote host, depending on the state of PMCCR1[PME_EN]. This bit can be cleared by writing a 1 to the bit location (writing zero has no effect). **Note:** This bit will not be affected by the wake-up event if the corresponding mask bit in PMCMR is cleared.
24	GPIO2	Wake-up event detected. 0 A wake-up event did not occur from this wake-up source. 1 A wake-up event occurred on GPIO2. This wake-up event was caused by an unmasked event of GPIO2 module. See Chapter 25, "General Purpose I/O (GPIO)," for more details. If the corresponding PMCMR bit is set, the PMC will assert interrupt request to the PowerPC core or external PME to the remote host, depending on the state of PMCCR1[PME_EN]. This bit can be cleared by writing a 1 to the bit location (writing zero has no effect). **Note:** This bit will not be affected by the wake-up event if the corresponding mask bit in PMCMR is cleared.
25	—	Reserved. Write has no effect, read returns 0.
26	eTSEC1	Wake-up event detected. 0 A wake-up event did not occur from this wake-up source. 1 A wake-up event occurred on eTSEC1. This wake-up event was caused by a detection of a Magic Packet on the receive path of eTSEC1. SeeChapter 18, "Enhanced Three-Speed Ethernet Controllers," for more details. I If the corresponding PMCMR bit is set, the PMC will assert interrupt request to the PowerPC core or external PME to the remote host, depending on the state of PMCCR1[PME_EN]. This bit can be cleared by writing a 1 to the bit location (writing zero has no effect). **Note:** This bit will not be affected by the wake-up event if the corresponding mask bit in PMCMR is cleared.
27	eTSEC2	Wake-up event detected. 0 A wake-up event did not occur from this wake-up source. 1 A wake-up event occurred on eTSEC2. This wake-up event was caused by a detection of a Magic Packet on the receive path of eTSEC2. See Chapter 18, "Enhanced Three-Speed Ethernet Controllers," for more details. I If the corresponding PMCMR bit is set, the PMC will assert interrupt request to the PowerPC core or external PME to the remote host, depending on the state of PMCCR1[PME_EN]. This bit can be cleared by writing a 1 to the bit location (writing zero has no effect). **Note:** This bit will not be affected by the wake-up event if the corresponding mask bit in PMCMR is cleared.
28	TIMER	Wake-up event detected. 0 A wake-up event did not occur from this wake-up source. 1 A wake-up event occurred on General Purpose Timer 1. This wake up event was caused by a detection of match between timer current value and timer reference value of the fourth 16 bit unit of GTM1. See Section 5.7, "General-Purpose Timers (GTMs),"for more details. If the corresponding PMCMR bit is set, the PMC will assert interrupt request to the PowerPC core or external PME to the remote host, depending on the state of PMCCR1[PME_EN]. This bit can be cleared by writing a 1 to the bit location (writing zero has no effect). **Note:** This bit will not be affected by the wake-up event if the corresponding mask bit in PMCMR is cleared.

Table 5-73. PMCER Bit Settings (continued)

Bits	Name	Description
29	INT1	Wake-up event detected. 0 A wake-up event did not occur from this wake-up source. 1 A wake-up event occurred on external interrupt request 1. This wake-up event was caused by a detection of an active state of the $\overline{IRQ1}$ external pin. If the corresponding PMCMR bit is set, the PMC will assert interrupt request to the PowerPC core or external PME to the remote host, depending on the state of PMCCR1[PME_EN]. This bit can be cleared by writing a 1 to the bit location (writing zero has no effect). **Note:** This bit will not be affected by the wake-up event if the corresponding mask bit in PMCMR is cleared.
30	INT2	Wake-up event detected. 0 A wake-up event did not occur from this wake-up source. 1 A wake-up event occurred on external interrupt request 2. This wake-up event was caused by a detection of an active state of the $\overline{IRQ2}$ external pin. If the corresponding PMCMR bit is set, the PMC will assert interrupt request to the PowerPC core or external PME to the remote host, depending on the state of PMCCR1[PME_EN]. This bit can be cleared by writing a 1 to the bit location (writing zero has no effect). **Note:** This bit will not be affected by the wake-up event if the corresponding mask bit in PMCMR is cleared.
31	PMCI	Power management controller interrupt. When set, indicates that one of the following events has occurred: • One of the unmasked wake-up events (bits 23–30) occurred and PMCCR1[PME_vPEN] is cleared, or • PM current state (as indicated in PMCCR1[CURR_STATE]) is different than PM next state (as written to PCIPMR1[Power_State] and indicated in PMCCR1[NEXT_STATE]) and PMCCR1[USE_STATE] is set, or • CSB platform is in low power mode and a new CSB bus request is detected If PMCMR[PMCIE] is set, the PMC interrupt request to the PowerPC core is driven, causing the PowerPC core to exit its low power state. PMCI can be cleared by writing a 1 to it (writing zero has no effect).

5.8.2.3 Power Management Controller Mask Register (PMCMR)

The power management controller mask register (PMCMR), shown in Figure 5-59, controls through the PMCIE bit whether the PMC interrupt request to the PowerPC core is enabled. The PMC interrupt request causes the PowerPC core to exit its low power state before any transaction on the system bus occurs. Bits 23–30 are mask bits for the defined low power wake-up events.

Offset 0x00B08 Access: Read/Write

	0–15
R W	—
Reset	All zeros

	16–22	23	24	25	26	27	28	29	30	31
R W	—	GPIO1	GPIO2	—	eTSEC1	eTSEC2	Timer	Int1	Int2	PMCIE
Reset	All zeros									

Figure 5-59. Power Management Controller Mask Register

Table 5-74 defines the bit fields of PMCMR.

Table 5-74. PMCMR Bit Settings

Bits	Name	Description
0–22	—	Reserved. Write has no effect, read returns 0.
23–30	GPIO1, GPIO2, eTSEC1, eTSEC2, Timer, Int1, Int2	Wake-up event masking. 0 Mask wake-up events from Int2, Int1, Timer, eTSEC2, eTSEC1, or GPIO, respectively. 1 Do not mask wake-up events
31	PMCIE	Power management controller interrupt enable. 0 PMC interrupt request (PMCI) is disabled. 1 PMC interrupt request (PMCI) is enabled.

NOTE

The user is also required to enable the PMC interrupt in the programmable interrupt controller by setting SIMSR_L[PMC].

5.8.2.4 Power Management Controller Configuration Register 1 (PMCCR1)

The power management controller configuration register 1 (PMCCR1), shown in Figure 5-60, controls the sequencing of the device into its low power state including PME (power management event) signaling and indication of current and desired power states.

Offset 0x00B0C | Access: Read/Write

	0	1–15
R/W	USE_STATE	—
Reset	All zeros	

	16–23	24	25	26–27	28–29	30–31
R	—	PME_EN	ASSERT_PME	—	NEXT_STATE	CURR_STATE
W						
Reset	All zeros					

Figure 5-60. Power Management Controller Configuration Register 1

Table 5-75 defines the bit fields of PMCCR1.

Table 5-75. PMCCR1 Bit Settings

Bits	Name	Description
0	USE_STATE	Controls whether the next and current state values should be used. This typically depends on the device operating mode (PCI host or agent). 0 Ignore the next and current state values (host mode). 1 Use the next and current state values (agent mode). When next state does not equal current state, an interrupt will be sent to the e300 CPU.
1–23	—	Reserved, must be cleared.

Table 5-75. PMCCR1 Bit Settings (continued)

Bits	Name	Description
24	PME_EN	PME (power management event) signaling enable. Clearing this bit typically indicates the device is acting as PCI host. Setting this bit typically indicates that the device is a PCI agent and will assert PCI_PME on wake-up. 0 Mask PME signaling when wake-up events occur. Unmasked wake-up events will assert an interrupt to the local e300 CPU. 1 Allow PME signaling when wake-up events occur. Unmasked wake-up events will assert PCI_PME signal, indicating to the remote host that service is required.
25	ASSERT_PME	Normally PCI_PME output is asserted automatically by PMC when a defined wake-up event occurs assuming PMCCR1[PME_EN] = 1 and PCIPMCR1[PME_EN] = 1. A defined wake-up event refers to those events that are registered in bits 23–30 of PMCER. ASSERT_PME allows PCI_PME to be asserted manually, by the e300. This would be done to inform the host of a power state change when PMC does not generate PCI_PME automatically, for example waking from D1 due to CSB bus activity. ASSERT_PME is not qualified by PMCCR1[PME_EN]. ASSERT_PME is cleared when the PCI_PME signal is asserted. 0 PCI_PME will only be asserted automatically when a defined wake-up event occurs. 1 Assert the PCI_PME signal under software control (manually).
26–27	—	Reserved
28–29	NEXT_STATE	Indicate the power state as programmed by the PCI host in PCIPMR1[Power_State]. The host will write the Power_State bits to request that the device enter a certain low power state. The host may also write the Power_State to request that the device return to D0 from some low power state. When the NEXT_STATE field is different than the CURR_STATE field, an interrupt is asserted to the e300 processor through the IPIC. This field is read-only. 00 Host's desired power state is D0 01 Host's desired power state is D1 10 Host's desired power state is D2 11 Host's desired power state is D3
30–31	CURR_STATE	Indicate the current power state of the device. These bits are written by the e300 just before entering a requested low power state, or when the device has returned to the full on state (D0). Writing these bits causes the PCIPMR1[Power_State] field to be updated informing the host that the device has entered the requested power state. 00 Current power state is D0 01 Current power state is D1 10 Current power state is D2 11 Current power state is D3Hot

5.8.3 Functional Description

The device has features to minimize power consumption at several levels. Dynamic power management locally minimizes power consumption when a block is idle. Software can also shut down clocks to individual blocks when they are not needed through a memory-mapped register in the clock unit (SCCR). Additionally, software running on the PowerPC core can access the core's SPRs to put the device into doze, nap, or sleep power down states. Finally, software can access the PMCCR register to enable the device to go to low power state whenever the PowerPC core enters nap or sleep states. The PMC supports features that work in concert with the PCI power management (PM) block (PME context) to provide support for PCI power management capabilities such as asserting or responding to power management events (PMEs). These power management features are described in further detail in this section.

5.8.3.1 Dynamic Power Management

Many blocks in the device can dynamically turn off clocks within the block when sections of the block are idle. This feature is always enabled and occurs automatically.

5.8.3.2 Shutting Down Unused Blocks

As described in Section 4.5.2.3, "System Clock Control Register (SCCR)," SCCR provides a way to shut down certain functional blocks within the device when they are not needed in a particular system. SCCR can be written by the PowerPC core or by an external master. Powering down a block in this way turns off all clocks to that block. It does not remove power. It is required that the SCCR is written to shut down a certain functional block only when that block is idle.

NOTE

Functional blocks disabled using SCCR cannot respond to configuration accesses. Any access to configuration, control, and status registers of a disabled block is a programming error.

5.8.3.3 Software-Controlled Power-Down States

PowerPC software can place the core in doze, nap, or sleep power-down states by writing to HID0 in the core, as described in detail in the section "Hardware Implementation Register 0 (HID0)," of the *e300 PowerPC Core Reference Manual*. In addition, if PMCCR[SLPEN] is set when the PowerPC core request to enter nap or sleep modes, it will also cause the system internal logic units to enter low power mode.

5.8.3.4 Support of PCI Power Management Interface Specification

The device will support the PCI power states D0, D1, D2, D3Hot, and D3Cold as defined in the Rev. 1.2 of the PCI Power Management Interface Specification. Table 5-76 defines these D*x* states. The PCI power management specification defines a power management event, or PME, as the process by which a PCI agent signals a request to the host for a change in its power consumption state. When in agent mode, the device has the capability to generate PME signaling through the assertion of an external $\overline{\text{PCI\_PME}}$ signal. As host, the device is able to respond to PME signaling as a wake-up event.

Table 5-76 defines the PCI power states.

Table 5-76. PCI Defined Power Management State Support

PCI Dx Power State	Supported	How Supported
D0	Yes	Default state; full power; PME signaling supported through software control
D1	Yes	e300 in Doze mode; e300 PLL running; PME signaling supported
D2	Yes	e300 in Nap state; e300 PLL running; PME signaling supported

Table 5-76. PCI Defined Power Management State Support (continued)

PCI Dx Power State	Supported	How Supported
D3Hot	Yes	e300 in Sleep mode. PME signaling is supported.
D3Cold	Yes*	*PME signaling is not supported in D3Cold. For PME signaling, use the D3Hot state. According to PCI PM Specification 1.2, in D3Cold the PCI clock is removed and all devices will be reset when power is restored. The device can be placed into D3Cold state, but wake-up events cannot be recognized or generated and when power is restored, the device must go through a normal power-on reset boot sequence as it needs to be re-initialized.

For completeness, Table 5-77 shows the PCI bus power management states (B0–B3) that the device supports. If the device is used as a PCI agent, the supported bus states are B0 and B1, since the PCI clock is running in these two states. Otherwise, as a host, the device could be configured to support all the bus states.

Table 5-77. PCI Bus Power Management State Support

PCI Bx Bus State	PCI Bus VDD	PCI Bus Clock	PCI Bus Activity	Support in PCI Agent Mode	Support in PCI Host Mode
B0 (full on)	On	Yes	Any PCI transaction, function interrupt, or PME event	Yes	Yes
B1	On	Yes	PME event; bus is idle	Yes	Yes[1]
B2	On	No	PME event	No[2] (no bus clock)	Yes[3]
B3	Off	No	PME event	No[4] (no bus clock or VDD)	Yes[5]

1 Requires the device to hold the PCI bus in idle state

2 The device cannot process wake-up events with the PCI bus in this state and the bus must be returned to B0 state through a PORESET.

3 Requires the device to turn off PCI bus clock

4 The device cannot process wake-up events with the PCI bus in this state and the bus must be returned to B0 state through a PORESET.

5 Requires the device to turn off the PCI bus clock through the output clock control register (OCCR), and to turn off the PCI bus VDD through some customer-defined method (perhaps GPIO).

5.8.3.4.1 Entering Low Power States—Core-Only Mode

Entering Doze mode is controlled only by the e300 PowerPC core itself, and does not involve the power management controller or other blocks. For a more detailed description, see Table 7-2.

Entering Nap or Sleep modes occurs by writing to HID0 in the core, causing the core to make a quiesce request to the power management controller while PMCCR[SLPEN] is cleared. The core is immediately enabled to enter low power state, regardless of the system status. Note that since the core does not snoop the bus in this mode, it is the user's responsibility to keep the cache coherent. Other device peripheral and internal units continue to operate in full-on mode while the core is in low power state in this mode.

5.8.3.4.2 Entering Low Power States—Core and System Mode

Core and system mode is achieved when the core makes a quiesce request to the power management controller after PMCCR[SLPEN] is set. To preserve cache coherency and otherwise avoid loss of system state, the core's transition to low-power modes is coordinated with other functional blocks. The power management controller allows the core to enter power down mode only when the rest of the system is idle.

When the power management controller detects that the internal system bus is idle, and there are no outstanding transactions, it signals the internal logic units to enter low power state.

If PMCCR[DLPEN] is set, the DDR SDRAM is first set to self-refresh mode (if enabled by DDR_SDRAM_CFG[SREN] memory controller register) before the memory controller stops driving refresh commands. Self-refresh mode guarantees that the memory content will remain valid while the memory controller and its clocks are off. The DDR clocks are then disabled. Finally the DDR SDRAM memory controller enters low power state and acknowledges the power management controller.

The power management controller then signals the core and acknowledges it's request to enter power down mode. Finally the $\overline{\text{QUIESCE}}$ output signal is asserted.

5.8.3.5 Exiting Core and System Low Power States

The device can exit low power state and return to full-on mode for one of the following reasons:

- The core internal time base unit invokes a request to exit low power state.
- The core has received an interrupt request.
- The device is a PCI host, and the power management controller has detected that the system is not idle and there are outstanding requests for transactions on the internal bus.
- The device is a PCI agent, and the power management controller has detected that the PCI power next state does not equal the PCI power current state (meaning that the remote PCI host has requested a change from non D0 to D0 state).

The actions taken to exit low power state depend on the mode and whether the system or only the core are in this state. The following sections describe the various scenarios.

5.8.3.5.1 Exiting Low Power States—Core-Only Mode

Exit from Doze mode is controlled only by the core itself and does not involve the power management controller or other blocks in the device. For a detailed description, see Table 7-2.

Nap or Sleep modes are exited when the core has received an interrupt request, or according to the internal time base unit of the core (Nap mode only). The source of the interrupt can be an internal block or external signal. When the core returns to full-on state, it signals to the power management controller that it is ready and is immediately acknowledged to access the rest of the system.

5.8.3.5.2 Exiting Low Power States—Core and System Mode

The power management controller decides to exit low power state when it detects that the system is not idle anymore. The device may exit idle state when one of the peripheral interfaces makes a request to access the internal bus or when the core returns to full-on state, as described above, and makes a request to access the internal bus. For example, the TSEC receives an Ethernet frame, and requires to store it on the DDR SDRAM memory.

If the DDR SDRAM memory controller is in low power state (PMCCR[DxLPEN] was set when entering low power state), the power management controller initially enables the DDR SDRAM memory controller. DDR SDRAM clocks (MCK*n)* are enabled and the memory controller exit self-refresh and returns to auto-refresh mode.

The power management controller then enables other internal units and interrupts the PowerPC core. When all internal units, including the core, are ready, the power management controller enables the device to return to full-on state, negate the $\overline{\text{QUIESCE}}$ output, and clear PMCCR[SLPEN]. Outstanding requests for transactions are now granted to execute on the internal bus.

NOTE

Software is required to enable PMCI interrupt by setting PMCMR[PMCIE], otherwise exiting from low power state is not possible.

NOTE

It is the software's responsibility to clear PMCER[PMCI].

5.8.4 Initialization/Application Information

5.8.4.1 Core Disable in Low Power Mode

If the device is required to operate with the core permanently disabled, the following steps must be taken:

1. Initialize the device with the core enable.
2. Clear PMCCR[SLPEN] and disable the core time base unit by clearing SPCR[TBEN]. See Section 5.3.2.4, "System Priority and Configuration Register (SPCR)," for more information.
3. The e300 core enters low power state by accessing the HID0 register.
4. Set ACR[COREDIS] in the system arbiter register with an external master (that is, PCI). This steers all device interrupts to the $\overline{\text{PCI\_INTA}}$ so the core cannot receive any interrupt requests.

NOTE

Make sure that the core cannot receive any interrupt requests during the time interval between setting HID0 and setting ACR[COREDIS]. This can be achieved if the rest of the system is idle (during the initialization).

By following this flow, the e300 core remains in low power state while the rest of the system is operational, and does not get out of this state as a result of any interrupt or time-based event.

Chapter 6
Arbiter and Bus Monitor

This chapter describes operation theory of the arbiter in the device. In addition, it describes configuration, control, and status registers of the arbiter.

6.1 Overview

The arbiter is responsible for providing coherent system bus arbitration. It tracks all address and data tenures and provides all the arbitration signals to masters and slaves. In addition, it monitors the bus and reports on errors and protocol violations.

The arbiter includes the following features:

- Supports a programmable pipeline depth (from 1 to 4)
- Supports four levels of priority for bus arbitration
- Supports repeat-request mode: number of programmable consecutive transactions from the same master (up to eight transactions)
- Supports data streaming operations
- Supports programmable address bus parking mode: disable, park to last bus owner, park to software selected master
- Claims address only, reserved and illegal transaction types, report on it and can raise maskable interrupt
- Provides timers for address tenure time out and data tenure time out detection and can issue maskable interrupt, if any timer expired
- Reports on transfer error and can issue maskable interrupt
- Can issue regular or machine check interrupt for each type of error event (programmable)

6.1.1 Coherent System Bus Overview

The coherent system bus is the central bus of the device. Any data transaction from master to slave in the device passes through the coherent system bus. The device coherent system bus supports pipelined transactions. It has independent address and data tenures. Pipeline depth determines the number of address tenures that can be started before the first data tenure is finished.

Basic burst size is equal to cache line length of the core, which is 32 bytes. Using repeat request mode enables up to eight consecutive bursts to be executed by the same master. Maximum number of consecutive transactions can be limited by programming arbiter configuration register. See Section 6.2.1, "Arbiter Configuration Register (ACR)," for more details.

NOTE

Write accesses to different interfaces are not guaranteed to finish in order.

6.2 Arbiter Memory Map/Register Definition

Table 6-1 shows the memory map for arbiter's configuration, control and status registers.

Table 6-1. Arbiter Register Map

Memory Offset (Hex)	Register	Access	Reset	Section/Page
0x00	Arbiter configuration register (ACR)	R/W	0x0000_0000/ 0x0010_0000[1]	6.2.1/6-2
0x04	Arbiter timers register (ATR)	R/W	0xFFFF_FFFF	6.2.2/6-4
0x0C	Arbiter event register (AER)	w1c	0x0000_0000	6.2.3/6-5
0x10	Arbiter interrupt definition register (AIDR)	R/W	0x0000_0000	6.2.4/6-6
0x14	Arbiter mask register (AMR)	R/W	0x0000_0000	6.2.5/6-7
0x18	Arbiter event attributes register (AEATR)	R	0x0000_0000[2]	6.2.6/6-8
0x1C	Arbiter event address register (AEADR)	R	0x0000_0000[2]	6.2.7/6-9
0x20	Arbiter event response register (AERR)	R/W	0x0000_0000	6.2.8/6-10

[1] Reset value is determined from the core PLL configuration of the reset configuration word. See Chapter 4, "Reset, Clocking, and Initialization," for details.

[2] The registers AEATR and AEADR are affected only by the assertion of $\overline{PORESET}$

6.2.1 Arbiter Configuration Register (ACR)

Arbiter configuration register (ACR) defines the arbiter modes and parked master on the bus. Figure 6-1 shows the fields of ACR.

Offset 0x00 — Access: User read/write

Bits	0–6	7	8–12	13–15
R/W	—	COREDIS	—	PIPE_DEP
Reset	All zeros[1]			

Bits	16	17–19	20	21–23	24–25	26–27	28–31
R/W	—	PCI_RPTCNT	—	RPTCNT	—	APARK	PARKM
Reset	All zeros[1]						

[1] Note that the reset value of COREDIS and bits 10–11 are determined from reset configuration word. (See Section 4.3.2, "Reset Configuration Words," for more details on reset configuration word.)

Figure 6-1. Arbiter Configuration Register (ACR)

Table 6-2 describes ACR fields.

Table 6-2. ACR Field Descriptions

Bits	Name	Description
0–6	—	Write reserved, read = 0
7	COREDIS	Core disable. Specifies whether CPU is disabled. When CPU is disabled, it cannot be granted on the bus by the arbiter. After reset, this bit receives it's value from the reset configuration bit of COREDIS and can be configured by software. Also, if boot source is boot sequencer, COREDIS must be set to 1 at reset and the last transaction of the boot sequencer must set COREDIS to 0, if CPU enable is needed. 0 CPU enabled. 1 CPU disabled.
8–9	—	Write reserved, read = 0
10–11	—	Reserved. Write should preserve reset value.The reset value is a function of the core PLL configuration, which is part of the reset configuration word. When the core is set to operate at 1:1 or 3:2 bus clock, these bits are set to '01' during reset; otherwise, they are set to '00'.
12	—	Write reserved, read = 0
13–15	PIPE_DEP	Pipeline depth (number of outstanding transactions). 000 Pipeline depth 1 (1 outstanding transaction) 001 Pipeline depth 2 (2 outstanding transactions) 010 Pipeline depth 3 (3 outstanding transactions) 011 Pipeline depth 4 (4 outstanding transactions) 1xx Reserved
16	—	Write reserved, read = 0
17–19	PCI_RPTCNT	PCI repeat count. Specifies the maximum number of consecutive transactions, that PCI master can perform, using $\overline{\text{REPEAT}}$ request mode. 000 One consecutive transaction ($\overline{\text{REPEAT}}$ request mode disable) 001 Two consecutive transactions 010 Three consecutive transactions 011 Four consecutive transactions 100 Five consecutive transactions 101 Six consecutive transactions 110 Seven consecutive transactions 111 Eight consecutive transactions
20	—	Write reserved, read = 0
21–23	RPTCNT	Repeat count. Specifies the maximum number of consecutive transactions, that any master (except PCI) can perform, using $\overline{\text{REPEAT}}$ request mode. 000 1 consecutive transactions ($\overline{\text{REPEAT}}$ request mode disable) 001 2 consecutive transactions 010 3 consecutive transactions 011 4 consecutive transactions 100 5 consecutive transactions 101 6 consecutive transactions 110 7 consecutive transactions 111 8 consecutive transactions **Note:** It is recommended not to program this field for more than four consecutive transactions.
24–25	—	Write reserved, read = 0

Table 6-2. ACR Field Descriptions (continued)

Bits	Name	Description
26–27	APARK	Address parking. Specifies arbiter bus parking mode. 00 Park to master. Arbiter parks the address bus to the master, that is selected by numeric value of PARKM field. 01 Park to last owner. Arbiter parks the address bus to last bus owner. 10 Disable. Arbiter does not assert $\overline{BG}$ to any master, if no $\overline{BR}$ is present. 11 Reserved
28–31	PARKM	Parking master. 0000 e300 core 0001 PCI, DMA 0010 TSEC1, TSEC2 0011 eSDHC 0100 PCI Express 1 0101 USB, Encryption core 0110 PCI Express 2 0111 SATA1, SATA2, SATA3, SATA4 1000–1111 Reserved

6.2.2 Arbiter Timers Register (ATR)

The arbiter timers register (ATR) defines the arbiter address time out (ATO) and data time out (DTO) values. Figure 6-2 shows the fields of ATR.

Offset 0x04 | Access: User read/write

	0–15	16–31
R/W	DTO	ATO
Reset	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 6-2. Arbiter Timers Register (ATR)

Table 6-3 describes ATR fields.

Table 6-3. ATR Field Descriptions

Bits	Name	Description
0–15	DTO	Data time out. Specifies the time-out period for the data tenure. The granularity of this field is128 bus clocks. The maximum value is8355840 coherent system bus clocks. Data time_out occurs if the data tenure does not end before the specified time-out period. When DTO = n, the timeout cycle is n*128. 0000 Reserved 0001 128 clock cycles 0002 256 clock cycles 0003 384 clock cycles ... FFFF 8355840 clock cycles
16–31	ATO	Address time out. Specifies the time-out period for the address tenure. The granularity of this field is128 bus clocks. Maximum value is8355840 coherent system bus clocks. Address time-out occurs if the address tenure did not end before the specified time-out period. When ATO = n, the timeout cycle is n*128. 0000 Reserved 0001 128 clock cycles 0002 256 clock cycles 0003 384 clock cycles ... FFFF 8355840 clock cycles

6.2.3 Arbiter Event Register (AER)

The arbiter uses arbiter event register (AER) to report on erroneous transactions. This register is cleared by writing ones to the fields to be cleared. Figure 6-3 shows the fields of AER.

Offset 0x0C — Access: User w1c

	0–25	26	27	28	29	30	31
R	—	ETEA	RES	ECW	AO	DTO	ATO
W		w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros						

Figure 6-3. Arbiter Event Register (AER)

Table 6-4 describes AER fields.

Table 6-4. AER Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error.Reports on detection of transfer error by one of the slaves. 0 No transfer error detected by one of the slaves. 1 Transfer error detected by one of the slaves.
27	RES	Reserved transfer type. Reports on transaction with reserved transfer type. See Section 6.3.2.5, “Reserved Transaction Type,” for more information. 0 No transaction with reserved transfer type occurred. 1 Transaction with reserved transfer type occurred.

Table 6-4. AER Field Descriptions (continued)

Bits	Name	Description
28	ECW	External control word transfer type. Reports on transaction with external control word transfer type. See Section 6.3.2.6, "Illegal (eciwx/ecowx) Transaction Type," for more information. 0 No transaction with external control word transfer type occurred. 1 Transaction with external control word transfer type occurred.
29	AO	Address Only transfer type. Reports on transaction with address only transfer type. See Section 6.3.2.4, "Address Only Transaction Type," for more information. 0 No transaction with address only transfer type occurred. 1 Transaction with address only transfer type occurred.
30	DTO	Data time out. Reports on data tenure time out. 0 Data time out timer is not expired. 1 Data time out timer is expired.
31	ATO	Address time out. Reports on address tenure time out. 0 Address time out timer is not expired. 1 Address time out timer is expired.

6.2.4 Arbiter Interrupt Definition Register (AIDR)

Arbiter interrupt definition register (AIDR) determines the interrupt that responds to different error conditions. Setting a bit defines the corresponding interrupt as MCP interrupt; clearing a bit defines the corresponding interrupt as regular interrupt. Figure 6-4 shows the fields of AIDR.

Offset 0x10 Access: User read/write

	0–25	26	27	28	29	30	31
R W	—	ETEA	RES	ECW	AO	DTO	ATO
Reset	All zeros						

Figure 6-4. Arbiter Interrupt Definition Register (AIDR)

Table 6-5 describes AIDR fields.

Table 6-5. AIDR Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error.Detection of transfer error by one of the slaves interrupt definition. 0 Detection of transfer error by one of the slaves causes regular interrupt. 1 Detection of transfer error by one of the slaves causes MCP interrupt.
27	RES	Reserved transfer type. Transaction with reserved transfer type interrupt definition. 0 Transaction with reserved transfer type causes regular interrupt. 1 Transaction with reserved transfer type causes MCP interrupt.
28	ECW	External control word transfer type. Transaction with external control word transfer type interrupt definition. 0 Transaction with external control word transfer type causes regular interrupt. 1 Transaction with external control word transfer type causes MCP interrupt.

Table 6-5. AIDR Field Descriptions (continued)

Bits	Name	Description
29	AO	Address only transfer type. Transaction with address only transfer type interrupt definition. 0 Transaction with address only transfer type causes regular interrupt. 1 Transaction with address only transfer type causes MCP interrupt.
30	DTO	Data time out. Data tenure time out interrupt definition. 0 Data tenure time out causes regular interrupt. 1 Data tenure time out causes MCP interrupt.
31	ATO	Address time out. Address tenure time out interrupt definition. 0 Address tenure time out causes regular interrupt. 1 Address tenure time out causes MCP interrupt.

6.2.5 Arbiter Mask Register (AMR)

Arbiter mask register (AMR) is used to mask interrupts or reset requests. Setting a mask bit enables the corresponding interrupt or reset request; clearing a bit masks it. Regular interrupts, MCP interrupts and reset requests can be masked by AMR register. Figure 6-5 shows the fields of AMR.

Offset 0x14 Access: User read/write

	0–25	26	27	28	29	30	31
R W	—	ETEA	RES	ECW	AO	DTO	ATO
Reset	All zeros						

Figure 6-5. Arbiter Mask Register (AMR)

Table 6-6 describes AMR fields.

Table 6-6. AMR Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error. Detection of transfer error by one of the slaves interrupt mask bit. 0 Detection of transfer error by one of the slaves interrupt disabled. 1 Detection of transfer error by one of the slaves interrupt enabled.
27	RES	Reserved transfer type.Transaction with reserved transfer type interrupt mask bit. 0 Transaction with reserved transfer type interrupt disabled. 1 Transaction with reserved transfer type interrupt enabled.
28	ECW	External control word transfer type.Transaction with external control word transfer type interrupt mask bit. 0 Transaction with external control word transfer type interrupt disabled. 1 Transaction with external control word transfer type interrupt enabled.
29	AO	Address only transfer type. Transaction with address only transfer type interrupt mask bit. 0 Transaction with address only transfer type interrupt disabled. 1 Transaction with address only transfer type interrupt enabled.

Table 6-6. AMR Field Descriptions (continued)

Bits	Name	Description
30	DTO	Data time out. Data tenure time out interrupt mask bit. 0 Data tenure time out interrupt disabled. 1 Data tenure time out interrupt enabled.
31	ATO	Address time out. Address tenure time out interrupt mask bit. 0 Address tenure time out interrupt disabled. 1 Address tenure time out interrupt enabled.

6.2.6 Arbiter Event Attributes Register (AEATR)

Arbiter event attributes register (AEATR) reports the type of transaction that causes error, which is specified in the event register. See Section 6.2.3, “Arbiter Event Register (AER),” for more information. AEATR is cleared only by power-on reset. The attributes of the first error event are stored. Note that this means that AEATR does not change its value when AER is not clear. As AEATR is not affected by soft or hard reset, software can read this register and determine the cause of the bus failure, even if the failure caused a deadlock situation. Refer to Section 6.4.2, “Error Handling Sequence,” for more information.

Figure 6-6 shows the fields of AEATR.

Offset 0x18 Access: User read/write

	0–4	5–7	8–10	11–15	16–19	20	21–23	24–26	27–31
R W	—	EVENT	—	MSTR_ID	—	TBST	TSIZE	—	TTYPE
Reset	All zeros								

Figure 6-6. Arbiter Event Attributes Register (AEATR)

Table 6-7 describes AEATR fields.

Table 6-7. AEATR Field Descriptions

Bits	Name	Description
0–4	—	Write reserved, read = 0
5–7	EVENT	Event type. 000 Address time out 001 Data time out 010 Address only transfer type 011 External control word transfer type 100 Reserved transfer type 101 Transfer error 11*c* Reserved
8–10	—	Write reserved, read = 0

Table 6-7. AEATR Field Descriptions (continued)

Bits	Name	Description
11–15	MSTR_ID	Master Id. 00000 e300 core data transaction 00001 Reserved 00010 e300 core instruction fetch 00011 Reserved 00100 eTSEC1 00101 eTSEC2 00110 Reserved 00111 USB DR 01000 Encryption core 01001 I2C (boot sequencer) 01010 JTAG 01011 Reserved 01100 eSDHC 01101 PCI 01110 Reserved 01111 DMA 10000–10111 Reserved 11000 SATA1 11001 SATA2 11010 SATA3 11011 SATA4 11100 Reserved 11101 PCI Express 1 11110 PCI Express 2 11111 Reserved **Note:** Master Id reflects the source of transaction and is used for debug purpose.
16–19	—	Write reserved, read = 0
20	TBST	Transfer burst. 0 Burst transaction. Transfer size is greater than 8 bytes 1 Single-beat transaction. Transfer size is up to 8 bytes
21–23	TSIZE	Transfer size. Transfer size encoding depends on the value of the field TBST. TBST = 1: 001 1 byte 010 2 bytes 011 3 bytes 100 4 bytes 101 5 bytes 110 6 bytes 111 7 bytes 000 8 bytes TBST = 0: 000 16 bytes 001 24 bytes 010 32 bytes 011–111 Reserved
24–26	—	Write reserved, read = 0
27–31	TTYPE	Transfer type. 00000 Address-only 00001 Address-only 00010 Single-beat or burst write 00011 Reserved 00100 Address-only 00101 Reserved 00110 Burst write 00111 Reserved 0100x Address-only 0101x Single-beat or burst read 0110x Address-only 01110 Burst read 01111 Reserved 10000 Address-only 1XX01 Reserved 10010 Single-beat write 1XX11 Reserved 10100 **ecowx**—Illegal single-beat write 10110 Reserved 11000 Address-only 11010 Single-beat or burst read 11100 **eciwx**—Illegal single-beat read 11110 Burst read

6.2.7 Arbiter Event Address Register (AEADR)

Arbiter event address register (AEADR) reports the address of transaction that causes the error, which is specified in the event register. See Section 6.2.3, “Arbiter Event Register (AER),” for more information.

AEADR is cleared only by power-on reset. The address of the first error event is stored. Note that this means that AEADR does not change its value when AER is not clear. As AEADR is not effected by soft or hard reset, software can read this register and determine the cause of the bus failure, even if the bus failure had caused a deadlock situation. Refer to Section 6.4.2, "Error Handling Sequence," for more information.

Figure 6-7 shows the fields of AEADR.

Offset 0x1C   Access: User read/write

	0–31
R W	ADDR
Reset	All zeros

Figure 6-7. Arbiter Event Address Register (AEADR)

Table 6-8 describes AEADR fields.

Table 6-8. AEADR Field Descriptions

Bits	Name	Description
0–31	ADDR	Address of the event reported in AEATR register. See Section 6.2.6, "Arbiter Event Attributes Register (AEATR)," for more information.

6.2.8 Arbiter Event Response Register (AERR)

Arbiter event response register (AERR) determines whether different error conditions cause interrupt or reset request. Setting a bit defines the corresponding error condition to cause reset request; clearing a bit defines the corresponding error condition to cause interrupt. Figure 6-8 shows the fields of AERR.

Offset 0x20   Access: User read/write

	0–25	26	27	28	29	30	31
R W	—	ETEA	RES	ECW	AO	DTO	ATO
Reset	All zeros						

Figure 6-8. Arbiter Event Response Register (AERR)

Table 6-9 describes AERR field.

Table 6-9. AERR Field Descriptions

Bits	Name	Description
0–25	—	Write reserved, read = 0
26	ETEA	Transfer error. Detection of transfer error by one of the slaves event response. 0 Detection of transfer error by one of the slaves causes interrupt. 1 Detection of transfer error by one of the slaves causes reset request.
27	RES	Reserved transfer type. Transaction with reserved transfer type interrupt definition. 0 Transaction with reserved transfer type causes interrupt. 1 Transaction with reserved transfer type causes reset request.

Table 6-9. AERR Field Descriptions

Bits	Name	Description
28	ECW	External control word transfer type. Transaction with external control word transfer type interrupt definition. 0 Transaction with external control word transfer type causes interrupt. 1 Transaction with external control word transfer type causes reset request.
29	AO	Address only transfer type. Transaction with address only transfer type interrupt definition. 0 Transaction with address only transfer type causes interrupt. 1 Transaction with address only transfer type causes reset request.
30	DTO	Data time out. Data tenure time out interrupt definition. 0 Data tenure time out causes interrupt. 1 Data tenure time out causes reset request.
31	ATO	Address time out. Address tenure time out interrupt definition. 0 Address tenure time out causes interrupt. 1 Address tenure time out causes reset request.

6.3 Functional Description

The following sections describe arbitration policy and bus error detection.

6.3.1 Arbitration Policy

The arbitration process involves the masters and the arbiter. Masters arbitrate on the privilege to own an address tenure. For data tenures, the arbiter uses the same order of transactions as address tenures. Figure 6-9 shows the interface signals between the arbiter and masters that are involved in address bus arbitration.

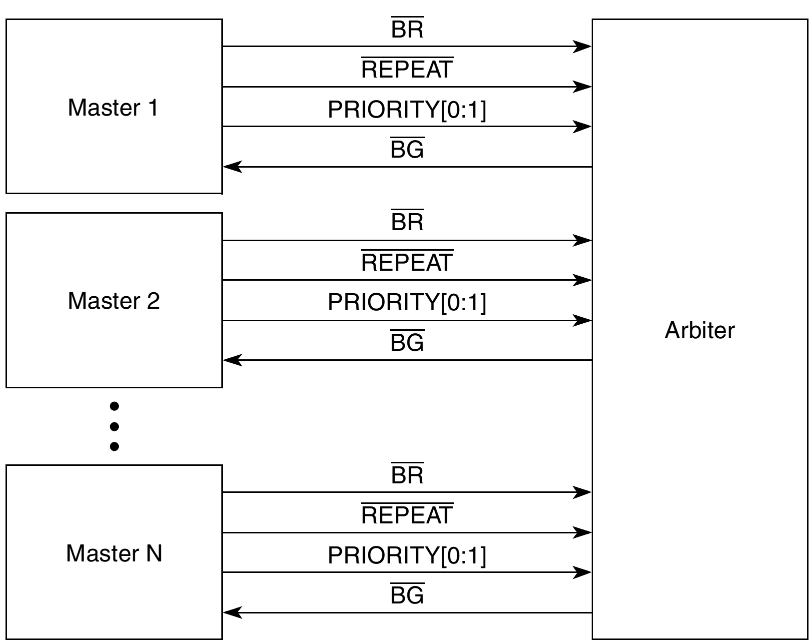


Figure 6-9. Address Bus Arbitration

A master has to acquire address bus ownership before it starts any transaction. The master asserts its own bus request signal along with the arbitration attribute signals $\overline{\text{REPEAT}}$ & PRIORITY[0:1]. The arbiter later asserts the corresponding address bus grant signal to the requesting master depending on the system states and arbitration scheme. See Section 6.3.1.1, "Address Bus Arbitration with PRIORITY[0:1]," for details on arbitration scheme. When address bus grant is received the master can start the address tenure.

6.3.1.1 Address Bus Arbitration with PRIORITY[0:1]

Whenever a master asserts its bus request to acquire address bus ownership, it can drive its PRIORITY[0:1] signals to indicate request priority. The master would be served sooner because of its higher priority level. The arbiter takes this extra information into consideration in order to yield better service for a higher priority request than a lower priority request. Therefore, the arbiter operates according to the following priority based arbitration scheme:

1. For every priority level a fair arbitration scheme is used (a simple round robin scheme)
2. For every priority level other than 0, one place is reserved as a place holder for lower level arbitration rings.
3. Each master can change its priority level at any time.

Figure 6-10 shows an example of priority-based arbitration algorithm with four priority levels. In this example, if all masters request the bus continuously, the following order of bus grants occurs with the specific bandwidth:

- M6 gets 1/2 of the bus bandwidth
- M4 and M5 each gets 1/6 of the bus bandwidth
- M0 and M3 each gets 1/18 of the bus bandwidth
- M1 and M2 each gets 1/36 of the bus bandwidth

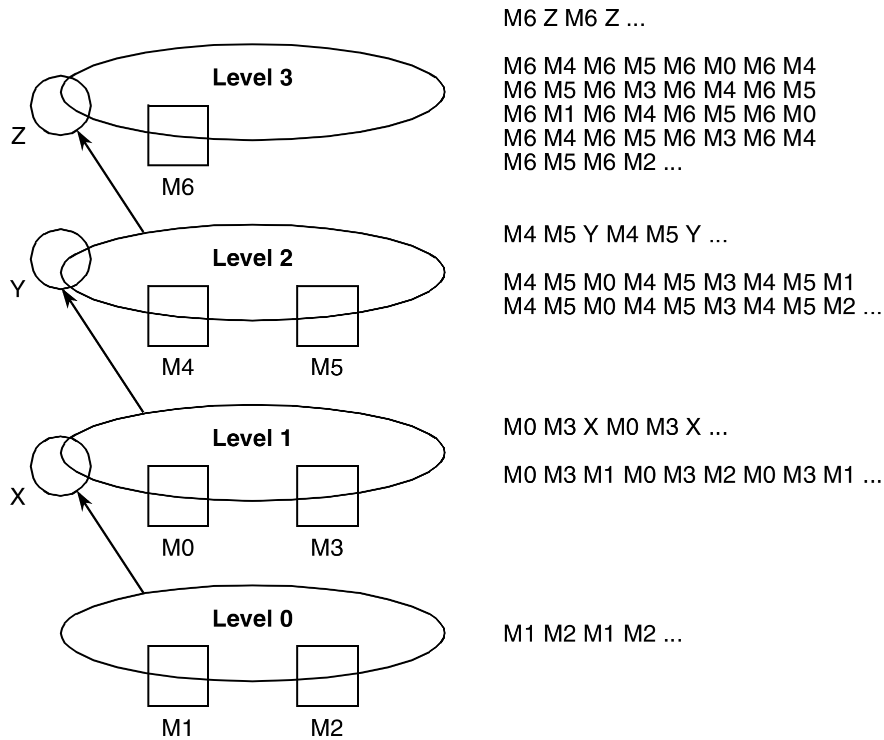


Figure 6-10. An Example of Priority-Based Arbitration Algorithm

NOTE

See each bus master's chapter and Section 5.3.2.4, "System Priority and Configuration Register (SPCR)," for more details about priority programming.

6.3.1.2 Address Bus Arbitration with $\overline{\text{REPEAT}}$

When a master owns the address bus and wants to perform another transaction, it can assert bus request along with $\overline{\text{REPEAT}}$, to make a repeat request to the arbiter. Consequently, the arbiter asserts bus grant to the same master if the current address tenure is not being $\overline{\text{ARTRY}}$ed. This happens regardless of the priority level of bus request from other masters. In another word, "repeat request" overrides the priority scheme.

Even though repeat request can improve the page hit ratio and the overall memory bandwidth efficiency, it can increase the worst case latency of individual master. Therefore, the arbiter has programmable counter to limit the maximum number of consecutive transactions that are performed by masters. Whenever the counter expires, arbiter ignores the $\overline{\text{REPEAT}}$ signal and falls back to the regular arbitration scheme. PCI master has a dedicated repeat counter as it might need more repeated transactions before accepting read requests. PCI ordering rules require that the PCI bridge should empty all queued write operations before any new read operation can begin. See Section 3.2.5, "Transaction Ordering and Posting," of the *PCI Local Bus Specifications Rev 2.2*, for more information.

See Section 6.2.1, "Arbiter Configuration Register (ACR)," for more details about programming ACR[RPTCNT]PCI.

6.3.1.3 Address Bus Arbitration after ARTRY

The ARTRY protocol is used primarily by the CPU to interrupt a transaction that hits to a modified line in its D-cache, so that it can maintain data coherency by performing the snoop copyback. When CPU asserts ARTRY, the bus is immediately granted to the CPU to perform snoop copyback. After the completion of snoop copyback, the arbiter grants the bus back to the master that had its transaction ARTRYed.

6.3.1.4 Address Bus Parking

The arbiter supports address bus parking. This feature implies that when no master is requesting the bus (all bus requests are negated), the arbiter can choose to park the address bus (or assert the address bus grant) to a master. The parked master can skip the bus request and assume the bus mastership directly. This reduces the access latency for parked master.

See Section 6.2.1, "Arbiter Configuration Register (ACR)," for more details about ACR[APARK] and ACR[PARKM].

6.3.1.5 Data Bus Arbitration

For every committed address tenure a data tenure is required to complete the transaction.

In the device system, the arbiter controls the issuing of data bus grants to a master and a slave, which are involved in a data tenure of a previously performed address tenure.

6.3.2 Bus Error Detection

The arbiter is responsible for tracking the following cases on the bus:

- Address time out
- Data time out
- Transfer error
- Address only transaction type
- Reserved transaction type
- Illegal (**eciwx**/**ecowx**) transaction type

6.3.2.1 Address Time Out

Address time out occurs, if the address tenure was not ended before the specified time-out period (programmed by ATR[ATO]). In this case, the arbiter performs as follows:

1. Ends the address tenure.
2. Starts data tenure and ends it by asserting transfer error.
3. Reports on the event to AER[ATO].
4. Issues reset request, MCP or regular interrupt according to AERR[ATO] and AIDR[ATO], if enabled by AMR[ATO].
5. Updates transaction attributes and address of AEATR and AEADR for the first error event.

6.3.2.2 Data Time Out

Data time out occurs, if the data tenure was not ended before the specified time-out period (programmed by ATR[DTO]). In this case, the arbiter performs as follows:

1. Ends the data tenure by asserting transfer error.
2. Reports on this event in AER[DTO].
3. Issues reset request, MCP or regular interrupt according to AERR[DTO] and AIDR[DTO], if enabled by AMR[DTO].
4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

6.3.2.3 Transfer Error

The arbiter tracks the transfer error asserted by one of the slaves. In this case, the arbiter performs as follows:

1. Reports on the event to AER[ETEA].
2. Issues reset request, MCP or regular interrupt according to AERR[ETEA] and AIDR[ETEA] if enabled by AMR[ETEA].
3. Updates transaction attributes and address of AEATR and AEADR for the first error event.

6.3.2.4 Address Only Transaction Type

Table 6-10 shows transaction types, which are defined as address only:

Table 6-10. Address Only Transaction Type Encoding

ttype[0:4]	Bus Commands
00000	Clean block
00100	Flush block
01000	Sync
01100	Kill block
10000	eieio
11000	TLB Invalidate
00001	lwarx reservation set
01001	tlbsync
01101	icbi

The arbiter allows address-only (AO) transactions on the bus and the G2 core has ability to issue address-only (AO) transactions (see HID0 [ABE] in the *G2 PowerPC Core Reference Manual*, Rev 1). As there is no advantage in using AO transaction in this system, the bus monitor allows the detection of AO transactions and treats them as an error.

The arbiter performs the transaction with an address only transfer type as follows:

1. Ends the address tenure by asserting $\overline{\text{AACK}}$.
2. Reports on the event to AER[AO].

3. Issues reset request, MCP or regular interrupt according to AERR[AO] and AIDR[AO] if enabled by AMR[AO].
4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

6.3.2.5 Reserved Transaction Type

Table 6-11 shows transaction types defined as reserved.

Table 6-11. Reserved Transaction Type Encoding

ttype[0:4]	Bus Commands
00101	Reserved
1xx01	Reserved for customer
10110	Reserved
00011	Reserved
00111	Reserved
01111	Reserved
1xx11	Reserved for customer

The arbiter performs the transaction with a reserved transfer type as follows:

1. Ends the address tenure by asserting $\overline{AACK}$.
2. Reports on the event to AER[RES].
3. Issues reset request, MCP or regular interrupt according to AERR[RES] and AIDR[RES], if enabled by AMR[RES].
4. Updates transaction attributes and address of AEATR and AEADR for the first error event.

6.3.2.6 Illegal (eciwx/ecowx) Transaction Type

Table 6-12 shows transaction types defined as illegal.

Table 6-12. Illegal Transaction Type Encoding

ttype[0:4]	Bus command
10100	External control word write (**ecowx**)
11100	External control word read (**eciwx**)

The arbiter performs the transaction with an illegal (eciwx, ecowx) transfer type as follows:

1. Ends the address tenure by asserting $\overline{AACK}$.
2. Starts data tenure and ends data tenure by asserting $\overline{TEA}$.
3. Reports on the event in AER[ECW].
4. Issues reset request, MCP or regular interrupt according to AERR[ECW] and AIDR[ECW], if enabled by AMR[ECW].
5. Updates transaction attributes and address of AEATR and AEADR for the first error event.

See Section 6.2.3, “Arbiter Event Register (AER),” Section 6.2.4, “Arbiter Interrupt Definition Register (AIDR),” Section 6.2.5, “Arbiter Mask Register (AMR),” Section 6.2.6, “Arbiter Event Attributes

Register (AEATR)," Section 6.2.7, "Arbiter Event Address Register (AEADR)," and Section 6.2.8, "Arbiter Event Response Register (AERR)," for more information.

6.4 Initialization/Applications Information

The following sections describe the initialization and error handling sequences for the arbiter.

6.4.1 Initialization Sequence

The following initialization sequence is recommended:

1. Write to ACR to configure pipeline depth, address bus parking mode, global maximum repeat count PCI maximum.
2. Write to AERR defines whether different error events cause a reset request or an interrupt.
3. Write to AIDR defines the kind of interrupt (regular or MCP) caused by each error event. Note that this is necessary only if interrupts are enabled and AERR defines error events to cause interrupt.
4. Write to AMR to enable interrupts.
5. Write to ATR to set the ATO and DTO timers. Note that this is only necessary if the required timers are less than the maximum value (which is default).

6.4.2 Error Handling Sequence

The following error handling sequence is recommended:

1. Read to AER to find out about the error that occurred in the system. Also, read the values of AEATR and AEADR to check on the first error event in the system.
2. If those registers are not accessible because of a bus deadlock situation, reset the chip and read the values of the AEATR and AEADR registers to check on the event that causes this problem to the system. Use $\overline{\text{HRESET}}$ to reset the chip to guarantee that the information stored in AEATR and AEADR is not lost.
3. Clear all the previous events by writing ones to the AER. This register is also cleared after reset.

Chapter 7
e300 Processor Core Overview

This chapter provides an overview of features for the embedded microprocessors in the e300 core family, which are PowerPC microprocessors built on Power Architecture technology. Throughout this chapter, the terms ‘e300 core’, ‘core’, and ‘processor’ are used interchangeably. The term ‘e300c4’ is used when describing an implementation-specific feature or when a difference exists between different configurations. The term ‘e300’ is used when describing a feature that pertains to the family of e300 processors. The MPC8379E uses an e300c4 core.

7.1 Overview

This section describes the details of the e300 core, provides a block diagram showing the major functional units, and briefly describes how these units interact. All differences between the e300 and previous PowerPC implementations derived from the MPC603e processor are noted. For additional information, please refer to the *e300 PowerPC Core Family Reference Manual*.

The e300 core is a low-power implementation of this microprocessor family of reduced instruction set computing (RISC) microprocessors. The core implements the 32-bit portion of the PowerPC architecture, which defines 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 and 64 bits.

The core is a superscalar processor that can issue and retire as many as three instructions per clock cycle. Instructions can execute out of program order for increased performance; however, the core makes completion appear sequential.

The e300 core integrates independent execution units including: an integer unit (IU) a floating-point unit (FPU), a branch processing unit (BPU), a load/store unit (LSU), and a system register unit (SRU). The e300c4 integrates an additional integer unit for a total of two IUs. The ability to execute instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for e300-core-based systems. Most integer instructions execute in one clock cycle. The additional IUs along with enhanced multipliers in the e300c4 improve multiply instructions to a maximum two-cycle latency, a significant improvement from previous processors. In the e300c4 core, the FPU is pipelined so a single-precision multiply-add instruction can be issued and completed every clock cycle. The e300c4 core provide hardware support for all single- and double-precision floating-point operations for most value representations and all rounding modes.

Figure 7-1 shows a block diagram of the e300c4 core. Note that the e300c4 supports floating-point operations and includes two integer units.

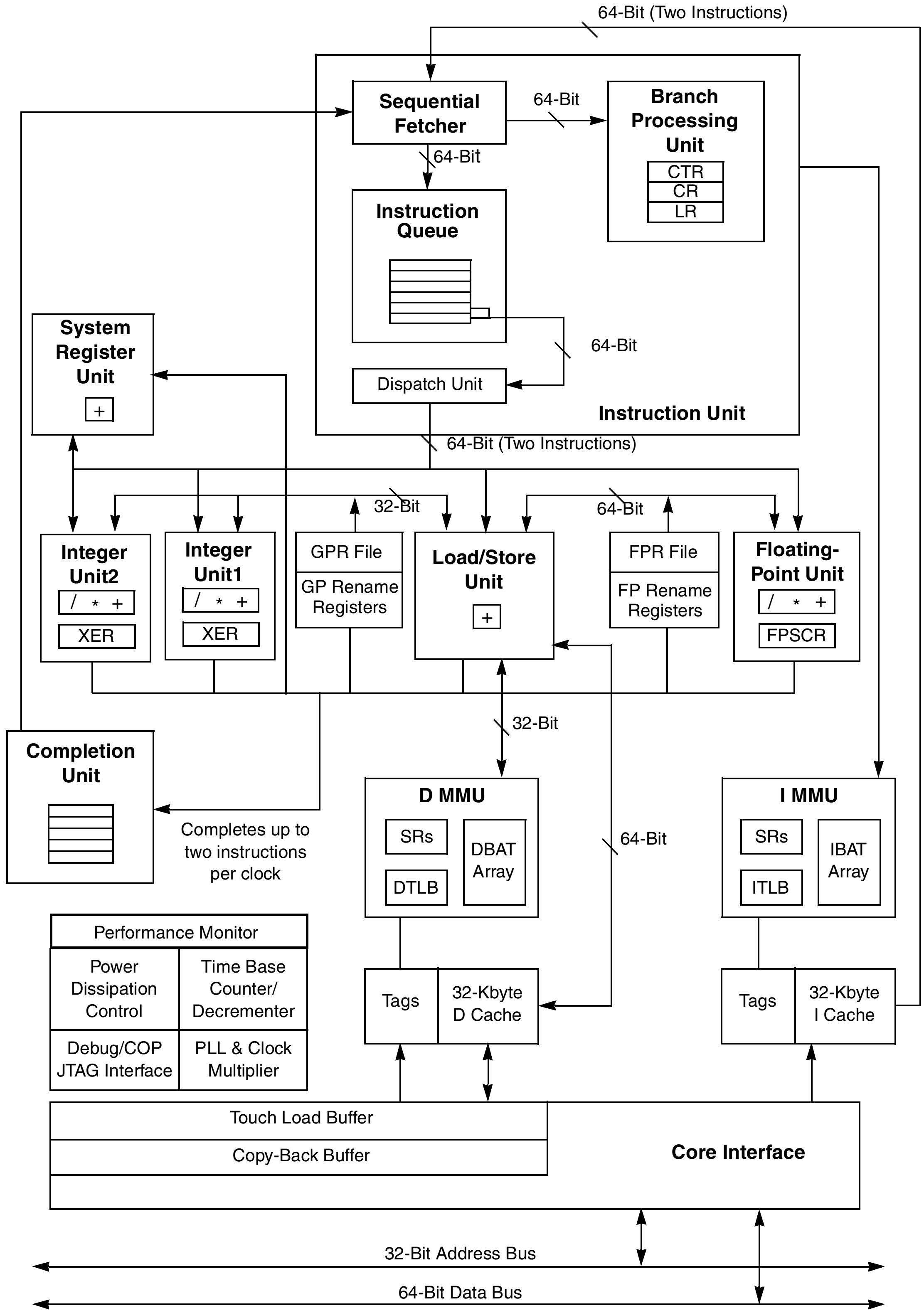


Figure 7-1. e300c4 Core Block Diagram

The e300c4 provide independent, on-chip, 32-Kbyte, eight-way, set-associative, physically-addressed caches for instructions and data, and on-chip instruction and data memory management units (MMUs). s The MMUs contain 64-entry, two-way, set-associative, data and instruction translation lookaside buffers (DTLB and ITLB) that provide support for demand-paged, virtual-memory, address translation, and variable-sized block translation. The TLBs use a least recently used (LRU) replacement algorithm and the caches use a pseudo least recently used algorithm (PLRU).

The core also supports block address translation through the use of two independent instruction and data block address translation (IBAT and DBAT) arrays, each containing eight pairs of BATs, an increase from four pairs of each type of BATs in the G2 core. This increase provides more flexibility in protecting accesses and providing translation on a segment, block, or page basis for memory accesses and I/O accesses. Effective addresses are compared simultaneously with all eight entries in the BAT array during block translation. In accordance with the PowerPC architecture, if an effective address hits in both the TLB and BAT array, the BAT translation takes priority.

As part of the coherent system bus (CSB), the e300 core has a 64-bit data bus and a 32-bit address bus. During normal operation, the e300 core provides a three-state (modified, exclusive, and invalid) coherency protocol which is a compatible subset of a four-state (modified/exclusive/shared/invalid) MESI protocol. However, the e300 data cache contains a programmable MESI extension that supports the shared cache coherency state (similar to other PowerPC processors). Both protocols operate coherently in systems that contain four-state caches. Although MESI is supported by the e300 core, it is not implemented on the MPC8379E. The core also supports single-beat and burst data transfers for memory accesses and supports memory-mapped I/O operations.

The true little-endian mode is another enhanced capability of the e300 core. Unlike the PowerPC little-endian mode (which manipulates only the address bits), no longer supported on the e300, the true little-endian mode actually operates on true little-endian instructions and data from memory.

The critical interrupt is an additional interrupt in the e300 core and has higher priority order than the system management interrupt. Also, debug features are improved in the e300. Additional SPRG interrupt handling registers are provided for enhancing flexibility for the operating system.

The and e300c4 include a performance monitor facility that provides the ability to monitor and count predefined events such as core clocks, misses in the instruction cache, data cache, or L2 cache, types of instructions dispatched, mispredicted branches, and other occurrences. The count of such events (which may be an approximation) can be used to trigger the performance monitor interrupt. Section 7.1.7.5, “Core Performance Monitor,” describes the operation of the performance monitor diagnostic tool.

7.1.1 Features

This section describes the major features of the e300 core:

- High-performance, superscalar microprocessor core
  - As many as three instructions issued and retired per clock (two instructions plus one branch instruction)
  - As many as five instructions in execution per clock
  - Single-cycle execution for most instructions

 - Pipelined floating-point unit (FPU) for all single- and double-precision operations
- Independent execution units and two register files
  - Branch processing unit (BPU) featuring static branch prediction
  - Two 32-bit integer units (IU) in the e300c4.
  - FPU based on the IEEE Std 754™ for both single- and double-precision operations
  - Load/store unit (LSU) for data transfer between data-cache and general-purpose registers (GPRs) and floating-point registers (FPRs)
  - System register unit (SRU) that executes condition register (CR), special-purpose register (SPR), and integer add/compare instructions. Add/compare instructions are also executed in the IUs.
  - Thirty-two 32-bit GPRs for integer operands
  - Thirty-two 64-bit FPRs for single- or double-precision operands
- High instruction and data throughput
  - Zero-cycle branch capability (branch folding)
  - Programmable static branch prediction on unresolved conditional branches
  - Two integer units with enhanced multipliers in the e300c4 for increased integer instruction throughput and a maximum two-cycle latency for multiply instructions
  - Instruction fetch unit capable of fetching two instructions per clock from the instruction cache
  - A six-entry instruction queue (IQ) that provides lookahead capability
  - Independent pipelines with feed-forwarding that reduces data dependencies in hardware
  - 32-Kbyte data cache and 32-Kbyte instruction cache with parity—eight-way, set-associative, physically addressed, PLRU replacement algorithm on the e300c4.
  - Cache write-back or write-through operation programmable on a per-page or per-block basis
  - Features for instruction and data cache locking and protection
  - BPU that performs CR lookahead operations
  - Address translation facilities for 4-Kbyte page size, variable block size, and 256-Mbyte segment size
  - A 64-entry, two-way, set-associative ITLB and DTLB
  - Eight-entry data and instruction BAT arrays providing 128-Kbyte to 256-Mbyte blocks
  - Software table search operations and updates supported through fast trap mechanism
  - 52-bit virtual address; 32-bit physical address
- Facilities for enhanced system performance
  - A 64-bit split-transaction internal data bus interface to the coherent system bus (CSB) with burst transfers
  - Support for one-level address pipelining on the CSB interface
  - True little-endian mode for compatibility with other true little-endian devices
  - Critical interrupt support
  - Hardware support for misaligned little-endian accesses

  - Configurable processor bus frequency multipliers as defined in the *MPC8379E Integrated Processor Hardware Specifications*
- Integrated power management
  - Internal processor/bus clock multiplier ratios
  - Three power-saving modes: doze, nap, and sleep
  - Automatic dynamic power reduction when internal functional units are idle
- In-system testability and debugging features through JTAG boundary-scan capability

Features specific to the e300 core not present on the G2 processors follow:

- Enhancements to the register set
  - The e300 core has one more HID0 bit than the G2:
    - The enable cache parity checking (ECPE) bit, HID0[1], gives the e300 core the ability to enable the taking of a machine check interrupt based on the detection of a cache parity error
- Enhancements to cache implementation
  - 32-Kbyte data cache and 32-Kbyte instruction cache with parity—eight-way, set-associative, physically addressed, PLRU replacement algorithm on the e300c4.
  - Full parity checking is performed on both instruction and data cache memory arrays
  - Lockable L1 instruction and data caches—entire cache or on a per-way basis up to 7 of 8 ways on the e300c4
  - New **icbt** instruction supports initialization of instruction cache
  - Data cache supports four-state MESI coherency protocol (not implemented on MPC8379E)
  - The instruction cache is blocked only until the critical load completes (hit under reloads allowed)
  - Instruction cancel mechanism improves utilization of instruction cache by supporting hits-under-cancels and misses-under-cancels.
  - The critical double word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.
  - Data cache queue sharing makes cast-outs and snoop pushes more efficient
  - Provides for an optional data cache operation broadcast feature (enabled by HID0[ABE]) that allows for coherent system management. All of the data cache control instructions, except **dcbz** (**dcbi**, **dcbf**, and **dcbst**) require that HID0[ABE] be enabled to broadcast.
  - Instruction fetch burst feature allows all instruction fetches from caching-inhibited space to be performed on the bus as burst transactions
- Interrupts
  - The e300 core offers hardware support for misaligned little-endian accesses. Little-endian load/store accesses that are not on a word boundary, except for strings and multiples, generate interrupts under the same circumstances as big-endian accesses.
  - The e300 core supports true little-endian mode to minimize the impact on software porting from true little-endian systems.

  - An input interrupt signal, $\overline{cint}$, is provided to trigger the critical interrupt exception on the e300 core. The pm_event_in input signal can be used by the performance monitor counters to trigger an interrupt upon overflow on the and e300c4.
- Bus clock—PLL configuration signals include seven signals for settings and control: *pll_cfg[0:6]*.
- Debug features
  - Breakpoint status recorded in DBCR and IBCR control registers
  - Two signals for the debug interface: $\overline{stopped}$ and $\overline{ext\_halt}$
  - Performance monitor registers for system analysis in the e300c4

,Figure 7-1 provides a block diagram of the e300 core that shows how the execution units—IU, FPU, BPU, LSU, and SRU—operate independently and in parallel. It should be noted that this is a conceptual diagram and does not attempt to show how these features are physically implemented on the device.

The e300 core provides address translation and protection facilities, including an ITLB, DTLB, and instruction and data BAT arrays. Instruction fetching and issuing are handled in the instruction unit. Translation of addresses for cache or external memory accesses are handled by the MMUs. Both units are discussed in more detail in Section 7.1.2, “Instruction Unit,” and Section 7.1.5.1, “Memory Management Units (MMUs).”

7.1.2 Instruction Unit

As shown in Figure 7-1, the e300 core instruction unit, containing a fetch unit, instruction queue, dispatch unit, and BPU, provides centralized control of instruction flow to the execution units. The instruction unit determines the address of the next instruction to be fetched based on information from the sequential fetcher and from the BPU.

The instruction unit fetches the instructions from the instruction cache into the instruction queue. The BPU receives branch instructions from the fetcher and uses static branch prediction to allow fetching from a predicted instruction stream while a conditional branch is evaluated. The BPU folds out for unconditional branch instructions and conditional branch instructions unaffected by instructions in the execution pipeline.

Instructions issued beyond a predicted branch cannot complete execution until the branch is resolved, preserving the programming model of sequential execution. If any of these are branch instructions, they are decoded but not issued. Instructions to be executed by the FPU, IU, LSU, and SRU are issued and allowed to progress up to the register write-back stage. Write-back is allowed when a correctly predicted branch is resolved, and execution continues along the predicted path.

If branch prediction is incorrect, the instruction unit flushes all predicted path instructions, and instructions are issued from the correct path.

7.1.2.1 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 7-1, holds as many as six instructions and loads up to two instructions from the instruction unit during a single cycle. The instruction fetch unit continuously loads as many instructions as space in the IQ allows. Instructions are dispatched to their respective execution units from the dispatch unit at a maximum rate of two instructions per cycle. Dispatching is facilitated to the IUs, FPU, LSU, and SRU by the provision of a reservation station at each unit. The dispatch unit performs source and destination register dependency checking, determines dispatch serializations, and inhibits subsequent instruction dispatching as required.

For a more detailed overview of instruction dispatch, see Section 7.4.6, "Instruction Timing."

7.1.2.2 Branch Processing Unit (BPU)

The BPU receives branch instructions from the fetch unit and performs CR lookahead operations on conditional branches to resolve them early, achieving the effect of a zero-cycle branch in many cases.

The BPU uses a bit in the instruction encoding to predict the direction of the conditional branch. Therefore, when an unresolved conditional branch instruction is encountered, the core fetches instructions from the predicted target stream until the conditional branch is resolved.

The BPU contains an adder to compute branch target addresses and three user-control registers: the link register (LR), the count register (CTR), and the conditional register (CR). The BPU calculates the return pointer for sub-routine calls and saves it into the LR for certain types of branch instructions. The LR also contains the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction. The CTR contains the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction. The contents of the LR and CTR can be copied to or from any GPR. Because the BPU uses dedicated registers rather than GPRs or FPRs, execution of branch instructions is largely independent from execution of integer and floating-point instructions.

7.1.3 Independent Execution Units

The PowerPC architecture's support for independent execution units allows implementation of processors with out-of-order instruction execution. For example, because branch instructions do not depend on GPRs or FPRs, branches can often be resolved early, eliminating stalls caused by taken branches.

The four other execution units and the completion unit are described in the following sections.

7.1.3.1 Integer Unit (IU)

The IU executes all integer instructions. The IU executes one integer instruction at a time, performing computations with its arithmetic logic unit (ALU), multiplier, divider, and XER register. Most integer instructions are single-cycle instructions. The 32 GPRs hold integer operands. Stalls due to contention for GPRs are minimized by the automatic allocation of rename registers. The core writes the contents of the rename registers to the appropriate GPR when integer instructions are retired by the completion unit. The e300c4 provides two integer units for greater integer instruction throughput along with enhanced multipliers in each IU for faster multiply-instruction execution.

7.1.3.2 Floating-Point Unit (FPU)

The FPU contains a single-precision multiply-add array and the floating-point status and control register (FPSCR). The multiply-add array allows the core to efficiently implement multiply and multiply-add operations. The FPU is pipelined so that single- and double-precision instructions can be issued back-to-back. The 32 FPRs are provided to support floating-point operations. Stalls due to contention for

FPRs are minimized by the automatic allocation of rename registers. The core writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The e300c4 core supports all floating-point data types based on the IEEE 754 standard (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software interrupt routines.

7.1.3.3 Load/Store Unit (LSU)

The LSU executes all load and store instructions and provides the data transfer interface between the GPRs, FPRs, and the cache/memory subsystem. The LSU calculates effective addresses, performs data alignment, and provides sequencing for load/store string and multiple instructions.

Load and store instructions are issued and executed in program order; however, the memory accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering.

Cacheable loads, when free of data bus dependencies, can execute out of order with a maximum throughput of one per cycle and with a two-cycle total latency. Data returned from the cache is held in a rename register until the completion logic commits the value to a GPR or FPR. Stores cannot be executed in a predicted manner and are held in the store queue until the completion logic signals that the store operation is to be completed to memory. The core executes store instructions with a maximum throughput of one per cycle and with a three-cycle total latency. The time required to perform the actual load or store depends on whether the operation involves the cache, system memory, or an I/O device.

7.1.3.4 System Register Unit (SRU)

The SRU executes various system-level instructions, including condition register logical operations and move to/from special-purpose register instructions. It also executes integer add/compare instructions. In order to maintain system state, most instructions executed by the SRU are completion-serialized; that is, the instruction is held for execution in the SRU until all prior instructions issued have completed. Results from completion-serialized instructions executed by the SRU are not available or forwarded for subsequent instructions until they complete.

7.1.4 Completion Unit

The completion unit tracks instructions in program order from dispatch through execution and then completes. Completing an instruction commits the core to any architectural register changes caused by that instruction. In-order completion ensures the correct architectural state when the core must recover from a mispredicted branch or an interrupt.

Instruction state and other information required for completion is kept in a five-entry FIFO completion queue. A single completion queue entry is allocated for each instruction once it enters the execution unit from the dispatch unit. An available completion queue entry is a required resource for dispatch; if no completion entry is available, dispatch stalls. A maximum of two instructions per cycle are completed in order from the queue.

7.1.5 Memory Subsystem Support

The core provides separate instruction and data caches and MMUs. The core also provides an efficient processor bus interface to facilitate access to main memory and other bus subsystems. The memory subsystem support functions are described in the following sections.

7.1.5.1 Memory Management Units (MMUs)

The core MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 4 Gigabytes (2^{32}) of physical memory (referred to as real memory in the architecture specification) for instruction and data. The MMUs also control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the processor for each page to assist implementation of a demand-paged virtual memory system. Note that software assistance is required for the device to maintain reference and changed status. A key bit is implemented to provide information about memory protection violations prior to page table search operations.

The LSU calculates effective addresses for data loads and stores, performs data alignment to and from cache memory, and provides the sequencing for load and store string and multiple word instructions. The instruction unit calculates effective addresses for instruction fetching.

After an EA is generated, its higher-order bits are translated by the appropriate MMU into physical address bits. The lower-order EA bits are the same on the physical address which are directed to the on-chip cache and formed the index into a four-way set-associative tag array. After translating the address, the MMU passes the higher-order physical address bits to the cache and the cache lookup completes. For caching-inhibited accesses or accesses that miss in the cache, the untranslated lower-order address bits are concatenated with the translated higher-order address bits; the resulting 32-bit physical address is then used by the memory unit and the core interface to access external memory.

The MMU also directs the address translation and enforces the protection hierarchy programmed by the operating system in relation to the supervisor/user privilege level of the access and in relation to whether the access is a load or store.

For instruction fetches, the IMMU looks for the address in the ITLB and in the IBAT array. If an address hits both, the IBAT array translation is used. Data accesses cause a lookup in the DTLB and DBAT array. In most cases, the translation is in a TLB and the physical address bits are available to the on-chip cache.

The e300 core implements four more IBAT and four more DBAT entries than the G2.

When the EA misses in the TLBs, the core provides hardware assistance for software to perform a search of the translation tables in memory. The hardware assist consists of the following features:

- Automatic storage of the missed effective address in IMISS and DMISS
- Automatic generation of the primary and secondary hashed real addresses of the page-table entry group (PTEG), which are readable from the HASH1 and HASH2 register locations.

  The HASH data is generated from the contents of the IMISS or DMISS register. The register that is selected depends on the miss (instruction or data) that was last acknowledged.
- Automatic generation of the first word of the page table entry (PTE) of the tables being searched
- A real page address (RPA) register that matches the format of the lower word of the PTE

- TLB access instructions (**tlbli** and **tlbld**) that are used to load an address translation into the instruction or data TLBs
- Shadow registers for GPR0–GPR3 that allow miss code to execute without corrupting the state of any of the existing GPRs. Shadow registers are used only for servicing a TLB miss.

See Section 7.4.5.2, "Implementation-Specific Memory Management," for more information about memory management for the core.

7.1.5.2 Cache Units

The and e300c4 provides independent, 32-Kbyte, eight-way, set-associative, instruction and data caches. s The cache block is 32 bytes long. The caches adhere to a write-back policy, but the e300 core allows control of cacheability, write policy, and memory coherency at the page and block levels. The caches use a pseudo LRU replacement policy.

As shown in Figure 7-1, the caches provide a 64-bit interface to the instruction fetch unit and LSU. The surrounding logic selects, organizes, and forwards the requested information to the requesting unit. Write operations to the cache can be performed on a byte basis, and a complete read-modify-write operation to the cache can occur in each cycle.

The load/store and instruction fetch units provide the caches with the address of the data or instruction to be fetched. In the case of a cache hit, the cache returns two words to the requesting unit.

Because the data cache tags are single-ported, simultaneous load/store and snoop accesses cause resource contention. Snoop accesses have the highest priority and are given first access to the tags, unless the snoop access coincides with a tag write; in this case the snoop is retried and must rearbitrate for cache access. Loads or stores deferred due to snoop accesses are performed on the clock cycle following the snoop.

The e300 core includes a new instruction cancel extension. The instruction cancel extension improves utilization of the instruction cache during cancel operations. It allows a new instruction fetch to be issued to the cache or to the bus if a canceled instruction fetch is pending or active on the bus. This supports hit-under-cancel and miss-under-cancel instruction fetch operations.

7.1.6 Bus Interface Unit (BIU)

Because the caches are on-chip, write-back caches, the most common transactions are burst-read memory operations, burst-write memory operations, and single-beat (noncacheable or write-through) memory read and write operations. There can also be address-only operations, variants of the burst and single-beat operations, (for example, global memory operations that are snooped and atomic memory operations), and address retry activity (for example, when a snooped read access hits a modified cache block).

Memory accesses can occur in single-beat (1–8 bytes) and four-beat burst (32 bytes) data transfers on the 64-bit data bus. The address and data buses operate independently to support pipelining and split transactions during memory accesses.

The e300 bus interface unit (BIU) has been enhanced to allow a pipeline slot to become available once a previous transaction has been granted the data bus (that is, as early as when the data tenure starts rather than after the data tenure completes), thus allowing for greater bus utilization in systems that support it. This is sometimes referred to as 1 1/2-level pipelining.

Typically, memory accesses are weakly ordered, meaning that sequences of operations, including load/store string and multiple instructions, do not necessarily complete in the order they begin. This weak ordering maximizes the efficiency of the bus without sacrificing coherency of the data. The core allows read operations to precede store operations (except when a dependency exists, or in cases where a noncacheable access is performed), and provides support for a write operation to proceed a previously queued read data tenure (for example, allowing a snoop push to be enveloped by the address and data tenures of a read operation). Because the processor can dynamically optimize run-time ordering of load/store traffic, overall performance is improved.

7.1.7 System Support Functions

The e300 core implements several support functions that include power management, time base/decrementer registers for system timing tasks, a JTAG (based on IEEE Std 1149.1™ standard) interface, hardware debug, and a phase-locked loop (PLL) clock multiplier. These system support functions are described in the following sections.

7.1.7.1 Power Management

The e300 core provides four power modes, selectable by setting the appropriate control bits in the machine state register (MSR) and the hardware implementation register 0 (HID0). When entering into a power mode other than full-power, the core will request entry via a $\overline{qreq}$ signal and will only enter another power mode after an acknowledge ($\overline{qack}$) is received. The four power modes are as follows:

- Full-power—This is the default power state of the e300 core. The e300 core is fully powered and the internal functional units are operating at the full processor clock speed. If the dynamic power management mode is enabled, functional units that are idle will automatically enter a low-power state without affecting performance, software execution, or external hardware.
- Doze—All the functional units of the e300 core are disabled except for the time base/decrementer registers and the bus snooping logic. When the processor is in doze mode, an external asynchronous interrupt, system management interrupt, decrementer interrupt, hard or soft reset, or machine check brings the e300 core into the full-power state. The core in doze mode maintains the PLL in a fully-powered state and locked to the system external clock input (*sysclk*), so a transition to the full-power state takes only a few processor clock cycles.
- Nap—The nap mode further reduces power consumption by disabling bus snooping, leaving only the time base register and the PLL in a powered state. The core returns to the full-power state on receipt of an external asynchronous interrupt, system management interrupt, decrementer interrupt, hard or soft reset, or machine check input ($\overline{mcp}$) signal. A return to full-power state from a nap state takes only a few processor clock cycles.
- Sleep—Sleep mode reduces power consumption to a minimum by disabling all internal functional units; then external system logic may disable the PLL and *sysclk*. Returning the core to the full-power state requires the enabling of the PLL and *sysclk*, followed by the assertion of an external asynchronous interrupt, system management interrupt, hard or soft reset, or $\overline{mcp}$ signal after the time required to relock the PLL.

7.1.7.2 Time Base/Decrementer

The time base is a 64-bit register (accessed as two 32-bit registers) that is incremented once every four bus clock cycles; external control of the time base is provided through the time base/decrementer clock base enable (*tben*) signal. The decrementer is a 32-bit register that generates a decrementer interrupt after a programmable delay. The contents of the decrementer register are decremented once every four bus clock cycles, and the decrementer interrupt is generated as the count passes through zero.

7.1.7.3 JTAG Test and Debug Interface

The core provides JTAG and hardware debug functions for facilitating board testing and chip debugging. The JTAG test interface (based on IEEE Std. 1149.1 standard) provides a means for boundary-scan testing of the core and the attached system logic. The hardware debug function accesses the JTAG test port, providing a means for executing test routines and facilitating chip and software debugging.

All instruction and data address breakpoints are accessible in the IBCR and DBCR. See Section 7.4.8, “Debug Features,” for more information.

7.1.7.4 Clock Multiplier

The internal clocking of the e300 core is generated from and synchronized to the external clock signal, *sysclk*, by means of a voltage-controlled, oscillator-based PLL. The PLL provides programmable internal processor clock multiplier ratios which multiply the externally supplied clock frequency. The bus clock is the same frequency and is synchronous with *sysclk*. The configuration of the PLL can be read by software from the hardware implementation register 1 (HID1).

7.1.7.5 Core Performance Monitor

The performance monitor provides the ability to count predefined events and processor clocks associated with particular operations, such as cache misses, mispredicted branches, or the number of cycles an execution unit stalls. The count of such events can be used to trigger the performance monitor interrupt.

The performance monitor can be used to do the following:

- Improve system performance by monitoring software execution and then recoding algorithms for more efficiency. For example, memory hierarchy behavior can be monitored and analyzed to optimize task scheduling or data distribution algorithms.
- Characterize processors in environments not easily characterized by benchmarking.
- Help system developers bring up and debug their systems.

The performance monitor uses the following resources:

- The performance monitor mark bit in the MSR (MSR[PMM]). This bit controls which programs are monitored.
- The move to/from performance monitor registers (PMR) instructions, **mtpmr** and **mfpmr**.
- The external core input, *pm_event_in*. On the MPC8379E, *pm_event_in*, for example, can be used to count the number of transactions started on the coherent system bus (CSB).
- PMRs:

- — The performance monitor counter registers (PMC0–PMC3) are 32-bit counters used to count software-selectable events. Each counter counts up to 128 events. UPMC0–UPMC3 provide user-level read access to these registers. They are identified in Table 7-2.
- — The performance monitor global control register (PMGC0) controls the counting of performance monitor events. It takes priority over all other performance monitor control registers. UPMGC0 provides user-level read access to PMGC0.
- — The performance monitor local control registers (PMLCa0–PMLCa3) control each individual performance monitor counter. Each counter has a corresponding PMLCa register. UPMLCa0–UPMLCa3 provide user-level read access to PMLCa0–PMLCa3).
- The performance monitor interrupt is assigned to interrupt vector 0x0F00.

Software communication with the performance monitor is achieved through PMRs rather than SPRs. The PMRs are used for enabling conditions that can trigger the performance monitor interrupt.

7.2 e300 Processor and System Version Numbers

Table 7-1 lists the revision codes in the processor version register (PVR) and the system version register (SVR) to the revision level marked on the device. These registers can be accessed as SPRs through the e300 core (see Figure 7-2).

Table 7-1. Device Revision Level Cross-Reference

MPC8379E Revision	Processor Version Register (PVR)	System Version Register (SVR)
1.0	0x8086_1010	MPC8379E (with security): 80C2_0010 MPC8379 (without security): 80C3_0010 MPC8377E (with security): 80C6_0010 MPC8377 (without security): 80C7_0010 MPC8378E (with security): 80C4_0010 MPC8378 (without security): 80C5_0010
1.1	—	—

7.3 PowerPC Architecture Implementation

The PowerPC architecture consists of the following layers, and adherence to the PowerPC architecture can be measured in terms of which of the following levels of the architecture is implemented:

- User instruction set architecture (UISA)—Defines the base user-level instruction set, user-level registers, data types, floating-point interrupt model, memory models for a uniprocessor environment, and programming model for a uniprocessor environment.
- Virtual environment architecture (VEA)—Describes the memory model for a multiprocessor environment, defines cache control instructions, and describes other aspects of virtual environments. Implementations that conform to the VEA also adhere to the UISA but may not necessarily adhere to the OEA.
- Operating environment architecture (OEA)—Defines the memory management model, supervisor-level registers, synchronization requirements, and interrupt model. Implementations that conform to the OEA also adhere to the UISA and VEA.

The PowerPC architecture allows a wide range of designs for such features as cache and core interface implementations.

7.4 Implementation-Specific Information

This section describes the PowerPC architecture in general and provides specific details about the implementation of the e300 core as a low-power, 32-bit member of this PowerPC core family. The main topics addressed are as follows:

- Section 7.4.1, "Register Model," describes the registers for the operating environment architecture common among e300 cores that implement the PowerPC architecture and describes the programming model. It also describes the additional registers that are unique to the core.
- Section 7.4.2, "Instruction Set and Addressing Modes," describes the PowerPC instruction set and addressing modes for the OEA, and defines and describes the instructions implemented in the core.
- Section 7.4.3, "Cache Implementation," describes the cache model that is defined generally for cores that implement the PowerPC architecture by the VEA. It also provides specific details about the e300 core cache implementation.
- Section 7.4.4, "Interrupt Model," describes the interrupt model of the OEA and the differences in the core interrupt model.
- Section 7.4.5, "Memory Management," describes generally the conventions for memory management among these cores. This section also describes the core implementation of the 32-bit PowerPC memory management specification.
- Section 7.4.6, "Instruction Timing," provides a general description of the instruction timing provided by the superscalar, parallel execution supported by the PowerPC architecture and the e300 core.
- Section 7.1.6, "Bus Interface Unit (BIU)," describes the signals implemented on the core.

The e300 core is a high-performance, superscalar processor core. The PowerPC architecture allows optimizing compilers to schedule instructions to maximize performance through efficient use of the PowerPC instruction set and register model. The multiple, independent execution units allow compilers to optimize instruction throughput. Compilers that take advantage of the flexibility of the PowerPC architecture can additionally optimize system performance.

The following sections summarize the features of the core, including both those that are defined by the architecture and those that are unique to the various core implementations.

Specific features of the core are listed in Section 7.1.1, "Features."

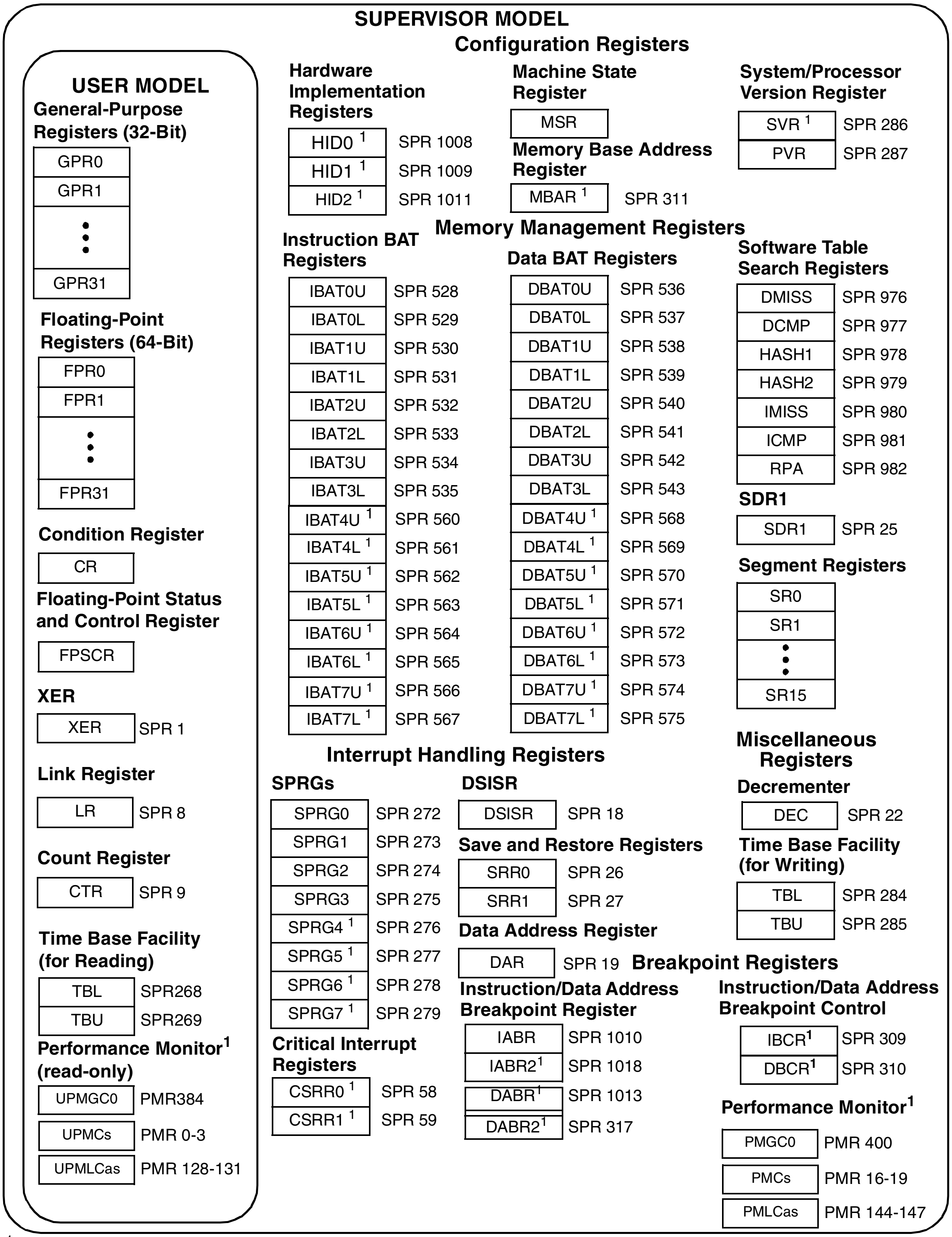
7.4.1 Register Model

The PowerPC architecture defines register-to-register operations for most computational instructions. Source operands for these instructions are accessed from the registers or are provided as immediate values embedded in the instruction opcode. The three-register instruction format allows specification of a target register distinct from the two-source operands. Load and store instructions transfer data between registers and memory.

The e300 core has two levels of privilege: supervisor mode of operation (typically used by the operating system) and user mode of operation (used by the application software). The programming models incorporate 32 GPRs, 32 FPRs, special-purpose registers (SPRs), and several miscellaneous registers. Each core also has its own unique set of hardware implementation (HID) registers.

Having access to privileged instructions, registers, and other resources allows the operating system to control the application environment (providing virtual memory and protecting operating system and critical machine resources). Instructions that control the state of the e300 core, the address translation mechanism, and supervisor registers can be executed only when the core is operating in supervisor mode.

Figure 7-2 shows all the core registers available at the user and supervisor level. The numbers to the right of the SPRs indicate the number that is used in the syntax of the instruction operands for the move to/from SPR instructions.



[1] These registers are e300 core implementation-specific (not defined by the PowerPC architecture).

Figure 7-2. e300 Programming Model—Registers

The following sections describe the e300-core-implementation-specific features as they apply to registers.

7.4.1.1 UISA Registers

UISA registers are user-level registers that include the following.

7.4.1.1.1 General-Purpose Registers (GPRs)

The PowerPC architecture defines 32 user-level GPRs that are 32 bits wide in 32-bit cores. The GPRs serve as the data source or destination for all integer instructions.

7.4.1.1.2 Floating-Point Registers (FPRs)

The PowerPC architecture also defines 32 user-level, 64-bit FPRs. The FPRs serve as the data source or destination for floating-point instructions. These registers can contain data objects of either single- or double-precision floating-point formats.

7.4.1.1.3 Condition Register (CR)

The CR is a 32-bit user-level register that provides a mechanism for testing and branching. It consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point comparisons, arithmetic, and logical operations.

7.4.1.1.4 Floating-Point Status and Control Register (FPSCR)

The user-level FPSCR contains all floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits needed for compliance with the IEEE Std. 754™ standard.

7.4.1.1.5 User-Level SPRs

The PowerPC architecture defines numerous special purpose registers that serve a variety of functions, such as providing controls, indicating status, configuring the core, and performing special operations. During normal execution, a program can access the registers, as shown in Figure 7-2, depending on the program's access privilege (supervisor or user, determined by the privilege-level bit, MSR[PR]). Note that GPRs and FPRs are accessed through operands that are part of the instructions. Access to registers can be explicit (that is, through the use of specific instructions for that purpose such as Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions) or implicit, as the part of the execution of an instruction. Some registers are accessed both explicitly and implicitly. In the e300 core, all SPRs are 32 bits wide.

The following SPRs are accessible by user-level software:

- Link register (LR)—The LR can be used to provide the branch target address and to hold the return address after branch and link instructions. The LR is 32 bits wide in 32-bit implementations.
- Count register (CTR)—The CTR is decremented and tested automatically as a result of branch-and-count instructions. The CTR is 32 bits wide in 32-bit implementations.
- XER register—The 32-bit XER contains the summary overflow bit, integer carry bit, overflow bit, and a field specifying the number of bytes to be transferred by a Load String Word Indexed (**lswx**) or Store String Word Indexed (**stswx**) instruction.

7.4.1.2 VEA Registers

The VEA introduces the time base facility (TB) for reading. The TB is a 64-bit register pair whose contents are incremented once every four core input clock cycles. The TB consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL). Note that the time base registers are read-only in user state.

7.4.1.3 OEA Registers

OEA registers are supervisor-level registers that include the following.

7.4.1.3.1 Machine State Register (MSR)

The MSR is a supervisor-level register that defines the state of the core. The contents of this register are saved when an interrupt is taken, and restored when the interrupt handling completes. A critical interrupt interrupt is taken in the e300 core when the $\overline{cint}$ signal is asserted and MSR[CE] is set. The e300 core implements the MSR as a 32-bit register.

Table 7-2 shows the bit definitions for MSR.

Table 7-2. MSR Bit Descriptions

Bits	Name	Description
0[1]	—	Reserved. Full function.
1–4[1]	—	Reserved. Partial function.
5–9[1]	—	Reserved. Full function.
10–12[1]	—	Reserved. Partial function.
13	POW	Power management enable (implementation-specific) 0 Disables programmable power modes (normal operation mode) 1 Enables programmable power modes (nap, doze, or sleep mode). This bit controls the programmable power modes only; it has no effect on dynamic power management (DPM). MSR[POW] may be altered with an **mtmsr** instruction only. Also, when altering the POW bit, software may alter only this bit in the MSR and no others. The **mtmsr** instruction must be followed by a context-synchronizing instruction.
14	TGPR	Temporary GPR remapping (implementation-specific) 0 Normal operation 1 TGPR mode. GPR0–GPR3 are remapped to TGPR0–TGPR3 for use by TLB miss routines. The contents of GPR0–GPR3 remain unchanged while MSR[TGPR] = 1. Attempts to use GPR4–GPR31 with MSR[TGPR] = 1 yield undefined results. Temporarily replaces TGPR0–TGPR3 with GPR0–GPR3 for use by TLB miss routines. The TGPR bit is set when either an instruction TLB miss, data read miss, or data write miss interrupt is taken. The TGPR bit is cleared by an **rfi** instruction.
15	ILE	Interrupt little-endian mode. When an interrupt occurs, this bit is copied into MSR[LE] to select the endian mode for the context established by the interrupt.
16	EE	External interrupt enable 0 The processor ignores external interrupts, system management interrupts, and decrementer interrupts. 1 The processor is enabled to take an external interrupt, system management interrupt, or decrementer interrupt.
17	PR	Privilege level 0 The processor can execute both user- and supervisor-level instructions 1 The processor can only execute user-level instructions

Table 7-2. MSR Bit Descriptions (continued)

Bits	Name	Description
18	FP	Floating-point available 0 The processor prevents dispatch of floating-point instructions, including floating-point loads, stores, and moves. 1 The processor can execute floating-point instructions and can take floating-point enabled exception type program interrupts.
19	ME	Machine check enable 0 Machine check interrupts are disabled 1 Machine check interrupts are enabled
20	FE0	Floating-point exception mode 0
21	SE	Single-step trace enable 0 The processor executes instructions normally 1 The processor generates a trace interrupt upon the successful completion of the next instruction
22	BE	Branch trace enable 0 The processor executes branch instructions normally 1 The processor generates a trace interrupt upon the successful completion of a branch instruction
23	FE1	Floating-point exception mode 1
24	CE	Critical interrupt enable 0 Critical interrupts disabled 1 Critical interrupts enabled; critical interrupt and **rfci** instruction enabled The critical interrupt is an asynchronous implementation-specific interrupt. The critical interrupt vector offset is 0x00A00. The **rfci** instruction is implemented to return from these interrupt handlers. Also, CSRR0 and CSRR1 are used to save and restore the processor state for critical interrupts.
25	IP	Interrupt prefix. The setting of this bit specifies whether an interrupt vector offset is prepended with Fs or 0s. In the following description, *nnnnn* is the offset of the interrupt. 0 Interrupts are vectored to the physical address 0x000*n_nnnn* 1 Interrupts are vectored to the physical address 0xFFF*n_nnnn*
26	IR	Instruction address translation 0 Instruction address translation is disabled 1 Instruction address translation is enabled
27	DR	Data address translation 0 Data address translation is disabled 1 Data address translation is enabled
28–29[1]	—	Reserved. Full function. Bit 29 not reserved on e300c4.
29	PMM	Performance monitor mark bit (e300c4). System software can set PMM when a marked process is running to enable statistics to be gathered only during the execution of the marked process. MSR[PR] and MSR[PMM] together define a state that the processor (supervisor or user) and the process (marked or unmarked) may be in at any time. If this state matches an individual state specified in the PMLCa*n*, the state for which monitoring is enabled, counting is enabled.
30	RI	Recoverable interrupt (for system reset and machine check interrupts) 0 Interrupt is not recoverable 1 Interrupt is recoverable
31	LE	Little-endian mode enable 0 The processor runs in big-endian mode 1 The processor runs in little-endian mode.

[1] All reserved bits should be set to zero for future compatibility.

7.4.1.3.2 Segment Registers (SRs)

For memory management, 32-bit processors implement sixteen 32-bit SRs. To speed access, the core implements the SRs as two arrays: a main array, for data memory accesses, and a shadow array, for instruction memory accesses. Loading a segment entry with the Move to Segment Register (**mtsr**) instruction loads both arrays.

7.4.1.3.3 Supervisor-Level SPRs

The e300 core, like the G2_LE core, has additional supervisor-level SPRs, which are shown in Figure . Two critical interrupt SPRs (CSRR0 and CSRR1), eight SPRGs (SPRG0–SPRG7), eight pairs of instruction BATs (IBAT0–IBAT7), eight pairs of data BATs (DBAT0–DBAT7), one system version register (SVR), one system memory base address (MBAR), one instruction address breakpoint control (IBCR), one data address breakpoint control (DBCR), a new instruction breakpoint register (IABR2), and two data address breakpoint registers (DABR and DABR2) are integrated into the core.

Supervisor-level SPRs include the following:

- The DSISR defines the cause of data access and alignment interrupts. The cause of a DSI interrupt for a data breakpoint (match with DABR and DABR2) can be determined by the value of the DSISR[DABR] bit (bit 9).
- The data address register (DAR) holds the address of an access after an alignment or DSI interrupt. For example, it contains the address of the breakpoint match condition.
- The decrementer register (DEC) is a 32-bit decrementing counter that provides a mechanism for causing a decrementer interrupt after a programmable delay.
- SDR1 specifies the page table format used in virtual-to-physical address translation for pages. (Note that physical address is referred to as ‘real address’ in the architecture specification.)
- The machine status save/restore register 0 (SRR0) is used for saving the address of the instruction that caused the interrupt, and the address to return to when a Return from Interrupt (**rfi**) instruction is executed.
- The machine status save/restore register 1 (SRR1) is used to save machine status on interrupts and to restore machine status when an **rfi** instruction is executed.
- The SPRG0–SPRG7 registers are provided for operating system use. They reduce the latency that may be incurred in the saving of registers to memory while in a handler. Note that the e300 implements four more SPRGs than the G2 (SPRG0–SPRG3).
- The time base register (TB) is a 64-bit register that maintains the time of day and operates interval timers. It consists of two 32-bit fields: time base upper (TBU) and time base lower (TBL).
- The processor version register (PVR) is a read-only register that identifies the version (model) and revision level of the processor. See Table 7-9 for the version and revision level of the PVR for the e300 processor core.
- Block address translation (BAT) arrays—The PowerPC architecture defines 16 BAT registers. The e300 core includes a total of eight pairs of DBAT and eight pairs of IBAT registers. See Figure 7-2 for a list of the SPR numbers for the BAT arrays.

The following supervisor-level SPRs are implementation-specific (not defined in the PowerPC architecture):

- DMISS and IMISS are read-only registers that are loaded automatically on an instruction or data TLB miss.
- HASH1 and HASH2 contain the physical addresses of the primary and secondary page table entry groups (PTEGs).
- ICMP and DCMP contain a duplicate of the first word in the page table entry (PTE) for which the table search is looking.
- The required physical address (RPA) register is loaded by the core with the second word of the correct PTE during a page table search.
- The system version register (SVR) is available on the e300 core, which identifies the specific version (model) and revision level of the system-on-a-chip (SOC) integration.
- System memory base address (MBAR) is an implementation-specific register available on the e300 core. It supports a temporary storage for the system-level memory map.
- The instruction and data address breakpoint registers (IABR, IABR2, DABR, DABR2) are loaded with an instruction or data address, respectively, that is compared to instruction addresses in the dispatch queue or to the data address in the LSU. When an address match occurs, a breakpoint interrupt is generated.
- One instruction breakpoint control register (IBCR) and one data breakpoint control register (DBCR) are implemented in the e300 core.
- To support critical interrupts, two registers (CSRR0 and CSRR1) are included in the e300 core.
- Eight SPRG registers (SPRG0–SPRG7) are in the e300 core.
- Block address translation (BAT) arrays—The e300 core has eight instruction and eight data BAT registers.
- The hardware implementation (HID0 and HID1) registers provide the means for enabling core checkstops and features and allow software to read the configuration of the PLL configuration signals. The HID2 register enables the true little-endian mode, cache way-locking, and the additional BAT registers.

Table 7-3 shows the bit definitions for HID0.

Table 7-3. e300 HID0 Bit Descriptions

Bits	Name	Function
0	EMCP	Enable $\overline{mcp}$. The purpose of this bit is to mask out machine check interrupts caused by assertion of $\overline{mcp}$, similar to how MSR[EE] can mask external interrupts. 0 Masks $\overline{mcp}$. Asserting $\overline{mcp}$ does not generate a machine check interrupt or a checkstop. 1 Asserting $\overline{mcp}$ causes checkstop if MSR[ME] = 0 or a machine check interrupt if ME = 1
1	ECPE	Enable cache parity errors. 0 Disables instruction and data cache parity error reporting 1 Allows a detected cache parity error to cause a machine check interrupt if MSR[ME] = 1 or a checkstop if MSR[ME] = 0

Table 7-3. e300 HID0 Bit Descriptions (continued)

Bits	Name	Function
2	EBA	Enable *ap_in[0:3]* and *ape* for address parity checking. 0 Disables address parity checking during a snoop operation 1 Allows an address parity error during snoop operations to cause a checkstop if MSR[ME] = 0 or a machine check interrupt if MSR[ME] = 1
3	EBD	Enable *dpe* for data parity checking. 0 Disables data parity checking 1 Allows a data parity error during reads to cause a checkstop if MSR[ME] = 0 or a machine check interrupt if MSR[ME] = 1
4	SBCLK	*clk_out* output enable. Used in conjunction with HID0[ECLK] and *hreset* to configure *clk_out*. See Table 7-4 for settings.
5	—	Reserved, should be cleared
6	ECLK	*clk_out* output enable. Used in conjunction with HID0[SBCLK] and the *hreset* signal to configure *clk_out*. See Table 7-4 for settings.
7	PAR	Disable precharge of *artry_out* 0 Precharge of *artry_out* enabled 1 Alters bus protocol slightly by preventing the processor from driving *artry_out* to high (negated) state. If this is done, the integrated device must restore the signals to the high state.
8	DOZE	Doze mode enable. Operates in conjunction with MSR[POW]. 0 Doze mode disabled 1 Doze mode enabled. Doze mode is invoked by setting MSR[POW] while this bit is set. In doze mode, the PLL, time base, and snooping remain active.
9	NAP	Nap mode enable. Operates in conjunction with MSR[POW]. 0 Nap mode disabled 1 Nap mode enabled. Nap mode is invoked by setting MSR[POW] while this bit is set. In nap mode, the PLL and time base remain active.
10	SLEEP	Sleep mode enable. Operates in conjunction with MSR[POW]. 0 Sleep mode disabled 1 Sleep mode enabled. Sleep mode is invoked by setting MSR[POW] while this bit is set. *qreq* is asserted to indicate that the processor is ready to enter sleep mode. If the system logic determines that the processor may enter sleep mode, the quiesce acknowledge signal, *qack,* is asserted back to the processor. Once *qack* assertion is detected, the processor enters sleep mode after several processor clocks. At this point, the system logic may turn off the PLL by first configuring *pll_cfg[0:6]* to PLL bypass mode, then disabling *sysclk*.
11	DPM	Dynamic power management enable 0 Dynamic power management is disabled 1 Functional units enter a low-power mode automatically if the unit is idle. This does not affect operational performance and is transparent to software or any external hardware.
12–15	—	Reserved, should be cleared.
16	ICE	Instruction cache enable 0 The instruction cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all instruction fetches are propagated to the coherent system bus (CSB) as single-beat transactions. For those transactions, however, *ci* reflects the state of the I bit in the MMU for that page regardless of cache disabled status. ICE is zero at power-up. 1 The instruction cache is enabled

Table 7-3. e300 HID0 Bit Descriptions (continued)

Bits	Name	Function
17	DCE	Data cache enable 0 The data cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIM = x1x). Potential cache accesses from the bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all data read and write accesses are propagated to the CSB as single-beat transactions. For those transactions, however, $\overline{ci}$ reflects the state of the I bit in the MMU for that page regardless of cache disabled status. DCE is zero at power-up. 1 The data cache is enabled
18	ILOCK	Instruction cache lock 0 Normal operation 1 The entire instruction cache is locked (that is, all eight ways of the cache are locked). A locked cache supplies data normally on a hit, but the access is treated as a cache-inhibited transaction on a miss. On a miss, the transaction to the bus is single-beat; however, $\overline{ci}$ still reflects the state of the I bit in the MMU for that page independent of cache locked or disabled status. To prevent locking during a cache access, an **isync** instruction must precede the setting of ILOCK.
19	DLOCK	Data cache lock 0 Normal operation 1 The entire data cache is locked (that is, all eight ways of the cache are locked). A locked cache supplies data normally on a hit, but is treated as a cache-inhibited transaction on a miss. On a miss, the transaction to the bus is single-beat; however, $\overline{ci}$ still reflects the state of the I bit in the MMU for that page independent of cache locked or disabled status. A snoop hit to a locked L1 data cache performs as if the cache were not locked. A cache block invalidated by a snoop remains invalid until the cache is unlocked. To prevent locking during a cache access, a **sync** instruction must precede the setting of DLOCK.
20	ICFI	Instruction cache Flash invalidate 0 The instruction cache is not invalidated. The bit is cleared when the invalidation operation begins (usually the next cycle after the write operation to the register). The instruction cache must be enabled for the invalidation to occur. 1 An invalidate operation is issued that marks the state of each instruction cache block as invalid. Cache access is blocked during this time. Setting ICFI clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set. For the e300 core, the proper use of the ICFI and DCFI bits is to set and clear them with two consecutive **mtspr** operations.
21	DCFI	Data cache Flash invalidate 0 The data cache is not invalidated. The bit is cleared when the invalidation operation begins (usually the next cycle after the write operation to the register). The data cache must be enabled for the invalidation to occur. 1 An invalidate operation is issued that marks the state of each data cache block as invalid without writing back modified cache blocks to memory. Cache access is blocked during this time. Bus accesses to the cache are signaled as a miss during invalidate-all operations. Setting DCFI clears all the valid bits of the blocks and the PLRU bits to point to way L0 of each set. For the e300 core, the proper use of the ICFI and DCFI bits is to set and clear them with two consecutive **mtspr** operations.
22–23	—	Reserved, should be cleared.
24	IFEM	Enable M bit on bus for instruction fetches 0 M bit not reflected on bus for instruction fetches. Instruction fetches are treated as nonglobal on the bus. 1 Instruction fetches reflect the M bit from the WIM settings

Table 7-3. e300 HID0 Bit Descriptions (continued)

Bits	Name	Function
25	DECAREN	Decrementer auto reload 0 Normal operation. 1 Decrementer loads last mtdec value for precise periodic interrupt.
26	—	Reserved, should be cleared.
27	FBIOB	Force branch indirect on the bus 0 Register indirect branch targets are fetched normally 1 Forces register indirect branch targets to be fetched externally
28	ABE	Address broadcast enable. Controls whether certain address-only operations (such as cache operations) are broadcast on the bus. 0 Address-only operations affect only local caches and are not broadcast 1 Address-only operations are broadcast on the bus Affected instructions are **dcbi**, **dcbf**, and **dcbst**. Note that these cache control instruction broadcasts are not snooped by the e300 core. Refer to Section 4.3.3, "Data Cache Control," for more information.
29–30	—	Reserved, should be cleared.
31	NOOPTI	No-op the data cache touch instructions 0 The **dcbt** and **dcbtst** instructions are enabled 1 The **dcbt** and **dcbtst** instructions are no-oped internal to the e300 core

Table 7-4 shows how HID0[ECLK] and HID0[SBCLK] are used to configure the *clk_out* signal.

Table 7-4. Using HID0[ECLK] and HID0[SBCLK] to Configure *clk_out*

hreset	ECLK	SBCLK	***clk_out***
Asserted	x	x	Bus clock (small pulse for every rising edge of sysclk)
Negated	0	0	Clock output off
	0	1	Core clock/2
	1	0	Core clock
	1	1	Bus clock

Table 7-5 shows the bit definitions for HID1.

Table 7-5. HID1 Bit Descriptions

Bits	Name	Description
0	PC0	PLL configuration bit 0 (read-only)
1	PC1	PLL configuration bit 1 (read-only)
2	PC2	PLL configuration bit 2 (read-only)
3	PC3	PLL configuration bit 3 (read-only)
4	PC4	PLL configuration bit 4 (read-only)
5	PC5	PLL configuration bit 5 (read-only)
6	PC6	PLL configuration bit 6 (read-only)

Table 7-5. HID1 Bit Descriptions (continued)

Bits	Name	Description
7–31	—	Reserved, should be cleared

Note: The clock configuration bits reflect the state of the *pll_cfg[0:6]* signals.

Table 7-6 shows the bit definitions for HID2.

Table 7-6. e300HID2 Bit Descriptions

Bits	Name	Description
0–3	—	Reserved, should be cleared.
4	LET	True little-endian. This bit enables true little-endian mode operation for instruction and data accesses. This bit is set to reflect the state of the *tle* signal at the negation of *hreset*. This bit is used in conjunction with MSR[LE] to determine the endian mode of operation. 0 No function 1 True little-endian mode, when MSR[LE] = 1 Changing the value of this bit during normal operation is not recommended
5	IFEB	Instruction fetch burst extension. This bit enables the instruction fetch burst extension. 0 Instruction fetch burst extension disabled 1 Instruction fetch burst extension enabled
6	—	Reserved, should be cleared.
7	MESISTATE	MESI state enable. This bit enables the four-state MESI cache coherency protocol. 0 MESI disabled. The data cache uses a three-state MEI coherency protocol. 1 MESI enabled. The data cache uses a four-state MESI protocol.
8	IFEC	Instruction fetch cancel extension. This bit enables the instruction fetch cancel extension. 0 Instruction fetch cancel extension disabled 1 Instruction fetch cancel extension enabled
9	EBQS	Enable BIU queue sharing. This bit enables data cache queue sharing. 0 Data cache queue sharing disabled 1 Data cache queue sharing enabled
10	EBPX	Enable BIU pipeline extension.This bit enables the bus interface unit pipeline extension. 0 BIU pipeline extension disabled; 1 level pipeline 1 BIU pipeline extension enabled; 1-1/2 level pipeline
11–12	—	Reserved for e300c1, should be cleared.
11	ELRW	Enable weighted LRU. This bit enables the use of an adjusted (weighted) LRU. 0 Normal operation. 1 The dcbt, dcbtst, and dcbz instructions use and adjusted (weighted) LRU such that they always select and replace the lowest unlocked way in the data cache.
12	NOKS	No kill for snoop. This bit enables the forcing of kill-type snoops to flush data instead of killing it. 0 Normal operation. 1 Forces write-with-kill snoops to flush instead of kill (snoop can never kill data).
13	HBE	High BAT enable. Regardless of the setting of HID2[HBE], these BATs are accessible by **mfspr** and **mtspr**. 0 IBAT[4–7] and DBAT[4–7] are disabled 1 IBAT[4–7] and DBAT[4–7] are enabled
14–15	—	Reserved, should be cleared.

Table 7-6. e300HID2 Bit Descriptions (continued)

Bits	Name	Description
16–18	IWLCK[0–2]	Instruction cache way-lock. Useful for locking blocks of instructions into the instruction cache for time-critical applications that require deterministic behavior. 000 no ways locked 001 way 0 locked 010 way 0 through way 1 locked 011 way 0 through way 2 locked 100 way 0 through way 3 locked in and e300c4. 101 way 0 through way 4 locked in and e300c4. 110 way 0 through way 5 locked in and e300c4. 111 way 0 through way 6 locked in and e300c4. Setting HID0[ILOCK] will lock all ways.
19	ICWP	Instruction cache way protection. Used to protect locked ways in the instruction cache from being invalidated. 0 Instruction cache way protection disabled 1 Instruction cache way protection enabled
20–23	—	Reserved, should be cleared.
24–26	DWLCK[0–2]	Data cache way-lock. Useful for locking blocks of data into the data cache for time-critical applications where deterministic behavior is required. 000 no ways locked 001 way 0 locked 010 way 0 through way 1 locked 011 way 0 through way 2 locked 100 way 0 through way 3 locked in e300c4. 101 way 0 through way 4 locked in e300c4. 110 way 0 through way 5 locked in e300c4. 111 way 0 through way 6 locked in e300c4. Setting HID0[DLOCK] will lock all ways.
27–31	—	Reserved, should be cleared.

7.4.2 Instruction Set and Addressing Modes

The following sections describe the PowerPC instruction set and addressing modes in general.

7.4.2.1 PowerPC Instruction Set and Addressing Modes

All PowerPC instructions are encoded as single-word (32-bit) opcodes. Instruction formats are consistent among all instruction types, permitting efficient decoding to occur in parallel with operand accesses. This fixed instruction length and consistent format simplifies instruction pipelining.

The PowerPC instructions are divided into the following categories:

- Integer instructions—These include computational and logical instructions.
  - Integer arithmetic instructions
  - Integer compare instructions
  - Integer logical instructions
  - Integer rotate and shift instructions
- Floating-point instructions—These include floating-point computational instructions, as well as instructions that affect the FPSCR.
  - Floating-point arithmetic instructions
  - Floating-point multiply/add instructions
  - Floating-point rounding and conversion instructions
  - Floating-point compare instructions
  - Floating-point status and control instructions
- Load/store instructions—These include integer and floating-point load and store instructions.
  - Integer load and store instructions
  - Integer load and store multiple instructions
  - Floating-point load and store
  - Primitives used to construct atomic memory operations (**lwarx** and **stwcx.** instructions)
- Flow control instructions—These include branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow.
  - Branch and trap instructions
  - Condition register logical instructions
- Processor control instructions—These instructions are used for synchronizing memory accesses and management of caches, TLBs, and the segment registers.
  - Move to/from SPR instructions
  - Move to/from MSR
  - Move to/from PMR
  - Synchronize
  - Instruction synchronize

- Memory control instructions—These instructions provide control of caches, TLBs, and segment registers.
  - — Supervisor-level cache management instructions
  - — Translation lookaside buffer management instructions. Note that there are additional implementation-specific instructions.
  - — User-level cache instructions
  - — Segment register manipulation instructions
- The e300 core implements the following instructions which are defined as optional by the PowerPC architecture:
  - — Floating Select (**fsel**)
  - — Floating Reciprocal Estimate Single-Precision (**fres**)
  - — Floating Reciprocal Square Root Estimate (**frsqrte**)
  - — Store Floating-Point as Integer Word (**stfiwx**)

Note that this grouping of instructions does not indicate the execution unit that executes a particular instruction or group of instructions.

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision (one word) and double-precision (one double word) floating-point operands. The PowerPC architecture uses instructions that are 4 bytes long and word-aligned. It provides for byte, half-word, and word operand loads and stores between memory and a set of 32 GPRs. It also provides for word and double-word operand loads and stores between memory and a set of 32 FPRs.

Computational instructions do not modify memory. To use a memory operand in a computation and then modify the same or another memory location, the memory contents must be loaded into a register, modified, and then written back to the target location with distinct instructions.

The core follows the program flow when it is in the normal execution state. However, the flow of instructions can be interrupted directly by the execution of an instruction or by an asynchronous event. Either kind of interrupt may cause one of several components of the system software to be invoked.

7.4.2.2 Implementation-Specific Instruction Set

The e300 core instruction set is defined as follows:

- The core provides hardware support for all 32-bit PowerPC instructions.
- The core provides two implementation-specific instructions used for software table search operations following TLB misses:
  - — Load Data TLB Entry (**tlbld**)
  - — Load Instruction TLB Entry (**tlbli**)
- The core implements the following instruction which is added to support critical interrupts (also supported on the G2_LE). This is a supervisor-level, context synchronizing instruction.
  - — Return from Critical Interrupt (**rfci**)

- The core implements the following instruction which is added to support easy start-up initialization or reloading of the instruction cache.
  - Instruction Cache Block Touch (**icbt**)
- The core provides the following performance monitor instructions:
  - Move to Performance Monitor Register (**mtpmr**)
  - Move from Performance Monitor Register (**mfpmr**)

7.4.3 Cache Implementation

The following sections describe the general cache characteristics as implemented in the PowerPC architecture and the core implementation.

7.4.3.1 PowerPC Cache Characteristics

The PowerPC architecture does not define hardware aspects of cache implementations. The e300 core controls the following memory access modes on a page or block basis:

- Write-back/write-through mode
- Caching-inhibited mode
- Memory coherency

Note that in the core, a cache block is defined as eight words. The VEA defines cache management instructions that provide a means by which the application programmer can affect the cache contents.

7.4.3.2 Implementation-Specific Cache Organization

The e300c4 provides independent, 32-Kbyte, eight-way, set-associative, instruction and data caches.s The caches are physically addressed, and the data cache can operate in either write-back or write-through mode as specified by the PowerPC architecture.

The data cache is configured as 128 sets of 8 blocks each on the e300c4. Each block consists of 32 bytes, 2 state bits, and an address tag. The two state bits implement the three-state MEI (modified/exclusive/invalid) protocol. Each block contains eight 32-bit words. Note that the PowerPC architecture defines the term ‘block’ as the cacheable unit. For the core, the block size is equivalent to a cache line. A block diagram of the data cache organization is shown in Figure 7-3.

The instruction cache is configured as 128 sets of 8 blocks each on the e300c4. Each block consists of 32 bytes, an address tag, and a valid bit. The instruction cache may not be written to, except through a block fill operation. In the e300 core, the instruction cache is blocked only until the critical load completes. The e300 core supports instruction fetching from other instruction cache lines following the forwarding of the critical-first-double-word of a cache line load operation. Successive instruction fetches from the cache line being loaded are forwarded, and accesses to other instruction cache lines can proceed during the cache line load operation. The instruction cache is not snooped, and cache coherency must be maintained by software. A fast hardware invalidation capability is provided to support cache maintenance. The organization of the instruction cache for the e300c4 is very similar to the data cache shown in Figure 7-3.

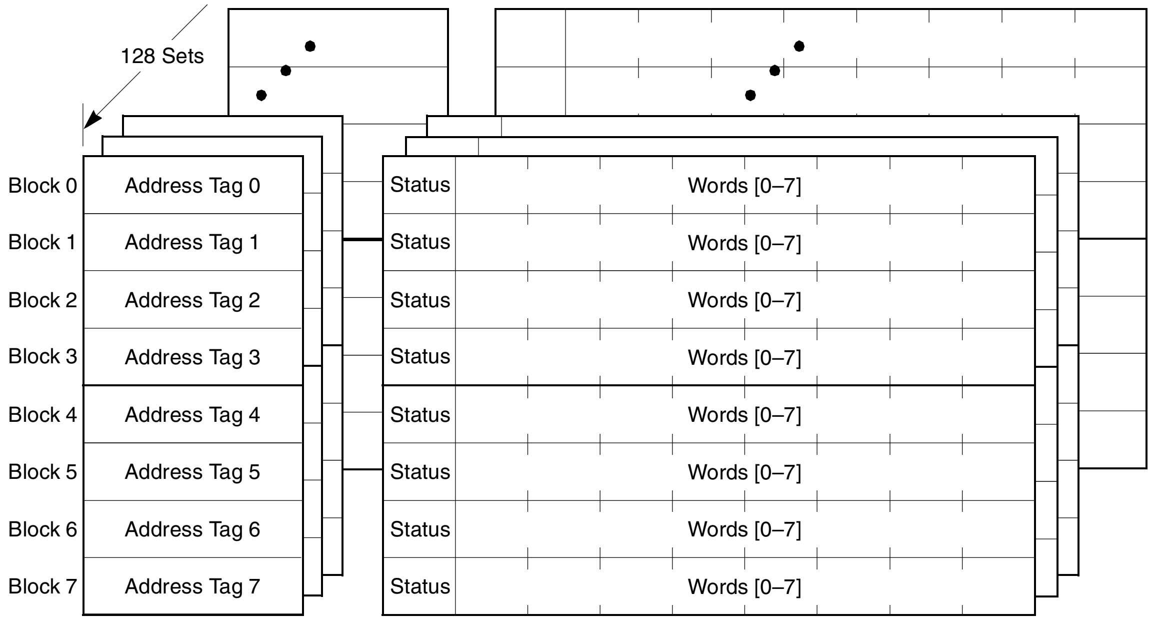


Figure 7-3. e300c4 Data Cache Organization

Each cache block contains eight contiguous words from memory that are loaded from an 8-word boundary (that is, bits A[27–31] of the effective addresses are zero); thus, a cache block never crosses a page boundary. Misaligned accesses across a page boundary can incur a performance penalty.

The e300 core cache blocks are loaded in four beats of 64 bits each on the 64-bit data bus. The burst load is performed as critical-double-word-first. The data cache is blocked to internal accesses until the load completes; the instruction cache allows sequential fetching during a cache block load. In the core, the critical-double-word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to load delays.

To ensure coherency among caches in a multiprocessor (or multiple caching-device) implementation, the core implements the MEI protocol during normal operation of the data cache. The new data cache MESI extension supports the additional fourth cache coherency shared state for the data cache. To support this feature, the shared signal, *shd*, has been added to the bus interface. Although the MESI protocol is supported by the e300 core, it is not implemented on MPC8379E. The following four states indicate the state of the cache block:

- Modified—The cache block is modified with respect to system memory; that is, data for this address is valid only in the cache and not in system memory.
- Exclusive—This cache block holds valid data that is identical to the data at this address in system memory. No other cache has this data.
- Shared—Only available if HID2[MESISTATE] register bit is set. The address block is valid in the cache and in at least one other cache. This block is always consistent with system memory. That is, the shared state is shared-unmodified; there is no shared-modified state. Although the MESI protocol is supported by the e300 core, it is not implemented on MPC8379E.

- Invalid—This cache block does not hold valid data.

Cache coherency is enforced by on-chip bus snooping logic. Because the e300 core data cache tags are single-ported, a simultaneous load/store and snoop access represents a resource contention. The snoop access is given first access to the tags. The load or store then occurs on the clock following the snoop.

Parity is now integrated into both instruction and data cache memory. A machine check interrupt is now taken upon the detection of an instruction or data cache parity error. Parity is checked whenever valid data is returned from the instruction or data cache for a cache hit or whenever valid data is read out of the cache for a castout or snoop-push operation.

7.4.3.3 Instruction and Data Cache Way-Locking

The e300 core implements instruction and data cache way-locking, which guarantees that certain memory accesses will hit in the cache. This provides deterministic access times for those accesses.

7.4.4 Interrupt Model

This section describes the PowerPC interrupt model and the e300 core implementation specifically.

7.4.4.1 PowerPC Interrupt Model

The PowerPC interrupt mechanism allows the core to change to supervisor state as a result of external signals, errors, or unusual conditions arising in the execution of instructions. The conditions that can cause interrupts are called exceptions. When interrupts occur, information about the state of the core is saved to certain registers and the core begins execution at an address (interrupt vector) predetermined for each interrupt type. Interrupts are processed in supervisor mode.

Some interrupts, such as program interrupts, can be triggered by a broad range of exception conditions. Other interrupts, such as the decrementer interrupt, have only a single exception condition. Although multiple exception conditions can map to a single interrupt vector, a more specific condition may be determined by examining a register associated with the interrupt—for example, the DSISR and the FPSCR. Additionally, some exception conditions can be explicitly enabled or disabled by software.

The PowerPC architecture requires that interrupts be handled in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are presented strictly in order. When an instruction-caused interrupt is recognized, any unexecuted instructions that appear earlier in the instruction stream, including any that have not yet entered the execute stage, are required to complete before the interrupt is taken. Any interrupts caused by those instructions are handled first. Likewise, asynchronous, precise interrupts are recognized when they occur, but are not handled until the instruction currently in the completion stage successfully completes execution or generates an interrupt, and the completed store queue is emptied.

Unless a catastrophic condition causes a system reset or machine check interrupt, only one interrupt is handled at a time. If, for example, a single instruction encounters multiple interrupt conditions, those conditions are handled sequentially. After the interrupt handler completes, the instruction execution continues until the next interrupt condition is encountered. However, in many cases there is no attempt to re-execute the instruction. This method of recognizing and handling interrupts sequentially guarantees that interrupts are recoverable.

To prevent the program state from being lost due to a system reset, a machine check interrupt, or an instruction-caused interrupt in the interrupt handler, interrupt handlers should save the information stored in SRR0 and SRR1 early and before enabling external interrupts.

The PowerPC architecture supports four types of interrupts:

- Synchronous, precise—These are caused by instructions. All instruction-caused interrupts are handled precisely; that is, the machine state at the time the interrupt occurs is known and can be completely restored. This means that (excluding the trap and system call interrupts) the address of the faulting instruction is provided to the interrupt handler and neither the faulting instruction nor subsequent instructions in the code stream will complete execution before the interrupt is taken. Once the interrupt is processed, execution resumes at the address of the faulting instruction (or at an alternate address provided by the interrupt handler). When an interrupt is taken due to a trap or system call instruction, execution resumes at an address provided by the handler.
- Synchronous, imprecise—The PowerPC architecture defines two imprecise floating-point exception modes: recoverable and nonrecoverable. Even though the core provides a means to enable the imprecise modes, it implements these modes identically to the precise mode (that is, all enabled floating-point exceptions are always precise on the core).
- Asynchronous, maskable—The external system management interrupt (SMI) and decrementer interrupts are maskable, asynchronous interrupts. When these interrupts occur, their handling is postponed until the next instruction and any of its associated interrupts complete execution. If there are no instructions in the execution units, the interrupt is taken immediately upon determination of the correct restart address (for loading SRR0).
- Asynchronous, nonmaskable—The system reset and the machine check interrupt are nonmaskable, asynchronous interrupts. They may not be recoverable, or they may provide a limited degree of recoverability. All interrupts report recoverability through MSR[RI].

7.4.4.2 Implementation-Specific Interrupt Model

As specified by the PowerPC architecture, all interrupts can be described as either precise or imprecise and either synchronous or asynchronous. Asynchronous interrupts (some of which are maskable) are caused by events external to the processor’s execution; synchronous interrupts, which are all handled precisely by the e300 core, are caused by instructions. A system management interrupt is an implementation-specific interrupt. The interrupt classes are shown in Table 7-7.

Table 7-7. Interrupt Classifications

Synchronous/Asynchronous	Precise/Imprecise	Interrupt Type
Asynchronous, nonmaskable	Imprecise	Machine check System reset
Asynchronous, maskable	Precise	External interrupt Decrementer System management interrupt Critical interrupt
Synchronous	Precise	Instruction-caused interrupts

Although interrupts have other characteristics, such as whether they are maskable, the distinctions shown in Table 7-7 define categories of interrupts that the core handles uniquely. Note that Table 7-7 includes no synchronous, imprecise instructions. While the PowerPC architecture supports imprecise handling of floating-point exceptions, the core implements floating-point exception modes as precise.

The e300 core interrupts and exception conditions that cause them are listed in Table 7-8.

Table 7-8. Exceptions and Interrupts

Interrupt Type	Vector Offset (hex)	Exception Conditions
Reserved	00000	—
System reset	00100	Caused by the assertion of either *hreset*.
Machine check	00200	Caused by the assertion of the *tea* signal during a data bus transaction, assertion of *mcp*, an address or data parity error, or an instruction or data cache parity error. Note that the e300 has SRR1 register values that are different from the G2/G2_LE cores' when a machine check occurs.
DSI	00300	Determined by the bit settings in the DSISR, listed as follows: 1 Set if the translation of an attempted access is not found in the primary hash table entry group (HTEG), or in the rehashed secondary HTEG, or in the range of a DBAT register; otherwise cleared 4 Set if a memory access is not permitted by the page or DBAT protection mechanism; otherwise cleared 6 Set for a store operation and cleared for a load operation 9 Set if a data address breakpoint interrupt occurs when the data [0–28] in the DABR or DABR2 matches the next data access (load or store instruction) to complete in the completion unit. The different breakpoints are enabled as follows: • Write breakpoints enabled when DABR[30] is set • Read breakpoints enabled when DABR[31] is set
ISI	00400	Caused when an instruction fetch cannot be performed for any of the following reasons: • The effective (logical) address cannot be translated. That is, there is a page fault for this portion of the translation, so an ISI interrupt must be taken to load the PTE (and possibly the page) into memory. • The fetch access violates memory protection (indicated by SRR1[4] set). If the key bits (Ks and Kp) in the segment register and the PP bits in the PTE are set to prohibit read access, instructions cannot be fetched from this location.

Table 7-8. Exceptions and Interrupts (continued)

Interrupt Type	Vector Offset (hex)	Exception Conditions
External interrupt	00500	Caused when MSR[EE] = 1 and the *int* signal is asserted.
Alignment	00600	Caused when the core cannot perform a memory access for any of the reasons described below: • The operand of a floating-point load or store instruction is not word-aligned. • The operands of **lmw**, **stmw**, **lwarx**, and **stwcx.** instructions are not aligned. • The instruction is **lswi**, **lswx**, **stswi**, **stswx**, and the core is in little-endian mode. Note that PowerPC little-endian mode is not supported on the e300 core. • The operand of **dcbz** is in memory that is write-through-required or caching-inhibited.
Program	00700	Caused by one of the following exception conditions, which correspond to bit settings in SRR1 and arise during execution of an instruction. Floating-point enabled exception—A floating-point enabled exception condition is generated when the following condition is met: (MSR[FE0] | MSR[FE1]) and FPSCR[FEX] is 1. • FPSCR[FEX] is set by the execution of a floating-point instruction that causes an enabled exception or by the execution of one of the Move to FPSCR instructions that results in both an exception condition bit and its corresponding enable bit being set in the FPSCR. • Illegal instruction—An illegal instruction program interrupt is generated when execution of an instruction is attempted with an illegal opcode or illegal combination of opcode and extended opcode fields (including PowerPC instructions not implemented in the core), or when execution of an optional instruction not provided in the core is attempted (these do not include those optional instructions that are treated as no-ops). • Privileged instruction—A privileged instruction program interrupt is generated when the execution of a privileged instruction is attempted and the MSR register user privilege bit, MSR[PR], is set. In the e300 core, this interrupt is generated for **mtspr** or **mfspr** with an invalid SPR field if SPR[0] = 1 and MSR[PR] = 1. This may not be true for all cores that implement the PowerPC architecture. • Trap—A trap type program interrupt is generated when any of the conditions specified in a trap instruction are met.
Floating-point unavailable	00800	Caused by an attempt to execute a floating-point instruction (including floating-point load, store, and move instructions) when the floating-point available bit (MSR[FP]) is cleared.
Decrementer	00900	Occurs when DEC[0] changes from 0 to 1. This interrupt is enabled with MSR[EE].
Critical interrupt	00A00	Taken when *cint* is asserted and MSR[CE] = 1.
Reserved	00B00–00BFF	—
System call	00C00	Occurs when a System Call (**sc**) instruction is executed.
Trace	00D00	Taken when MSR[SE] =1 or when the currently completing instruction is a branch and MSR[BE] =1.
Reserved	00E00	The e300 core does not generate an interrupt to this vector. Other devices may use this vector for floating-point assist interrupts.
Performance monitor	00F00	Caused when a configured PM counter using the pm_event_in to transition overflows.
Instruction translation miss	01000	Caused when the effective address for an instruction fetch cannot be translated by the ITLB.
Data load translation miss	01100	Caused when the effective address for a data load operation cannot be translated by the DTLB.

Table 7-8. Exceptions and Interrupts (continued)

Interrupt Type	Vector Offset (hex)	Exception Conditions
Data store translation miss	01200	Caused when the effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs and the change bit in the PTE must be set due to a data store operation.
Instruction address breakpoint	01300	Occurs when the address (bits 0–29) in the IABR matches the next instruction to complete in the completion unit, and IABR[30] is set. Note that the e300 core also implements IABR2, which functions identically to IABR.
System management interrupt	01400	Caused when MSR[EE] = 1 and the $\overline{smi}$ input signal is asserted.
Reserved	01500–02FFF	—

7.4.5 Memory Management

The following sections describe the memory management features of the PowerPC architecture and the e300 core implementation, respectively.

7.4.5.1 PowerPC Memory Management

The primary functions of the MMU are to translate logical (effective) addresses to physical addresses for memory accesses and to provide access protection on blocks and pages of memory.

The core generates two types of accesses that require address translation: instruction accesses and data accesses to memory generated by load and store instructions.

The PowerPC MMU and interrupt model support demand-paged virtual memory. Virtual memory management permits execution of programs larger than the size of physical memory; demand-paged implies that individual pages are loaded into physical memory from system memory only when they are first accessed by an executing program.

The hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of two, and its starting address is a multiple of its size.

The page table contains a number of page-table entry groups (PTEGs). A PTEG contains eight page-table entries (PTEs) of 8 bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

Address translations are enabled by setting bits in the MSR—MSR[IR] enables instruction address translations, and MSR[DR] enables data address translations.

7.4.5.2 Implementation-Specific Memory Management

The instruction and data memory management units in the e300 core provide 4 Gbytes of logical address space accessible to supervisor and user programs with a 4-Kbyte page size and 256-Mbyte segment size. Block sizes range from 128 Kbytes to 256 Mbytes and are software selectable. In addition, the core uses an interim 52-bit virtual address and hashed page tables for generating 32-bit physical addresses. The MMUs in the e300 core rely on the interrupt processing mechanism for the implementation of the paged virtual memory environment and for enforcing protection of designated memory areas.

Instruction and data TLBs provide address translation in parallel with the on-chip cache access, incurring no additional time penalty in the event of a TLB hit. A TLB is a cache of the most recently used page table entries. Software is responsible for maintaining the consistency of the TLB with memory. The core TLBs are 64-entry, two-way, set-associative caches that contain instruction and data address translations. The core provides hardware assist for software table search operations through the hashed page table on TLB misses. Supervisor software can invalidate TLB entries selectively.

For instructions and data that correspond to block address translation, the e300 core provides independent eight-entry BAT arrays. These entries define blocks that can vary from 128 Kbytes to 256 Mbytes. The BAT arrays are maintained by system software. HID2[HBE] is added to the e300 for enabling or disabling the four additional pairs of BAT registers. However, regardless of the setting of HID2[HBE], these BATs are accessible by **mfspr** and **mtspr**.

As specified by the PowerPC architecture, the hashed page table is a variable-sized data structure that defines the mapping between virtual page numbers and physical page numbers. The page table size is a power of two, and its starting address is a multiple of its size.

Also as specified by the PowerPC architecture, the page table contains a number of PTEGs. A PTEG contains 8 PTEs of 8 bytes each; therefore, each PTEG is 64 bytes long. PTEG addresses are entry points for table search operations.

7.4.6 Instruction Timing

The e300 core is a pipelined superscalar processor core. Because instruction processing is reduced into a series of stages, an instruction does not require all of the resources of an execution unit at the same time. For example, after an instruction completes the decode stage, it can pass on to the next stage, while the subsequent instruction can advance into the decode stage. This improves the throughput of the instruction flow. For example, it may take three cycles for a single floating-point instruction to execute, but if there are no stalls in the floating-point pipeline, a series of floating-point instructions can have a throughput of one instruction per cycle.

The core instruction pipeline has four major pipeline stages, described as follows:

- The fetch pipeline stage primarily involves retrieving instructions from the memory system and determining the location of the next instruction fetch. Additionally, if possible, the BPU decodes branches during the fetch stage and folds out branch instructions before the dispatch stage.
- The dispatch pipeline stage is responsible for decoding the instructions supplied by the instruction fetch stage and determining which of the instructions are eligible to be dispatched in the current cycle. In addition, the source operands of the instructions are read from the appropriate register file and dispatched with the instruction to the execute pipeline stage. At the end of the dispatch pipeline stage, the dispatched instructions and their operands are latched by the appropriate execution unit.
- In the execute pipeline stage, each execution unit with an instruction executes the selected instruction (perhaps over multiple cycles), writes the instruction's result into the appropriate rename register, and notifies the completion stage when the execution has finished. In the case of

an internal interrupt, the execution unit reports the interrupt to the completion/write-back pipeline stage and discontinues instruction execution until the interrupt is handled. The interrupt is not signaled until that instruction is the next to be completed. Execution of most floating-point instructions is pipelined within the FPU, allowing up to three instructions to execute in the FPU concurrently. The FPU pipeline stages are multiply, add, and round-convert. The LSU has two pipeline stages: the first stage, for effective address calculation and MMU translation, and the second, for accessing data in the cache.

- The complete/write-back pipeline stage maintains the correct architectural machine state and transfers the contents of the rename registers to the GPRs and FPRs as instructions are retired. If the completion logic detects an instruction causing an interrupt, all subsequent instructions are canceled, their execution results in rename registers are discarded, and instructions are fetched from the correct instruction stream.

A superscalar processor core issues multiple, independent instructions into multiple pipelines, allowing instructions to execute in parallel. The e300c1 core has independent execution units for: integer instructions, floating-point instructions, branch instructions, load/store instructions, and system register instructions.The e300c4 provides two IUs, which improves the throughput of integer instructions. The e300c4 provides two integer units for greater integer instruction throughput along with enhanced multipliers in each IU that reduce the multiply instruction latency to a maximum of two cycles. The IU and the FPU each have dedicated register files for maintaining operands (GPRs and FPRs, respectively), allowing integer and floating-point calculations to occur simultaneously without interference.

The core provides support for single-cycle store, and it provides an adder/comparator in the system register unit that allows the dispatch and execution of multiple integer add and compare instructions on each cycle.

Because the PowerPC architecture can be applied to such a wide variety of implementations, instruction timing among processor cores varies accordingly.

7.4.7 Core Interface

The core interface is specific for each processor core implementation.

The MPC8379E contains an internal coherent system bus (CSB) that interfaces the processor core to the peripheral logic. This internal bus is very similar in function to the external 60x bus interface on the MPC603e. In the case of the MPC8379E, the CSB system logic decodes e300-initiated transactions and directs all accesses to the appropriate interface.

The e300 core can operate at a variety of frequencies allowing the designer to trade off performance for power consumption. The processor core is clocked from a separate PLL, which is referenced to the CSB frequency. This allows the processor core and the peripheral logic to operate at different frequencies.

The e300 core provides a versatile core interface that allows for a wide range of implementations. The interface includes a 32-bit address bus, a 64-bit data bus, and 56 control and information signals (see Figure 7-4). The core interface allows for address-only transactions, as well as address and data transactions. The core control and information signals include the address arbitration, address start, address transfer, transfer attribute, address termination, data arbitration, data transfer, data termination, and core state signals. Test and control signals provide diagnostics for selected internal circuits.

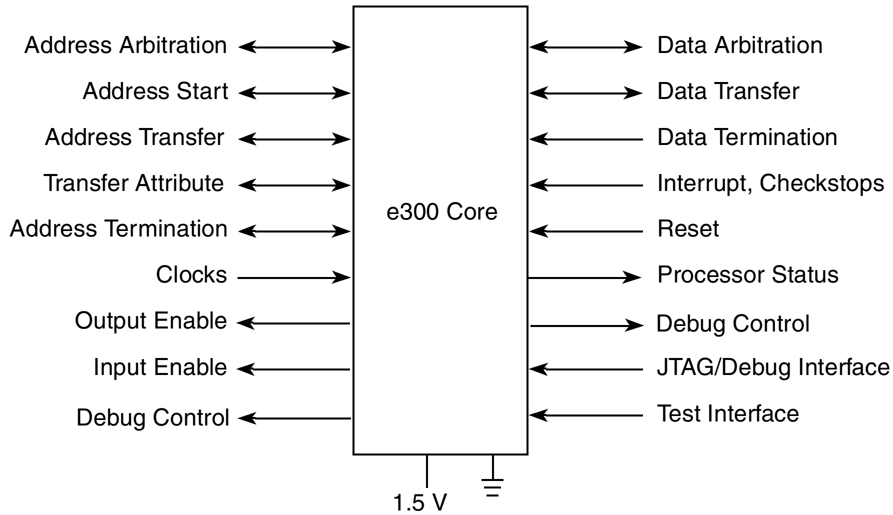


Figure 7-4. Core Interface

The core interface supports bus pipelining, allowing the address tenure of one transaction to overlap the data tenure of another. The extent of the pipelining depends on external arbitration and control circuitry. Similarly, the core supports split-bus transactions for systems with multiple potential bus masters—one device can have mastership of the address bus while another has mastership of the data bus. Allowing multiple bus transactions to occur simultaneously increases the available bus bandwidth for other activity and, as a result, improves performance.

The core clocking structure allows the bus to operate at integer multiples of the core cycle time.

The following sections describe the core bus support for memory operations. Note that some signals perform different functions depending on the addressing protocol used.

7.4.7.1 Memory Accesses

The e300 core CSB is a 64-bit data bus.

With a 64-bit CSB, memory accesses allow transfer sizes of 8, 16, 24, 32, 40, 48, 56, or 64 bits in one bus clock cycle. Data transfers occur in either single-beat transactions or four-beat burst transactions. Single-beat transactions are caused by noncached accesses that access memory directly (that is, reads and writes when caching is disabled, caching-inhibited accesses, and stores in write-through mode). Four-beat burst transactions, which always transfer an entire cache block (32 bytes), are initiated when a line is read from or written to memory.

7.4.7.2 Signals

The e300 core signals are grouped as follows:

- Interrupts/Resets—These signals include the external interrupt signal ($\overline{int}$), critical interrupt signal ($\overline{cint}$), checkstop signals, performance monitor signal (*pm_event_in*) via the PM counters, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the core.

- JTAG/debug interface signals—The JTAG (based on the IEEE 1149.1 standard) interface and debug unit provides a serial interface to the system for performing monitoring and boundary tests. Two additional signals are added to the e300 core to allow observation of the internal clock state of the core (*stopped*) and to allow the external input to force the core into a halted state (*ext_halt*).
- Core status and control—These signals include the memory reservation signal, machine quiesce control signals, time base/decrementer clock base enable signal, and the *tlbisync* signal.
- Clock control—These signals provide for system clock input and frequency control.
- Test interface signals—Signals like address matching, combinational matching, and watchpoint are used in the core for production testing.
- Transfer attribute signals—These signals provide information about the type of transfer, such as the transfer size and whether the transaction is bursted, write-through, or cache-inhibited.

7.4.8 Debug Features

Some new debug features are specific to the e300 core. Accesses to the debug facilities are available only in supervisor mode by using the **mtspr** and **mfspr** instructions. The e300 provides the following additional feature in the JTAG/debug interface: Inclusion of breakpoint status and control pins: *stopped* and *ext_halt*.

7.4.8.1 Breakpoint Signaling

The breakpoint signaling provided on the e300 core allows observability of breakpoint matches external to the core. The *iabr*, *iabr2*, *dabr*, and *dabr2* breakpoint signals are asserted for at least one bus clock cycle when the respective breakpoint occurs. The status of the run state of the e300 core is indicated by the *stopped* pin. An asynchronous external breakpoint can be asserted to the e300 core using the *ext_halt* pin:

- When DBCR and IBCR are configured for an OR combinational signal type, the breakpoint signals *iabr*, *iabr2* and *dabr*, *dabr2* reflect their respective breakpoints.
- When the DBCR and IBCR are configured for AND combinational signal type, only the *iabr2* and *dabr2* breakpoint signals are asserted after the AND condition is met (that is, both instruction breakpoints occurred or both data breakpoints occurred).
- When the core_stopped pin is asserted, the e300 core has entered a stopped state and all internal clocking has stopped, indicating that a hardware debug event has occurred.
- The *ext_halt* input pin can be used to force the core into halted state. The halted state may be a hardstop, conditional upon the HARDSTOP condition being set through the JTAG/debug interface

7.5 Differences Between Cores

The e300 core has similar functionality to the G2_LE core. Table 7-9 describes the differences between the G2_LE and the e300.

Table 7-9. Differences Between e300 and G2_LE Cores

e300 Core	G2_LE Core	Impact
New HID0 bits	—	The e300 core has a new HID0 bit defined to enable cache parity error reporting (ECPE).
New HID1 bits	—	The e300 core has new HID1 bits defined to extend the number of PLL configuration signals to seven (PC5, PC6).
New HID2 bits	—	The e300 core has new HID2 bits defined to support instruction fetch bursting (IFEB), MESI coherency protocol (MESI), instruction fetch cancels (IFEC), data cache queue sharing (EBQS), pipelining extension (EBPX), additional cache way locking (IWLCK and DWLCK), and instruction cache way protection (ICWP).
New PVR register value	—	The processor version register values differ.
New IBCR and DBCR bits	—	The e300 core has new IBCR[IABRSTAT, IABR2STAT] and DBCR[DABR1STAT, DABR2STAT] fields to provide instruction and data address breakpoint status.
—	16-Kbyte, four-way, set-associative, instruction and data caches	Some e300 cores may have different cache sizes than the G2_LE. See the *e300 PowerPC Core Reference Manual* for detailed information.
L1 cache parity	—	The e300 core supports parity for both instruction and data caches; the G2_LE does not support cache parity.
MEI or MESI coherency protocols	MEI protocol only	The e300 supports two coherency protocols: MEI and MESI; the G2_LE only supports the MEI protocol.
Instruction cancel extension	—	The e300 instruction cancel mechanism improves utilization of instruction cache by supporting 'hits-under-cancels' and 'misses-under-cancels'; the G2_LE requires the cancel to complete before new instruction fetches can begin.
Instruction fetch bursts to caching-inhibited space	Single-beat instruction fetches to caching-inhibited space	The e300's instruction fetch burst extension allows all caching-inhibited instruction fetches to be performed on the bus as burst transactions, even though the instructions are not cached. This improves performance for instruction space that is caching-inhibited, because up to eight instructions are returned with one bus operation. The G2_LE core must use single-beat instruction fetches for caching-inhibited space, returning only two instructions per bus operation.
Instruction cache way protection	—	The e300 core can protect locked ways in the instruction cache from invalidation; the G2_LE does not support instruction cache way protection.

Table 7-9. Differences Between e300 and G2_LE Cores (continued)

e300 Core	G2_LE Core	Impact
Data cache queue sharing	—	The e300 has a new data cache queue sharing extension that allows the two burst-write queues in the bus unit to be used interchangeably for cache replacements and snoop pushes. Thus, the data cache can support two outstanding cache replacements or two outstanding snoop push operations on the bus at any given time.
icbt instruction	—	The e300 supports a new instruction cache block touch instruction that facilitates preloading the instruction cache before locking; the G2_LE core requires speculatively fetching instructions before locking the instruction cache.
1-1/2-level bus pipelining	1-level bus pipelining	For the e300, a new transaction can complete an address tenure when the previous transaction has been granted the data bus; for the G2_LE, a new transaction must wait until the previous data tenure has completed before completing its address tenure.
PowerPC little-endian not supported	PowerPC little-endian supported	PowerPC little-endian will not be supported in the e300 core, although true little-endian will be fully supported.
Data retry mode removed	Data retry mode available	$\overline{drtry}$ and *drtrymode* will no longer be supported on the e300 and future versions.
External control instructions removed	External control instructions available	The **eciwx** and **ecowx** instruction pair will not be supported on the e300 core. These are optional instructions in the PowerPC architecture.
Reduced pin mode removed	Reduced pin mode available	Reduced pinout mode and the signal *redpinmode* will not be supported in the e300 core.

Chapter 8
Integrated Programmable Interrupt Controller (IPIC)

This chapter describes the integrated programmable interrupt controller (IPIC), including a definition of the external signals and their functions. Also, the configuration, control, and status registers are described in this chapter. Note that individual chapters in this reference manual describe specific initialization aspects for each individual block.

8.1 Introduction

This chapter describes the IPIC interrupt protocol, various types of interrupt sources controlled by the IPIC unit, and the IPIC registers with some programming guidelines. The programming model is similar to the interrupt controller of the MPC8260. The interrupt controller provides interrupt management that is responsible for receiving hardware-generated interrupts from different sources (both internal and external). It also prioritizes and delivers the interrupts to the CPU for servicing. The IPIC prioritizes and manages interrupts from the following controller units:

- DDR memory controller (DDR)
- Enhanced local bus memory controller (eLBC)
- PCI
- Four-channel DMA controller (DMA)
- Message unit (MU)
- Dual three-speed Ethernet controllers (eTSEC1 and eTSEC2)
- DUART communication module (DUART)
- USB 2.0 dual role controller (USB DR)
- Security engine (SEC)
- System bus arbiter (SBA)
- Periodic interval timer (PIT)
- Real time clock timer (RTC ALR and RTC SEC)
- Eight global timers (GTM1–GTM8)
- Software watchdog timer (WDT)
- I^2C controllers (I^2C1 and I^2C2)
- SPI controller (SPI)
- Power management controller (PMC)
- Two general-purpose I/O controllers (GPIO1 and GPIO2)
- External pins ($\overline{IRQ}$[0:7])
- Serial ATA (SATA) controller (SATA1–4)

- PCI Express controllers (PCI Express1 and PCI Express2)
- eSDHC controller
- Message Shared Interrupt (MSI)

The interrupt sources controlled by the IPIC unit cause exceptions in the processor core. The internal interrupt (*int*) signal is the main interrupt output from the IPIC to the core and it causes the regular interrupt exception. The *cint* signal is the critical interrupt output from the IPIC to the processor core and causes the critical interrupt exception. The smi signal is the system management interrupt output from the IPIC to the processor core and causes the system management interrupt exception. The machine check exception is caused by the internal *mcp* signal generated by the IPIC, informing the host processor of error conditions, assertion of the external IRQ0 machine-check request (enabled when SEMSR[SIRQ0] = 1), and other conditions.

Figure 8-1 shows the relationship of the various functional blocks and external signals of the device to the IPIC unit.

The IPIC receives interrupt request signals from the following two sources:

- External to the integrated device
- Internal to the integrated device

The unit selects the highest priority interrupt from all current interrupts and forwards it to the internal processor core, or off-chip for external servicing.

The IPIC also manages an internal non-maskable machine-check processor (*mcp*) signal and the interrupt generated by the off-chip interrupt sources (IRQ[0:7]).

The interrupt router of the IPIC monitors the outputs of the internal configuration registers. When the priority is highest in one of the received interrupt signals, the IPIC sets the corresponding bit in one of the interrupt pending registers—system internal interrupt pending register (SIPNR)/system external interrupt pending register (SEPNR). If the interrupt is not masked, the IPIC asserts the *int* signal to indicate an interrupt request to the processor. When the processor is running the specific *int*, *cint*, or *smi* interrupt handler code, the processor must vectorize the external interrupt handler by explicitly (in software) reading the corresponding interrupt vector register (SIVCR, SCVCR or SMVCR). In response to this read, the IPIC unit returns the vector (associated with the interrupt source) to the interrupt handler routine. In addition, the handler can vectorize different branches of interrupt handling.

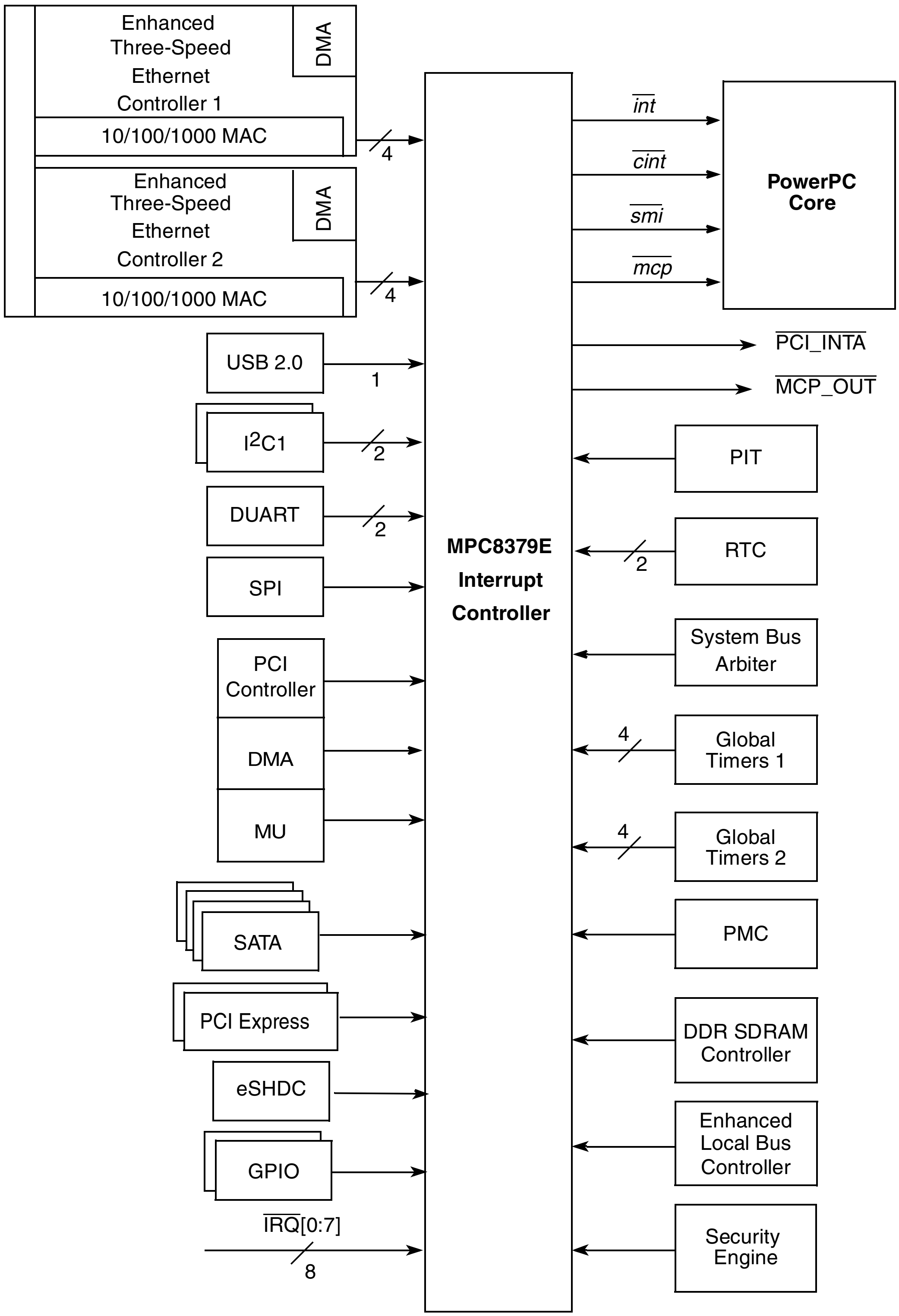


Figure 8-1. Interrupt Sources Block Diagram

The IPIC receives the following types of interrupts:

- External interrupt—triggered by the off-chip signals (IRQ*n*) listed in Table 8-1
- Internal interrupts—on-chip interrupts, triggered by the sources listed in Table 8-7 and Table 8-9
- External and internal non-maskable machine check conditions, signaled by the sources listed in Table 8-23 through *mcp*

The interrupt controller provides the ability to mask each interrupt source. Any source that can be caused by multiple events are also maskable.

When the IPIC receives an internal or external interrupt, its configuration register is checked to determine if it should be routed off-chip (to the external PCI_INTA) or serviced as a normal external interrupt by the processor core (through the *int* signal). As a third alternative, if the incoming interrupt has been configured as a critical or system management interrupt, the IPIC completes the processing of the interrupt by asserting *cint* or *smi* to the core. The assertion of the *cint* or *smi* signal to the core causes the interrupt to be serviced as a critical or a system management interrupt, respectively.

8.2 Features

The IPIC unit implements the following features:

- Functional and programming compatibility with the MPC8260 interrupt controller
- Support for external and internal discrete vectorized interrupt sources
- Support for external and internal non-maskable machine check conditions, signaled by *mcp*
- Support for programmable polarity of external interrupt signals.
- Programmable highest priority request (can be programmed to support a critical (*cint*) or system management interrupt (*smi*) type)
- Two programmable priority mixed groups of four on-chip and four external interrupt signals with two priority schemes for each group: grouped and spread
- Six programmable priority internal groups of eight on-chip interrupt signals with two priority schemes for each group: grouped and spread
- Two highest priority interrupts from each group can be programmed to support a critical or system management interrupt type
- External and internal interrupts directed to host processor
- Unique vector number for each interrupt source

8.3 Modes of Operation

The IPIC unit can operate in the core enable or core disable mode.

8.3.1 Core Enable Mode

In core enable mode, all internal interrupts (including those from the PCI block) are routed to and from the IPIC; the interrupts are sent to the PowerPC core. The DMA controller can optionally (depending on the programming of the DMA registers) steer its interrupt to the PCI host through the PCI_INTA signal.

In this mode all machine check interrupts are gathered by the IPIC unit and sent to the PowerPC core. If the device performs as a PCI host, the interrupts of the other PCI agents should be connected to the implementation's IRQx signals and treated like normal external interrupts (sent to the core).

8.3.2 Core Disable Mode

In core disable mode, all internal interrupts (including those from the PCI block) are routed to and from the IPIC, the interrupts are then sent through the PCI_INTA signal to the PCI host CPU. Note that the core interrupt signal is masked. The user should use in this mode only the *int* output interrupt type (should not use *cint* or *smi* output interrupt types) to read an updated SIVCR. (See Section 8.5.9, "System Internal Interrupt Control Register (SICNR)," and Section 8.5.14, "System External Interrupt Control Register (SECNR).")

In this mode, machine check interrupts are driven either on PCI_INTA or on MCP_OUT as level-sensitive interrupts. SERCR[MCPR] (see Section 8.5.17, "System Error Control Register (SERCR)") controls which external signal is used.

8.4 External Signal Description

The following sections provide an overview and detailed descriptions of the IPIC signals.

8.4.1 Overview

The device has 8 distinct external interrupt request input signals (IRQ[0:7]) and one interrupt request output signal (PCI_INTA). The IPIC interface signals are defined in Table 8-1.

Table 8-1. IPIC Signal Properties

Name	Port	Function	I/O	Reset	Requires Pull Up
IRQ[0:7]	IRQ[0:7]	External interrupts	I	—	Yes
PCI_INTA	PCI_INTA	Interrupt request output	O	Z	Yes
MCP_OUT	MCP_OUT	Interrupt request output	O	Z	Yes

8.4.2 Detailed Signal Descriptions

Table 8-2 provides detailed descriptions of the external IPIC signals.

Table 8-2. IPIC External Signals—Detailed Signal Descriptions

Signal	I/O	Description	
IRQ[0:7]	I	Interrupt request 0–7. The sense (level or edge) of each of these signals is programmable. All of these inputs can be driven completely asynchronously.	
		State Meaning	Asserted—When an external interrupt request signal is asserted the priority is checked by the IPIC unit, and the interrupt is conditionally passed to the processor. Negated—There is no incoming interrupt from that source.
		Timing	Assertion—All of these inputs can be asserted completely asynchronously. Negation—Interrupts programmed as level-sensitive must remain asserted until serviced.

Table 8-2. IPIC External Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_INTA	OD	Interrupt request out. Active-low, open drain. When the IPIC is programmed in core disable mode, this output reflects the raw interrupts generated by on-chip sources. See Section 8.3, "Modes of Operation," for details.	
		State Meaning	Asserted—At least one interrupt is currently being signalled to the external system. Negated—Indicates no interrupt source currently routed to IRQ_OUT.
		Timing	Because external interrupts are asynchronous with respect to the system clock, both assertion and negation of IRQ_OUT occur asynchronously with respect to the interrupt source. All timing given here is approximate. Assertion—Internal interrupt source: 3 system bus clock cycles after the interrupt occurs. External interrupt source: 4 cycles after the interrupt occurs. Negation—Follows interrupt source negation with the following delay: Internal interrupt: 3 system bus clock cycles. External interrupt: 4 cycles.
MCP_OUT	OD	Non-maskable Interrupt (machine check) request out. Active-low, open drain. When the IPIC is programmed in core disable mode, this output reflects the *mcp* interrupts generated by on-chip sources. See Section 8.3, "Modes of Operation."	
		State Meaning	Asserted—At least one machine check interrupt is currently being signalled to the external system. Negated—Indicates no interrupt source currently routed to MCP_OUT.
		Timing	Because external interrupts are asynchronous with respect to the system clock, both assertion and negation of MCP_OUT occurs asynchronously with respect to the interrupt source. All timing given here is approximate. Assertion—Internal interrupt source: 2 system bus clock cycles after interrupt occurs. External interrupt source: 4 cycles after interrupt occurs. Negation—Follows interrupt source negation with the following delay: Internal interrupt: 2 system bus clock cycles. External interrupt: 4 cycles.

8.5 Memory Map/Register Definition

The IPIC programmable register map occupies 192 bytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect.

All IPIC registers are 32 bits wide and they are located on 32-bit address boundaries. Software can perform byte, half-word or word accesses to any IPIC registers. All addresses used in this chapter are offsets from the IPIC base, as defined in Chapter 2, "Memory Map."

Table 8-3 shows the memory map of the IPIC unit.

Table 8-3. IPIC Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
0x00	System global interrupt configuration register (SICFR)	R/W	0x0000_0000	8.5.1/8-7
0x04	System regular interrupt vector register (SIVCR)	R	0x0000_0000	8.5.2/8-9
0x08	System internal interrupt pending register (SIPNR_H)	R	0x0000_0000	8.5.3/8-11
0x0C	System internal interrupt pending register (SIPNR_L)	R	0x0000_0000	8.5.3/8-11
0x10	System internal interrupt group A priority register (SIPRR_A)	R/W	0x0530_9770	8.5.4/8-14
0x14	System internal interrupt group B priority register (SIPRR_B)	R/W	0x0530_9770	8.5.5/8-15

Table 8-3. IPIC Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
0x18	System internal interrupt group C priority register (SIPRR_C)	R/W	0x0530_9770	8.5.6/8-16
0x1C	System internal interrupt group D priority register (SIPRR_D)	R/W	0x0530_9770	8.5.7/8-17
0x20	System internal interrupt mask register (SIMSR_H)	R/W	0x0000_0000	8.5.8/8-18
0x24	System internal interrupt mask register (SIMSR_L)	R/W	0x0000_0000	8.5.8/8-18
0x28	System internal interrupt control register (SICNR)	R/W	0x0000_0000	8.5.9/8-19
0x2C	System external interrupt pending register (SEPNR)	R/W	Special	8.5.10/8-21
0x30	System mixed interrupt group A priority register (SMPRR_A)	R/W	0x0530_9770	8.5.11/8-21
0x34	System mixed interrupt group B priority register (SMPRR_B)	R/W	0x0530_9770	8.5.12/8-22
0x38	System external interrupt mask register (SEMSR)	R/W	0x0000_0000	8.5.13/8-23
0x3C	System external interrupt control register (SECNR)	R/W	0x0000_0000	8.5.14/8-24
0x40	System error status register (SERSR)	R/W	0x0000_0000	8.5.15/8-25
0x44	System error mask register (SERMR)	R/W	—	8.5.16/8-26
0x48	System error control register (SERCR)	R/W	0x0000_0000	8.5.17/8-27
0x4C	System external interrupt polarity control register (SEPCR)	R/W	0x0000_0000	8.5.18/8-28
0x4F	Reserved	—	—	—
0x50	System internal interrupt force register (SIFCR_H)	R/W	0x0000_0000	8.5.19/8-29
0x54	System internal interrupt force register (SIFCR_L)	R/W	0x0000_0000	8.5.19/8-29
0x58	System external interrupt force register (SEFCR)	R/W	0x0000_0000	8.5.20/8-30
0x5C	System error force register (SERFR)	R/W	0x0000_0000	8.5.21/8-30
0x60	System critical interrupt vector register (SCVCR)	R	0x0000_0000	8.5.22/8-31
0x64	System management interrupt vector register (SMVCR)	R	0x0000_0000	8.5.23/8-31
0x68–0xBF	Reserved	—	—	—

8.5.1 System Global Interrupt Configuration Register (SICFR)

SICFR, shown in Figure 8-2, defines the highest priority interrupt and whether interrupts are grouped or spread in the priority table. See Table 8-4 for more information.

Offset 0x00 Access: Read/write

Bits	0	1–7	8	9	10	11	12	13	14	15	16–21	22–23	24–31
R/W	—	HPI	—	MPSB	MPSA	—	IPSD	IPSC	IPSB	IPSA	—	HPIT	—
Reset	All zeros												

Figure 8-2. System Global Interrupt Configuration Register (SICFR)

Table 8-4 defines the bit fields of SICFR.

Table 8-4. SICFR Field Descriptions

Bits	Name	Description
0	—	Write ignored, read = 0
1–7	HPI	Highest priority interrupt. Specifies the 7-bit unique interrupt number/vector (see Table 8-6) of the single interrupt controller interrupt source that is advanced to the highest priority in the IPIC priority table (see Table 8-34). HPI can be modified dynamically.
8	—	Write ignored, read = 0
9	MPSB	Mixed interrupts priority scheme for group B. Selects the relative MIXB priority scheme. It cannot be changed dynamically. 0 Grouped. The MIXBs are grouped by priority at the top of the table. 1 Spread. The MIXBs are spread by priority in the table.
10	MPSA	Mixed interrupts priority scheme for group A. Selects the relative MIXA priority scheme. It cannot be changed dynamically. 0 Grouped. The MIXAs are grouped by priority at the top of the table. 1 Spread. The MIXAs are spread by priority in the table.
11	—	Write ignored, read = 0
12	IPSD	Internal interrupts priority scheme for group D. Selects the relative SYSD priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSDs are grouped by priority at the top of the table. 1 Spread. The SYSDs are spread by priority in the table.
13	IPSC	Internal interrupts priority scheme for group C. Selects the relative SYSC priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSCs are grouped by priority at the top of the table. 1 Spread. The SYSCs are spread by priority in the table.
14	IPSB	Internal interrupts priority scheme for group B. Selects the relative SYSB priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSBs are grouped by priority at the top of the table. 1 Spread. The SYSBs are spread by priority in the table.
15	IPSA	Internal interrupts priority scheme for group A. Selects the relative SYSA priority scheme. It cannot be changed dynamically. 0 Grouped. The SYSAs are grouped by priority at the top of the table. 1 Spread. The SYSAs are spread by priority in the table.
16–21	—	Write ignored, read = 0
22–23	HPIT	HPI priority position IPIC output interrupt type. Defines which type of IPIC output interrupt signal ($\overline{int}$, $\overline{cint}$, or $\overline{smi}$) asserts its request to the core in the HPI priority position. These bits cannot be changed dynamically. (If software really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change). The definition of HPIT is as follows: 00 $\overline{int}$ request is asserted to the core for HPI. 01 $\overline{smi}$ request is asserted to the core for HPI. 10 $\overline{cint}$ request is asserted to the core for HPI. 11 Reserved.
24–31	—	Write ignored, read = 0

8.5.2 System Global Interrupt Vector Register (SIVCR)

SIVCR, shown in Figure 8-3, contains a 7-bit code (Table 8-5) representing the regular unmasked interrupt source ($\overline{INT}$) of the highest priority level.

NOTE

Note that in core disabled mode the user should use SIVCR only in order to read an updated interrupt vector (SCVCR and SMVCR should not be used).

Offset 0x04 — Access: Read only

	0–5	6–24	25–31
R	IVECx	—	IVEC
W			
Reset	All zeros		

Figure 8-3. System Global Interrupt Vector Register (SIVCR)

Table 8-5 defines the bit fields of SIVCR.

Table 8-5. SIVCR Field Descriptions

Bits	Name	Description
0–5	IVECx	Backward (MPC8260) compatible regular interrupt vector. Specifies a 6-bit unique number of the IPIC's highest priority regular interrupt source, pending to the core. When a regular interrupt request occurs, SIVCR can be read. If there are multiple regular interrupt sources, SIVCR latches the highest priority regular interrupt. Note that IVECx field correctly reflects only the first 64 interrupt vectors (See Table 8-6 for details). The value of SIVEC cannot change while it is being read.
6–24	—	Write ignored, read = 0
25–31	IVEC	Regular interrupt vector. Specifies a 7-bit unique number of the IPIC's highest priority regular interrupt source, pending to the core. Note that the when a regular interrupt request occurs, SIVCR can be read. If there are multiple regular interrupt sources, SIVCR latches the highest priority regular interrupt. Note that the IVEC field correctly reflects all interrupt vectors (see Table 8-6 for details). The value of SIVCR cannot change while it is being read.

Table 8-6 shows the definition of IVEC.

Table 8-6. IVEC/CVEC/MVEC Field Definition

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
0	Error (no interrupt)	0b000_0000
1	PEX1 CNT	0b000_0001
2	PEX2 CNT	0b000_0010
3	Reserved	0b000_0011
4	MSIR1	0b0000_0100
5–8	Reserved	0b000_0101–0b000_1000
9	UART1	0b000_1001

Table 8-6. IVEC/CVEC/MVEC Field Definition (continued)

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
10	UART2	0b000_1010
11	SEC	0b000_1011
12	eTSEC1 1588 timer	0b000_1100
13	eTSEC2 1588 timer	0b000_1101
14	I2C1	0b000_1110
15	I2C2	0b000_1111
16	SPI	0b001_0000
17	IRQ1	0b001_0001
18	IRQ2	0b001_0010
19	IRQ3	0b001_0011
20	IRQ4	0b001_0100
21	IRQ5	0b001_0101
22	IRQ6	0b001_0110
23	IRQ7	0b001_0111
24–31	Reserved	0b001_1000–0b001_1111
32	TSEC1 Tx	0b010_0000
33	TSEC1 Rx	0b010_0001
34	TSEC1 Err	0b010_0010
35	TSEC2 TX	0b010_0011
36	TSEC2 Rx	0b010_0100
37	TSEC2 Err	0b010_0101
38	USB DR	0b010_0110
39	Reserved	0b010_0111
40–41	Reserved	0b010_1000–0b010_1001
42	eSDHC	0b010_1010
43	Reserved	0b010_1011
44	SATA1	0b010_1100
45	SATA2	0b010_1101
46	SATA3	0b010_1110
47	SATA4	0b010_1111
48	IRQ0	0b011_0000
49–63	Reserved	0b011_0001–0b011_1111
64	RTC SEC	0b100_0000
65	PIT	0b100_0001
66	PCI	0b100_0010

Table 8-6. IVEC/CVEC/MVEC Field Definition (continued)

Interrupt ID Number	Interrupt Meaning	Interrupt Vector
67	MSIR0	0b100_0011
68	RTC ALR	0b100_0100
69	MU	0b100_0101
70	SBA	0b100_0110
71	DMA	0b100_0111
72	GTM4	0b100_1000
73	GTM8	0b100_1001
74	GPIO1	0b100_1010
75	GPIO2	0b100_1011
76	DDR	0b100_1100
77	LBC	0b100_1101
78	GTM2	0b100_1110
79	GTM6	0b100_1111
80	PMC	0b101_0000
81	MSIR2	0b101_0001
82	MSIR3	0b101_0010
83	Reserved	0b101_0011
84	GTM3	0b101_0100
85	GTM7	0b101_0101
86	MSIR4	0b101_0110
87	MSIR5	0b101_0111
88	MSIR6	0b101_1000
89	MSIR7	0b101_1001
90	GTM1	0b101_1010
91	GTM5	0b101_1011
92–127	Reserved	0b101_1100–0b111_1111

8.5.3 System Internal Interrupt Pending Registers (SIPNR_H and SIPNR_L)

Each bit in SIPNR_H and SIPNR_L, shown in Figure 8-4 and Figure 8-5, may be assigned an internal interrupt source. (Implemented bits are listed in Table 8-7.) When an interrupt request is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is handled, the user clears the SIPNR bit by clearing the corresponding event register bit.

Note that SIPNR bit positions are not changed according to relative priority.

Offset 0x08 Access: Read only

	0 … 31
R	INT*n* (Implemented bits are listed in Table 8-7.)
W	
Reset	All zeros

Figure 8-4. System Internal Interrupt Pending Register (SIPNR_H)

Table 8-7 lists implemented SIPNR_H fields. Note that these field descriptions are also valid for SIFCR_H and SIMSR_H.

Table 8-7. SIPNR_H/SIFCR_H/SIMSR_H Bit Assignments

Bits	Field
0	TSEC1 Tx
1	TSEC1 Rx
2	TSEC1 Err
3	TSEC2 Tx
4	TSEC2 Rx
5	TSEC2 Err
6	USB DR
7–9	—
10	eSDHC
11	—
12	SATA1
13	SATA2
14	SATA3
15	SATA4
16	PEX1 CNT
17	PEX2 CNT
19	MSIR1
20-23	—
24	UART1
25	UART2
26	SEC
27	eTSEC1 1588 timer
28	eTSEC2 1588 timer
29	I2C1
30	I2C2
31	SPI

Table 8-8 defines the bit fields of SIPNR_H.

Table 8-8. SIPNR_H Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit (listed in Table 8-7) corresponds to an internal interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is handled, the user clears the SIPNR bit by clearing the corresponding event register bit. SIPNR bits are read only. Writing to this register has no effect. Note that the SIPNR bit positions are not changed according to their relative priority. For unimplemented bits, writes are ignored, read = 0.

SIPNR_L is shown in Figure 8-4.

Offset 0x0C — Access: Read only

	0 … 31
R	INT*n* (Implemented bits are listed in Table 8-9.)
W	
Reset	All zeros

Figure 8-5. System Internal Interrupt Pending Register (SIPNR_L)

Table 8-9 lists implemented SIPNR_L fields. Note that these field assignments are also valid for SIFCR_L and SIMSR_L.

Table 8-9. SIPNR_L/SIFCR_L/SIMSR_L Bit Assignments

Bits	Field
0	RTC SEC
1	PIT
2	PCI
3	MSIR0
4	RTC ALR
5	MU
6	SBA
7	DMA
8	GTM4
9	GTM8
10	GPIO1
11	GPIO2
12	DDR
13	LBC
14	GTM2
15	GTM6
16	PMC
17	MSIR2
18	MSIR3

Table 8-9. SIPNR_L/SIFCR_L/SIMSR_L Bit Assignments (continued)

Bits	Field
19	—
20	GTM3
21	GTM7
22	MSIR4
23	MSIR5
24	MSIR6
25	MSIR7
26	GTM1
27	GTM5
28–30	—
31	—

Table 8-10 defines the bit fields of SIPNR_L.

Table 8-10. SIPNR_L Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit (listed in Table 8-9) corresponds to an internal interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SIPNR bit. When a pending interrupt is handled, the user clears the SIPNR bit by clearing the corresponding event register bit. SIPNR bits are read only. Writing to this register has no effect. Note that the SIPNR bit positions are not changed according to their relative priority. For unimplemented bits, writes are ignored, read = 0.

8.5.4 System Internal Interrupt Group A Priority Register (SIPRR_A)

SIPRR_A, shown in Figure 8-6, defines the priority between TSEC1 transmit request (TSEC1 Tx), TSEC1 receive request (TSEC1 Rx), TSEC1 transmit/receive error (TSEC1 Err), TSEC2 transmit request (TSEC2 Tx), TSEC2 receive request (TSEC2 Rx) TSEC2 transmit/receive error (TSEC2 Err), USB DR, internal interrupt signals.

For more information, see Section 8.6.3, "Internal Interrupts Group Relative Priority."

Offset 0x10 — Access: Read/write

Bits	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R/W	SYSA0P	SYSA1P	SYSA2P	SYSA3P	—	SYSA4P	SYSA5P	SYSA6P	SYSA7P	—
Reset	0 0 0	0 0 1	0 1 0	0 1 1	0 0 0 0	1 0 0	1 0 1	1 1 0	1 1 1	0 0 0 0

Figure 8-6. System Internal Interrupt Group A Priority Register (SIPRR_A)

Table 8-11 defines the bit fields of SIPRR_A.

Table 8-11. SIPRR_A Field Descriptions

Bits	Name	Description
0–2	SYSA0P	SYSA0 priority order. Defines which interrupt source asserts its request in the SYSA0 priority position. The user should not program the same code to multiple priority positions (0–7). These bits can be changed dynamically. The definition of SYSA0P is as follows: 000 TSEC1 Tx asserts its request in the SYSA0 position. 001 TSEC1 Rx asserts its request in the SYSA0 position. 010 TSEC1 Err asserts its request in the SYSA0 position. 011 TSEC2 Tx asserts its request in the SYSA0 position. 100 TSEC2 Rx 101 TSEC2 Err asserts its request in the SYSA0 position. 110 USB DR asserts its request in the SYSA0 position. 111 Reserved
3–11, 16–27	SYSA1P–SYSA7P	Same as SYSA0P, but for SYSA1P–SYSA7P.
12–15, 28–31	—	Write ignored, read = 0

8.5.5 System Internal Interrupt Group B Priority Register (SIPRR_B)

The system internal interrupt group B priority register (SIPRR_B), shown in Figure 8-7, defines the priority between internal interrupt signals.

For more information, see Section 8.6.3, "Internal Interrupts Group Relative Priority."

Offset 0x14 Access: Read/write

	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R/W	SYSB0P	SYSB1P	SYSB2P	SYSB3P	—	SYSB4P	SYSB5P	SYSB6P	SYSB7P	—
Reset	0 0 0	0 0 1	0 1 0	0 1 1	0 0 0 0	1 0 0	0 1 0	1 1 0	1 1 1	0 0 0 0

Figure 8-7. System Internal Interrupt Group B Priority Register (SIPRR_B)

Table 8-12 defines the bit fields of SIPRR_B.

Table 8-12. SIPRR_B Field Descriptions

Bits	Name	Description
0–2	SYSB0P	SYSB0 Priority order.Defines which interrupt source asserts its request in the SYSB0 priority position. The user should not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSB0P is shown as follows: 000 Reserved. 001 Reserved. 010 eSDHC asserts its request in the SYSB0 position. 011 Reserved 100 SATA1 asserts its request in the SYSB0 position. 101 SATA2 asserts its request in the SYSB0 position. 110 SATA3 asserts its request in the SYSB0 position. 111 SATA4 asserts its request in the SYSB0 position.
3–11, 16–27	SYSB1P–SYSB7P	Same as SYSB0P, but for SYSB1P–SYSB7P.
12–15, 28–31	—	Write ignored, read = 0

8.5.6 System Internal Interrupt Group C Priority Register (SIPRR_C)

The system internal interrupt group C priority register (SIPRR_C), shown in Figure 8-8, defines the priority between internal interrupt signals.

For more information about interrupt priorities, see Section 8.6.3, “Internal Interrupts Group Relative Priority.”

Offset 0x18 Access: Read/write

Bits	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R/W	SYSC0P	SYSC1P	SYSC2P	SYSC3P	—	SYSC4P	SYSC5P	SYSC6P	SYSC7P	—
Reset	0 0 0	0 0 1	0 1 0	0 1 1	0 0 0 0	1 0 0	0 1 0	1 1 0	1 1 1	0 0 0 0

Figure 8-8. System Internal Interrupt Group C Priority Register (SIPRR_C)

Table 8-13 defines the bit fields of SIPRR_C.

Table 8-13. SIPRR_C Field Descriptions

Bits	Name	Description
0–2	SYSC0P	SYSC0 priority order.Defines which interrupt source asserts its request in the SYSC0 priority position. The user should not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of SYSC0P is shown as follows: 000 PEX1 CNT asserts its request in the SYSC0 position. 001 PEX2 CNT asserts its request in the SYSC0 position. 010 Reserved 011 MSIR1 asserts its request in the SYSC0 position 100–111Reserved
3–11, 16–27	SYSC1P–SYSC7P	Same as SYSC0P, but for SYSC1P–SYSC7P.
12–15, 28–31	—	Write ignored, read = 0

8.5.7 System Internal Interrupt Group D Priority Register (SIPRR_D)

SIPRR_D, shown in Figure 8-9, defines the priority among the interrupt sources listed in Table 8-14.

Offset 0x1C — Access: Read/write

Bits	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R/W	SYSD0P	SYSD1P	SYSD2P	SYSD3P	—	SYSD4P	SYSD5P	SYSD6P	SYSD7P	—
Reset	0 0 0	0 0 1	0 1 0	0 1 1	0 0 0 0	1 0 0	1 0 1	1 1 0	1 1 1	0 0 0 0

Figure 8-9. System Internal Interrupt Group D Priority Register (SIPRR_D)

Table 8-14 defines the bit fields of SIPRR_D.

Table 8-14. SIPRR_D Field Descriptions

Bits	Name	Description
0–2	SYSD0P	SYSD0 priority order. Defines which interrupt source asserts its request in the SYSD0 priority position. The user should not program the same code to more than one priority position (0–7). These bits can be changed dynamically. SYSD0P is defined as follows: 000 UART1 asserts its request in the SYSD0 position. 001 UART2 asserts its request in the SYSD0 position. 010 SEC asserts its request in the SYSD0 position. 011 eTSEC1 1588 timer asserts its request in the SYSD0 position. 100 eTSEC2 1588 timer asserts its request in the SYSD0 position. 101 I2C1 asserts its request in the SYSD0 position. 110 I2C2 asserts its request in the SYSD0 position. 111 SPI asserts its request in the SYSD0 position.
3–11, 16–27	SYSD1P–SYSD7P	Same as SYSD0P, but for SYSD1P–SYSD7P.
12–15, 28–31	—	Write ignored, read = 0

8.5.8 System Internal Interrupt Mask Register (SIMSR_H and SIMSR_L)

Each implemented bit in SIMSR_H and SIMSR_L, shown in Figure 8-10 and Figure 8-11, corresponds to an internal interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. When an interrupt request occurs, the corresponding SIPNR bit is set, regardless of the SIMSR bit. However, if the corresponding SIMSR bit is cleared, no interrupt request is passed to the core.

When an SIMSR bit is cleared by the user at the same time corresponding interrupt source requests an interrupt service, the request stops. If the user sets the SIMSR bit later, the core processes any pending corresponding interrupt requests according to its priority.

Offset 0x20 Access: Read/write

	0–31
R W	INT*n* (Implemented bits are listed in Table 8-7.)
Reset	All zeros

Figure 8-10. System Internal Interrupt Mask Register (SIMSR_H)

Table 8-15 defines the bit fields of SIMSR_H.

Table 8-15. SIMSR_H Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit (listed in Table 8-7) corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. An interrupt is unmasked (enable) by setting the corresponding SIMSR bit. The SIMSR can be read by the user at any time. **Note:** • SIMSR bit positions do not change according to their relative priority. • The user can clear pending register bits that were set by multiple interrupt events only by clearing all unmasked events in the corresponding event register. • If an SIMSR bit is masked at the same time that the corresponding SIPNR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts are pending). Thus, the user should always include an error vector routine, even if it contains only an **rfi** instruction. The error vector cannot be masked. Unimplemented bits, shown as reserved in Table 8-7, are ignored on writes; read = 0.

Figure 8-11 shows SIMSR_L.

Offset 0x24 Access: Read/write

	0–31
R W	INT*n* (Implemented bits are listed in Table 8-9.)
Reset	All zeros

Figure 8-11. System Internal Interrupt Mask Register (SIMSR_L)

Table 8-16 defines the bit fields of SIMSR_L.

Table 8-16. SIMSR_L Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit (listed in Table 8-9) corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SIMSR bit. An interrupt is unmasked (enabled) by setting the corresponding SIMSR bit. The SIMSR can be read by the user at any time. **Note:** • SIMSR bit positions are not changed according to their relative priority. • The user can clear pending register bits that were set by multiple interrupt events only by clearing all unmasked events in the corresponding event register. • If an SIMSR bit is masked at the same time that the corresponding SIPNR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts are pending). Thus, the user should always include an error Unimplemented bits, shown as reserved in Figure 8-11, are ignored on writes; read = 0.

8.5.9 System Internal Interrupt Control Register (SICNR)

SICNR, shown in Figure 8-12, defines the IPIC output interrupt type ($\overline{\text{int}}$, $\overline{\text{cint}}$, or $\overline{\text{smi}}$) in the SYSA0–SYSA1, SYSB0–SYSB1, SYSC0–SYSC1, and SYSD0–SYSD1 priority positions. All other priority positions assert $\overline{\text{int}}$ to the core.

Note that in core disabled mode the user should use the $\overline{\text{int}}$ output interrupt type (should not use $\overline{\text{cint}}$ or $\overline{\text{smi}}$ output interrupt types) to read an updated SIVCR.

Offset 0x28 Access: Read write

	0–1	2–3	4–7	8–9	10–11	12–15	16–17	18–19	20–23	24–25	26–27	28–31
R/W	SYSD0T	SYSD1T	—	SYSC0T	SYSC1T	—	SYSB0T	SYSB1T	—	SYSA0T	SYSA1T	—
Reset	All zeros											

Figure 8-12. System Internal Interrupt Control Register (SICNR)

Table 8-17 defines the bit fields of SICNR.

Table 8-17. SICNR Field Descriptions

Bits	Name	Description
0–1	SYSD0T	SYSD0 priority position IPIC output interrupt type.Defines which type of the IPIC output interrupt signal ($\overline{\textit{int}}$, $\overline{\textit{cint}}$, or $\overline{\textit{smi}}$) asserts its request to the core in the SYSD0 priority position. These bits cannot be changed dynamically. (to change it, software must make sure the corresponding interrupt source is masked or it cannot happen during the change). The definition of SYSD0T is as follows: 00 $\overline{\textit{int}}$ request is asserted to the core for SYSD0. 01 $\overline{\textit{smi}}$ request is asserted to the core for SYSD0. 10 $\overline{\textit{cint}}$ request is asserted to the core for SYSD0. 11 Reserved
2–3	SYSD1T	Same as SYSD0T, but for SYSD1T.
4–7	—	Write ignored, read = 0

Table 8-17. SICNR Field Descriptions (continued)

Bits	Name	Description
8–9	SYSC0T	SYSC0 priority position IPIC output interrupt Type.Defines which type of the IPIC output interrupt signal (*int*, *cint*, or *smi*) asserts its request to the core in the SYSC0 priority position. These bits can not be changed dynamically. (If s/w really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change). The definition of SYSC0T is as follows: 00 *int* request is asserted to the core for SYSC0. 01 *smi* request is asserted to the core for SYSC0. 10 *cint* request is asserted to the core for SYSC0. 11 Reserved
10–11	SYSC1T	Same as SYSC0T, but for SYSC1T.
12–15	—	Write ignored, read = 0
16–17	SYSB0T	SYSB0 priority position IPIC output interrupt Type.Defines which type of the IPIC output interrupt signal (*int*, *cint*, or *smi*) asserts its request to the core in the SYSB0 priority position. These bits can not be changed dynamically. (If s/w really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change). The definition of SYSB0T is as follows: 00 *int* request is asserted to the core for SYSB0. 01 *smi* request is asserted to the core for SYSB0. 10 *cint* request is asserted to the core for SYSB0. 11 Reserved
18–19	SYSB1T	Same as SYSB0T, but for SYSB1T.
20–23	—	Write ignored, read = 0
24–25	SYSA0T	SYSA0 priority position IPIC output interrupt type. Defines which type of the IPIC output interrupt signal (*int*, *smi*, or *cint*) asserts its request to the core in the SYSA0 priority position. These bits can not be changed dynamically. (If s/w really wants to change it, it has to make sure the corresponding interrupt source is masked or it won't happen during the change). The definition of SYSA0T is as follows: 00 *int* request is asserted to the core for SYSA0. 01 *smi* request is asserted to the core for SYSA0. 10 *cint* request is asserted to the core for SYSA0. 11 Reserved.
26–27	SYSA1T	Same as SYSA0T, but for SYSA1T
28–31	—	Write ignored, read = 0

8.5.10 System External Interrupt Pending Register (SEPNR)

Each bit in the SEPNR, shown in Figure 8-13, corresponds to an external interrupt source. When an interrupt is received, the interrupt controller sets the corresponding SEPNR bit.

Offset 0x2C — Access: Read/write

	0	1	2	3	4	5	6	7	8–15
R/W	IRQ0[1]	IRQ1	IRQ2	IRQ3	IRQ4	IRQ5	IRQ6	IRQ7	—
Reset	The reset values of implemented bits reflect the values of the external IRQ signals. Reserved bits are zeros. [2]								

	16–31
R/W	—
Reset	All zeros

[1] This bit is valid only if the IRQ0 signal is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

[2] The user should drive all IRQ inputs to an inactive state prior to reset negation

Figure 8-13. System External Interrupt Pending Register (SEPNR)

Table 8-18 defines the bit fields of SEPNR.

Table 8-18. SEPNR Field Descriptions

Bits	Name	Description
0–7	IRQ*n*	Each bit corresponds to an external interrupt source. When an external interrupt is received, the interrupt controller sets the corresponding SEPNR bit. When a pending interrupt is handled, the user must clear the corresponding SEPNR bit. For level triggered cases, the software needs to cause the $\overline{\text{IRQn}}$ to negate which automatically clears the bit in SEPNR. For edge-triggered cases, the software needs to clear the corresponding bit in SEPNR. SEPNR bits are cleared by writing ones to them. Because the user can only clear bits in this register, writing zeros to this register has no effect. Note that the SEPNR bit positions are not changed according to their relative priority.
8–31	—	Write ignored, read = 0

8.5.11 System Mixed Interrupt Group A Priority Register (SMPRR_A)

The SMPRR_A, shown in Figure 8-14, defines the priority among the sources listed in Table 8-19.

Offset 0x30 — Access: Read/write

	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R/W	MIXA0P	MIXA1P	MIXA2P	MIXA3P	—	MIXA4P	MIXA5P	MIXA6P	MIXA7P	—
Reset	0 0 0	0 0 1	0 1 0	0 1 1	0 0 0 0	1 0 0	1 0 1	1 1 0	1 1 1	0 0 0 0

Figure 8-14. System Mixed Interrupt Group A Priority Register (SMPRR_A)

Table 8-19 defines the bit fields of SMPRR_A.

Table 8-19. SMPRR_A Field Descriptions

Bits	Name	Description
0–2	MIXA0P	MIXA0 priority order. Defines which interrupt source asserts its request in the MIXA0 priority position. The user must not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of MIXA0P is as follows: 000 RTC SEC asserts its request to the MIXA0 position. 001 PIT asserts its request to the MIXA0 position. 010 PCI asserts its request to the MIXA0 position. 011 MSIR0 asserts its request to the MIXA0 position. 100 IRQ0 asserts its request to the MIXA0 position. This field for MIXA0 position is valid (must not be ignored) if IRQ0 signal configured as an external maskable interrupt (SEMSR[SIRQ0] = 0). 101 IRQ1 asserts its request to the MIXA0 position. 110 IRQ2 asserts its request to the MIXA0 position. 111 IRQ3 asserts its request to the MIXA0 position.
3–11, 16–27	MIXA1P–MIXA7P	Same as MIXA0P, but for MIXA1P–MIXA7P.
12–15, 28–31	—	Write ignored, read = 0

8.5.12 System Mixed Interrupt Group B Priority Register (SMPRR_B)

SMPRR_B, shown in Figure 8-15, defines the priority among the sources listed in Table 8-20.

Offset 0x34 — Access: Read/write

Bits	0–2	3–5	6–8	9–11	12–15	16–18	19–21	22–24	25–27	28–31
R/W	MIXB0P	MIXB1P	MIXB2P	MIXB3P	—	MIXB4P	MIXB5P	MIXB6P	MIXB7P	—
Reset	0 0 0	0 0 1	0 1 0	0 1 1	0 0 0 0	1 0 0	1 0 1	1 1 0	1 1 1	0 0 0 0

Figure 8-15. System Mixed Interrupt Group B Priority Register (SMPRR_B)

Table 8-20 defines the bit fields of SMPRR_B.

Table 8-20. SMPRR_B Field Descriptions

Bits	Name	Description
0–2 3–11, 16–27	MIXB*n*P	MIXB*n* priority order. Defines which interrupt source asserts its request in the MIXB*n* priority position. The user must not program the same code to more than one priority position (0–7). These bits can be changed dynamically. The definition of MIXB*n*P is as follows: 000 RTC ALR asserts its request to the MIXB*n* position. 001 MU asserts its request to the MIXB*n* position. 010 SBA asserts its request to the MIXB*n* position. 011 DMA asserts its request to the MIXB*n* position. 100 IRQ4 asserts its request to the MIXB*n* position. 101 IRQ5 asserts its request to the MIXB*n* position. 110 IRQ6 asserts its request to the MIXB*n* position. 111 IRQ7 asserts its request to the MIXB*n* position.
12–15, 28–31	—	Write ignored, read = 0

8.5.13 System External Interrupt Mask Register (SEMSR)

Each bit in the system external interrupt mask register (SEMSR), shown in Figure 8-13, corresponds to an external interrupt source. The user masks an interrupt by clearing the corresponding SEMSR bit. An interrupt is unmasked (enabled) by setting the corresponding SEMSR bit.

When an external interrupt request occurs, the corresponding SEPNR bit is set regardless of the setting of the corresponding SEMSR bit. However, if the corresponding SEMSR bit is cleared, no interrupt request is passed to the core.

When an SEMSR bit is cleared by the user at the same time that an interrupt source requests an interrupt service, the request stops. If the user sets the SEMSR bit later, a previously pending interrupt request is processed by the core according to its assigned priority. SEMSR can be read by the user at any time.

Offset 0x38 — Access: Read/write

	0	1	2	3	4	5	6	7	8–15
R/W	IRQ0[1]	IRQ1	IRQ2	IRQ3	IRQ4	IRQ5	IRQ6	IRQ7	—
Reset	The reset values of implemented bits reflect the values of the external IRQ signals. Reserved bits are zeros. [2]								

	16	17–31
R/W	SIRQ0	—
Reset	All zeros	

1 This bit is valid only if the IRQ0 signal is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

2 The user should drive all IRQ inputs to an inactive state prior to reset negation

Figure 8-16. System External Interrupt Mask Register (SEMSR)

Table 8-21 defines the bit fields of SEMSR.

Table 8-21. SEMSR Field Descriptions

Bits	Name	Description
0–7	—	Each bit corresponds to an external interrupt source. The user masks an interrupt by clearing the SEMSR bit. An interrupt can be enabled by setting the corresponding SEMSR bit. SEMSR can be read by the user at any time. **Note:** • SEMSR bit positions are not affected by their relative priority. • The user can clear pending register bits that were set by multiple interrupt events only by clearing all unmasked events in the corresponding event register. • If an SEMSR bit is masked at the same time that the corresponding SEPNR bit causes an interrupt request to the core, the error vector is issued (if no other interrupts pending). Thus, the user must always include an error vector routine, even if it contains only an **rfi** instruction. The error vector cannot be masked.
8–15	—	Write ignored, read = 0
16	SIRQ0	Steer IRQ0. 0 IRQ0 is used as external interrupt request 1 IRQ0 is used as external MCP request
17–31	—	Write ignored, read = 0

8.5.14 System External Interrupt Control Register (SECNR)

SECNR, shown in Figure 8-17, defines the edge detect mode for external $\overline{\text{IRQ}n}$ interrupt signals and determines whether the corresponding $\overline{\text{IRQ}n}$ signal asserts an interrupt request upon either a high-to-low change or assertion on the pin. It also defines the IPIC output interrupt type ($\overline{\text{int}}$, $\overline{\text{cint}}$, or $\overline{\text{smi}}$) in the MIXA0–MIXA1 and MIXB0–MIXB1 priority positions.

Note that in core disabled mode of operation the user should use the $\overline{int}$ output interrupt type (should not use $\overline{cint}$ or $\overline{smi}$ output interrupt types) in order to read an updated SIVCR.

Offset 0x3C Access: Read/write

	0–1	2–3	4–7	8–9	10–11	12–15
R/W	MIXB0T	MIXB1T	—	MIXA0T	MIXA1T	—
Reset	All zeros					

	16	17	18	19	20	21	22	23	24–31
R/W	EDI0	EDI1	EDI2	EDI3	EDI4	EDI5	EDI6	EDI7	—
Reset	All zeros								

Figure 8-17. System External Interrupt Control Register (SECNR)

Table 8-22 defines the bit fields of SECNR.

Table 8-22. SECNR Field Descriptions

Bits	Name	Description
0–1	MIXB0T	MIXB0 priority position IPIC output interrupt type. Defines which type of the IPIC output interrupt signal ($\overline{int}$, $\overline{cint}$, or $\overline{smi}$) asserts its request to the core in the MIXB0 priority position. These bits can be changed dynamically. The definition of MIXB0T is as follows: 00 $\overline{int}$ request is asserted to the core for MIXB0. 01 $\overline{smi}$ request is asserted to the core for MIXB0. 10 $\overline{cint}$ request is asserted to the core for MIXB0. 11 Reserved
2–3	MIXB1T	Same as MIXB0T, but for MIXB1T.
4–7	—	Write ignored, read = 0
8–9	MIXA0T	MIXA0 priority position IPIC output interrupt Type. Defines which type of the IPIC output interrupt signal ($\overline{int}$, $\overline{cint}$, or $\overline{smi}$) asserts its request to the core in the MIXA0 priority position. These bits can be changed dynamically. The definition of MIXA0T is as follows: 00 $\overline{int}$ request is asserted to the core for MIXA0. 01 $\overline{smi}$ request is asserted to the core for MIXA0. 10 $\overline{cint}$ request is asserted to the core for MIXA0. 11 Reserved
10–11	MIXA1T	Same as MIXA0T, but for MIXA1T.
12–15	—	Write ignored, read = 0
16–23	EDIx	Each bit defines the edge detect mode for the external $\overline{IRQ}n$ interrupt signals, determines whether the corresponding $\overline{IRQ}n$ signal asserts an interrupt request upon either a high-to-low change or low assertion on the pin. The corresponding $\overline{IRQ}n$ signal asserts an interrupt request as follows: 0 Low assertionon $\overline{IRQ}n$ generates an interrupt request (level sensitive). 1 High-to-lowchange on $\overline{IRQ}n$ generates an interrupt request (edge sensitive).
24–31	—	Write ignored, read = 0

8.5.15 System Error Status Register (SERSR)

The bits in the SERSR, shown in Figure 8-18, correspond to the external and internal non-maskable error source machine check (mcp) conditions listed in Table 8-23. When an error interrupt signal is received, the interrupt controller sets the corresponding SERSR bit.

Offset 0x40 Access: Read/write

	0 … 31
R W	INT*n* (Implemented bits are listed in Table 8-23)
Reset	All zeros

Figure 8-18. System Error Status Register (SERSR)

Table 8-23 lists the implemented SERSR bits. Note that these field assignments are valid for SERMR and SERFR.

Table 8-23. SERSR/SERMR/SERFR Bit Assignments

Bits	Field
0	IRQ0[1]
1	WDT
2	SBA
3	—
4	—
5	PCI
6	—
7	MU
8–14	—
15	—
16–31	—

[1] This bit is valid only if the IRQ0 signal is configured as an external MCP interrupt (SEMSR[SIRQ0] = 1)

Table 8-24 defines the bit fields of SERSR.

Table 8-24. SERSR Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit in the SERSR, listed in Table 8-23, corresponds to an external and an internal error source (mcp). When an error interrupt signal is received, the interrupt controller sets the corresponding SERSR bit. SERSR bits are cleared by writing ones to them. Unmasked event register bits should be cleared before clearing SERSR bits. Because the user can only clear bits in this register, writing zeros to this register has no effect. SERSR bits are cleared by power-on reset. Subsequent soft and hard resets do not affect SERSR bit states. For unimplemented bits (listed as reserved in Table 8-23), writes are ignored, read = 0

8.5.16 System Error Mask Register (SERMR)

Each implemented bit in SERMR, shown in Figure 8-19, corresponds to an external and an internal $\overline{mcp}$ source (MCP). The user masks an MCP by clearing and enables an interrupt by setting the corresponding SERMR bit. When a masked MCP occurs, the corresponding SERSR bit is set, regardless of the setting of the corresponding SERMR bit although no MCP request is passed to the core in this case. The SERMR can be read by the user at any time.

Offset 0x44 Access: Read/write

	0–31
R/W	INT*n* (Implemented bits are listed in Table 8-23.)
Reset	Implemented bits reset to ones; unimplemented (reserved) bits reset to zeros.

Figure 8-19. System Error Mask Register (SERMR)

Table 8-25 defines the bit fields of SERMR.

Table 8-25. SERMR Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented SERMR bit, listed in Table 8-23, corresponds to an external and an internal MCP source. The user masks an MCP by clearing and enables an interrupt by setting the corresponding SERMR bit. When a masked MCP occurs, the corresponding SERSR bit is set, regardless of the setting of the SERMR bit although no MCP request is passed to the core. The SERMR can be read by the user at any time. Writes to unimplemented (reserved) bits are ignored; read = 0

8.5.17 System Error Control Register (SERCR)

SERCR, shown in Figure 8-20, defines the control bits that route MCP requests in core disable mode to either MCP_OUT or PCI_INTA in core-disable mode.

Offset 0x48 Access: Read/write

	0–30	31
R/W	—	MCPR
Reset	All zeros	

Figure 8-20. System Error Control Register (SERCR)

Table 8-26 defines the bit fields of SERCR.

Table 8-26. SERCR Field Descriptions

Bits	Name	Description
0–30	—	Write ignored, read = 0
31	MCPR	MCP route. Route MCP request to either MCP_OUT or PCI_INTA (in core disable mode). 0 MCP routed to PCI_INTA (in core disable mode). 1 MCP routed to MCP_OUT (in core disable mode).

8.5.18 System External interrupt Polarity Control Register (SEPCR)

SEPCR, shown in Figure 8-21, defines the polarity for each one of the external $\overline{\text{IRQ}}n$ interrupt signals and determines whether the corresponding $\overline{\text{IRQ}}n$ signal is treated as active low or active high signal. The active low signals will assert an interrupt request upon either a high-to-low change or assertion (low state) on the pin. The active high signals will asserts an interrupt request upon either a low-to-high change or assertion (high state) on the pin. See Section 8.5.14, “System External Interrupt Control Register (SECNR),” on page 8-24 for more details.

NOTE

Note that the $\overline{\text{IRQ}}n$ signals are overbarred although the SEPCR could be programmed to accept active high signals. The overbar should be ignored in this case.

Offset 0x4C — Access: Read/write

	0	1	2	3	4	5	6	7	8–15
R/W	EIP0	EIP1	EIP2	EIP3	EIP4	EIP5	EIP6	EIP7	—
Reset	All zeros								

	16–31
R/W	—
Reset	All zeros

Figure 8-21. System External Interrupt Polarity Control Register (SEPCR)

Table 8-27 defines the bit fields of SEPCR.

Table 8-27. SEPCR Field Descriptions

Bits	Name	Description
0–7	EIPx	Each bit defines the active state for the external $\overline{\text{IRQ}}n$ interrupt signals. 0 Active Low. 1 Active High.
8–31	—	Write ignored, read = 0

8.5.19 System Internal Interrupt Force Registers (SIFCR_H and SIFCR_L)

Each implemented bit SIFCR_H and SIFCR_L, shown in Figure 8-22 and Figure 8-23, corresponds to an internal interrupt source. When a bit is set, the interrupt controller generates the corresponding interrupt (sets the corresponding SIPNR bit).The SIFCR can be read by the user at any time.

Offset 0x50 — Access: Read/write

	0–31
R W	INT*n* (Implemented bits are listed in Table 8-7.)
Reset	All zeros

Figure 8-22. System Internal Interrupt Force Register (SIFCR_H)

Table 8-28 defines the bit fields of SIFCR_H.

Table 8-28. SIFCR_H Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit, listed in Table 8-7, corresponds to an internal interrupt source. The user forces an interrupt by setting the corresponding SIFCR*x* bit. SIFCR*n* bit positions are not changed according to their relative priority. Writes to unimplemented (reserved) bits are ignored; read = 0

SIFCR_L is shown in Figure 8-23.

Offset 0x54 — Access: Read/write

	0–31
R W	INT*n* (Implemented bits are listed in Table 8-9).
Reset	All zeros

Figure 8-23. System Internal Interrupt Force Register (SIFCR_L)

Table 8-29 defines the bit fields of SIFCR_L.

Table 8-29. SIFCR_L Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit, listed in Table 8-9, corresponds to an internal interrupt source. The user forces an interrupt by setting the corresponding SIFCR*x* bit. SIFCR*x* bit positions are not changed according to their relative priority. Writes to unimplemented (reserved) bits are ignored; read = 0

8.5.20 System External Interrupt Force Register (SEFCR)

Each implemented bit in SEFCR, shown in Figure 8-24, corresponds to an external interrupt source. When a bit is set, the interrupt controller generates the corresponding external interrupt (sets the corresponding SEPNR bit). SEFCR can be read by the user at any time.

Offset 0x58 | Access: Read/write

	0	1	2	3	4	5	6	7	8–15
R/W	IRQ0[1]	IRQ1	IRQ2	IRQ3	IRQ4	IRQ5	IRQ6	IRQ7	—
Reset	All zeros								

	16–31
R/W	—
Reset	All zeros

1 This bit is valid only if IRQ0 is configured as an external maskable interrupt (SEMSR[SIRQ0] = 0)

Figure 8-24. System External Interrupt Force Register (SEFCR)

Table 8-30 defines the bit fields of SEFCR.

Table 8-30. SEFCR Field Descriptions

Bits	Name	Description
0–7	IRQ*n*	Each bit corresponds to an external interrupt source. The user force an interrupt by setting the SEFCR bit. **Note:** SEFCR bit positions are not affected by their relative priority.
8–31	—	Write ignored, read = 0

8.5.21 System Error Force Register (SERFR)

Each bit in the system error force register (SERFR), shown in Figure 8-25, corresponds to an external MCP source. When a bit is set, the interrupt controller generates the corresponding MCP interrupt (sets the corresponding SERSR bit). The SERFR can be read by the user at any time.

Offset 0x5C | Access: Read/write

	0–31
R/W	INT*n* (Implemented bits are listed in Table 8-23.)
Reset	All zeros

Figure 8-25. System Error Status Register (SERFR)

Table 8-31 defines the bit fields of SERFR.

Table 8-31. SERFR Field Descriptions

Bits	Name	Description
0–31	INT*n*	Each implemented bit, listed in Table 8-23, corresponds to an external MCP source. The user forces an MCP by setting the SERFR bit. SERFR bit positions are not affected by their relative priority. Attempts to write to unimplemented (reserved) bits are ignored; read = 0

8.5.22 System Critical Interrupt Vector Register (SCVCR)

SCVCR, shown in Figure 8-26, contains a 7-bit code (Table 8-32) representing the unmasked critical interrupt ($\overline{\text{CINT}}$) source of the highest priority level.

Note that in core-disabled mode the user should use SIVCR only to read an updated interrupt vector (SCVCR should not be used).

Offset 0x60 Access: Read only

	0–5	6–24	25–31
R	CVECx	—	CVEC
W			
Reset	All zeros		

Figure 8-26. System Critical Interrupt Vector Register (SCVCR)

Table 8-32 defines SCVCR bit fields.

Table 8-32. SCVCR Field Descriptions

Bits	Name	Description
0–5	CVECx	Backward (MPC8260) compatible critical interrupt vector. Specifies a 6-bit unique number of the IPIC’s highest priority critical interrupt source, pending to the core. When a critical interrupt request occurs, SCVCR can be read. If there are multiple critical interrupt sources, SCVCR latches the highest priority critical interrupt. Note that CVECx field will correctly reflect only first 64 interrupt vectors (See Table 8-6 for details). The value of SCVEC cannot change while it is being read.
6–24	—	Write ignored, read = 0
25–31	CVEC	Critical interrupt vector. Specifies a 7-bit unique number of the IPIC’s highest priority critical interrupt source, pending to the core. When a critical interrupt request occurs, SCVCR can be read. If there are multiple critical interrupt sources, SCVCR latches the highest priority critical interrupt. Note that CVEC field correctly reflects all of the interrupt vectors (See Table 8-6 for details). The value of SCVEC cannot change while it is being read.

8.5.23 System Management Interrupt Vector Register (SMVCR)

SMVCR, shown in Figure 8-27, contains a 7-bit code (Table 8-33) representing the unmasked system management interrupt ($\overline{\text{SMI}}$) source of the highest priority level.

Note that in core disabled mode the user should use SIVCR only read an updated interrupt vector (SMVCR should not be used).

Offset 0x64 — Access: Read only

	0–5	6–24	25–31
R	MVECx	—	MVEC
W			
Reset	All zeros		

Figure 8-27. System Management Interrupt Vector Register (SMVCR)

Table 8-33 defines the bit fields of SMVCR.

Table 8-33. SMVCR Field Descriptions

Bits	Name	Description
0–5	MVECx	Backward (MPC8260) compatible system management interrupt vector. Specifies a 6-bit unique number of the IPIC's highest priority system management interrupt source, pending to the core. When a system management interrupt request occurs, SMVCR can be read. If there are multiple system management interrupt sources, SMVCR latches the highest priority system management interrupt. Note that MVECx f correctly reflects only the first 64 interrupt vectors (See Table 8-6 for details). The value of SMVEC cannot change while it is being read.
6–24	—	Write ignored, read = 0
25–31	MVEC	System management interrupt vector. Specifies a 7-bit unique number of the IPIC's highest priority system management interrupt source, pending to the core. When a system management interrupt request occurs, SMVCR can be read. If there are multiple system management interrupt sources, SMVCR latches the highest priority system management interrupt. Note that MVEC field will correctly reflect all interrupt vectors (See Table 8-6 for details). The value of SMVEC cannot change while it is being read.

8.6 Functional Description

The following sections describe the types of interrupts, interrupt configurations, and their priorities.

8.6.1 Interrupt Types

The IPIC is responsible for receiving hardware-generated interrupts from different sources (both internal and external) along with prioritizing and delivering them to the CPU for servicing. The interrupt sources are controlled by the IPIC unit and may cause three types of exceptions in the processor core. The $\overline{int}$ signal is the main interrupt output from the IPIC to the processor core and causes the external interrupt exception.The $\overline{cint}$ signal is the critical interrupt output from the IPIC to the processor core and causes the critical external interrupt exception. The $\overline{smi}$ signal is the system management interrupt output from the IPIC to the processor core and causes the system management interrupt exception. The machine check exception is caused by the internal $\overline{mcp}$ signal generated by the IPIC, informing the processor of error conditions, assertion of the external MCP request, and other conditions.

8.6.2 Interrupt Configuration

Figure 8-28 shows the interrupt configuration.

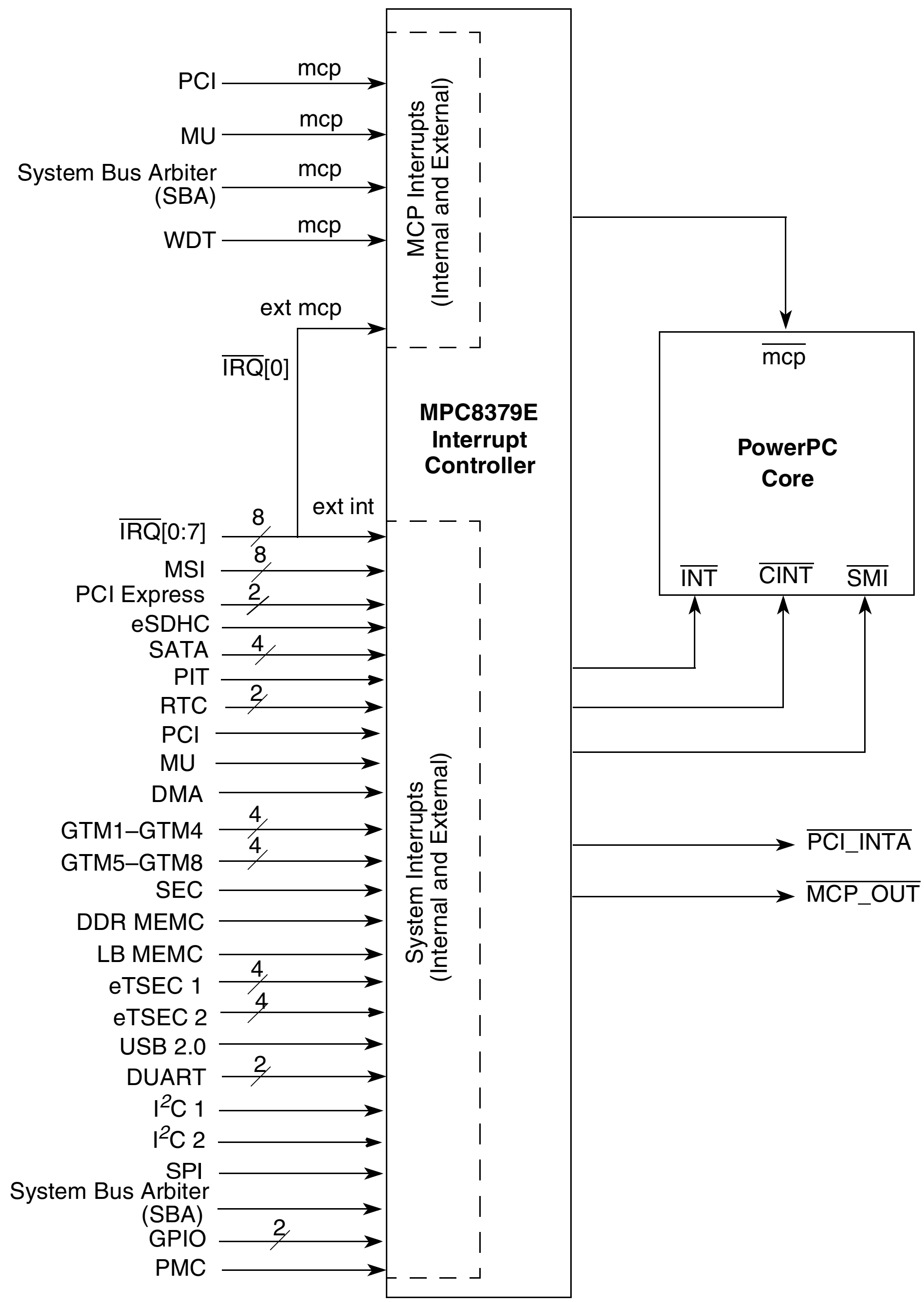


Figure 8-28. Interrupt Structure

The interrupt controller allows masking of each interrupt source. When an unmasked interrupt source is pending in the SIPNR register, the interrupt controller sends an interrupt request to the core. When an interrupt is taken, the interrupt mask bit in the machine state register is cleared to disable further interrupt requests to the PowerPC core until software can handle them.

All interrupt sources are prioritized and bits are set in the system interrupt pending register (SIPNR, SEPNR) as interrupts occur regardless of whether they are masked in the IPIC. The prioritization of the interrupt sources is flexible within the following groups:

- The relative priority of the eTSEC1 Tx, eTSEC1 Rx, eTSEC1 Err, eTSEC2 TX, eTSEC2 Rx, TSEC2 Err, and USB DR internal interrupt signals can be modified.
- The relative priority of the eSDHC, SATA1, SATA2, SATA3, and SATA4 internal interrupt signals can be modified.
- The relative priority of the PCI Express1, PCI Express2, and MSIR1 internal interrupts can be modified.
- The relative priority of the UART1, UART2, SPI, I2C1, I2C2, eTSEC1 1588 timer, eTSEC2 1588 timer, and SEC internal interrupt signals can be modified.
- The relative priority of the IRQ0, IRQ1, IRQ2, and IRQ3 external interrupts, and RTC SEC, PCI, and MSIR0 internal interrupts can be modified.
- The relative priority of the IRQ4, IRQ5, IRQ6, and IRQ7 external interrupts, and RTC ALR, MU, SBA, and DMA internal interrupts can be modified.
- One interrupt source can be assigned to be the programmable highest priority.

The SIVCR is updated with a 7-bit vector corresponding to the sub-block with the highest current priority.

8.6.3 Internal Interrupts Group Relative Priority

The relative priority in each internal group is programmable and can be changed dynamically. The group priorities are programmed in the IPIC internal interrupt priority registers (SIPRR*x*) and can be changed dynamically to implement a rotating priority.

In addition, the grouping of the locations of the interrupt entries has the following two options:

- Grouped.

  In the group scheme, all interrupts are grouped together at the top of Table 8-34, ahead of most other interrupt sources. This scheme is ideal for applications where all interrupt sources function at a very high data rate and interrupt latency is very important.
- Spread.

  In the spread scheme, priorities are spread over Table 8-34 so other sources can have lower interrupt latencies. This scheme is also programmed but cannot be changed dynamically.

8.6.4 Mixed Interrupts Group Relative Priority

The relative priority between up to four internal and four external interrupts in each group is programmable and can be changed dynamically. The group priorities are programmed in the IPIC mixed interrupt priority registers (SMPRR*x*) and can be changed dynamically to implement a rotating priority.

In addition, the grouping of the locations of the mixed interrupt entries has the following two options:

- Grouped.

  In the group scheme, all interrupts are grouped together at the top of the priority table, ahead of most other interrupt sources. See Table 8-34 for more information. This scheme is ideal for applications where all interrupt sources function at a very high data rate and interrupt latency is very important.

- Spread.

  In the spread scheme, priorities are spread over the table so other sources can have lower interrupt latencies. This scheme is also programmed but cannot be changed dynamically.

8.6.5 Highest Priority Interrupt

In addition to the group relative priority option, SICFR[HPI] can be used to specify one interrupt source as having the highest priority. This interrupt remains within the same interrupt level as the other interrupt controller interrupts, but is serviced before any other interrupt in Table 8-34.

If the highest priority feature is not used, the IPIC selects the interrupt request in MIXA0 to be the highest priority interrupt and the standard interrupt priority order is used from Table 8-34. SICFR[HPI] can be updated dynamically to allow the user to change a normally low priority source into a high priority-source for a period as needed.

8.6.6 Interrupt Source Priorities

Each of the IPIC's internal and external interrupt sources can independently assert one interrupt request to the core. Table 8-34 shows the prioritization of these interrupt sources. As described in previous sections, flexibility exists in the relative ordering of the interrupts, but, in general, relative priorities are as shown. A single interrupt priority number is associated with each table entry.

Table 8-34. Interrupt Source Priority Levels

Priority	Interrupt Source Description
1	Highest
2	MIXA0 (Grouped/Spread)
3	MIXA1 (Grouped)
4	MIXA2 (Grouped)
5	MIXA3 (Grouped)
6	MIXB0 (Spread)
7	SYSB0 (Grouped)
8	SYSB1 (Grouped)
9	SYSB2 (Grouped)
10	SYSB3 (Grouped)
11	MIXA1 (Spread)
12	SYSB4 (Grouped)
13	SYSB5 (Grouped)
14	SYSB6 (Grouped)
15	SYSB7 (Grouped)
16	MIXB0 (Grouped)
17	MIXB1 (Grouped)

Table 8-34. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
18	MIXB2 (Grouped)
19	MIXB3 (Grouped)
20	MIXB1 (Spread)
21	SYSA0 (Grouped)
22	SYSA1 (Grouped)
23	SYSA2 (Grouped)
24	SYSA3 (Grouped)
25	MIXA2 (Spread)
26	SYSA4 (Grouped)
27	SYSA5 (Grouped)
28	SYSA6 (Grouped)
29	SYSA7 (Grouped)
30	MIXA4 (Grouped)
31	MIXA5 (Grouped)
32	MIXA6 (Grouped)
33	MIXA7 (Grouped)
34	MIXB2 (Spread)
35	SYSC0 (Grouped)
36	SYSC1 (Grouped)
37	SYSC2 (Grouped)
38	SYSC3 (Grouped)
39	MIXA3 (Spread)
40	SYSC4 (Grouped)
41	SYSC5 (Grouped)
42	SYSC6 (Grouped)
43	SYSC7 (Grouped)
44	MIXB4 (Grouped)
45	MIXB5 (Grouped)
46	MIXB6 (Grouped)
47	MIXB7 (Grouped)
48	MIXB3 (Spread)
49	SYSD0 (Grouped)
50	SYSD1 (Grouped)
51	SYSD2 (Grouped)
52	SYSD3 (Grouped)

Table 8-34. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
53	MIXA4 (Spread)
54	SYSD4 (Grouped)
55	SYSD5 (Grouped)
56	SYSD6 (Grouped)
57	SYSD7 (Grouped)
58	MIXB4 (Spread)
59	GTM4
60	SYSB0 (Spread)
61	SYSA0 (Spread)
62	GTM8
63	SYSC0 (Spread)
64	SYSD0 (Spread)
65	Reserved
66	GPIO1
67	MIXA5 (Spread)
68	GPIO2
69	SYSB1 (Spread)
70	SYSA1 (Spread)
71	DDR
72	SYSC1 (Spread)
73	SYSD1 (Spread)
74	Reserved
75	LBC
76	MIXB5 (Spread)
77	GTM2
78	SYSB2 (Spread)
79	SYSA2 (Spread)
80	GTM6
81	SYSC2 (Spread)
82	SYSD2 (Spread)
83	Reserved
84	PMC
85	MIXA6 (Spread)
86	MSIR2
87	SYSB3 (Spread)
88	SYSA3 (Spread)

Table 8-34. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
89	MSIR3
90	SYSC3 (Spread)
91	SYSD3 (Spread)
92	Reserved
93	Reserved
94	MIXB6 (Spread)
95	GTM3
96	SYSB4 (Spread)
97	SYSA4 (Spread)
98	GTM7
99	SYSC4 (Spread)
100	SYSD4 (Spread)
101	Reserved
102	MSIR4
103	MIXA7 (Spread)
104	MSIR5
105	SYSB5 (Spread)
106	SYSA5 (Spread)
107	MSIR6
108	SYSC5 (Spread)
109	SYSD5 (Spread)
110	Reserved
111	MSIR7
112	MIXB7 (Spread)
113	GTM1
114	SYSB6 (Spread)
115	SYSA6 (Spread)
116	GTM5
117	SYSC6 (Spread)
118	SYSD6 (Spread)
119	Reserved
120	Reserved
121	Reserved
122	SYSB7 (Spread)
123	SYSA7 (Spread)

Table 8-34. Interrupt Source Priority Levels (continued)

Priority	Interrupt Source Description
124	Reserved
125	SYSC7 (Spread)
126	SYSD7 (Spread)
127	Reserved
128	Reserved

8.6.7 Masking Interrupt Sources

By programming the system interrupt mask registers, SIMSR*x* and SEMSR, the user can mask interrupt requests to the core. Each SIMSR*x* and SEMSR bit corresponds to an interrupt source. To enable an interrupt, set the corresponding SIMSR or SEMSR bit. When a masked interrupt source has a pending interrupt request, the corresponding SIPNR*x* or SEMSR bit is set, even though the interrupt is not generated to the core. The user can mask all interrupt sources to implement a polling interrupt servicing scheme.

When an interrupt source has multiple interrupting events, the user can individually mask these events by programming a mask register within that particular block. Table 8-34 shows which interrupt sources have multiple interrupting events.

Figure 8-29 shows an example of how the masking occurs, using a DDR block as an example.

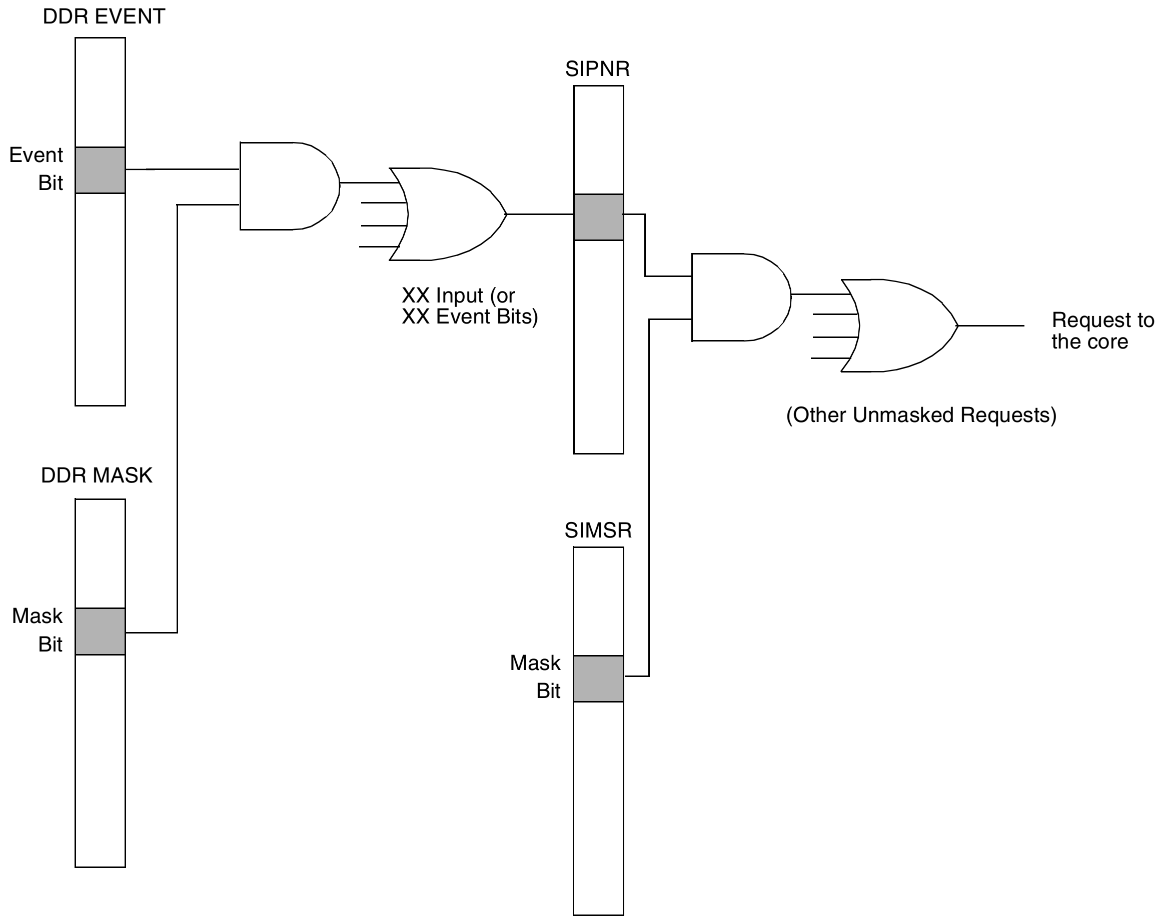


Figure 8-29. DDR Interrupt Request Masking

8.6.8 Interrupt Vector Generation and Calculation

Pending unmasked interrupts are presented to the core in the order of priority shown in Table 8-34. The interrupt vector that allows the core to locate the interrupt service routine is made available to the core by interrupt handler software reading SIVCR. The interrupt controller passes an interrupt vector corresponding to the highest-priority, unmasked, pending interrupt in response to a read of SIVCR. Table 8-5 lists the encodings for the seven low-order bits of the interrupt vector.

8.6.9 Machine Check Interrupts

The PIC supports the non-maskable machine check interrupts. When an error interrupt signal is received, the interrupt controller indicates the source by setting the corresponding SERSR bit. These sources are listed in Table 8-23.

8.7 Message Shared Interrupts

The massage shared interrupt (MSI) registers enable the system end points (PCI agents or PCI express end points) to generate interrupt requests to the local e300 CPU. Each end point can generate an interrupt and set a unique bit in one of the eight MSIR registers. Clearing the MSIR register happens immediately after the read of its content, and by then a new set operation can begin. MSIR*n* is considered active if it contains at least one bit set. Each active non masked MSIR*n* register will generate an interrupt.

8.7.1 Memory Map/Register Definition

The MSI programmable register map occupies 64 bytes of memory-mapped space. The MSI registers are 32 bit wide and must be accessed in a 32 bit read or write operation. The listed addresses are offset from the IPIC base address.

Table 8-35 provides address map for the message shared registers.

Table 8-35. Message Shared Registers Address Map

Offset	Register	Access	Reset	Section/page
0xC0	MSIR0—Message shared interrupt register 0	Special	0x0000_0000	8.7.2.1/8-42
0xC4	MSIR1—Message shared interrupt register 1	Special	0x0000_0000	8.7.2.1/8-42
0xC8	MSIR2—Message shared interrupt register 2	Special	0x0000_0000	8.7.2.1/8-42
0xCC	MSIR3—Message shared interrupt register 3	Special	0x0000_0000	8.7.2.1/8-42
0xD0	MSIR4—Message shared interrupt register 4	Special	0x0000_0000	8.7.2.1/8-42
0xD4	MSIR5—Message shared interrupt register 5	Special	0x0000_0000	8.7.2.1/8-42
0xD8	MSIR6—Message shared interrupt register 6	Special	0x0000_0000	8.7.2.1/8-42
0xDC	MSIR7—Message shared interrupt register 7	Special	0x0000_0000	8.7.2.1/8-42
0xE0–0xEC	Reserved	—	—	—
0xF0	MSIMR—Message shared interrupt mask register	R/W	0x0000_0000	8.7.2.2/8-42
0xF4	MSISR—Message shared interrupt status register	R	0x0000_0000	8.7.2.3/8-43
0xF8	MSIIR—Message shared interrupt index register	W	0x0000_0000	8.7.2.4/8-44
0xFC	Reserved	—	—	—

8.7.2 Message Shared Registers

This section contains the description of all of the message shared interrupt registers.

8.7.2.1 Message Shared Interrupt Register (MSIRs)

Figure 8-30 shows the eight MSIRs, which indicate which of the interrupt sources sharing the message have pending interrupts. Up to 32 sources can share any individual message register. These registers are cleared when read. A write to these registers has no effect.

Offset 0xC0, 0xC4, 0xC8, 0xCC, 0xD0, 0xD4, 0xD8, 0xDC — Access: Special

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SH31	SH30	SH29	SH28	SH27	SH26	SH25	SH24	SH23	SH22	SH21	SH20	SH19	SH18	SH17	SH16
W																
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SH15	SH14	SH13	SH12	SH11	SH10	SH9	SH8	SH7	SH6	SH5	SH4	SH3	SH2	SH1	SH0
W																
Reset	All zeros															

Figure 8-30. Message Shared Interrupt Register (MSIRs)

Table 8-36 describes the bits of the MSIRs.

Table 8-36. MSIRs Field Descriptions

Bits	Name	Description
31–0	SH*n*	Message sharer *n* (*n* = 31–0) has a pending interrupt.

8.7.2.2 Message Shared Interrupt Mask Register (MSIMR)

Figure 8-31 shows the MSIMR, which contains the mask bits for the message shared interrupt register interrupts. The mask bit corresponding to a message shared interrupt register must be clear to enable interrupt generation when the message input region is written and a bit in the message shared interrupt register is set. This is a read-write register.

Offset 0xF0 — Access: Read/Write

	0–23	24	25	26	27	28	29	30	31
R/W	—	M7	M6	M5	M4	M3	M2	M1	M0
Reset	All zeros								

Figure 8-31. Message Shared Interrupt Mask Register (MSIMR)

Table 8-37 describes the bits of the MSIMR.

Table 8-37. MSIMR Field Descriptions

Bits	Name	Description
0–23	—	Reserved.
24	M7	Mask 7. Set to 1 masks interrupt generation for message shared interrupt register 7
25	M6	Mask 6. Set to 1 masks interrupt generation for message shared interrupt register 6
26	M5	Mask 5. Set to 1 masks interrupt generation for message shared interrupt register 5
27	M4	Mask 4. Set to 1 masks interrupt generation for message shared interrupt register 4
28	M3	Mask 3. Set to 1 masks interrupt generation for message shared interrupt register 3
29	M2	Mask 2. Set to 1 masks interrupt generation for message shared interrupt register 2
30	M1	Mask 1. Set to 1 masks interrupt generation for message shared interrupt register 1
31	M0	Mask 0. Set to 1 masks interrupt generation for message shared interrupt register 0

8.7.2.3 Message Shared Interrupt Status Register (MSISR)

Figure 8-32 shows the MSISR, which contains the status bits for the message shared interrupts. The status bit is set to 1 when the corresponding message shared interrupt is active. The status bit is 0 if all the corresponding shared interrupt sources are cleared in the message shared interrupt register (MSIR). This register is read-only.

Offset 0xF4 Access: Read Only

	0–23	24	25	26	27	28	29	30	31
R	—	S7	S6	S5	S4	S3	S2	S1	S0
W									
Reset	All zeros								

Figure 8-32. Message Shared Interrupt Status Register (MSISR)

Table 8-38 describes the bits of the MSISR.

Table 8-38. MSISR Field Descriptions

Bits	Name	Description
0–23	—	Reserved.
24	S7	Status 7. Set to 1 when message shared interrupt 7 is active
25	S6	Status 6. Set to 1 when message shared interrupt 6 is active
26	S5	Status 5. Set to 1 when message shared interrupt 5 is active
27	S4	Status 4. Set to 1 when message shared interrupt 4 is active
28	S3	Status 3. Set to 1 when message shared interrupt 3 is active
29	S2	Status 2. Set to 1 when message shared interrupt 2 is active
30	S1	Status 1. Set to 1 when message shared interrupt 1 is active
31	S0	Status 0. Set to 1 when message shared interrupt 0 is active

8.7.2.4 Message Shared Interrupt Index Register (MSIIR)

Figure 8-33 shows the MSIIR, which provides a mechanism for setting an interrupt in the message shared interrupt registers. There are two fields. When this register is written, one field selects the register in which an interrupt bit is to be set and the other field selects the bit in the selected register to set. This register is write-only.

Offset 0xF8 — Access: Write only

	0–2	3–7	8–31
R			—
W	SRS	IBS	
Reset	All zeros		

Figure 8-33. Message Shared Interrupt Index Register (MSIIR)

Table 8-39 describes the bits of the MSIIRs.

Table 8-39. MSIIR Field Descriptions

Bits	Name	Description
0–2	SRS	Shared interrupt register select. Select the message shared interrupt register. 000 Message shared interrupt register 0 001 Message shared interrupt register 1 010 Message shared interrupt register 2 ... 111 Message shared interrupt register 7
3–7	IBS	Interrupt bit select. Select the bit to set. 00000 Set bit 31 (SH0) 00001 Set bit 30 (SH1) 00010 Set bit 29 (SH2) ... 11111 Set bit 0 (SH31)
8–31	—	Reserved

Chapter 9
DDR Memory Controller

9.1 Introduction

The fully programmable DDR SDRAM controller supports most JEDEC standard ×8, ×16, or ×32 DDR2 and DDR memories available. In addition, unbuffered and registered DIMMs are supported. However, mixing different memory types or unbuffered and registered DIMMs in the same system is not supported. Built-in error checking and correction (ECC) ensures very low bit-error rates for reliable high-frequency operation. Dynamic power management and auto-precharge modes simplify memory system design. A large set of special features, including ECC error injection, support rapid system debug.

NOTE

In this chapter, the word 'bank' refers to a physical bank specified by a chip select; 'logical bank' refers to one of the four or eight sub-banks in each SDRAM chip. A sub-bank is specified by the 2 or 3 bits on the bank address (MBA) pins during a memory access.

Figure 9-1 is a high-level block diagram of the DDR memory controller with its associated interfaces. Section 9.5, “Functional Description,” contains detailed figures of the controller.

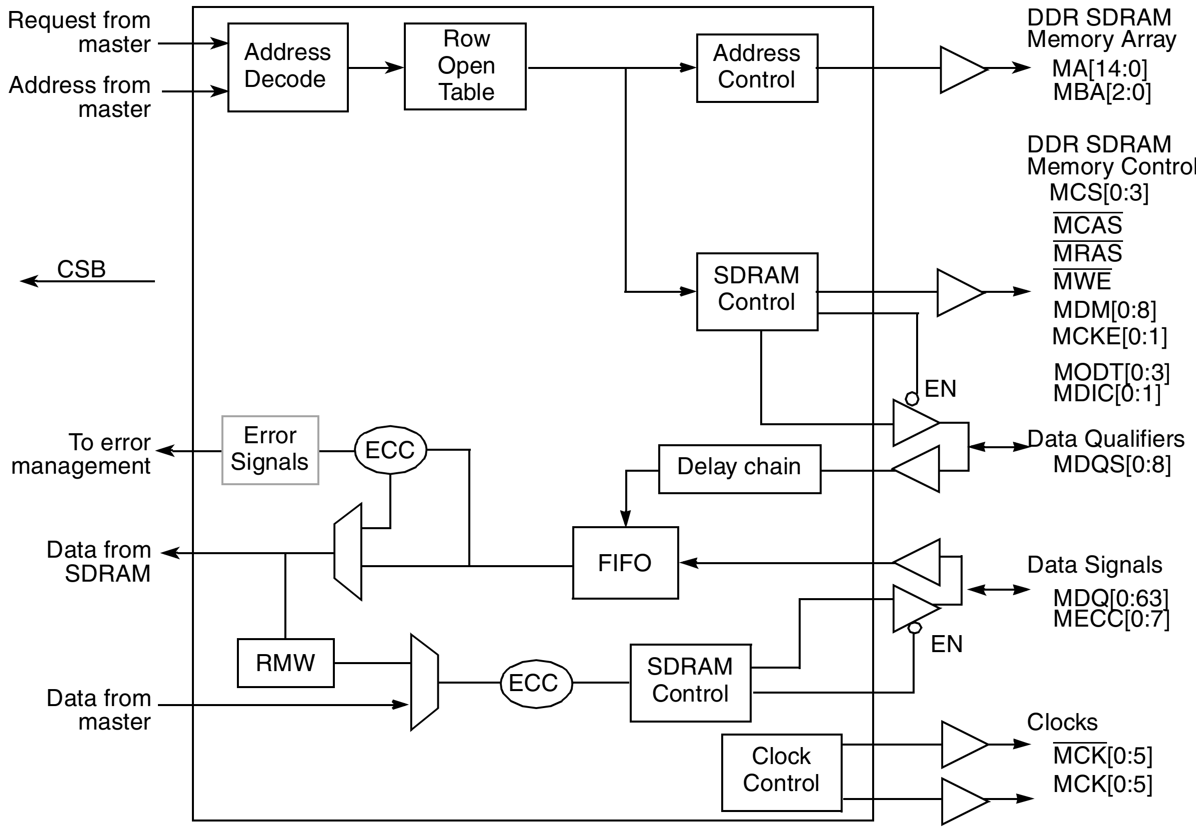


Figure 9-1. DDR Memory Controller Simplified Block Diagram

9.2 Features

The DDR memory controller includes these distinctive features:

- Support for DDR2 and DDR SDRAM
- 64-/72-bit SDRAM data bus. 32-/40-bit SDRAM for DDR and DDR2
- Programmable settings for meeting all SDRAM timing parameters
- The following SDRAM configurations are supported:
  - As many as four physical banks (chip selects), each bank independently addressable
  - 64-Mbit to 4-Gbit devices depending on internal device configuration with x8/x16/x32 data ports (no direct x4 support)
  - Unbuffered and registered DIMMs
- Chip select interleaving support
- Support for data mask signals and read-modify-write for sub-double-word writes. Note that a read-modify-write sequence is only necessary when ECC is enabled.
- Support for double-bit error detection and single-bit error correction ECC (8-bit check word across 64-bit data)

- Open page management (dedicated entry for each logical bank)
- Automatic DRAM initialization sequence or software-controlled initialization sequence
- Automatic DRAM data initialization
- Support for up to eight posted refreshes
- Memory controller clock frequency of two times the SDRAM clock with support for sleep power management
- Support for error injection

9.2.1 Modes of Operation

The DDR memory controller supports the following modes:

- Dynamic power management mode. The DDR memory controller can reduce power consumption by negating the SDRAM CKE signal when no transactions are pending to the SDRAM.
- Auto-precharge mode. Clearing DDR_SDRAM_INTERVAL[BSTOPRE] causes the memory controller to issue an auto-precharge command with every read or write transaction. Auto-precharge mode can be enabled for separate chip selects by setting CS*n*_CONFIG[AP_*n*_EN].

9.3 External Signal Descriptions

This section provides descriptions of the DDR memory controller's external signals. It describes each signal's behavior when the signal is asserted or negated and when the signal is an input or an output.

9.3.1 Signals Overview

Memory controller signals are grouped as follows:

- Memory interface signals
- Clock signals
- Debug signals

Table 9-1 shows how DDR memory controller signals are grouped. The device hardware specification has a pinout diagram showing pin numbers. It also lists all electrical and mechanical specifications.

Table 9-1. DDR Memory Interface Signal Summary

Name	Function/Description	Reset	Pins	I/O
0–63	Data bus	All zeros	64	I
0–8	Data strobes	All zeros	9	I
	Error checking and correcting	All zeros	8	I
	Column address strobe	One	1	O
14–0	Address bus	All zeros	15	O
	Logical bank address	All zeros	3	O

Table 9-1. DDR Memory Interface Signal Summary (continued)

Name	Function/Description	Reset	Pins	I/O
[0–3]	Chip selects	All ones	4	O
	Write enable	One	1	O
	Row address strobe	One	1	O
0–8	Data mask	All zeros	9	O
[0–1]	DRAM clock enable	All zeros	2	O
[0–3]	DRAM on-die termination external control.	All zeros	4	O
	Memory debug data valid	Zero	1	O
	Memory debug source ID	All zeros	5	O
MDIC[0–1]	Driver impedance calibration	High Z	2	I/O

Table 9-2 shows the memory address signal mappings.

Table 9-2. Memory Address Signal Mappings

Signal Name (Outputs)		JEDEC DDR DIMM Signals (Inputs)
msb	MA14	A14
	MA13	A13
	MA12	A12
	MA11	A11
	MA10	A10 (AP for DDR)[1]
	MA9	A9
	MA8	A8 (alternate AP for DDR)[2]
	MA7	A7
	MA6	A6
	MA5	A5
	MA4	A4
	MA3	A3
	MA2	A2
	MA1	A1
lsb	MA0	A0
msb	MBA2	MBA2
	MBA1	MBA1
lsb	MBA0	MBA0

1 Auto-precharge for DDR signaled on A10 when DDR_SDRAM_CFG[PCHB8] = 0.

2 Auto-precharge for DDR signaled on A8 when DDR_SDRAM_CFG[PCHB8] = 1.

9.3.2 Detailed Signal Descriptions

The following sections describe the DDR SDRAM controller input and output signals, the meaning of their different states, and relative timing information for assertion and negation.

9.3.2.1 Memory Interface Signals

Table 9-3 describes the DDR controller memory interface signals.

Table 9-3. Memory Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
0–63	I/O	Data bus. Both input and output signals on the DDR memory controller.	
	O	As outputs for the bidirectional data bus, these signals operate as described below.	
		State Meaning	Asserted/Negated—Represent the value of data being driven by the DDR memory controller.
		Timing	Assertion/Negation—Driven coincident with corresponding data strobes (MDQS) signal. High impedance—No READ or WRITE command is in progress; data is not being driven by the memory controller or the DRAM.
	I	As inputs for the bidirectional data bus, these signals operate as described below.	
		State Meaning	Asserted/Negated—Represents the state of data being driven by the external DDR SDRAMs.
		Timing	Assertion/Negation—The DDR SDRAM drives data during a READ transaction. High impedance—No READ or WRITE command in progress; data is not being driven by the memory controller or the DRAM.
0–8	I/O	Data strobes. Inputs with read data, outputs with write data.	
	O	As outputs, the data strobes are driven by the DDR memory controller during a write transaction. The memory controller always drives these signals low unless a read has been issued and incoming data strobes are expected. This keeps the data strobes from floating high when there are no transactions on the DRAM interface.	
		State Meaning	Asserted/Negated—Driven high when positive capture data is transmitted and driven low when negative capture data is transmitted. Centered in the data “eye” for writes; coincident with the data eye for reads. Treated as a clock. Data is valid when signals toggle. See Table 9-36 for byte lane assignments.
		Timing	Assertion/Negation—If a WRITE command is registered at clock edge *n*, data strobes at the DRAM assert centered in the data eye on clock edge *n* + 1. See the JEDEC DDR SDRAM specification for more information.
	I	As inputs, the data strobes are driven by the external DDR SDRAMs during a read transaction. The data strobes are used by the memory controller to synchronize data latching.	
		State Meaning	Asserted/Negated—Driven high when positive capture data is received and driven low when negative capture data is received. Centered in the data eye for writes; coincident with the data eye for reads. Treated as a clock. Data is valid when signals toggle. See Table 9-36 for byte lane assignments.
		Timing	Assertion/Negation—If a READ command is registered at clock edge *n*, and the latency is programmed in TIMING_CFG_1[CASLAT] to be *m* clocks, data strobes at the DRAM assert coincident with the data on clock edge *n* + *m*. See the JEDEC DDR SDRAM specification for more information.

Table 9-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
	I/O	Error checking and correcting codes. for the DDR controller's bidirectional ECC bus. MECC[0–5] function in both normal and debug modes.	
	O	As normal mode outputs the ECC signals represent the state of ECC driven by the DDR controller on writes. As debug mode outputs MECC[0–5] provide source ID and data-valid information. See Section 9.5.11, "Error Checking and Correcting (ECC)," for more details.	
		State Meaning	Asserted/Negated—Represents the state of ECC being driven by the DDR controller on writes.
		Timing	Assertion/Negation—Same timing as MDQ High impedance—Same timing as MDQ
	I	As inputs, the ECC signals represent the state of ECC driven by the SDRAM devices on reads.	
		State Meaning	Asserted/Negated—Represents the state of ECC being driven by the DDR SDRAMs on reads.
		Timing	Assertion/Negation—Same timing as MDQ High impedance—Same timing as MDQ
14–0	O	Address bus. Memory controller outputs for the address to the DRAM. 14–0 carry 15 of the address bits for the DDR memory interface corresponding to the row and column address bits. MA0 the lsb of the address output from the memory controller.	
		State Meaning	Asserted/Negated—Represents the address driven by the DDR memory controller. Contains different portions of the address depending on the memory size and the DRAM command being issued by the memory controller. See Table 9-40Table 9-41 for a complete description of the mapping of these signals.
		Timing	Assertion/Negation—The address is always driven when the memory controller is enabled. It is valid when a transaction is driven to DRAM (when $\overline{\text{MCS}}n$ is active). High impedance—When the memory controller is disabled
	O	Logical bank address. Outputs that drive the logical (or internal) bank address pins of the SDRAM. Each SDRAM supports four or eight addressable logical sub-banks. Bit zero of the memory controller's output bank address be connected to bit zero of the SDRAM's input bank address. MBA0, the least-significant bit of the three bank address signals, is asserted during the mode register set command to specify the extended mode register.	
		State Meaning	Asserted/Negated—Selects the DDR SDRAM logical (or internal) bank to be activated during the row address phase and selects the SDRAM internal bank for the read or write operation during the column address phase of the memory access. Table 9-40Table 9-41 describes the mapping of these signals in all cases.
		Timing	Assertion/Negation—Same timing as MA*n* High impedance—Same timing as MA*n*
	O	Column address strobe. Active-low SDRAM address multiplexing signal. $\overline{\text{MCAS}}$ is asserted for read or write transactions and for mode register set, refresh, and precharge commands.	
		State Meaning	Asserted—Indicates that a valid SDRAM column address is on the address bus for read and write transactions. See Table 9-45 for more information on the states required on $\overline{\text{MCAS}}$ for various other SDRAM commands. Negated—The column address is not guaranteed to be valid.
		Timing	Assertion/Negation—Assertion and negation timing is directed by the values described in Section 9.4.1.4, "DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)," Section 9.4.1.5, "DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)," Section 9.4.1.6, "DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)," and Section 9.4.1.3, "DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)." High impedance—$\overline{\text{MCAS}}$ is always driven unless the memory controller is disabled.

Table 9-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
	O	Row address strobe. Active-low SDRAM address multiplexing signal. Asserted for activate commands. In addition; used for mode register set commands and refresh commands.	
		State Meaning	Asserted—Indicates that a valid SDRAM row address is on the address bus for read and write transactions. See Table 9-45 for more information on the states required on $\overline{\text{MRAS}}$ for various other SDRAM commands. Negated—The row address is not guaranteed to be valid.
		Timing	Assertion/Negation—Assertion and negation timing is directed by the values described in Section 9.4.1.4, "DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)," Section 9.4.1.5, "DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)," Section 9.4.1.6, "DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)," and Section 9.4.1.3, "DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)." High impedance—$\overline{\text{MRAS}}$ is always driven unless the memory controller is disabled.
[0–3]	O	Chip selects. Four chip selects supported by the memory controller.	
		State Meaning	Asserted—Selects a physical SDRAM bank to perform a memory operation as described in Section 9.4.1.1, "Chip Select Memory Bounds (CSn_BNDS)," and Section 9.4.1.2, "Chip Select Configuration (CSn_CONFIG)." The DDR controller asserts one of the [0–3]signals to begin a memory cycle. Negated—Indicates no SDRAM action during the current cycle.
		Timing	Assertion/Negation—Asserted to signal any new transaction to the SDRAM. The transaction must adhere to the timing constraints set in TIMING_CFG_0–TIMING_CFG_3. High impedance—Always driven unless the memory controller is disabled.
	O	Write enable. Asserted when a write transaction is issued to the SDRAM. This is also used for mode registers set commands and precharge commands.	
		State Meaning	Asserted—Indicates a memory write operation. See Table 9-45 for more information on the states required on $\overline{\text{MWE}}$ for various other SDRAM commands. Negated—Indicates a memory read operation.
		Timing	Assertion/Negation—Similar timing as $\overline{\text{MRAS}}$ and $\overline{\text{MCAS}}$. Used for write commands. High impedance—$\overline{\text{MWE}}$ is always driven unless the memory controller is disabled.
	O	DDR SDRAM data output mask. Masks unwanted bytes of data transferred during a write. They are needed to support sub-burst-size transactions (such as single-byte writes) on SDRAM where all I/O occurs in multi-byte bursts. MDM0 corresponds to the most significant byte (MSB) and MDM7 corresponds to the LSB, while MDM8 corresponds to the ECC byte. Table 9-36 shows byte lane encodings.	
		State Meaning	Asserted—Prevents writing to DDR SDRAM. Asserted when data is written to DRAM if the corresponding byte(s) should be masked for the write. Note that the MDM*n* signals are active-high for the DDR controller. MDM*n* is part of the DDR command encoding. Negated—Allows the corresponding byte to be read from or written to the SDRAM.
		Timing	Assertion/Negation—Same timing as MDQx as outputs. High impedance—Always driven unless the memory controller is disabled.
[0–3]	O	On-Die termination. Memory controller outputs for the ODT to the DRAM. [0–3] represents the on-die termination for the associated data, data masks, ECC, and data strobes.	
		State Meaning	Asserted/Negated—Represents the ODT driven by the DDR memory controller.
		Timing	Assertion/Negation—Driven in accordance with JEDEC DRAM specifications for on-die termination timings. It is configured through the CS*n*_CONFIG[ODT_RD_CFG] and CS*n*_CONFIG[ODT_WR_CFG] fields. High impedance—Always driven.

Table 9-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
MDIC[0–1]	I/O	Driver impedance calibration. Section 5.3.2.8, “DDR Control Driver Register (DDRCDR)	
		State Meaning	These pins are used for automatic calibration of the DDR IOs.
		Timing	These are driven for four DRAM cycles at a time while the DDR controller is executing the automatic driver compensation.

9.3.2.2 Clock Interface Signals

Table 9-4 contains the detailed descriptions of the clock signals of the DDR controller.

Table 9-4. Clock Signals—Detailed Signal Descriptions

Signal	I/O	Description	
MCK[0–5], $\overline{\text{MCK}}$[0–5]	O	DRAM clock outputs and their complements. See Section 9.5.4.1, “Clock Distribution.”	
		State Meaning	Asserted/Negated—The JEDEC DDR SDRAM specifications require true and complement clocks. A clock edge is seen by the SDRAM when the true and complement cross.
		Timing	Assertion/Negation—Timing is controlled by the DDR_CLK_CNTL register at offset 0x130.
[0–1]	O	Clock enable. Output signals used as the clock enables to the SDRAM. [0–1]can be negated to stop clocking the DDR SDRAM. The MCKE signals should be connected to the same rank of memory as the corresponding $\overline{\text{MCS}}$ and MODT signals. For example, MCKE[0] should be connected to the same rank of memory as $\overline{\text{MCS}}$[0] and MODT[0].	
		State Meaning	Asserted—Clocking to the SDRAM is enabled. Negated—Clocking to the SDRAM is disabled and the SDRAM should ignore signal transitions on MCK or $\overline{\text{MCK}}$. MCK/$\overline{\text{MCK}}$ are don’t cares while MCKE[0–1] are negated.
		Timing	Assertion/Negation—Asserted when DDR_SDRAM_CFG[MEM_EN] is set. Can be negated when entering dynamic power management or self refresh. Are asserted again when exiting dynamic power management or self refresh. High impedance—Always driven.

9.3.2.3 Debug Signals

The debug signals and have no function in normal DDR controller operation. Section 5.3.2.7, “Debug Configuration.”

9.4 Memory Map/Register Definition

Table 9-5 shows the register memory map for the DDR memory controller.

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.

- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

Table 9-5. DDR Memory Controller Memory Map

Offset	Register	Access	Reset	Section/Page
DDR Memory Controller—Block Base Address 0x0_2000				
0x000	CS0_BNDS—Chip select 0 memory bounds	R/W	0x0000_0000	9.4.1.1/9-10
0x008	CS1_BNDS—Chip select 1 memory bounds	R/W	0x0000_0000	9.4.1.1/9-10
0x010	CS2_BNDS—Chip select 2 memory bounds	R/W	0x0000_0000	9.4.1.1/9-10
0x018	CS3_BNDS—Chip select 3 memory bounds	R/W	0x0000_0000	9.4.1.1/9-10
0x080	CS0_CONFIG—Chip select 0 configuration	R/W	0x0000_0000	9.4.1.2/9-10
0x084	CS1_CONFIG—Chip select 1 configuration	R/W	0x0000_0000	9.4.1.2/9-10
0x088	CS2_CONFIG—Chip select 2 configuration	R/W	0x0000_0000	9.4.1.2/9-10
0x08C	CS3_CONFIG—Chip select 3 configuration	R/W	0x0000_0000	9.4.1.2/9-10
0x100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	0x0000_0000	9.4.1.3/9-12
0x104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	9.4.1.4/9-13
0x108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	0x0000_0000	9.4.1.5/9-15
0x10C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	0x0000_0000	9.4.1.6/9-17
0x110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	9.4.1.7/9-19
0x114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	0x0000_0000	9.4.1.8/9-22
0x118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	0x0000_0000	9.4.1.9/9-24
0x11C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	0x0000_0000	9.4.1.10/9-25
0x120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	0x0000_0000	9.4.1.11/9-25
0x124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	0x0000_0000	9.4.1.12/9-28
0x128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	0x0000_0000	9.4.1.13/9-28
0x130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	9.4.1.14/9-29
0x148	DDR_INIT_ADDR—DDR training initialization address	R/W	0x0000_0000	9.4.1.15/9-29
0x150–0xBF4	Reserved	—	—	—
0xBF8	DDR_IP_REV1—DDR IP block revision 1	R	0x*nnnn_nnnn*[1]	9.4.1.16/9-30
0xBFC	DDR_IP_REV2—DDR IP block revision 2	R	0x00*nn*_00*nn*[1]	9.4.1.17/9-30
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	0x0000_0000	9.4.1.18/9-31
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	0x0000_0000	9.4.1.19/9-31
0xE08	ERR_INJECT—Memory data path error injection mask ECC	R/W	0x0000_0000	9.4.1.20/9-32
0xE20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	0x0000_0000	9.4.1.21/9-32
0xE24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	0x0000_0000	9.4.1.22/9-33
0xE28	CAPTURE_ECC—Memory data path read capture ECC	R/W	0x0000_0000	9.4.1.23/9-33
0xE40	ERR_DETECT—Memory error detect	w1c	0x0000_0000	9.4.1.24/9-33
0xE44	ERR_DISABLE—Memory error disable	R/W	0x0000_0000	9.4.1.25/9-34
0xE48	ERR_INT_EN—Memory error interrupt enable	R/W	0x0000_0000	9.4.1.26/9-35

Table 9-5. DDR Memory Controller Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0xE4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	0x0000_0000	9.4.1.27/9-36
0xE50	CAPTURE_ADDRESS—Memory error address capture	R/W	0x0000_0000	9.4.1.28/9-37
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	0x0000_0000	9.4.1.29/9-37

[1] Implementation-dependent reset values are listed in specified section/page.

9.4.1 Register Descriptions

This section describes the DDR memory controller registers. Shading indicates reserved fields that should not be written.

9.4.1.1 Chip Select Memory Bounds (CS*n*_BNDS)

The chip select bounds registers (CS*n*_BNDS) define the starting and ending address of the memory space that corresponds to the individual chip selects. Note that the size specified in CS*n*_BNDS should equal the size of physical DRAM. Also, note that EA*n* must be greater than or equal to SA*n*.

If chip select interleaving is enabled, all fields in the lower interleaved chip select are used, and the other chip selects' bounds registers are unused. For example, if chip selects 0 and 1 are interleaved, all fields in CS0_BNDS are used, and all fields in CS1_BNDS are unused.

CS*n*_BNDS are shown in Figure 9-2.

Offset 0x000, 0x008, 0x010, 0x018 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	—	SA*n*	—	EA*n*
Reset	All zeros			

Figure 9-2. Chip Select Bounds Registers (CS*n*_BNDS)

Table 9-6 describes the CS*n*_BNDS register fields.

Table 9-6. CS*n*_BNDS Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	SA*n*	Starting address for chip select (bank) *n*. This value is compared against the 8 msbs of the 32-bit address.
16–23	—	Reserved
24–31	EA*n*	Ending address for chip select (bank) *n*. This value is compared against the 8 msbs of the 32-bit address.

9.4.1.2 Chip Select Configuration (CS*n*_CONFIG)

The chip select configuration (CS*n*_CONFIG) registers shown in Figure 9-3 enable the DDR chip selects and set the number of row and column bits used for each chip select. These registers should be loaded with the correct number of row and column bits for each SDRAM. Because CS*n*_CONFIG[ROW_BITS_CS_*n*, COL_BITS_CS_*n*] establish address multiplexing, the user should take great care to set these values correctly.

If chip select interleaving is enabled, then all fields in the lower interleaved chip select are used, and the other registers' fields are unused, with the exception of the ODT_RD_CFG and ODT_WR_CFG fields. For example, if chip selects 0 and 1 are interleaved, all fields in CS0_CONFIG are used, but only the ODT_RD_CFG and ODT_WR_CFG fields in CS1_CONFIG are used.

Offset 0x080, 0x084, 0x088, 0x08C — Access: Read/Write

	0	1–7	8	9–11	12	13–15
R/W	CS_*n*_EN	—	AP_*n*_EN	ODT_RD_CFG	—	ODT_WR_CFG
Reset	All zeros					

	16–17	18–20	21–23	24–28	29–31
R/W	BA_BITS_CS_*n*	—	ROW_BITS_CS_*n*	—	COL_BITS_CS_*n*
Reset	All zeros				

Figure 9-3. Chip Select Configuration Register (CS*n*_CONFIG)

Table 9-7 describes the CS*n*_CONFIG register fields.

Table 9-7. CS*n*_CONFIG Field Descriptions

Bits	Name	Description
0	CS_*n*_EN	Chip select *n* enable 0 Chip select *n* is not active 1 Chip select *n* is active and assumes the state set in CS*n*_BNDS.
1–7	—	Reserved
8	AP_*n*_EN	Chip select *n* auto-precharge enable 0 Chip select *n* is only auto-precharged if global auto-precharge mode is enabled (DDR_SDRAM_INTERVAL[BSTOPRE] = 0). 1 Chip select *n* always issues an auto-precharge for read and write transactions.
9–11	ODT_RD_CFG	ODT for reads configuration. Note that CAS latency plus additive latency must be at least 3 cycles for ODT_RD_CFG to be enabled. ODT should only be used with DDR2 memories. 000 Never assert ODT for reads 001 Assert ODT only during reads to CS*n* 010 Assert ODT only during reads to other chip selects 011 Assert ODT only during reads to other DIMM modules. It is assumed that CS0 and CS1 are on the same DIMM module, whereas CS2 and CS3 are on a separate DIMM module. 100 Assert ODT for all reads 101–111 Reserved
12	—	Reserved

Table 9-7. CS*n*_CONFIG Field Descriptions (continued)

Bits	Name	Description
13–15	ODT_WR_CFG	ODT for writes configuration. Note that write latency plus additive latency must be at least 3 cycles for ODT _WR_CFG to be enabled. ODT should only be used with DDR2 memories. 000 Never assert ODT for writes 001 Assert ODT only during writes to CS*n* 010 Assert ODT only during writes to other chip selects 011 Assert ODT only during writes to other DIMM modules. It is assumed that CS0 and CS1 are on the same DIMM module, whereas CS2 and CS3 are on a separate DIMM module. 100 Assert ODT for all writes 101–111 Reserved
16–17	BA_BITS_CS_*n*	Number of bank bits for SDRAM on chip select *n*. These bits correspond to the sub-bank bits driven on MBA*n* in Table 9-41Table 9-40and Table 9-41. 00 2 logical bank bits 01 3 logical bank bits 10–11 Reserved
18–20	—	Reserved
21–23	ROW_BITS_CS_*n*	Number of row bits for SDRAM on chip select *n*. See Table 9-41Table 9-40and Table 9-41 for details. 000 12 row bits 001 13 row bits 010 14 row bits 011 15 row bits 000–111 Reserved
24–28	—	Reserved
29–31	COL_BITS_CS_*n*	Number of column bits for SDRAM on chip select *n.* For DDR, the decoding is as follows: 000 8 column bits 001 9 column bits 010 10 column bits 011 11 column bits 100–111 Reserved

9.4.1.3 DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)

DDR SDRAM timing configuration register 3, shown in Figure 9-4, sets the extended refresh recovery time, which is combined with TIMING_CFG_1[REFREC] to determine the full refresh recovery time.

Offset 0x100 Access: Read/Write

	0–12	13–15	16–31
R W	—	EXT_REFREC	—
Reset	All zeros		

Figure 9-4. DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)

Table 9-8 describes TIMING_CFG_3 fields.

Table 9-8. TIMING_CFG_3 Field Descriptions

Bits	Name	Description
0–12	—	Reserved, should be cleared.
13–15	EXT_REFREC	Extended refresh recovery time (t_{RFC}). Controls the number of clock cycles from a refresh command until an activate command is allowed. This field is concatenated with TIMING_CFG_1[REFREC] to obtain a 7-bit value for the total refresh recovery. Note that hardware adds an additional 8 clock cycles to the final, 7-bit value of the refresh recovery. t_{RFC} = {EXT_REFREC || REFREC} + 8, such that t_{RFC} is calculated as follows: 000 0 clocks 001 16 clocks 010 32 clocks 011 48 clocks 100 64 clocks 101 80 clocks 110 96 clocks 111 112 clocks
16–31	—	Reserved, should be cleared.

9.4.1.4 DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)

DDR SDRAM timing configuration register 0, shown in Figure 9-5, sets the number of clock cycles between various SDRAM control commands.

Offset 0x104 — Access: Read/Write

Bits	0–1	2–3	4–5	6–7	8	9–11	12	13–15	16–19	20–23	24–27	28–31
R/W	RWT	WRT	RRT	WWT	—	ACT_PD_EXIT	—	PRE_PD_EXIT	—	ODT_PD_EXIT	—	MRS_CYC
Reset	0 0	0 0	0 0	0 0	0	0 0 1	0	0 0 1	0 0 0 0	0 0 0 1	0 0 0 0	0 1 0 1

Figure 9-5. DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)

Table 9-9 describes TIMING_CFG_0 fields.

Table 9-9. TIMING_CFG_0 Field Descriptions

Bits	Name	Description
0–1	RWT	Read-to-write turnaround (t_{RTW}). Specifies how many extra cycles are added between a read to write turnaround. If 0 clocks is chosen, then the DDR controller uses a fixed number based on the CAS latency and write latency. Choosing a value other than 0 adds extra cycles past this default calculation. As a default the DDR controller determines the read-to-write turnaround as CL – WL + BL/2 + 2. In this equation, CL is the CAS latency rounded up to the next integer, WL is the programmed write latency, and BL is the burst length. 00 0 clocks 01 1 clock 10 2 clocks 11 3 clocks

Table 9-9. TIMING_CFG_0 Field Descriptions (continued)

Bits	Name	Description
2–3	WRT	Write-to-read turnaround. Specifies how many extra cycles are added between a write to read turnaround. If 0 clocks is chosen, then the DDR controller uses a fixed number based on the, read latency, and write latency. Choosing a value other than 0 adds extra cycles past this default calculation. As a default, the DDR controller determines the write-to-read turnaround as WL – CL + BL/2 + 1. In this equation, CL is the CAS latency rounded down to the next integer, WL is the programmed write latency, and BL is the burst length. 00 0 clocks 01 1 clock 10 2 clocks 11 3 clocks
4–5	RRT	Read-to-read turnaround. Specifies how many extra cycles are added between reads to different chip selects. As a default, 3 cycles are required between read commands to different chip selects. Extra cycles may be added with this field. Note: If 8-beat bursts are enabled, then 5 cycles are the default. Note that DDR2 does not support 8-beat bursts. 00 0 clocks 01 1 clock 10 2 clocks 11 3 clocks
6–7	WWT	Write-to-write turnaround. Specifies how many extra cycles are added between writes to different chip selects. As a default, 2 cycles are required between write commands to different chip selects. Extra cycles may be added with this field. Note: If 8-beat bursts are enabled, then 4 cycles are the default. Note that DDR2 does not support 8-beat bursts. 00 0 clocks 01 1 clock 10 2 clocks 11 3 clocks
8	—	Reserved, should be cleared.
9–11	ACT_PD_EXIT	Active powerdown exit timing (t_{XARD} and t_{XARDS}). Specifies how many clock cycles to wait after exiting active powerdown before issuing any command. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
12	—	Reserved, should be cleared.
13–15	PRE_PD_EXIT	Precharge powerdown exit timing (t_{XP}). Specifies how many clock cycles to wait after exiting precharge powerdown before issuing any command. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
16–19	—	Reserved, should be cleared.

Table 9-9. TIMING_CFG_0 Field Descriptions (continued)

Bits	Name	Description
20–23	ODT_PD_EXIT	ODT powerdown exit timing (t_{AXPD}). Specifies how many clocks must pass after exiting powerdown before ODT may be asserted. 0000 0 clock 0001 1 clock 0010 2 clocks 0011 3 clocks 0100 4 clocks 0101 5 clocks 0110 6 clocks 0111 7 clocks 1000 8 clocks 1001 9 clocks 1010 10 clocks 1011 11 clocks 1100 12 clocks 1101 13 clocks 1110 14 clocks 1111 15 clocks
24–27	—	Reserved, should be cleared.
28–31	MRS_CYC	Mode register set cycle time (t_{MRD}). Specifies the number of cycles that must pass after a Mode Register Set command until any other command. 0000 Reserved 0001 1 clock 0010 2 clocks 0011 3 clocks 0100 4 clocks 0101 5 clocks 0110 6 clocks 0111 7 clocks 1000 8 clocks 1001 9 clocks 1010 10 clocks 1011 11 clocks 1100 12 clocks 1101 13 clocks 1110 14 clocks 1111 15 clocks

9.4.1.5 DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)

DDR SDRAM timing configuration register 1, shown in Figure 9-6, sets the number of clock cycles between various SDRAM control commands.

Offset 0x108 Access: Read/Write

	0	1–3	4–7	8	9–11	12–15	16–19	20	21–23	24	25–27	28	29–31
R W	—	PRETOACT	ACTTOPRE	—	ACTTORW	CASLAT	REFREC	—	WRREC	—	ACTTOACT	—	WRTORD
Reset	All zeros												

Figure 9-6. DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)

Table 9-10 describes TIMING_CFG_1 fields.

Table 9-10. TIMING_CFG_1 Field Descriptions

Bits	Name	Description
0	—	Reserved, should be cleared.

Table 9-10. TIMING_CFG_1 Field Descriptions (continued)

Bits	Name	Description
1–3	PRETOACT	Precharge-to-activate interval (t_{RP}). Determines the number of clock cycles from a precharge command until an activate or refresh command is allowed. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
4–7	ACTTOPRE	Activate to precharge interval (t_{RAS}). Determines the number of clock cycles from an activate command until a precharge command is allowed. 0000 16 clocks 0001 17 clocks 0010 18 clocks 0011 19 clocks 0100 4 clocks 0101 5 clocks 0110 6 clocks 0111 7 clocks ... 1111 15 clocks
8	—	Reserved, should be cleared.
9–11	ACTTORW	Activate to read/write interval for SDRAM (t_{RCD}). Controls the number of clock cycles from an activate command until a read or write command is allowed. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
12–15	CASLAT	$\overline{\text{MCAS}}$ latency from READ command. Number of clock cycles between registration of a READ command by the SDRAM and the availability of the first output data. If a READ command is registered at clock edge n and the latency is m clocks, data is available nominally coincident with clock edge $n + m$. This value must be programmed at initialization as described in Section 9.4.1.8, "DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2).") 0000 Reserved 0001 1 clock 0010 1.5 clocks 0011 2 clocks 0100 2.5 clocks 0101 3 clocks 0110 3.5 clocks 0111 4 clocks 1000 4.5 clocks 1001 5 clocks 1010 5.5 clocks 1011 6 clocks 1100 6.5 clocks 1101 7 clocks 1110 7.5 clocks 1111 8 clocks

Table 9-10. TIMING_CFG_1 Field Descriptions (continued)

Bits	Name	Description
16–19	REFREC	Refresh recovery time (t_{RFC}). Controls the number of clock cycles from a refresh command until an activate command is allowed. This field is concatenated with TIMING_CFG_3[EXTREFREC] to obtain a 7-bit value for the total refresh recovery. Note that hardware adds an additional 8 clock cycles to the final, 7-bit value of the refresh recovery, such that t_{RFC} is calculated as follows: t_{RFC} = {EXT_REFREC || REFREC} + 8. 0000 8 clocks 0001 9 clocks 0010 10 clocks 0011 11 clocks ... 1111 23 clocks
20	—	Reserved, should be cleared.
21–23	WRREC	Last data to precharge minimum interval (t_{WR}). Determines the number of clock cycles from the last data associated with a write command until a precharge command is allowed. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
24	—	Reserved, should be cleared.
25–27	ACTTOACT	Activate-to-activate interval (t_{RRD}). Number of clock cycles from an activate command until another activate command is allowed for a different logical bank in the same physical bank (chip select). 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
28	—	Reserved, should be cleared.
29–31	WRTORD	Last write data pair to read command issue interval (t_{WTR}). Number of clock cycles between the last write data pair and the subsequent read command to the same physical bank. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks

9.4.1.6 DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)

DDR SDRAM timing configuration 2, shown in Figure 9-7, sets the clock delay to data for writes.

Offset 0x10C — Access: Read/Write

Bits	0	1–3	4–8	9	10–12	13–15	16–18	19–21	22	23–25	26–31
R/W	—	ADD_LAT	CPO	—	WR_LAT	—	RD_TO_PRE	WR_DATA_DELAY	—	CKE_PLS	FOUR_ACT
Reset	All zeros										

Figure 9-7. DDR SDRAM Timing Configuration 2 Register (TIMING_CFG_2)

Table 9-11 describes the TIMING_CFG_2 fields.

Table 9-11. TIMING_CFG_2 Field Descriptions

Bits	Name	Description
0	—	Reserved
1–3	ADD_LAT	Additive latency. The additive latency must be set to a value less than TIMING_CFG_1[ACTTORW]. (DDR2-specific) 000 0 clocks 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 Reserved 111 Reserved
4–8	CPO[1]	$\overline{\text{MCAS}}$-to-preamble override. Defines the number of DRAM cycles between when a read is issued and when the corresponding DQS preamble is valid for the memory controller. For these decodings, “READ_LAT” is equal to the CAS latency plus the additive latency. 00000 READ_LAT + 1 00001 Reserved 00010 READ_LAT 00011 READ_LAT + 1/4 00100 READ_LAT + 1/2 00101 READ_LAT + 3/4 00110 READ_LAT + 1 00111 READ_LAT + 5/4 01000 READ_LAT + 3/2 01001 READ_LAT + 7/4 01010 READ_LAT + 2 01011 READ_LAT + 9/4 01100 READ_LAT + 5/2 01101 READ_LAT + 11/4 01110 READ_LAT + 3 01111 READ_LAT + 13/4 10000 READ_LAT + 7/2 10001 READ_LAT + 15/4 10010 READ_LAT + 4 10011 READ_LAT + 17/4 10100 READ_LAT + 9/2 10101 READ_LAT + 19/4 10110–11111 Reserved
9	—	Reserved
10–12	WR_LAT	Write latency. Note that the total write latency for DDR2 is equal to WR_LAT + ADD_LAT; the write latency for DDR1 is 1. 000 Reserved 001 1 clock 010 2 clocks 011 3 clocks 100 4 clocks 101 5 clocks 110 6 clocks 111 7 clocks
13–15	—	Reserved
16–18	RD_TO_PRE	Read to precharge (t_{RTP}). For DDR2, with a non-zero ADD_LAT value, takes a minimum of ADD_LAT + t_{RTP} cycles between read and precharge. For DDR1 with burst length of 4, must be set to 010; for DDR1 with burst length of 8, must be set to 100. 000 Reserved 001 1 cycle 010 2 cycles 011 3 cycles 100 4 cycles 101–111 Reserved

Table 9-11. TIMING_CFG_2 Field Descriptions (continued)

Bits	Name	Description
19–21	WR_DATA_DELAY	Write command to write data strobe timing adjustment. Controls the amount of delay applied to the data and data strobes for writes. See Section 9.5.7, “DDR SDRAM Write Timing Adjustments,” for details. 000 0 clock delay 001 1/4 clock delay 010 1/2 clock delay 011 3/4 clock delay 100 1 clock delay 101 5/4 clock delay 110 3/2 clock delay 111 Reserved
22	—	Reserved
23–25	CKE_PLS	Minimum CKE pulse width (t_{CKE})Can be set to 001 for DDR1. 000 Reserved 001 1 cycle 010 2 cycles 011 3 cycles 100 4 cycles 101–111 Reserved
26–31	FOUR_ACT	Window for four activates (t_{FAW}). This is applied to DDR2 with eight logical banks only. Must be set to 000001 for DDR1. 000000 Reserved 000001 1 cycle 000010 2 cycles 000011 3 cycles 000100 4 cycles ... 01001119 cycles 010100 20 cycles 010101–111111 Reserved

[1] For CPO decodings other than 00000 and 11111, ‘READ_LAT’ is rounded up to the next integer value.

9.4.1.7 DDR SDRAM Control Configuration (DDR_SDRAM_CFG)

The DDR SDRAM control configuration register, shown in Figure 9-8, enables the interface logic and specifies certain operating features such as self refreshing, error checking and correcting, registered DIMMs, and dynamic power management.

Offset 0x110 Access: Read/Write

Bits	0	1	2	3	4	5–7	8–9	10	11	12	13	14	15
R/W	MEM_EN	SREN	ECC_EN	RD_EN	—	SDRAM_TYPE	—	DYN_PWR	—	32_BE	8_BE	NCAP	—
Reset	0	0	0	0	0	0 1 0	0 0	0	0	0	0	0	0

Bits	16	17–23	24–25	26	27	28	29	30	31
R/W	2T_EN	BA_INTLV_CTL	—	x32_EN	PCHB8	HSE	—	MEM_HALT	BI
Reset	All zeros								

Figure 9-8. DDR SDRAM Control Configuration Register (DDR_SDRAM_CFG)

Table 9-12 describes the DDR_SDRAM_CFG fields.

Table 9-12. DDR_SDRAM_CFG Field Descriptions

Bits	Name	Description
0	MEM_EN	DDR SDRAM interface logic enable. 0 SDRAM interface logic is disabled. 1 SDRAM interface logic is enabled. Must not be set until all other memory configuration parameters have been appropriately configured by initialization code.
1	SREN	Self refresh enable (during sleep). 0 SDRAM self refresh is disabled during sleep. Whenever self-refresh is disabled, the system is responsible for preserving the integrity of SDRAM during sleep. 1 SDRAM self refresh is enabled during sleep.
2	ECC_EN	ECC enable. Note that uncorrectable read errors may cause an interrupt. 0 No ECC errors are reported. No ECC interrupts are generated. 1 ECC is enabled.
3	RD_EN	Registered DIMM enable. Specifies the type of DIMM used in the system. 0 Indicates unbuffered DIMMs. 1 Indicates registered DIMMs. **Note:** RD_EN and 2T_EN must not both be set at the same time.
4	—	Reserved
5–7	SDRAM_TYPE	Type of SDRAM device to be used. This field is used when issuing the automatic hardware initialization sequence to DRAM through Mode Register Set and Extended Mode Register Set commands. Default value is 010 designating DDR1 SDRAM. 000–001 Reserved 010 DDR1 SDRAM 011 DDR2 SDRAM 100 Reserved 101 Reserved 110 111 Reserved
8–9	—	Reserved
10	DYN_PWR	Dynamic power management mode 0 Dynamic power management mode is disabled. 1 Dynamic power management mode is enabled. If there is no ongoing memory activity, the SDRAM CKE signal is negated.
11	—	Reserved
12	32_BE	32-bit bus enable. 0 64-bit bus is used. 1 32-bit bus is used.
13	8_BE	8-beat burst enable. 0 4-beat bursts are used on the DRAM interface. 1 8-beat bursts are used on the DRAM interface. **Note:** DDR1 (SDRAM_TYPE = 010) must use 8-beat bursts when using 32-bit bus mode (32_BE = 1) and 4-beat bursts when using 64-bit bus mode; DDR2 (SDRAM_TYPE = 011) must use 4-beat bursts, even when using 32-bit bus mode

Table 9-12. DDR_SDRAM_CFG Field Descriptions (continued)

Bits	Name	Description
14	NCAP	Non-concurrent auto-precharge. Some older DDR DRAMs do not support concurrent auto precharge. If one of these devices is used, then this bit needs to be set if auto precharge is used. 0 DRAMs in system support concurrent auto-precharge. 1 DRAMs in system do not support concurrent auto-precharge.
15	—	Reserved
16	2T_EN	Enable 2T timing. 0 1T timing is enabled. The DRAM command/address are held for only 1 cycle on the DRAM bus. 1 2T timing is enabled. The DRAM command/address are held for 2 full cycles on the DRAM bus for every DRAM transaction. However, the chip select is only held for the second cycle. **Note:** RD_EN and 2T_EN must not both be set at the same time.
17–23	BA_INTLV_CTL	Bank (chip select) interleaving control. Set this field only if you wish to use bank interleaving. (‘*x*’ denotes a don’t care bit value. All unlisted field values are reserved.) 0000000 No external memory banks are interleaved 1000000 External memory banks 0 and 1 are interleaved 0100000 External memory banks 2 and 3 are interleaved 1100000 External memory banks 0 and 1 are interleaved together **and** banks 2 and 3 are interleaved together *xx*00100 External memory banks 0 through 3 are all interleaved together
24–25	—	Reserved
26	x32_EN	x32 enable. 0 Either x8 or x16 discrete DRAM chips are used. In this mode, each data byte has a dedicated corresponding data strobe. 1 x32 discrete DRAM chips are used. In this mode, DQS0 is used to capture DQ[0–31], DQS4 is used to capture DQ[32–63] and DQS8 is used to capture ECC[0–7].
27	PCHB8	Precharge bit 8 enable. 0 MA[10] is used to indicate the auto-precharge and precharge all commands. 1 MA[8] is used to indicate the auto-precharge and precharge all commands. If x32_EN is cleared, then PCHB8 should be cleared as well.
28	HSE	Global half-strength override Sets I/O driver impedance to half strength. This impedance is used by the MDIC, address/command, data, and clock impedance values, but only if automatic hardware calibration is disabled and the corresponding group's software override is disabled in the DDR control driver register(s) described in Section 5.3.2.8, “DDR Control Driver Register (DDRCDR).” This bit should be cleared if using automatic hardware calibration. 0 I/O driver impedance is configured to full strength. 1 I/O driver impedance is configured to half strength.
29	—	Reserved

Table 9-12. DDR_SDRAM_CFG Field Descriptions (continued)

Bits	Name	Description
30	MEM_HALT	DDR memory controller halt. When this bit is set, the memory controller does not accept any new data read/write transactions to DDR SDRAM until the bit is cleared again. This can be used when bypassing initialization and forcing MODE REGISTER SET commands through software. 0 DDR controller accepts new transactions. 1 DDR controller finishes any remaining transactions, and then it remains halted until this bit is cleared by software.
31	BI	Bypass initialization 0 DDR controller cycles through initialization routine based on SDRAM_TYPE 1 Initialization routine is bypassed. Software is responsible for initializing memory through DDR_SDRAM_MODE2 register. If software is initializing memory, then the MEM_HALT bit can be set to prevent the DDR controller from issuing transactions during the initialization sequence. Note that the DDR controller does not issue a DLL reset to the DRAMs when bypassing the initialization routine, regardless of the value of DDR_SDRAM_CFG[DLL_RST_DIS]. If a DLL reset is required, then the controller should be forced to enter and exit self refresh after the controller is enabled. See Section 9.4.1.15, “DDR Initialization Address (DDR_INIT_ADDR),” for details on avoiding ECC errors in this mode.

9.4.1.8 DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)

The DDR SDRAM control configuration register 2, shown in Figure 9-9, provides more control configuration for the DDR controller.

Offset 0x114 Access: Read/Write

	0	1	2	3	4–5	6–8	9–10	11–15
R/W	FRC_SR	SR_IE	DLL_RST_DIS	—	DQS_CFG	—	ODT_CFG	—
Reset	All zeros							

	16–19	20–26	27	28–31
R/W	NUM_PR	—	D_INIT	—
Reset	All zeros			

Figure 9-9. DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2)

Table 9-13 describes the DDR_SDRAM_CFG_2 fields.

Table 9-13. DDR_SDRAM_CFG_2 Field Descriptions

Bits	Name	Description
0	FRC_SR	Force self refresh 0 DDR controller operates in normal mode. 1 DDR controller enters self-refresh mode.
1	SR_IE	Self-refresh interrupt enable. The DDR controller can be placed into self refresh mode if DDR_SR_REQ (IRQ1) is asserted. This is considered a 'panic interrupt' by the DDR controller, and it will enter self refresh as soon as possible. DDR_SDRAM_CFG[SREN] must also be set if the panic interrupt is used. 0 DDR controller will not enter self-refresh mode if panic interrupt is asserted. 1 DDR controller will enter self-refresh mode if panic interrupt is asserted.
2	DLL_RST_DIS	DLL reset disable. The DDR controller typically issues a DLL reset to the DRAMs when exiting self refresh. However, this function may be disabled by setting this bit during initialization. 0 DDR controller issues a DLL reset to the DRAMs when exiting self refresh. 1 DDR controller does not issue a DLL reset to the DRAMs when exiting self refresh.
3	—	Reserved
4–5	DQS_CFG	DQS configuration 00 Only true DQS signals are used. 01 Reserved 10 Reserved 11 Reserved
6–8	—	Reserved
9–10	ODT_CFG	ODT configuration. This field defines how ODT is driven to the on-chip IOs. See Section 5.3.2.8, "DDR Control Driver Register (DDRCDR)," which defines the termination value that is used. (DDR2-specific, must be cleared for DDR1) 00 Never assert ODT to internal IOs 01 Assert ODT to internal IOs only during writes to DRAM 10 Assert ODT to internal IOs only during reads to DRAM 11 Always keep ODT asserted to internal IOs
11–15	—	Reserved.
16–19	NUM_PR	Number of posted refreshes. This determines how many posted refreshes, if any, can be issued at one time. Note that if posted refreshes are used, then this field, along with DDR_SDRAM_INTERVAL[REFINT], must be programmed such that the maximum t_{ras} specification cannot be violated. For example, some DDR1 SDRAMs are not able to use more than 3 posted refreshes because the required refresh interval could then exceed the maximum constraint for t_{ras}. 0000 Reserved 0001 1 refresh is issued at a time 0010 2 refreshes is issued at a time 0011 3 refreshes is issued at a time ... 1000 8 refreshes is issued at a time 1001–1111 Reserved
20–26	—	Reserved, should be cleared.

Table 9-13. DDR_SDRAM_CFG_2 Field Descriptions (continued)

Bits	Name	Description
27	D_INIT	DRAM data initialization. This bit is set by software, and it is cleared by hardware. If software sets this bit before the memory controller is enabled, the controller automatically initializes DRAM after it is enabled. This bit is automatically cleared by hardware once the initialization is completed. This data initialization bit should only be set when the controller is idle. 0 There is not data initialization in progress, and no data initialization is scheduled 1 The memory controller initializes memory once it is enabled. This bit remains asserted until the initialization is complete. The value in DDR_DATA_INIT register is used to initialize memory.
28–31	—	Reserved

9.4.1.9 DDR SDRAM Mode Configuration (DDR_SDRAM_MODE)

The DDR SDRAM mode configuration register, shown in Figure 9-10, sets the values loaded into the DDR's mode registers.

Figure 9-10. DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE)

Table 9-14 describes the DDR_SDRAM_MODE fields.

Table 9-14. DDR_SDRAM_MODE Field Descriptions

Bits	Name	Description
0–15	ESDMODE	Extended SDRAM mode. Specifies the initial value loaded into the DDR SDRAM extended mode register. The range and meaning of legal values is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus (during the DDR SDRAM initialization sequence), MA[0] presents the lsb of ESDMODE, which, in the big-endian convention shown in Figure 9-10, corresponds to ESDMODE[15]. The msb of the SDRAM extended mode register value must be stored at ESDMODE[0].
16–31	SDMODE	SDRAM mode. Specifies the initial value loaded into the DDR SDRAM mode register. The range of legal values is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus (during DDR SDRAM initialization), MA[0] presents the lsb of SDMODE, which, in the big-endian convention shown in Figure 9-10, corresponds to SDMODE[15]. The msb of the SDRAM mode register value must be stored at SDMODE[0]. Because the memory controller forces SDMODE[7] to certain values depending on the state of the initialization sequence, (for resetting the SDRAM's DLL) the corresponding bits of this field are ignored by the memory controller. Note that SDMODE[7] is mapped to MA[8].

9.4.1.10 DDR SDRAM Mode 2 Configuration (DDR_SDRAM_MODE_2)

The DDR SDRAM mode 2 configuration register, shown in Figure 9-11, sets the values loaded into the DDR's extended mode 2 and 3 registers (for DDR2).

Offset 0x11C — Access: Read/Write

	0–15	16–31
R/W	ESDMODE2	ESDMODE3
Reset	All zeros	

Figure 9-11. DDR SDRAM Mode 2 Configuration Register (DDR_SDRAM_MODE_2)

Table 9-15 describes the DDR_SDRAM_MODE_2 fields.

Table 9-15. DDR_SDRAM_MODE_2 Field Descriptions

Bits	Name	Description
0–15	ESDMODE2	Extended SDRAM mode 2. Specifies the initial value loaded into the DDR SDRAM extended 2 mode register. The range and meaning of legal values is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus (during the DDR SDRAM initialization sequence), MA[0] presents the lsb bit of ESDMODE2, which, in the big-endian convention shown in Figure 9-11, corresponds to ESDMODE2[15]. The msb of the SDRAM extended mode 2 register value must be stored at ESDMODE2[0].
16–31	ESDMODE3	Extended SDRAM mode 3. Specifies the initial value loaded into the DDR SDRAM extended 3 mode register. The range of legal values of legal values is specified by the DDR SDRAM manufacturer. When this value is driven onto the address bus (during DDR SDRAM initialization), MA[0] presents the lsb of ESDMODE3, which, in the big-endian convention shown in Figure 9-11, corresponds to ESDMODE3[15]. The msb of the SDRAM extended mode 3 register value must be stored at ESDMODE3[0].

9.4.1.11 DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)

The DDR SDRAM mode control register, shown in Figure 9-12, allows the user to carry out the following tasks:

- Issue a mode register set command to a particular chip select
- Issue an immediate refresh to a particular chip select
- Issue an immediate precharge or precharge all command to a particular chip select
- Force the CKE signals to a specific value

Table 9-16 describes the fields of this register. Table 9-17 shows the user how to set the fields of this register to accomplish the above tasks.

Offset 0x120 — Access: Read/Write

	0	1	2–3	4	5–7	8	9	10–11	12–15	16–31
R/W	MD_EN	—	CS_SEL	—	MD_SEL	SET_REF	SET_PRE	CKE_CNTL	—	MD_VALUE
Reset	All zeros									

Figure 9-12. DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)

Table 9-16 describes the DDR_SDRAM_MD_CNTL fields.

NOTE

Note that MD_EN, SET_REF, and SET_PRE are mutually exclusive; only one of these fields can be set at a time.

Table 9-16. DDR_SDRAM_MD_CNTL Field Descriptions

Bits	Name	Description
0	MD_EN	Mode enable. Setting this bit specifies that valid data in MD_VALUE is ready to be written to DRAM as one of the following commands: • MODE REGISTER SET • EXTENDED MODE REGISTER SET • EXTENDED MODE REGISTER SET 2 • EXTENDED MODE REGISTER SET 3 The specific command to be executed is selected by setting MD_SEL. In addition, the chip select must be chosen by setting CS_SEL. MD_EN is set by software and cleared by hardware once the command has been issued. 0 Indicates that no mode register set command needs to be issued. 1 Indicates that valid data contained in the register is ready to be issued as a mode register set command.
1	—	Reserved
2–3	CS_SEL	Select chip select. Specifies the chip select that is driven active due to any command forced by software in DDR_SDRAM_MD_CNTL. 00 Chip select 0 is active 01 Chip select 1 is active 10 Chip select 2 is active 11 Chip select 3 is active
4	—	Reserved
5–7	MD_SEL	Mode register select. MD_SEL specifies one of the following: • During a mode select command, selects the SDRAM mode register to be changed • During a precharge command, selects the SDRAM logical bank to be precharged. A precharge all command ignores this field. • During a refresh command, this field is ignored. Note that MD_SEL contains the value that is presented onto the memory bank address pins (MBA*n*) of the DDR controller. 000 MR 001 EMR 010 EMR2 011 EMR3
8	SET_REF	Set refresh. Forces an immediate refresh to be issued to the chip select specified by DDR_SDRAM_MD_CNTL[CS_SEL]. This bit is set by software and cleared by hardware once the command has been issued. 0 Indicates that no refresh command needs to be issued. 1 Indicates that a refresh command is ready to be issued.
9	SET_PRE	Set precharge. Forces a precharge or precharge all to be issued to the chip select specified by DDR_SDRAM_MD_CNTL[CS_SEL]. This bit is set by software and cleared by hardware once the command has been issued. 0 Indicates that no precharge all command needs to be issued. 1 Indicates that a precharge all command is ready to be issued.

Table 9-16. DDR_SDRAM_MD_CNTL Field Descriptions (continued)

Bits	Name	Description
10–11	CKE_CNTL	Clock enable control. Allows software to globally clear or set all CKE signals issued to DRAM. Once software has forced the value driven on CKE, that value continues to be forced until software clears the CKE_CNTL bits. At that time, the DDR controller continues to drive the CKE signals to the same value forced by software until another event causes the CKE signals to change (such as, self refresh entry/exit, power down entry/exit). 00 CKE signals are not forced by software. 01 CKE signals are forced to a low value by software. 10 CKE signals are forced to a high value by software. 11 Reserved
12–15	—	Reserved
16–31	MD_VALUE	Mode register value. This field, which specifies the value that is presented on the memory address pins of the DDR controller during a mode register set command, is significant only when this register is used to issue a mode register set command or a precharge or precharge all command. For a mode register set command, this field contains the data to be written to the selected mode register. For a precharge command, only bit five is significant: 0 Issue a precharge command; MD_SEL selects the logical bank to be precharged 1 Issue a precharge all command; all logical banks are precharged

Table 9-17 shows how DDR_SDRAM_MD_CNTL fields should be set for each of the tasks described above.

Table 9-17. Settings of DDR_SDRAM_MD_CNTL Fields

Field	Mode Register Set	Refresh	Precharge	Clock Enable Signals Control
MD_EN	1	0	0	—
SET_REF	0	1	0	—
SET_PRE	0	0	1	—
CS_SEL	Chooses chip select (CS)			—
MD_SEL	Select mode register. See Table 9-16.	—	Selects logical bank	—
MD_VALUE	Value written to mode register	—	Only bit five is significant. See Table 9-16.	—
CKE_CNTL	0	0	0	See Table 9-16.

9.4.1.12 DDR SDRAM Interval Configuration (DDR_SDRAM_INTERVAL)

The DDR SDRAM interval configuration register, shown in Figure 9-13, sets the number of DRAM clock cycles between bank refreshes issued to the DDR SDRAMs. In addition, the number of DRAM cycles that a page is maintained after it is accessed is provided here.

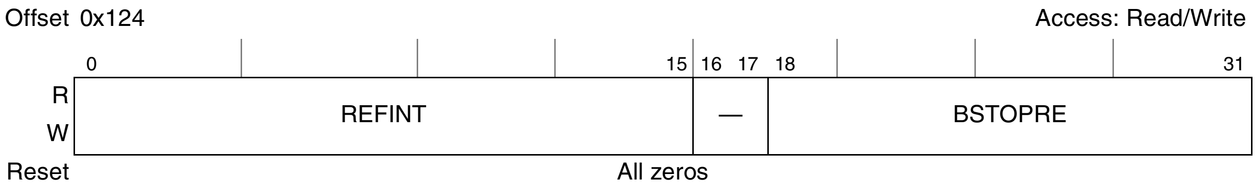


Figure 9-13. DDR SDRAM Interval Configuration Register (DDR_SDRAM_INTERVAL)

Table 9-18 describes the DDR_SDRAM_INTERVAL fields.

Table 9-18. DDR_SDRAM_INTERVAL Field Descriptions

Bits	Name	Description
0–15	REFINT	Refresh interval. Represents the number of memory bus clock cycles between refresh cycles. Depending on DDR_SDRAM_CFG_2[NUM_PR], some number of rows are refreshed in each DDR SDRAM physical bank during each refresh cycle. The value for REFINT depends on the specific SDRAMs used and the interface clock frequency. Refreshes are not issued when the REFINT is set to all 0s.
16–17	—	Reserved
18–31	BSTOPRE	Precharge interval. Sets the duration (in memory bus clocks) that a page is retained after a DDR SDRAM access. If BSTOPRE is zero, the DDR memory controller uses auto-precharge read and write commands rather than operating in page mode. This is called global auto-precharge mode.

9.4.1.13 DDR SDRAM Data Initialization (DDR_DATA_INIT)

The DDR SDRAM data initialization register, shown in Figure 9-14, provides the value that is used to initialize memory if DDR_SDRAM_CFG2[D_INIT] is set.

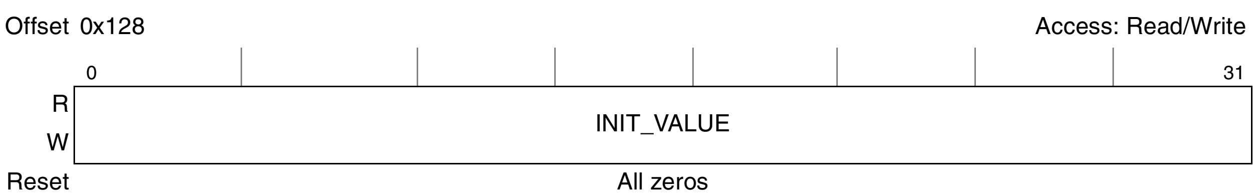


Figure 9-14. DDR SDRAM Data Initialization Configuration Register (DDR_DATA_INIT)

Table 9-19 describes the DDR_DATA_INIT fields.

Table 9-19. DDR_DATA_INIT Field Descriptions

Bits	Name	Description
0–31	INIT_VALUE	Initialization value. Represents the value that DRAM is initialized with if DDR_SDRAM_CFG2[D_INIT] is set.

9.4.1.14 DDR SDRAM Clock Control (DDR_SDRAM_CLK_CNTL)

The DDR SDRAM clock control configuration register, shown in Figure 9-15, provides a 1/4-cycle clock adjustment.

Offset 0x130 — Access: Read/Write

Bits	0–4	5–7	8–31
R/W	—	CLK_ADJUST	—
Reset	0 0 0 0 0	0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 9-15. DDR SDRAM Clock Control Configuration Register (DDR_SDRAM_CLK_CNTL)

Table 9-20 describes the DDR_SDRAM_CLK_CNTL fields.

Table 9-20. DDR_SDRAM_CLK_CNTL Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–7	CLK_ADJUST	Clock adjust. 000 Clock is launched aligned with address/command 001 Clock is launched 1/4 applied cycle after address/command 010 Clock is launched 1/2 applied cycle after address/command 011 Clock is launched 3/4 applied cycle after address/command 100 Clock is launched 1 applied cycle after address/command 101–111 Reserved
8	—	Reserved, should be cleared.
9–31	—	Reserved

9.4.1.15 DDR Initialization Address (DDR_INIT_ADDR)

The DDR SDRAM initialization address register, shown in Figure 9-16, provides the address that is used for the automatic $\overline{CAS}$ to preamble calibration after POR.

Offset 0x148 — Access: Read/Write

Bits	0–31
R/W	INIT_ADDR
Reset	All zeros

Figure 9-16. DDR Initialization Address Configuration Register (DDR_INIT_ADDR)

Table 9-21 describes the DDR_INIT_ADDR fields.

Table 9-21. DDR_INIT_ADDR Field Descriptions

Bits	Name	Description
0–31	INIT_ADDR	Initialization address. Represents the address that is used for the automatic CAS to preamble calibration at POR.

9.4.1.16 DDR IP Block Revision 1 (DDR_IP_REV1)

The DDR IP block revision 1 register, shown in Figure 9-17, provides read-only fields with the IP block ID, along with major and minor revision information.

Offset 0xBF8 — Access: Read Only

	0–15	16–23	24–31
R	IP_ID	IP_MJ	IP_MN
W			
Reset	n[1] n n n n n n n n n n n n n n n	n n n n n n n n	n n n n n n n n

Figure 9-17. DDR IP Block Revision 1 (DDR_IP_REV1)

[1] For reset values, see Table 9-22.

Table 9-22 describes the DDR_IP_REV1 fields.

Table 9-22. DDR_IP_REV1 Field Descriptions

Bits	Name	Description
0–15	IP_ID	IP block ID. For the DDR controller, this value is 0x0002.
16–23	IP_MJ	Major revision. This is currently set to 0x02.
24–31	IP_MN	Minor revision. This is currently set to 0x00.

9.4.1.17 DDR IP Block Revision 2 (DDR_IP_REV2)

The DDR IP block revision 2 register, shown in Figure 9-18, provides read-only fields with the IP block integration and configuration options.

Offset 0xBFC — Access: Read Only

	0–7	8–15	16–23	24–31
R	—	IP_INT	—	IP_CFG
W				
Reset	0 0 0 0 0 0 0 0	n n n n n n n n	0 0 0 0 0 0 0 0	n n n n n n n n

Figure 9-18. DDR IP Block Revision 2 (DDR_IP_REV2)

Table 9-23 describes the DDR_IP_REV2 fields.

Table 9-23. DDR_IP_REV2 Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	IP_INT	IP block integration options
16–23	—	Reserved
24–31	IP_CFG	IP block configuration options

9.4.1.18 Memory Data Path Error Injection Mask High (DATA_ERR_INJECT_HI)

The memory data path error injection mask high register is shown in Figure 9-19.

Offset 0xE00 Access: Read/Write

	0–31
R	EIMH
W	
Reset	All zeros

Figure 9-19. Memory Data Path Error Injection Mask High Register (DATA_ERR_INJECT_HI)

Table 9-24 describes the DATA_ERR_INJECT_HI fields.

Table 9-24. DATA_ERR_INJECT_HI Field Descriptions

Bits	Name	Description
0–31	EIMH	Error injection mask high data path. Used to test ECC by forcing errors on the high word of the data path. Setting a bit causes the corresponding data path bit to be inverted on memory bus writes.

9.4.1.19 Memory Data Path Error Injection Mask Low (DATA_ERR_INJECT_LO)

The memory data path error injection mask low register is shown in Figure 9-20.

Offset 0xE04 Access: Read/Write

	0–31
R	EIML
W	
Reset	All zeros

Figure 9-20. Memory Data Path Error Injection Mask Low Register (DATA_ERR_INJECT_LO)

Table 9-25 describes the DATA_ERR_INJECT_LO fields.

Table 9-25. DATA_ERR_INJECT_LO Field Descriptions

Bits	Name	Description
0–31	EIML	Error injection mask low data path. Used to test ECC by forcing errors on the low word of the data path. Setting a bit causes the corresponding data path bit to be inverted on memory bus writes.

9.4.1.20 Memory Data Path Error Injection Mask ECC (ERR_INJECT)

The memory data path error injection mask ECC register, shown in Figure 9-21, sets the ECC mask, enables errors to be written to ECC memory, and allows the ECC byte to mirror the most significant data byte.

Offset 0xE08 — Access: Read/Write

	0–21	22	23	24–31
R/W	—	EMB	EIEN	EEIM
Reset	All zeros			

Figure 9-21. Memory Data Path Error Injection Mask ECC Register (ERR_INJECT)

Table 9-26 describes the ERR_INJECT fields.

Table 9-26. ERR_INJECT Field Descriptions

Bits	Name	Description
0–21	—	Reserved
22	EMB	ECC mirror byte 0 Mirror byte functionality disabled. 1 Mirror the most significant data path byte onto the ECC byte.
23	EIEN	Error injection enable 0 Error injection disabled. 1 Error injection enabled. This applies to the data mask bits, the ECC mask bits, and the ECC mirror bit. Note that error injection should not be enabled until the memory controller has been enabled through DDR_SDRAM_CFG[MEM_EN].
24–31	EEIM	ECC error injection mask. Setting a mask bit causes the corresponding ECC bit to be inverted on memory bus writes.

9.4.1.21 Memory Data Path Read Capture High (CAPTURE_DATA_HI)

The memory data path read capture high register, shown in Figure 9-22, stores the high word of the read data path during error capture.

Offset 0xE20 — Access: Read/Write

	0–31
R/W	ECHD
Reset	All zeros

Figure 9-22. Memory Data Path Read Capture High Register (CAPTURE_DATA_HI)

Table 9-27 describes the CAPTURE_DATA_HI fields.

Table 9-27. CAPTURE_DATA_HI Field Descriptions

Bits	Name	Description
0–31	ECHD	Error capture high data path. Captures the high word of the data path when errors are detected.

9.4.1.22 Memory Data Path Read Capture Low (CAPTURE_DATA_LO)

The memory data path read capture low register, shown in Figure 9-23, stores the low word of the read data path during error capture.

Offset 0xE24 | Access: Read/Write

	0–31
R/W	ECLD
Reset	All zeros

Figure 9-23. Memory Data Path Read Capture Low Register (CAPTURE_DATA_LO)

Table 9-28 describes the CAPTURE_DATA_LO fields.

Table 9-28. CAPTURE_DATA_LO Field Descriptions

Bits	Name	Description
0–31	ECLD	Error capture low data path. Captures the low word of the data path when errors are detected.

9.4.1.23 Memory Data Path Read Capture ECC (CAPTURE_ECC)

The memory data path read capture ECC register, shown in Figure 9-24, stores the ECC syndrome bits that were on the data bus when an error was detected.

Offset 0xE28 | Access: Read/Write

	0–15	16–31
R/W	—	ECE
Reset	All zeros	

Figure 9-24. Memory Data Path Read Capture ECC Register (CAPTURE_ECC)

Table 9-29 describes the CAPTURE_ECC fields.

Table 9-29. CAPTURE_ECC Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	ECE	Error capture ECC. Captures the ECC bits on the data path whenever errors are detected. 16–23—8-bit ECC code for 1st 32 bits 24–31—8-bit ECC code for 2nd 32 bits **Note:** In 64-bit mode, only 24–31 should be used, although 16–23 shows the 8-bit ECC code replicated.

9.4.1.24 Memory Error Detect (ERR_DETECT)

The memory error detect register stores the detection bits for multiple memory errors, single- and multiple-bit ECC errors, and memory select errors. It is a read/write register. A bit can be cleared by writing a one to the bit. System software can determine the type of memory error by examining the contents of this register. If an error is disabled with ERR_DISABLE, the corresponding error is never detected or captured in ERR_DETECT.

ERR_DETECT is shown in Figure 9-25.

Offset 0xE40 Access: w1c

	0	1–23	24	25–27	28	29	30	31
R	MME	—	ACE	—	MBE	SBE	—	MSE
W	w1c		w1c		w1c	w1c		w1c
Reset	All zeros							

Figure 9-25. Memory Error Detect Register (ERR_DETECT)

Table 9-30 describes the ERR_DETECT fields.

Table 9-30. ERR_DETECT Field Descriptions

Bits	Name	Description
0	MME	Multiple memory errors. This bit is cleared by software writing a 1. 0 Multiple memory errors of the same type were not detected. 1 Multiple memory errors of the same type were detected.
1–23	—	Reserved
24	ACE	Automatic calibration error. This bit is cleared by software writing a 1. 0 An automatic calibration error has not been detected. 1 An automatic calibration error has been detected.
25–27	—	Reserved
28	MBE	Multiple-bit error. This bit is cleared by software writing a 1. 0 A multiple-bit error has not been detected. 1 A multiple-bit error has been detected.
29	SBE	Single-bit ECC error. This bit is cleared by software writing a 1. 0 The number of single-bit ECC errors detected has not crossed the threshold set in ERR_SBE[SBET]. 1 The number of single-bit ECC errors detected crossed the threshold set in ERR_SBE[SBET].
30	—	Reserved
31	MSE	Memory select error. This bit is cleared by software writing a 1. 0 A memory select error has not been detected. 1 A memory select error has been detected.

9.4.1.25 Memory Error Disable (ERR_DISABLE)

The memory error disable register, shown in Figure 9-26, allows selective disabling of the DDR controller’s error detection circuitry. Disabled errors are not detected or reported.

Offset 0xE44 Access: Read/Write

	0–23	24	25–27	28	29	30	31
R W	—	ACED	—	MBED	SBED	—	MSED
Reset	All zeros						

Figure 9-26. Memory Error Disable Register (ERR_DISABLE)

Table 9-31 describes the ERR_DISABLE fields.

Table 9-31. ERR_DISABLE Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24	ACED	Automatic calibration error disable 0 Automatic calibration errors are enabled. 1 Automatic calibration errors are disabled.
25–27	—	Reserved
28	MBED	Multiple-bit ECC error disable 0 Multiple-bit ECC errors are detected if DDR_SDRAM_CFG[ECC_EN] is set. They are reported if ERR_INT_EN[MBEE] is set. See Section 9.5.12, "Error Management," for more information. 1 Multiple-bit ECC errors are not detected or reported.
29	SBED	Single-bit ECC error disable 0 Single-bit ECC errors are enabled. 1 Single-bit ECC errors are disabled.
30	—	Reserved
31	MSED	Memory select error disable 0 Memory select errors are enabled. 1 Memory select errors are disabled.

9.4.1.26 Memory Error Interrupt Enable (ERR_INT_EN)

The memory error interrupt enable register, shown in Figure 9-27, enables ECC interrupts or memory select error interrupts. When an enabled interrupt condition occurs, the internal $\overline{int}$ signal is asserted to the programmable interrupt controller (PIC).

Offset 0xE48 — Access: Read/Write

	0–23	24	25–27	28	29	30	31
R/W	—	ACEE	—	MBEE	SBEE	—	MSEE
Reset	All zeros						

Figure 9-27. Memory Error Interrupt Enable Register (ERR_INT_EN)

Table 9-32 describes the ERR_INT_EN fields.

Table 9-32. ERR_INT_EN Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24	ACEE	Automatic calibration error interrupt enable 0 Automatic calibration errors cannot generate interrupts. 1 Automatic calibration errors generate interrupts.
25–27	—	Reserved
28	MBEE	Multiple-bit ECC error interrupt enable. See Section 9.5.12, "Error Management," for more information. 0 Multiple-bit ECC errors cannot generate interrupts. 1 Multiple-bit ECC errors generate interrupts.

Table 9-32. ERR_INT_EN Field Descriptions (continued)

Bits	Name	Description
29	SBEE	Single-bit ECC error interrupt enable 0 Single-bit ECC errors cannot generate interrupts. 1 Single-bit ECC errors generate interrupts.
30	—	Reserved
31	MSEE	Memory select error interrupt enable 0 Memory select errors do not cause interrupts. 1 Memory select errors generate interrupts.

9.4.1.27 Memory Error Attributes Capture (CAPTURE_ATTRIBUTES)

The memory error attributes capture register, shown in Figure 9-28, sets attributes for errors including type, size, source, and others.

Offset 0xE4C — Access: Read/Write

	0	1–3	4	5–7	8–10	11–15	16–17	18–19	20–30	31
R/W	—	BNUM	—	TSIZ	—	TSRC	—	TTYP	—	VLD
Reset	All zeros									

Figure 9-28. Memory Error Attributes Capture Register (CAPTURE_ATTRIBUTES)

Table 9-33 describes the CAPTURE_ATTRIBUTES fields.

Table 9-33. CAPTURE_ATTRIBUTES Field Descriptions

Bits	Name	Description
0	—	Reserved
1–3	BNUM	Data beat number. Captures the doubleword number for the detected error. Relevant only for ECC errors.
4	—	Reserved
5–7	TSIZ	Transaction size for the error. Captures the transaction size in double words. 000 4 double words 001 1 double word 010 2 double words 011 3 double words Others Reserved
8–10	—	Reserved

Table 9-33. CAPTURE_ATTRIBUTES Field Descriptions (continued)

Bits	Name	Description
11–15	TSRC	Transaction source for the error 00000 e300 core data transaction 00001 Reserved 00010 e300 core instruction fetch 00011 Reserved 00100 eTSEC 1 00101 eTSEC 2 00110 Reserved 00111 USB DR 01000 Encryption core 01001 I^{2}C (boot sequencer) 01010 JTAG 01100 eSDHC 01101 PCI 01110 Reserved 01111 DMA11000 SATA1 11001 SATA2 11010 SATA3 11011 SATA4 11100 Reserved 11101 PCI Express 1 11110 PCI Express 2 11111 Reserved
16–17	—	Reserved
18–19	TTYP	Transaction type for the error. 00 Reserved 01 Write 10 Read 11 Read-modify-write
20–30	—	Reserved
31	VLD	Valid. Set as soon as valid information is captured in the error capture registers.

9.4.1.28 Memory Error Address Capture (CAPTURE_ADDRESS)

The memory error address capture register, shown in Figure 9-29, holds the 32 lsbs of a transaction when a DDR ECC error is detected.

Offset 0xE50 Access: Read/Write

	0–31
R W	CADDR
Reset	All zeros

Figure 9-29. Memory Error Address Capture Register (CAPTURE_ADDRESS)

Table 9-34 describes the CAPTURE_ADDRESS fields.

Table 9-34. CAPTURE_ADDRESS Field Descriptions

Bits	Name	Description
0–31	CADDR	Captured address. Captures the 32 lsbs of the transaction address when an error is detected.

9.4.1.29 Single-Bit ECC Memory Error Management (ERR_SBE)

The single-bit ECC memory error management register, shown in Figure 9-30, stores the threshold value for reporting single-bit errors and the number of single-bit errors counted since the last error report. When the counter field reaches the threshold, it wraps back to the reset value (0). If necessary, software must clear the counter after it has managed the error.

Offset 0xE58 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	—	SBET	—	SBEC
Reset	All zeros			

Figure 9-30. Single-Bit ECC Memory Error Management Register (ERR_SBE)

Table 9-35 describes the ERR_SBE fields.

Table 9-35. ERR_SBE Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	SBET	Single-bit error threshold. Establishes the number of single-bit errors that must be detected before an error condition is reported.
16–23	—	Reserved
24–31	SBEC	Single-bit error counter. Indicates the number of single-bit errors detected and corrected since the last error report. If single-bit error reporting is enabled, an error is reported and an interrupt is generated when this value equals SBET. SBEC is automatically cleared when the threshold value is reached.

9.5 Functional Description

The DDR SDRAM controller controls processor and I/O interactions with system memory. It provides support for JEDEC-compliant DDR2 and DDR SDRAMs. The memory system allows a wide range of memory devices to be mapped to any arbitrary chip select, and support is provided for registered DIMMs and unbuffered DIMMs. However, registered DIMMs cannot be mixed with unbuffered DIMMs.

Figure 9-31 is a high-level block diagram of the DDR memory controller. Requests are received from the internal mastering device and the address is decoded to generate the physical bank, logical bank, row, and column addresses. The transaction is compared with values in the row open table to determine if the address maps to an open page. If the transaction does not map to an open page, an active command is issued.

The memory interface supports as many as four physical banks of 64-/72-bit wide or 32-/40-bit wide memory. Bank sizes up to 2 Gbytes (maximum total physical memory size of 4 Gbytes) are supported, providing up to a maximum of 4 Gbytes of DDR main memory.

Programmable parameters allow for a variety of memory organizations and timings. Optional error checking and correcting (ECC) protection is provided for the DDR SDRAM data bus. Using ECC, the DDR memory controller detects and corrects all single-bit errors within the 64- or 32-bit data bus, detects all double-bit errors within the 64- or 32-bit data bus, and detects all errors within a nibble. The controller allows as many as 32 pages to be open simultaneously. The amount of time (in clock cycles) the pages remain open is programmable with DDR_SDRAM_INTERVAL[BSTOPRE].

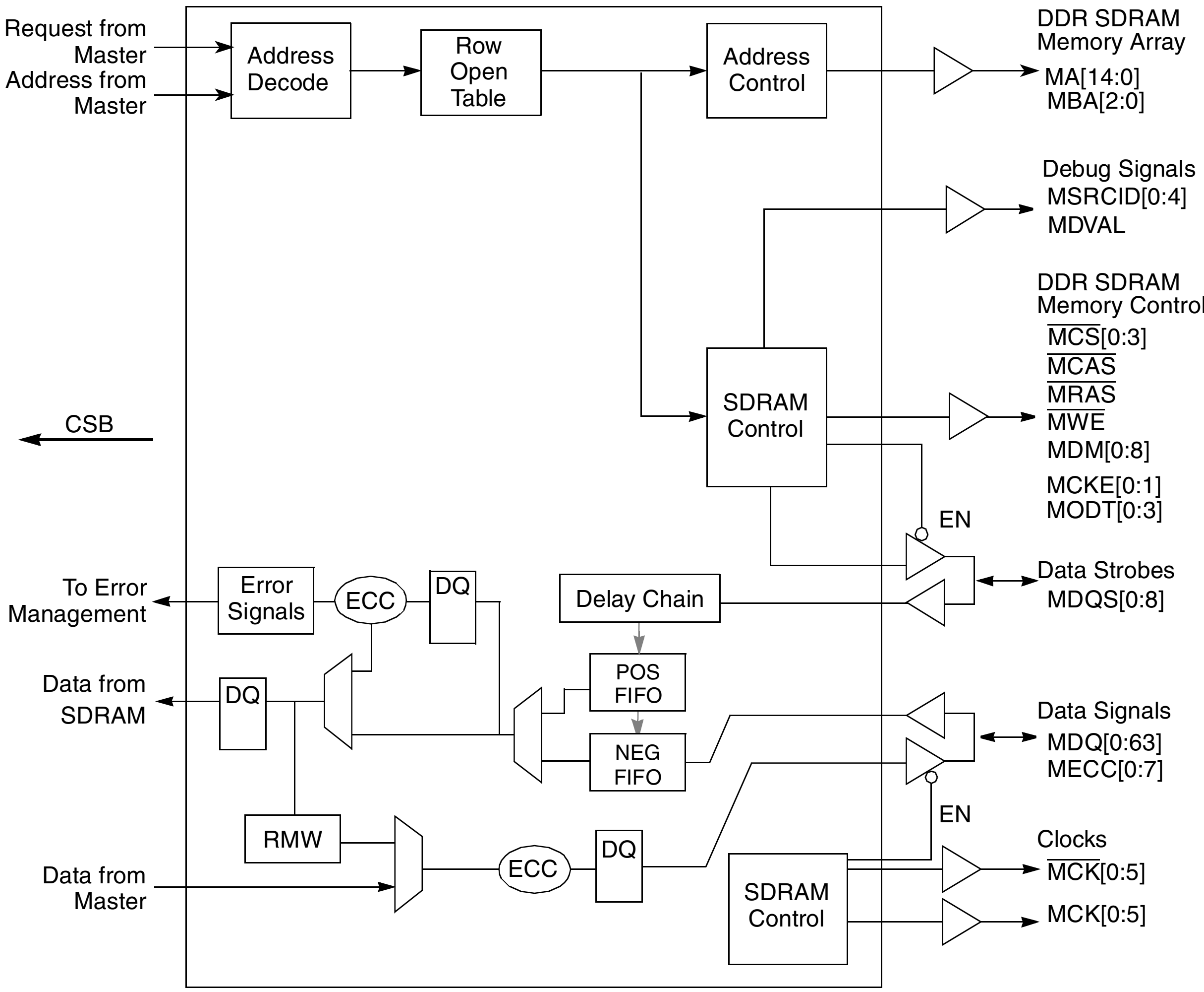


Figure 9-31. DDR Memory Controller Block Diagram

Read and write accesses to memory are burst oriented; accesses start at a selected location and continue for a programmed number of higher locations (4 or 8) in a programmed sequence. Accesses to closed pages start with the registration of an ACTIVE command followed by a READ or WRITE. (Accessing open pages does not require an ACTIVE command.) The address bits registered coincident with the activate command specifies the logical bank and row to be accessed. The address coincident with the READ or WRITE command specify the logical bank and starting column for the burst access.

The data interface is source synchronous, meaning whatever sources the data also provides a clocking signal to synchronize data reception. These bidirectional data strobes (MDQS[0–8]) are inputs to the controller during reads and outputs during writes. The DDR SDRAM specification requires the data strobe signals to be centered within the data tenure during writes and to be offset by the controller to the center of the data tenure during reads. This delay is implemented in the controller for both reads and writes.

When ECC is enabled, 1 clock cycle is added to the read path to check ECC and correct single-bit errors. ECC generation does not add a cycle to the write path.

The address and command interface is also source synchronous, although 1/8 cycle adjustments are provided for adjusting the clock alignment.

Figure 9-32 shows an example DDR SDRAM configuration with four logical banks.

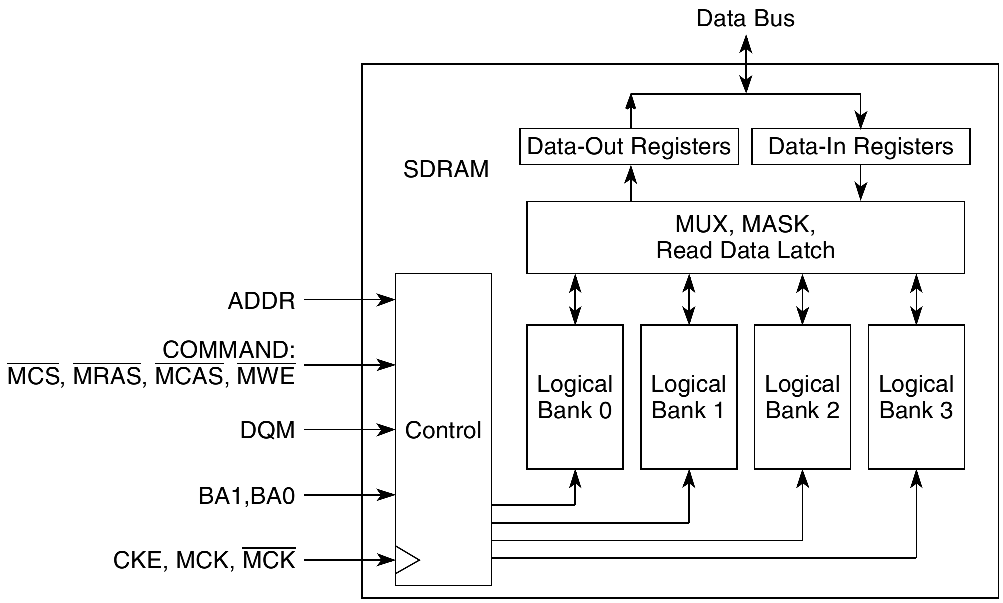


Figure 9-32. Typical Dual Data Rate SDRAM Internal Organization

Figure 9-33 shows some typical signal connections.

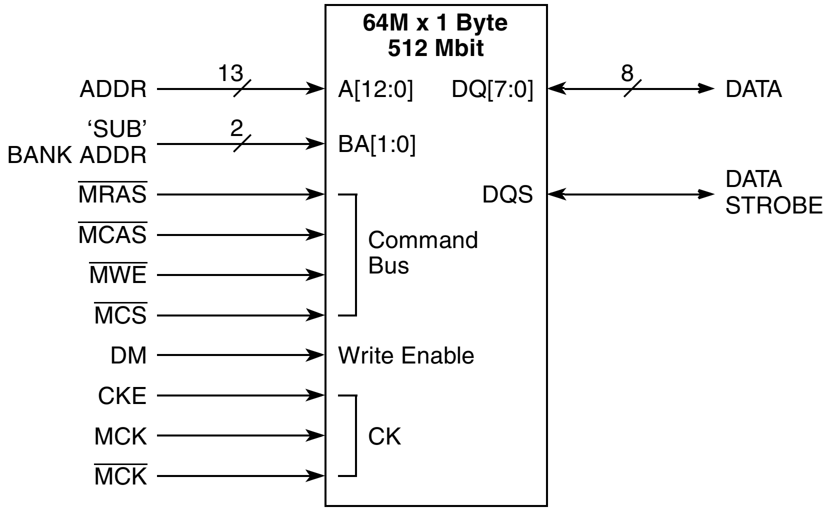
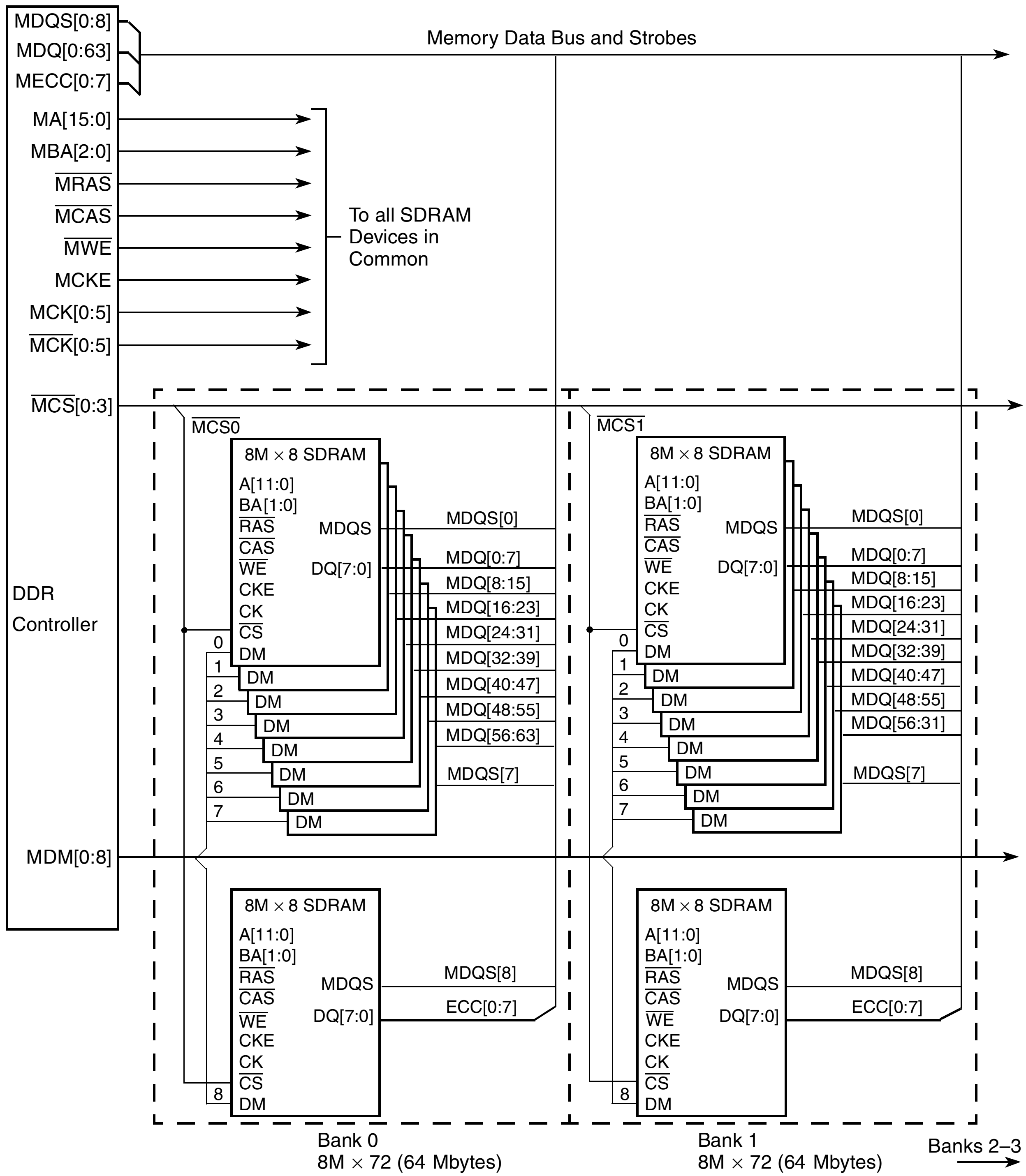


Figure 9-33. Typical DDR SDRAM Interface Signals

Figure 9-34 shows an example DDR SDRAM configuration with four physical banks each comprised of nine 8M × 8 DDR modules for a total of 256 Mbytes of system memory. One of the nine modules is used for the memory's ECC checking function. Certain address and control lines may require buffering. Analysis of the device's AC timing specifications, desired memory operating frequency, capacitive loads, and board routing loads can assist the system designer in deciding signal buffering requirements. The DDR memory controller drives 15 address pins, but in this example the DDR SDRAM devices use only 12 bits.



1. All signals are connected in common (in parallel) except for MCS[0:3], MCK[0:5], MDM[0:8], and the data bus signals.
2. Each of the MCS[0:3] signals correspond with a separate physical bank of memory.
3. Buffering may be needed if large memory arrays are used.
4. MCK[0:5] may be apportioned among all memory devices. Complementary bus is not shown.

Figure 9-34. Example 256-Mbyte DDR SDRAM Configuration With ECC

Section 9.5.12, “Error Management,” explains how the DDR memory controller handles errors.

9.5.1 DDR SDRAM Interface Operation

The DDR memory controller supports many different DDR SDRAM configurations. SDRAMs with different sizes can be used in the same system. Fifteen multiplexed address signals and three logical bank select signals support device densities from 64 Mbits to 4 Gbits. Four chip select ($\overline{CS}$) signals support up to two DIMMs of memory. The DDR SDRAM physical banks can be built from standard memory modules or directly-attached memory devices. The data path to individual physical banks is 64 or 32 bits wide, 72 or 40 bits with ECC. The DDR memory controller supports physical bank sizes from 16 Mbytes to 4 Gbytes. The physical banks can be constructed using ×8, ×16, or ×32 memory devices. The memory technologies supported are 64 Mbits, 128 Mbits, 256 Mbits, 512 Mbits, 1 Gbit, 2 Gbits, and 4 Gbits. Nine data qualifier (DQM) signals provide byte selection for memory accesses.

NOTE

An 8-bit DDR SDRAM device has a DQM signal and eight data signals (DQ[0–7]). A 16-bit DDR SDRAM device has two DQM signals associated with specific halves of the 16 data signals (DQ[0–7] and DQ[8–15]).

When ECC is enabled, all memory accesses are performed on double-word boundaries (that is, all DQM signals are set simultaneously). However, when ECC is disabled, the memory system uses the DQM signals for byte lane selection.

Table 9-36 shows the DDR memory controller's relationships between data byte lane0–7, MDM[0–7], MDQS[0–7], and MDQ[0–63] when DDR SDRAM memories are used with ×8 or ×16 devices.

Table 9-36. Byte Lane to Data Relationship

Data Byte Lane	Data Bus Mask	Data Bus Strobe	Data Bus 64-Bit Mode
0 (MSB)	MDM[0]	MDQS[0]	MDQ[0–7]
1	MDM[1]	MDQS[1]	MDQ[8–15]
2	MDM[2]	MDQS[2]	MDQ[16–23]
3	MDM[3]	MDQS[3]	MDQ[24–31]
4	MDM[4]	MDQS[4]	MDQ[32–39]
5	MDM[5]	MDQS[5]	MDQ[40–47]
6	MDM[6]	MDQS[6]	MDQ[48–55]
7 (LSB)	MDM[7]	MDQS[7]	MDQ[56–63]

9.5.1.1 Supported DDR SDRAM Organizations

Although the DDR memory controller multiplexes row and column address bits onto 15 memory address signals and 3 logical bank select signals, a physical bank may be implemented with memory devices requiring fewer than 31 address bits. The physical bank may be configured to provide from 12 to 15 row address bits, plus 2 or 3 logical bank-select bits and from 8–11 column address bits.

Table 9-38 describe DDR SDRAM device configurations supported by the DDR memory controller.

NOTE

DDR SDRAM is limited to 30 total address bits.

Table 9-37. Supported DDR1 SDRAM Device Configurations

SDRAM Device	Device Configuration	Row × Column × Sub-Bank Bits	64-Bit Bank Size	Banks of Memory
64 Mbits	8 Mbits × 8	12 × 9 × 2	64 Mbytes	
64 Mbits[1]	4 Mbits × 16	12 × 8 × 2	32 Mbytes	
128 Mbits	16 Mbits × 8	12 × 10 × 2	128 Mbytes	
128 Mbits	8 Mbits × 16	12 × 9 × 2	64 Mbytes	
256 Mbits	32 Mbits × 8	13 × 10 × 2	256 Mbytes	
256 Mbits	16 Mbits × 16	13 × 9 × 2	128 Mbytes	
512 Mbits	64 Mbits × 8	13 × 11 × 2	512 Mbytes	
512 Mbits	32 Mbits × 16	13 × 10 × 2	256 Mbytes	
1 Gbit	128 Mbits × 8	14 × 11 × 2	1 Gbyte	
1 Gbit	64 Mbits × 16	14 × 10 × 2	512 Mbytes	
2 Gbits	256 Mbits × 8	15 × 11 × 2	2 Gbytes	4 Gbytes (two banks)
2 Gbits	128 Mbits × 16	15 × 10 × 2	1 Gbyte	

[1] This configuration is not supported in 16-bit bus mode.

Table 9-38. Supported DDR2 SDRAM Device Configurations

SDRAM Device	Device Configuration	Row × Column × Sub-Bank Bits	64-Bit Bank Size	Banks of Memory
256 Mbits	32 Mbits × 8	13 × 10 × 2	256 Mbytes	
256 Mbits	16 Mbits × 16	13 × 9 × 2	128 Mbytes	
512 Mbits	64 Mbits × 8	14 × 10 × 2	512 Mbytes	
512 Mbits	32 Mbits × 16	13 × 10 × 2	256 Mbytes	
1 Gbit	128 Mbits × 8	14 × 10 × 3	1 Gbyte	
1 Gbit	64 Mbits × 16	13 × 10 × 3	512 Mbytes	
2 Gbits	256 Mbits × 8	15 × 10 × 3	2 Gbytes	4 Gbytes (two banks)
2 Gbits	128Mbits × 16	14 × 10 × 3	1 Gbyte	
4 Gbits	256 Mbits × 16	15 × 10 × 3	2 Gbytes	4 Gbytes (two banks)

If a transaction request is issued to the DDR memory controller and the address does not lie within any of the programmed address ranges for an enabled chip select, a memory select error is flagged. Errors are described in detail in Section 9.5.12, "Error Management."

Using a memory-polling algorithm at power-on reset or by querying the JEDEC serial presence detect capability of memory modules, system firmware uses the memory-boundary registers to configure the DDR memory controller to map the size of each bank in memory. The memory controller uses its bank map to assert the appropriate $\overline{MCSn}$ signal for memory accesses according to the provided bank starting and ending addresses. The memory banks are not required to be mapped to a contiguous address space.

9.5.2 DDR SDRAM Address Multiplexing

The following tables (Table 9-39, Table 9-40, Table 9-41, Table 9-42) show the address bit encodings for each DDR SDRAM configuration. The address presented at the memory controller signals MA[14–0] use MA[14] as the msb and MA[0] as the lsb. Also, MA[10] is used as the auto-precharge bit in DDR1/DDR2 modes for reads and writes, so the column address can never use MA[10].

Table 9-39. DDR1 Address Multiplexing for 64-Bit Data Bus with Interleaving Disabled

Row x Col		msb	Address from Core Master																												lsb
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29–31
15 x 11 x 2	$\overline{MRAS}$		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{MCAS}$																			11	9	8	7	6	5	4	3	2	1	0	
15 x 10 x 2	$\overline{MRAS}$			14	13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
14 x 11 x 2	$\overline{MRAS}$			13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{MCAS}$																			11	9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{MRAS}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 11 x 2	$\overline{MRAS}$				12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{MCAS}$																			11	9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{MRAS}$					12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 9 x 2	$\overline{MRAS}$						12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																			1	0										
	$\overline{MCAS}$																					8	7	6	5	4	3	2	1	0	
12 x 10 x 2	$\overline{MRAS}$						11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	

Table 9-39. DDR1 Address Multiplexing for 64-Bit Data Bus with Interleaving Disabled (continued)

Row x Col		msb	Address from Core Master																												lsb
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29–31
12 x 9 x 2	$\overline{MRAS}$							11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																			1	0										
	$\overline{MCAS}$																					8	7	6	5	4	3	2	1	0	
12 x 8 x 2	$\overline{MRAS}$								11	10	9	8	7	6	5	4	3	2	1	0											
	MBA																				1	0									
	$\overline{MCAS}$																						7	6	5	4	3	2	1	0	

Table 9-40. DDR1 Address Multiplexing for 32-Bit Data Bus with Interleaving Disabled

Row x Col		msb	Address from Core Master																													lsb
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30–31
15 x 11 x 2	$\overline{MRAS}$			14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	$\overline{MCAS}$																				11	9	8	7	6	5	4	3	2	1	0	
15 x 10 x 2	$\overline{MRAS}$				14	13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
14 x 11 x 2	$\overline{MRAS}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	$\overline{MCAS}$																				11	9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{MRAS}$					13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
13 x 11 x 2	$\overline{MRAS}$					12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	$\overline{MCAS}$																				11	9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{MRAS}$						12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
13 x 9 x 2	$\overline{MRAS}$							12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																				1	0										
	$\overline{MCAS}$																						8	7	6	5	4	3	2	1	0	
12 x 10 x 2	$\overline{MRAS}$							11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	

Table 9-40. DDR1 Address Multiplexing for 32-Bit Data Bus with Interleaving Disabled (continued)

Row x Col		msb	Address from Core Master																													lsb
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30–31
12 x 9 x 2	$\overline{\text{MRAS}}$								11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																				1	0										
	$\overline{\text{MCAS}}$																						8	7	6	5	4	3	2	1	0	
12 x 8 x 2	$\overline{\text{MRAS}}$									11	10	9	8	7	6	5	4	3	2	1	0											
	MBA																					1	0									
	$\overline{\text{MCAS}}$																							7	6	5	4	3	2	1	0	

Table 9-41. DDR2 Address Multiplexing for 64-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled

Row x Col		msb	Address from Core Master																												lsb
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29–31
15 x 10 x 3	$\overline{\text{MRAS}}$		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	2	1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
14 x 10 x 3	$\overline{\text{MRAS}}$			13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	2	1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{\text{MRAS}}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 3	$\overline{\text{MRAS}}$				12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	2	1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{\text{MRAS}}$					12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	
13 x 9 x 2	$\overline{\text{MRAS}}$						12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																			1	0										
	$\overline{\text{MCAS}}$																					8	7	6	5	4	3	2	1	0	

Table 9-42. DDR2 Address Multiplexing for 32-Bit Data Bus with Interleaving and Partial Array Self Refresh Disabled

Row x Col		msb	Address from Core Master																													lsb
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30–31
15 x 10 x 3	$\overline{MRAS}$			14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		2	1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
14 x 10 x 3	$\overline{MRAS}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		2	1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{MRAS}$					13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
13 x 10 x 3	$\overline{MRAS}$					12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		2	1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{MRAS}$						12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{MCAS}$																					9	8	7	6	5	4	3	2	1	0	
13 x 9 x 2	$\overline{MRAS}$							12	11	10	9	8	7	6	5	4	3	2	1	0												
	MBA																				1	0										
	$\overline{MCAS}$																						8	7	6	5	4	3	2	1	0	

Chip select interleaving is supported for the memory controller, and is programmed in DDR_SDRAM_CFG[BA_INTLV_CTL]. Interleaving is supported between chip selects 0 and 1 or chip selects 2 and 3. In addition, interleaving between all four chip selects can be enabled. When interleaving is enabled, the chip selects being interleaved must use the same size of memory. If two chip selects are interleaved, then 1 extra bit in the address decode is used for the interleaving to determine which chip select to access. If four chip selects are interleaved, then two extra bits are required in the address decode.

Table 9-43 illustrates examples of address decode when interleaving between two chip selects, and Table 9-44 shows examples of address decode when interleaving between four chip selects.

Table 9-43. Example of Address Multiplexing for 64-Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled

Row x Col		msb	Address from Core Master																											lsb
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29–31
14 x 10 x 3	$\overline{MRAS}$		13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	MBA																	2	1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{MRAS}$			13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL													
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 3	$\overline{MRAS}$			12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	MBA																	2	1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{MRAS}$				12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL													
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	

Table 9-44. Example of Address Multiplexing for 64-Bit Data Bus Interleaving between Four Banks with Partial Array Self Refresh Disabled

Row x Col		msb	Address from Core Master																											lsb
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29–31
14 x 10 x 3	$\overline{MRAS}$	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL															
	MBA																	2	1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
14 x 10 x 2	$\overline{MRAS}$		13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 3	$\overline{MRAS}$		12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL															
	MBA																	2	1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	
13 x 10 x 2	$\overline{MRAS}$			12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	MBA																		1	0											
	$\overline{MCAS}$																				9	8	7	6	5	4	3	2	1	0	

9.5.3 JEDEC Standard DDR SDRAM Interface Commands

The following section describes the commands and timings the controller uses when operating in DDR2 or DDR modes.

All read or write accesses to DDR SDRAM are performed by the DDR memory controller using JEDEC standard DDR SDRAM interface commands. The SDRAM device samples command and address inputs on rising edges of the memory clock; data is sampled using both the rising and falling edges of DQS. Data read from the DDR SDRAM is also sampled on both edges of DQS.

The following DDR SDRAM interface commands (summarized in Table 9-45) are provided by the DDR controller. All actions for these commands are described from the perspective of the SDRAM device.

- Row activate—Latches row address and initiates memory read of that row. Row data is latched in SDRAM sense amplifiers and must be restored by a precharge command before another row activate occurs.
- Precharge—Restores data from the sense amplifiers to the appropriate row. Also initializes the sense amplifiers in preparation for reading another row in the memory array (performing another activate command). Precharge must occur after read or write, if the row address changes on the next open page mode access.
- Read—Latches column address and transfers data from the selected sense amplifier to the output buffer as determined by the column address. During each succeeding clock edge, additional data is driven without additional read commands. The amount of data transferred is determined by the burst size which defaults to 4.
- Write—Latches column address and transfers data from the data pins to the selected sense amplifier as determined by the column address. During each succeeding clock edge, additional data is transferred to the sense amplifiers from the data pins without additional write commands. The amount of data transferred is determined by the data masks and the burst size, which is set to four by the DDR memory controller.
- Refresh (similar to $\overline{\text{MCAS}}$ before $\overline{\text{MRAS}}$)—Causes a row to be read in all logical banks (JEDEC SDRAM) as determined by the refresh row address counter. This refresh row address counter is internal to the SDRAM. After being read, the row is automatically rewritten in the memory array. All logical banks must be in a precharged state before executing a refresh. The memory controller also supports posted refreshes, where several refreshes may be executed at once, and the refresh interval may be extended.
- Mode register set (for configuration)—Allows setting of DDR SDRAM options. These options are: $\overline{\text{MCAS}}$ latency, additive latency (for DDR2), write recovery (for DDR2), burst type, and burst length. $\overline{\text{MCAS}}$ latency may be chosen as provided by the preferred SDRAM (some SDRAMs provide $\overline{\text{MCAS}}$ latency {1,2,3}, some provide $\overline{\text{MCAS}}$ latency {1,2,3,4,5}, and so on). Burst type is always sequential. Although some SDRAMs provide burst lengths of 1, 2, 4, 8, and page size, this memory controller supports a burst length of 4. A burst length of 8 is supported for DDR1memory only. For DDR2 in 32-bit bus mode, all 32-byte burst accesses from the platform are split into two 16-byte (that is, 4-beat) accesses to the SDRAMs in the memory controller. The mode register set command is performed by the DDR memory controller during system initialization. Parameters such as mode register data, $\overline{\text{MCAS}}$ latency, burst length, and burst type, are set by software in DDR_SDRAM_MODE[SDMODE] and transferred to the SDRAM array by

the DDR memory controller after DDR_SDRAM_CFG[MEM_EN] is set. If DDR_SDRAM_CFG[BI] is set to bypass the automatic initialization, then the MODE registers can be configured through software through use of the DDR_SDRAM_MD_CNTL register.

- Self refresh (for long periods of standby)—Used when the device is in standby for very long periods of time. Automatically generates internal refresh cycles to keep the data in all memory banks refreshed. Before execution of this command, the DDR controller places all logical banks in a precharged state.

Table 9-45. DDR SDRAM Command Table

Operation	CKE Prev.	CKE Current	$\overline{MCS}$	$\overline{MRAS}$	$\overline{MCAS}$	$\overline{MWE}$	MBA	MA10	MA
Activate	H	H	L	L	H	H	Logical bank select	Row	Row
Precharge select logical bank	H	H	L	L	H	L	Logical bank select	L	X
Precharge all logical banks	H	H	L	L	H	L	X	H	X
Read	H	H	L	H	L	H	Logical bank select	L	Column
Read with auto-precharge	H	H	L	H	L	H	Logical bank select	H	Column
Write	H	H	L	H	L	L	Logical bank select	L	Column
Write with auto-precharge	H	H	L	H	L	L	Logical bank select	H	Column
Mode register set	H	H	L	L	L	L	Opcode	Opcode	Opcode and mode
Auto refresh	H	H	L	L	L	H	X	X	X
Self refresh	H	L	L	L	L	H	X	X	X

9.5.4 DDR SDRAM Interface Timing

The DDR memory controller supports four-beat bursts to SDRAM. For single-beat reads, the DDR memory controller performs a four- (or eight-) beat burst read, but ignores the last three (or seven) beats. Single-beat writes are performed by masking the last three (or seven) beats of the four- (or eight-) beat burst using the data mask 0–8. If ECC is disabled, writes smaller than double words are performed by appropriately activating the data mask. If ECC is enabled, the controller performs a read-modify write.

NOTE

If a second read or write is pending, reads shorter than four beats are not terminated early even if some data is irrelevant.

To accommodate available memory technologies across a wide spectrum of operating frequencies, the DDR memory controller allows the setting of the intervals defined in Table 9-46 with granularity of one memory clock cycle, except for CASLAT, which can be programmed with ½ clock granularity.

Table 9-46. DDR SDRAM Interface Timing Intervals

Timing Intervals	Definition
ACTTOACT	The number of clock cycles from a bank-activate command until another bank-activate command within a physical bank. This interval is listed in the AC specifications of the SDRAM as t_{RRD}.
ACTTOPRE	The number of clock cycles from an activate command until a precharge command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RAS}.
ACTTORW	The number of clock cycles from an activate command until a read or write command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RCD}.
BSTOPRE	The number of clock cycles to maintain a page open after an access. The page open duration counter is reloaded with BSTOPRE each time the page is accessed (including page hits). When the counter expires, the open page is closed with an SDRAM precharge bank command as soon as possible.
CASLAT	Used in conjunction with additive latency to obtain the READ latency. The number of clock cycles between the registration of a READ command by the SDRAM and the availability of the first piece of output data. If a READ command is registered at clock edge n, and the read latency is m clocks, the data is available nominally coincident with clock edge $n + m$.
PRETOACT	The number of clock cycles from a precharge command until an activate or a refresh command is allowed. This interval is listed in the AC specifications of the SDRAM as t_{RP}
REFINT	Refresh interval. Represents the number of memory bus clock cycles between refresh cycles. Depending on DDR_SDRAM_CFG_2[NUM_PR], some number of rows are refreshed in each SDRAM bank during each refresh cycle. The value of REFINT depends on the specific SDRAMs used and the frequency of the interface as t_{RP}
REFREC	The number of clock cycles from the refresh command until an activate command is allowed. This can be calculated by referring to the AC specification of the SDRAM device. The AC specification indicates a maximum refresh-to-activate interval in nanoseconds.
WR_DATA_DELAY	Provides different options for the timing between a write command and the write data strobe. This allows write data to be sent later than the nominal time to meet the SDRAM timing requirement between the registration of a write command and the reception of a data strobe associated with the write command. The specification dictates that the data strobe may not be received earlier than 75% of a cycle, or later than 125% of a cycle, from the registration of a write command. This parameter is not defined in the SDRAM specification. It is implementation-specific, defined for the DDR memory controller in TIMING_CFG_2.
WRREC	The number of clock cycles from the last beat of a write until a precharge command is allowed. This interval, write recovery time, is listed in the AC specifications of the SDRAM as t_{WR}.
WRTORD	Last write pair to read command. Controls the number of clock cycles from the last write data pair to the subsequent read command to the same bank as t_{WTR}.

The value of the above parameters (in whole clock cycles) must be set by boot code at system start-up (in the TIMING_CFG_0, TIMING_CFG_1, TIMING_CFG_2, and TIMING_CFG_3 registers as described in Section 9.4.1.4, “DDR SDRAM Timing Configuration 0 (TIMING_CFG_0),” Section 9.4.1.5, “DDR SDRAM Timing Configuration 1 (TIMING_CFG_1),” Section 9.4.1.6, “DDR SDRAM Timing Configuration 2 (TIMING_CFG_2),” and Section 9.4.1.3, “DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)”) and be kept in the DDR memory controller configuration register space.

The following figures show SDRAM timing for various types of accesses. System software is responsible (at reset) for optimally configuring SDRAM timing parameters. The programmable timing parameters apply to both read and write timing configuration. The configuration process must be completed and the DDR SDRAM initialized before any accesses to SDRAM are attempted.

Figure 9-35 through Figure 9-37 show DDR SDRAM timing for various types of accesses; see Figure 9-35 for a single-beat read operation, Figure 9-36 for a single-beat write operation, and Figure 9-37 for a double word write operation. Note that all signal transitions occur on the rising edge of the memory bus clock and that single-beat read operations are identical to burst-reads. These figures assume the CLK_ADJUST is set to 1/2 DRAM cycle, an additive latency of 0 DRAM cycles is used, and the write latency is 1 DRAM cycle (for DDR1).

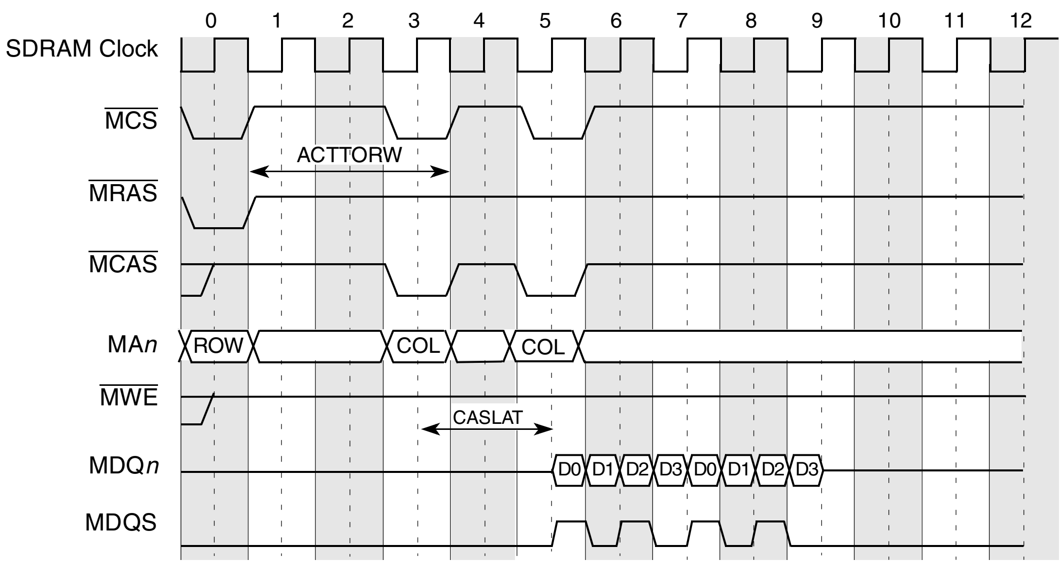


Figure 9-35. DDR SDRAM Burst Read Timing—ACTTORW = 3, MCAS Latency = 2

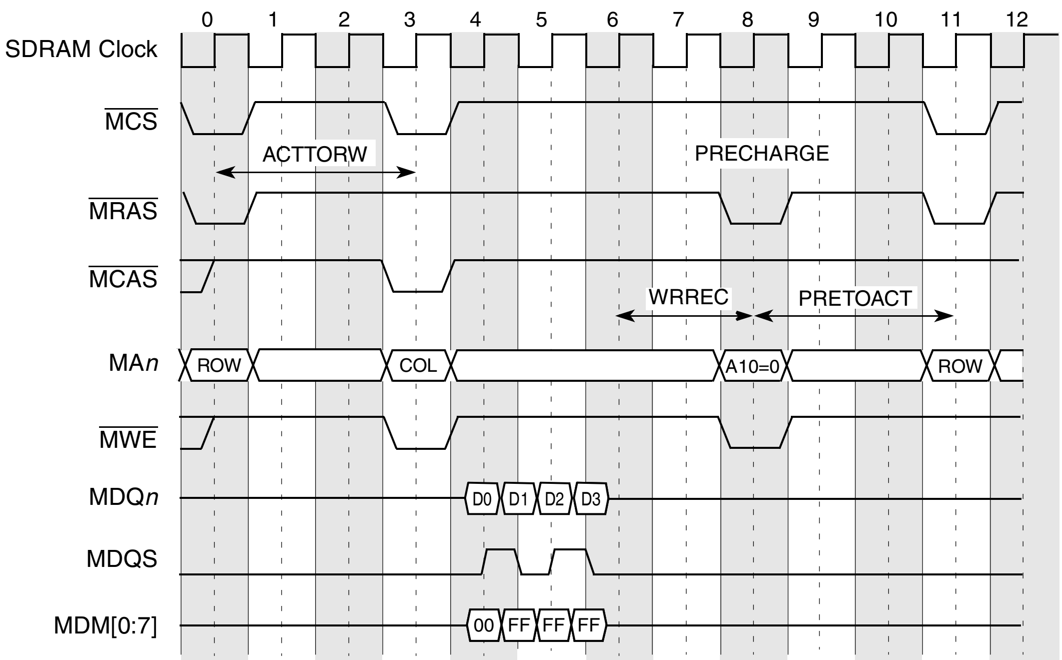


Figure 9-36. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTOR

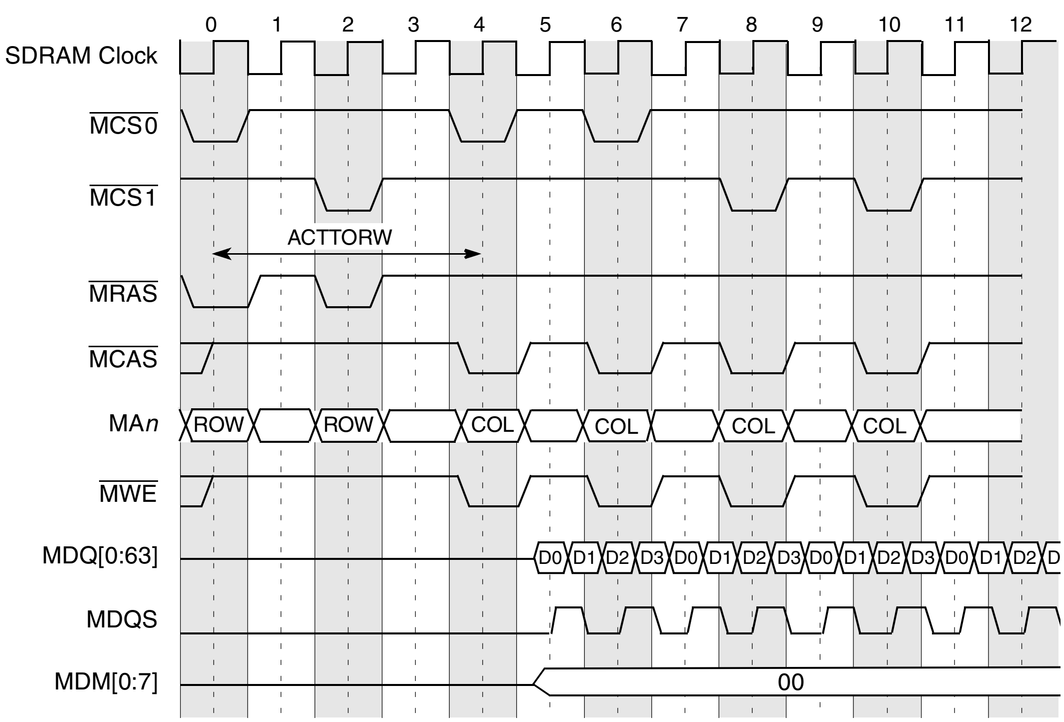


Figure 9-37. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3

9.5.4.1 Clock Distribution

- If running with many devices, zero-delay PLL clock buffers, JEDEC-JESD82 standard, should be used. These buffers were designed for DDR applications.
- A 72 bit x 64 Mbytes DDR bank has 9-byte-wide DDR chips, resulting in 18 DDR chips in a two-bank system. In this case, each MCK/$\overline{\text{MCK}}$ signal pair should drive exactly three devices.
- PCB traces for DDR clock signals should be short, all on the same layer, and of equal length and loading.
- DDR SDRAM manufacturers provide detailed information on PCB layout and termination issues.

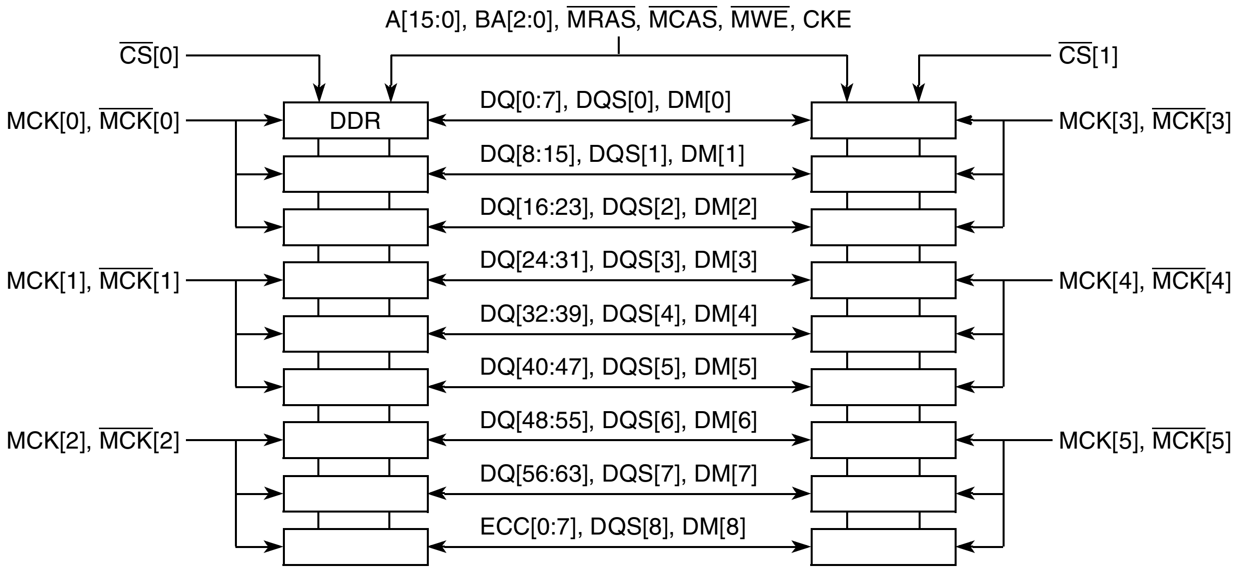


Figure 9-38. DDR SDRAM Clock Distribution Example for x8 DDR SDRAMs

9.5.5 DDR SDRAM Mode-Set Command Timing

The DDR memory controller transfers the mode register set commands to the SDRAM array, and it uses the setting of TIMING_CFG_0[MRS_CYC] for the Mode Register Set cycle time.

Figure 9-39 shows the timing of the mode-set command. The first transfer corresponds to the ESDMODE code; the second corresponds to SDMODE. The Mode Register Set cycle time is set to 2 DRAM cycles.

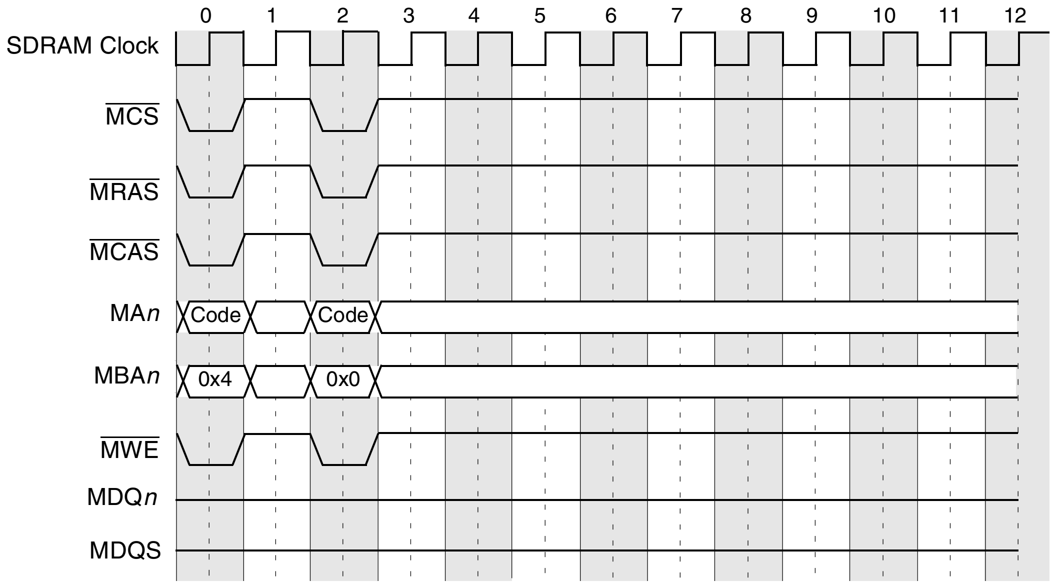


Figure 9-39. DDR SDRAM Mode-Set Command Timing

9.5.6 DDR SDRAM Registered DIMM Mode

To reduce loading, registered DIMMs latch the DDR SDRAM control signals internally before using them to access the array. Setting DDR_SDRAM_CFG[RD_EN] compensates for this delay on the DIMMs' control bus by delaying the data and data mask writes (on SDRAM buses) by an extra SDRAM clock cycle.

NOTE

Application system board must assert the reset signal on DDR memory devices until software is able to program the DDR memory controller configuration registers, and must deassert the reset signal on DDR memory devices before DDR_SDRAM_CFG[MEM_EN] is set. This ensures that the DDR memory devices are held in reset until a stable clock is provided and, further, that a stable clock is provided before memory devices are released from reset.

Figure 9-40 shows the registered DDR SDRAM DIMM single-beat write timing.

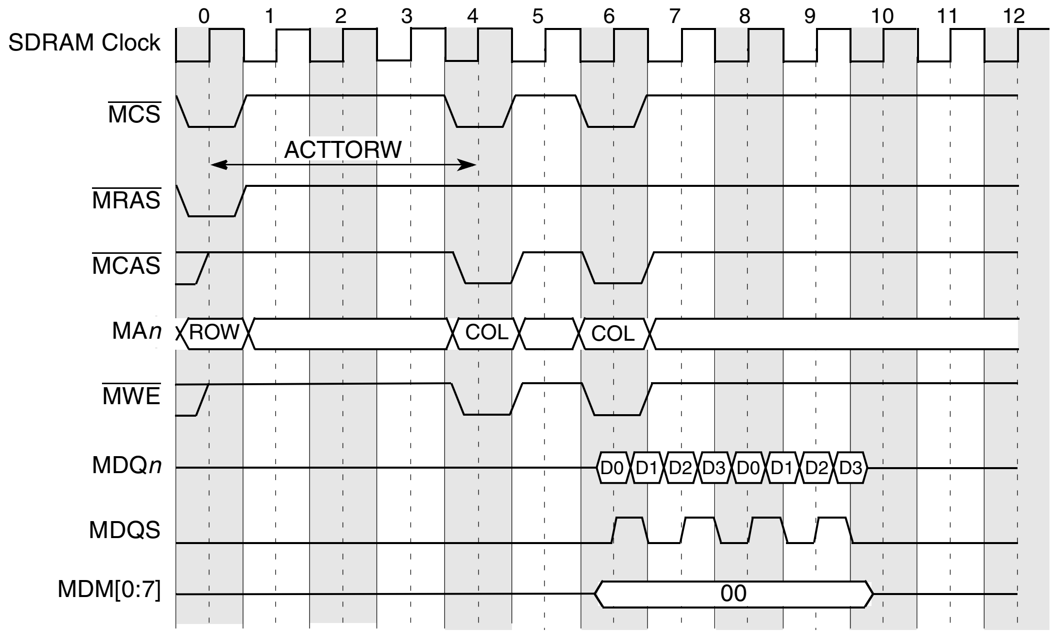


Figure 9-40. Registered DDR SDRAM DIMM Burst Write Timing

9.5.7 DDR SDRAM Write Timing Adjustments

The DDR memory controller facilitates system design flexibility by providing a write timing adjustment parameter, write data delay, (TIMING_CFG_2[WR_DATA_DELAY]) for data and DQS. The DDR SDRAM specification requires DQS be received no sooner than 75% of an SDRAM clock period—and no later than 125% of a clock period—from the capturing clock edge of the command/address at the SDRAM. The WR_DATA_DELAY parameter may be used to meet this timing requirement for a variety of system configurations, ranging from a system with one DIMM to a fully populated system with two DIMMs. TIMING_CFG_2[WR_DATA_DELAY] specifies how much to delay the launching of DQS and data from the first clock edge occurring one SDRAM clock cycle after the command is launched. The delay increment step sizes are in 1/4 SDRAM clock periods starting with the default value of 0.

Figure 9-41 shows the use of the WR_DATA_DELAY parameter.

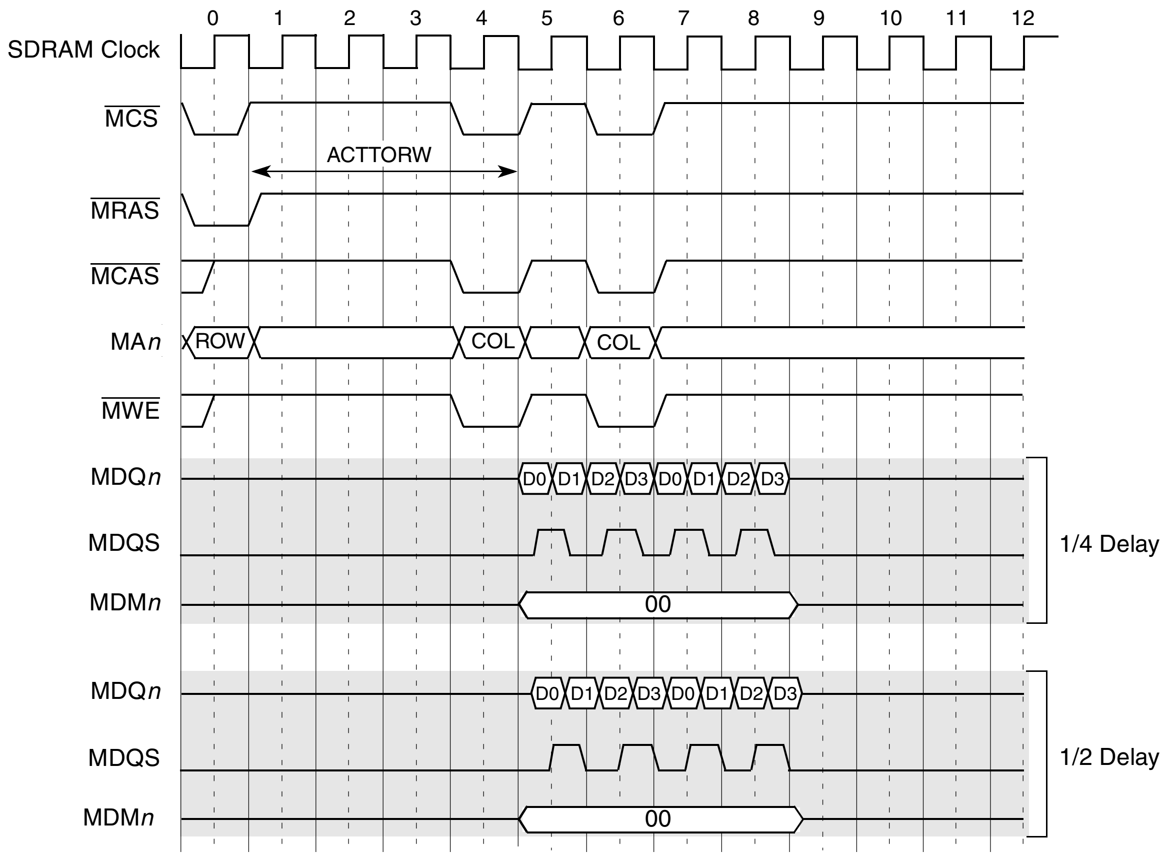


Figure 9-41. Write Timing Adjustments Example for Write Latency = 1

9.5.8 DDR SDRAM Refresh

The DDR memory controller supports auto-refresh and self-refresh. Auto refresh is used during normal operation and is controlled by the DDR_SDRAM_INTERVAL[REFINT] value; self-refresh is used only when the DDR memory controller is set to enter a sleep power management state. The REFINT value, which represents the number of memory bus clock cycles between refresh cycles, allow for possible outstanding transactions to complete before a refresh request is sent to the memory after the REFINT value is reached. If a memory transaction is in progress when the refresh interval is reached, the refresh cycle waits for the transaction to complete. In the worst case, the refresh cycle must wait the number of bus clock cycles required by the longest programmed access. To ensure that the latency caused by a memory transaction does not violate the device refresh period, it is recommended that the programmed value of REFINT be less than that required by the SDRAM.

When a refresh cycle is required, the DDR memory controller does the following:

1. Completes all current memory requests.
2. Closes all open pages with a PRECHARGE-ALL command to each DDR SDRAM bank with an open page (as indicated by the row open table).
3. Issues one or more auto-refresh commands to each DDR SDRAM bank (as identified by its chip select) to refresh one row in each logical bank of the selected physical bank.

The auto-refresh commands are staggered across the four possible banks to reduce the system's instantaneous power requirements. Three sets of auto refresh commands are issued on consecutive cycles when the memory is fully populated with two DIMMs. The initial PRECHARGE-ALL commands are also staggered in three groups for convenience. It is important to note that when entering self-refresh mode, only one refresh command is issued simultaneously to all physical banks. For this entire refresh sequence, no cycle optimization occurs for the usual case where fewer than four banks are installed. After the refresh sequence completes, any pending memory request is initiated after an inactive period specified by TIMING_CFG_1 [REFREC] and TIMING_CFG_3[EXT_REFREC]. In addition, posted refreshes are supported to allow the refresh interval to be set to a larger value.

9.5.8.1 DDR SDRAM Refresh Timing

Refresh timing for the DDR SDRAM is controlled by the programmable timing parameter TIMING_CFG_1 [REFREC], which specifies the number of memory bus clock cycles from the refresh command until a logical bank activate command is allowed. The DDR memory controller implements bank staggering for refreshes, as shown in Figure 9-42 (TIMING_CFG_1 [REFREC] = 10 in this example).

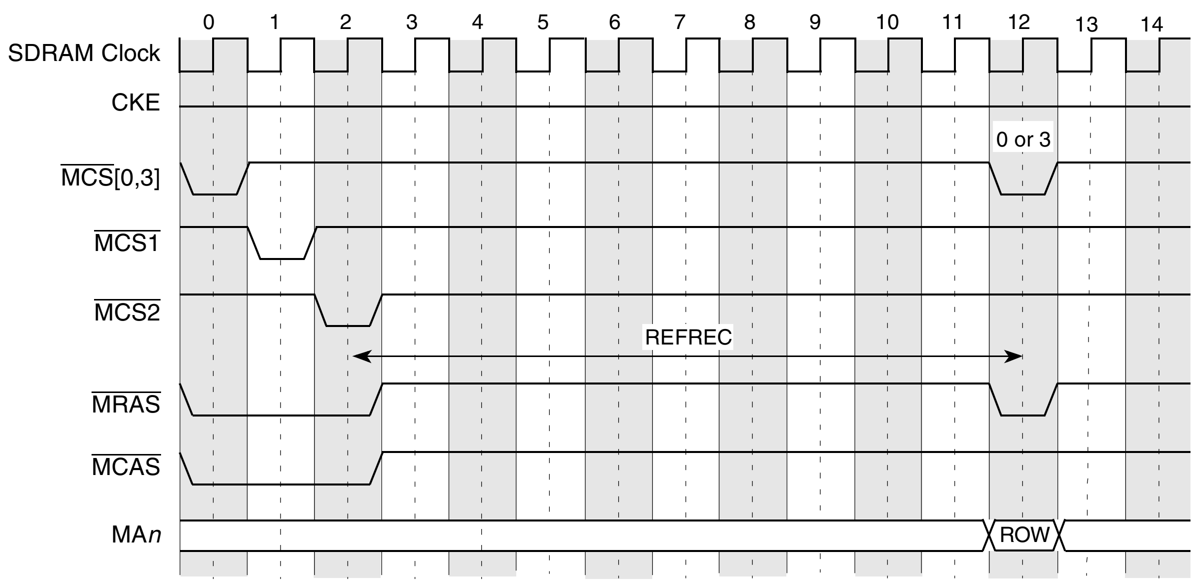


Figure 9-42. DDR SDRAM Bank Staggered Auto Refresh Timing

System software is responsible for optimal configuration of TIMING_CFG_1 [REFREC] and TIMING_CFG_3[EXT_REFREC] at reset. Configuration must be completed before DDR SDRAM accesses are attempted.

9.5.8.2 DDR SDRAM Refresh and Power-Saving Modes

In full-on mode, the DDR memory controller supplies the normal auto refresh to SDRAM. In sleep mode, the DDR memory controller can be configured to take advantage of self-refreshing SDRAMs or to provide no refresh support. Self-refresh support is enabled with the SREN memory control parameter.

summarizes the refresh types available in each power-saving mode.

Note that in the absence of refresh support, system software must preserve DDR SDRAM data (such as by copying the data to disk) before entering the power-saving mode.

The dynamic power-saving mode uses the CKE DDR SDRAM pin to dynamically power down when there is no system memory activity. The CKE pin is negated when both of the following conditions are met:

- No memory refreshes are scheduled
- No memory accesses are scheduled

CKE is reasserted when a new access or refresh is scheduled or the dynamic power mode is disabled. This mode is controlled with DDR_SDRAM_CFG[DYN_PWR_MGMT].

Dynamic power management mode offers tight control of the memory system's power consumption by trading power for performance through the use of CKE. Powering up the DDR SDRAM when a new memory reference is scheduled causes an access latency penalty, depending on whether active or precharge powerdown is used, along with the settings of TIMING_CFG_0[ACT_PD_EXIT] and TIMING_CFG_0[PRE_PD_EXIT]. A penalty of 1 cycle is shown in Figure 9-43.

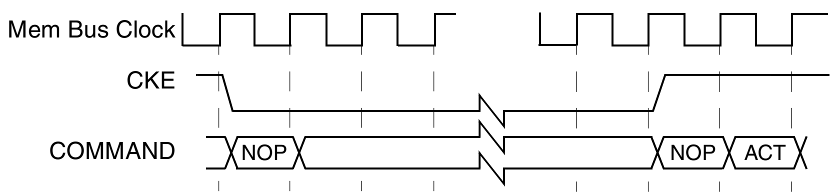


Figure 9-43. DDR SDRAM Power-Down Mode

9.5.8.2.1 Self-Refresh in Sleep Mode

The entry and exit timing for self-refreshing SDRAMs is shown in Figure 9-44 and Figure 9-45.

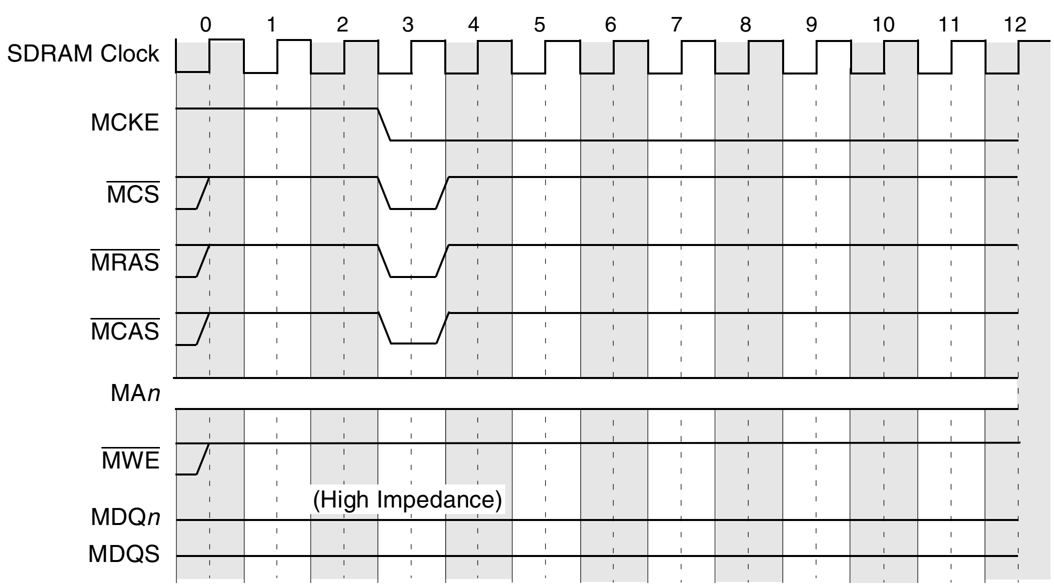


Figure 9-44. DDR SDRAM Self-Refresh Entry Timing

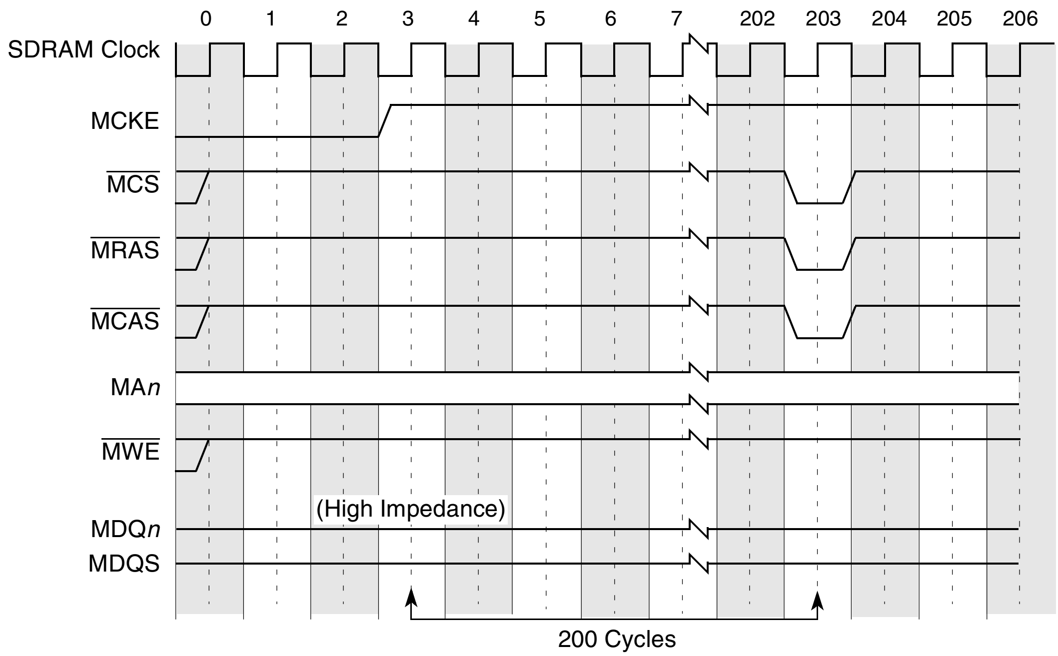


Figure 9-45. DDR SDRAM Self-Refresh Exit Timing

9.5.9 DDR Data Beat Ordering

Transfers to and from memory are always performed in four- or eight-beat bursts (four beats = 32 bytes when a 64-bit bus is used). For transfer sizes other than four or eight beats, the data transfers are still operated as four- or eight-beat bursts. If ECC is enabled and either the access is not doubleword aligned or the size is not a multiple of a doubleword, a full read-modify-write is performed for a write to SDRAM. If ECC is disabled or both the access is doubleword aligned with a size that is a multiple of a doubleword, the data masks (MDM[0–8] (MDM[0–4] for 32-bit bus) can be used to prevent the writing of unwanted data to SDRAM. The DDR memory controller also uses data masks to prevent all unintended full double words from writing to SDRAM. For example, if a write transaction is desired with a size of one double word (8 bytes), then the second, third, and fourth beats of data are not written to DRAM.

Table 9-47 lists the data beat sequencing to and from the DDR SDRAM and the data queues for each of the possible transfer sizes with each of the possible starting double-word offsets. All underlined double-word offsets are valid for the transaction.

Table 9-47. Memory Controller–Data Beat Ordering

Transfer Size	Starting Double-Word Offset	Double-Word Sequence[1] to/from DRAM and Queues
1 double word	0 1 2 3	**0** - 1 - 2 - 3 **1** - 2 - 3 - 0 **2** - 3 - 0 - 1 **3** - 0 - 1 - 2
2 double words	0 1 2	**0 - 1** - 2 - 3 **1 - 2** - 3 - 0 **2 - 3** - 0 - 1
3 double words	0 1	**0 - 1 - 2** - 3 **1 - 2 - 3** - 0

1 All underlined **Double**-word offsets are valid for the transaction.

9.5.10 Page Mode and Logical Bank Retention

The DDR memory controller supports an open/closed page mode with an allowable open page for each logical bank of DRAM used. In closed page mode for DDR SDRAMs, the DDR memory controller uses the SDRAM auto-precharge feature, which allows the controller to indicate that the page be automatically closed by the DDR SDRAM after the READ or WRITE access. This is performed using MA[10] of the address during the COMMAND phase of the access to enable auto-precharge. Auto-precharge is non-persistent in that it is either enabled or disabled for each individual READ or WRITE command. It can, however, be enabled or disabled separately for each chip select.

When the DDR memory controller operates in open page mode, it retains the currently active SDRAM page by not issuing a precharge command. The page remains opens until one of the following conditions occurs:

- Refresh interval is met.
- The user-programmable DDR_SDRAM_INTERVAL[BSTOPRE] value is exceeded.
- There is a logical bank row collision with another transaction that must be issued.

Page mode can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, using page mode can save two to three clock cycles for subsequent burst accesses that hit in an active page. Also, better performance can be obtained using more banks, especially in systems which use many different channels. Page mode is disabled by clearing DDR_SDRAM_INTERVAL[BSTOPRE] or setting CS*n*_CONFIG[AP_*n*EN].

9.5.11 Error Checking and Correcting (ECC)

The DDR memory controller supports error checking and correcting (ECC) for the data path between the core master and system memory. The memory detects all double-bit errors, detects all multi-bit errors within a nibble, and corrects all single-bit errors. Other errors may be detected, but are not guaranteed to be corrected or detected. Multiple-bit errors are always reported when error reporting is enabled. When a single-bit error occurs, the single-bit error counter register is incremented, and its value compared to the single-bit error trigger register. An error is reported when these values are equal. The single-bit error registers can be programmed such that minor memory faults are corrected and ignored, but a catastrophic memory failure generates an interrupt.

For writes that are smaller than 64 bits, the DDR memory controller performs a double-word read from system memory of the address for the write (checking for errors), and merges the write data with the data read from memory. Then, a new ECC code is generated for the merged double word. The data and ECC code is then written to memory. If a multi-bit error is detected on the read, the transaction completes the read-modify-write to keep the DDR memory controller from hanging. However, the corrupt data is masked on the write, so the original contents in SDRAM remain unchanged.

The syndrome encodings for the ECC code are shown in Table 9-48 and Table 9-49.

In 32-bit mode, Table 9-48 is split into 2 halves. The first half, consisting of rows 0–31, is used to calculate the ECC bits for the first 32 data bits of any 64-bit granule of data. This always applies to the odd data beats on the DDR data bus. The second half of the table, consisting of rows 32–63, is used to calculate the ECC bits for the second 32 bits of any 64-bit granule of data. This always applies to the even data beats on the DDR data bus.

Table 9-48. DDR SDRAM ECC Syndrome Encoding

Data Bit	Syndrome Bit								Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
0	•	•						•	32			•	•				•
1	•		•					•	33			•		•			•
2	•			•				•	34	•		•		•			
3	•				•			•	35		•	•		•			
4	•	•				•			36			•	•		•		
5	•		•			•			37			•		•	•		
6	•			•		•			38	•		•		•	•		•
7	•				•	•			39		•	•		•	•		•
8	•	•					•		40			•	•			•	
9	•		•				•		41			•		•		•	

Table 9-48. DDR SDRAM ECC Syndrome Encoding (continued)

Data Bit	Syndrome Bit								Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7
10	•			•			•		42	•		•		•		•	•
11	•				•		•		43		•	•		•		•	•
12	•	•				•	•	•	44			•	•		•	•	•
13	•		•			•	•	•	45			•		•	•	•	•
14	•			•		•	•	•	46	•		•		•	•	•	
15	•				•	•	•	•	47		•	•		•	•	•	
16		•	•					•	48		•				•	•	
17		•		•				•	49			•			•	•	
18		•			•			•	50				•		•	•	
19	•	•			•				51	•					•	•	
20		•	•			•			52		•				•		•
21		•		•		•			53			•			•		•
22		•			•	•			54				•		•		•
23	•	•			•	•		•	55	•					•		•
24		•	•				•		56		•					•	•
25		•		•			•		57			•				•	•
26		•			•		•		58				•			•	•
27	•	•			•		•	•	59	•						•	•
28		•	•			•	•	•	60				•	•		•	
29		•		•		•	•	•	61	•			•	•		•	•
30		•			•	•	•	•	62		•		•	•		•	•
31	•	•			•	•	•		63			•	•	•		•	•

Table 9-49. DDR SDRAM ECC Syndrome Encoding (Check Bits)

Check Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
0	•							
1		•						
2			•					
3				•				
4					•			
5						•		
6							•	
7								•

9.5.12 Error Management

The DDR memory controller detects four different kinds of errors: training, single-bit, multi-bit, and memory select errors. The following discussion assumes all the relevant error detection, correction, and reporting functions are enabled as described in Section 9.4.1.26, “Memory Error Interrupt Enable (ERR_INT_EN),” Section 9.4.1.25, “Memory Error Disable (ERR_DISABLE),” and Section 9.4.1.24, “Memory Error Detect (ERR_DETECT).”

Single-bit errors are counted and reported based on the ERR_SBE value. When a single-bit error is detected, the DDR memory controller does the following:

- Corrects the data
- Increments the single-bit error counter ERR_SBE[SBEC]
- Generates a critical interrupt if the counter value ERR_SBE[SBEC] equals the programmable threshold ERR_SBE[SBET]
- Completes the transaction normally

If a multi-bit error is detected for a read, the DDR memory controller logs the error and generates the interrupt (if enabled, as described in Section 9.4.1.25, “Memory Error Disable (ERR_DISABLE)”). Another error the DDR memory controller detects is a memory select error, which causes the DDR memory controller to log the error and generate a critical interrupt (if enabled, as described in Section 9.4.1.24, “Memory Error Detect (ERR_DETECT)”). This error is detected if the address from the memory request does not fall into any of the enabled, programmed chip select address ranges. For all memory select errors, the DDR memory controller does not issue any transactions onto the pins after the first read has returned data strobes. If the DDR memory controller is not using sample points, then a dummy transaction is issued to DDR SDRAM with the first enabled chip select. In this case, the source port on the pins is forced to 0x1F to show the transaction is not real. Table 9-50 shows the errors with their descriptions. The final error the memory controller detects is the automatic calibration error. This error is set if the memory controller detects an error during its training sequence.

Table 9-50. Memory Controller Errors

Category	Error	Descriptions	Action	Detect Register
Notification	Single-bit ECC threshold	The number of ECC errors has reached the threshold specified in the ERR_SBE.	The error is reported through interrupt if enabled.	The error control register only logs read versus write, not full type
Access Error	Multi-bit ECC error	A multi-bit ECC error is detected during a read, or read-modify-write memory operation.		
	Memory select error	Read, or write, address does not fall within the address range of any of the memory banks.		

9.6 Initialization/Application Information

System software must configure the DDR memory controller, using a memory polling algorithm at system start-up, to correctly map the size of each bank in memory. Then, the DDR memory controller uses its bank map to assert the appropriate $\overline{MCSn}$ signal for memory accesses according to the provided bank depths. System software must also configure the DDR memory controller at system start-up to appropriately multiplex the row and column address bits for each bank. Refer to row-address configuration in Section 9.4.1.2, “Chip Select Configuration (CSn_CONFIG).” Address multiplexing occurs according to these configuration bits.

At system reset, initialization software (boot code) must set up the programmable parameters in the memory interface configuration registers. See Section 9.4.1, “Register Descriptions,” for more detailed descriptions of the configuration registers. These parameters are shown in Table 9-51.

Table 9-51. Memory Interface Configuration Register Initialization Parameters

Name	Description	Parameter		Section/page
CS*n*_BNDS	Chip select memory bounds	SA*n* EA*n*		9.4.1.1/9-10
CS*n*_CONFIG	Chip select configuration	CS_*n*_EN AP_*n*_EN ODT_RD_CFG ODT_WR_CFG	BA_BITS_CS_*n* ROW_BITS_CS_*n* COL_BITS_CS_*n*	9.4.1.2/9-10
TIMING_CFG_3	Extended timing parameters for fields in TIMING_CFG_1	EXT_REFREC		9.4.1.3/9-12
TIMING_CFG_0	Timing configuration	RWT WRT RRT WWT	ACT_PD_EXIT PRE_PD_EXIT ODT_PD_EXIT MRS_CYC	9.4.1.4/9-13
TIMING_CFG_1	Timing configuration	PRETOACT ACTTOPRE ACTTORW CASLAT	REFREC WRREC ACTTOACT WRTORD	9.4.1.5/9-15
TIMING_CFG_2	Timing configuration	ADD_LAT CPO WR_LAT RD_TO_PRE	WR_DATA_DELAY CKE_PLS FOUR_ACT	9.4.1.6/9-17

Table 9-51. Memory Interface Configuration Register Initialization Parameters (continued)

Name	Description	Parameter		Section/page
DDR_SDRAM_CFG	Control configuration	SREN ECC_EN RD_EN SDRAM_TYPE DYN_PWR 32_BE 8_BE DBW	NCAP 2T_EN BA_INTLV_CTL x32_EN HSE BI	9.4.1.7/9-19
DDR_SDRAM_CFG_2	Control configuration	SR_IE DLL_RST_DIS DQS_CFG ODT_CFG	NUM_PR D_INIT	9.4.1.8/9-22
DDR_SDRAM_MODE	Mode configuration	ESDMODE SDMODE		9.4.1.9/9-24
DDR_SDRAM_MODE_2	Mode configuration	ESDMODE2 ESDMODE3		9.4.1.10/9-25
DDR_SDRAM_INTERVAL	Interval configuration	REFINT BSTOPRE		9.4.1.12/9-28
DDR_DATA_INIT	Data initialization configuration register	INIT_VALUE		9.4.1.13/9-28
DDR_SDRAM_CLK_CNTL	Clock adjust	CLK_ADJUST		9.4.1.14/9-29
DDR_INIT_ADDR	Initialization address	INIT_ADDR		9.4.1.15/9-29

9.6.1 Programming Differences Between Memory Types

Depending on the memory type used, certain fields must be programmed differently. Table 9-52 illustrates the differences in certain fields for different memory types. Note that this table does not list all fields that must be programmed.

Table 9-52. Programming Differences between Memory Types

Parameter	Description	Differences		Section/page
AP*n*_EN	Chip Select *n* Auto Precharge Enable	DDR1	Can be used to place chip select *n* in auto precharge mode	9.4.1.2/9-10
		DDR2	Can be used to place chip select *n* in auto precharge mode	
ODT_RD_CFG	Chip Select ODT Read Configuration	DDR1	Should always be set to 000	9.4.1.2/9-10
		DDR2	Can be enabled to assert ODT if desired. This could be set differently depending on system topology. However, systems with only 1 chip select typically not uses ODT when issuing reads to the memory.	

Table 9-52. Programming Differences between Memory Types (continued)

Parameter	Description	Differences		Section/page
ODT_WR_CFG	Chip Select ODT Write Configuration	DDR1	Should always be set to 000	9.4.1.2/9-10
		DDR2	Can be enabled to assert ODT if desired. This could be set differently depending on system topology. However, ODT typically is set to assert for the chip select that is getting written to (value would be set to 001).	
ODT_PD_EXIT	ODT Powerdown Exit	DDR1	Should be set to 0001	9.4.1.4/9-13
		DDR2	Should be set according to the DDR2 specifications for the memory used. The JEDEC parameter this applies to is t_{AXPD}.	
PRETOACT	Precharge to Activate Timing	DDR1	Should be set according to the specifications for the memory used (t_{RP})	9.4.1.5/9-15
		DDR2	Should be set according to the specifications for the memory used (t_{RP})	
ACTTOPRE	Activate to Precharge Timing	DDR1	Should be set, along with the Extended Activate to Precharge Timing, according to the specifications for the memory used (t_{RAS})	9.4.1.5/9-15
		DDR2	Should be set, along with the Extended Activate to Precharge Timing, according to the specifications for the memory used (t_{RAS})	
ACTTORW	Activate to Read/Write Timing	DDR1	Should be set according to the specifications for the memory used (t_{RCD})	9.4.1.5/9-15
		DDR2	Should be set according to the specifications for the memory used (t_{RCD})	
CASLAT	CAS Latency	DDR1	Should be set, along with the Extended CAS Latency, to the desired CAS latency	9.4.1.5/9-15
		DDR2	Should be set, along with the Extended CAS Latency, to the desired CAS latency	
REFREC	Refresh Recovery	DDR1	Should be set, along with the Extended Refresh Recovery, to the specifications for the memory used (t_{RFC})	9.4.1.5/9-15
		DDR2	Should be set, along with the Extended Refresh Recovery, to the specifications for the memory used (T_{RFC})	
WRREC	Write Recovery	DDR1	Should be set according to the specifications for the memory used (t_{WR})	9.4.1.5/9-15
		DDR2	Should be set according to the specifications for the memory used (t_{WR})	
ACTTOACT	Activate *A* to Activate *B*	DDR1	Should be set according to the specifications for the memory used (t_{RRD})	9.4.1.5/9-15
		DDR2	Should be set according to the specifications for the memory used (t_{RRD})	

Table 9-52. Programming Differences between Memory Types (continued)

Parameter	Description	Differences		Section/page
WRTORD	Write to Read Timing	DDR1	Should be set according to the specifications for the memory used (t_{WTR})	9.4.1.5/9-15
		DDR2	Should be set according to the specifications for the memory used (t_{WTR})	
ADD_LAT	Additive Latency	DDR1	Should be set to 000	9.4.1.6/9-17
		DDR2	Should be set to the desired additive latency. This must be set to a value less than TIMING_CFG_1[ACTTORW]	
WR_LAT	Write Latency	DDR1	Should be set to 001	9.4.1.6/9-17
		DDR2	Should be set to CAS latency – 1 cycle. For example, if the CAS latency if 5 cycles, then this field should be set to 100 (4 cycles).	
RD_TO_PRE	Read to Precharge Timing	DDR1	Should be set to 010 if burst length is 4 and 100 if burst length is 8	9.4.1.6/9-17
		DDR2	Should be set according to the specifications for the memory used (t_{RTP}). Time between read and precharge for non-zero value of additive latency (AL) is a minimum of AL + t_{RTP} cycles.	
CKE_PLS	Minimum CKE Pulse Width	DDR1	Can be set to 001	9.4.1.6/9-17
		DDR2	Should be set according to the specifications for the memory used (t_{CKE})	
FOUR_ACT	Four Activate Window	DDR1	Should be set to 00001	9.4.1.6/9-17
		DDR2	Should be set according to the specifications for the memory used (t_{FAW}). Only applies to eight logical banks.	
RD_EN	Registered DIMM Enable	DDR1	If registered DIMMs are used, then this field should be set to 1	9.4.1.7/9-19
		DDR2	If registered DIMMs are used, then this field should be set to 1	
8_BE	8-beat burst enable	DDR1	If a 32-bit bus is used, and 8-beat bursts are desired, then this field should be set to 1	9.4.1.7/9-19
		DDR2	Should be set to 0	
2T_EN	2T Timing Enable	DDR1	In heavily loaded systems, this can be set to 1 to gain extra timing margin on the interface at the cost of address/command bandwidth.	9.4.1.7/9-19
		DDR2	In heavily loaded systems, this can be set to 1 to gain extra timing margin on the interface at the cost of address/command bandwidth.	

Table 9-52. Programming Differences between Memory Types (continued)

Parameter	Description	Differences		Section/page
DLL_RST_DIS	DLL Reset Disable	DDR1	Should typically be set to 0, unless it is desired to bypass the DLL reset when exiting self refresh.	9.4.1.8/9-22
		DDR2	Should typically be set to 0, unless it is desired to bypass the DLL reset when exiting self refresh.	
DQS_CFG	DQS Configuration	DDR1	Should be set to 00	9.4.1.8/9-22
		DDR2	Should be set to 00	
ODT_CFG	ODT Configuration	DDR1	Should be set to 00	9.4.1.8/9-22
		DDR2	Can be set for termination at the IOs according to system topology. Typically, if ODT is enabled, then the internal IOs should be set up for termination only during reads to DRAM.	
BSTOPR	Burst To Precharge Interval	DDR1	Can be set to any value, depending on the application. Auto precharge can be enabled by setting this field to all 0s.	9.4.1.12/9-28
		DDR2	Can be set to any value, depending on the application. Auto precharge can be enabled by setting this field to all 0s.	

9.6.2 DDR SDRAM Initialization Sequence

After configuration of all parameters is complete, system software must set DDR_SDRAM_CFG[MEM_EN] to enable the memory interface. Note that 200 μs must elapse after DRAM clocks are stable (DDR_SDRAM_CLK_CNTL[CLK_ADJUST] is set and any chip select is enabled) before MEM_EN can be set, so a delay loop in the initialization code may be necessary if software is enabling the memory controller. If DDR_SDRAM_CFG[BI] is not set, the DDR memory controller conducts an automatic initialization sequence to the memory, which follows the memory specifications. If the bypass initialization mode is used, then software can initialize the memory through the DDR_SDRAM_MD_CNTL register.

9.6.3 Using Forced Self-Refresh Mode to Implement a Battery-Backed RAM System

This section describes the options offered by this device to support battery-backed main memory.

9.6.3.1 Hardware Based Self-Refresh

An external voltage sense device can be connected to this device through the IRQ1/DDR_SR_REQ signal. The assertion of this signal, when a voltage drop has been identified, is then be sensed by the DDR controller. The DDR controller immediately responds by sending a self refresh command to main memory, telling it to enter self-refresh mode.The DDR controller to the assertion of DDR_SR_REQ is controlled by DDR_SDRAM_CFG_2[SR_IE]; see Section 9.4.1.8, "DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)," for further information on this bit. Note that if IRQ1/DDR_SR_REQ is defined to be used as DDR_SR_REQ, it precludes any other usage of this pin as IRQ1; thus IRQ1 source should be disabled in the interrupt controller. Section 8.5.13, “System External Interrupt Mask Register (SEMSR),” contains a description of the register used to accomplish this task.

9.6.3.2 Software Based Self-Refresh

The DDR controller also has a software-programmable bit, DDR_SDRAM_CFG_2[FRC_SR], that immediately puts main memory into self-refresh mode. See Section 9.4.1.8, “DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2),” for a description of this register.

It is expected that a critical interrupt routine triggered by an external voltage sensing device has time to set this bit.

9.6.3.3 Bypassing Re-initialization During Battery-Backed Operation

The DDR controller offers an initialization bypass feature (DDR_SDRAM_CFG[BI]), which system designers may use to prevent re-initialization of main memory during system power-on following an abnormal shutdown. See Section 9.4.1.7, “DDR SDRAM Control Configuration (DDR_SDRAM_CFG),” for information on this bit and Section 9.4.1.15, “DDR Initialization Address (DDR_INIT_ADDR),” for a discussion of avoiding possible ECC errors in this mode.

Note that the DDR controller automatically waits 200 DRAM cycles before issuing any command after the assertion of MCKE[0:1] when this mode is used.

Chapter 10
Enhanced Local Bus Controller

This chapter describes the enhanced local bus controller (eLBC) block. It describes the external signals and the memory-mapped registers as well as a functional description of the general-purpose chip-select machine (GPCM), NAND Flash control machine (FCM), and user-programmable machines (UPMs) of the eLBC. Finally, it includes an initialization and applications information section with many specific examples of its use.

10.1 Introduction

Figure 10-1 is a functional block diagram of the eLBC, which supports three interfaces: GPCM, FCM, and UPM controllers.

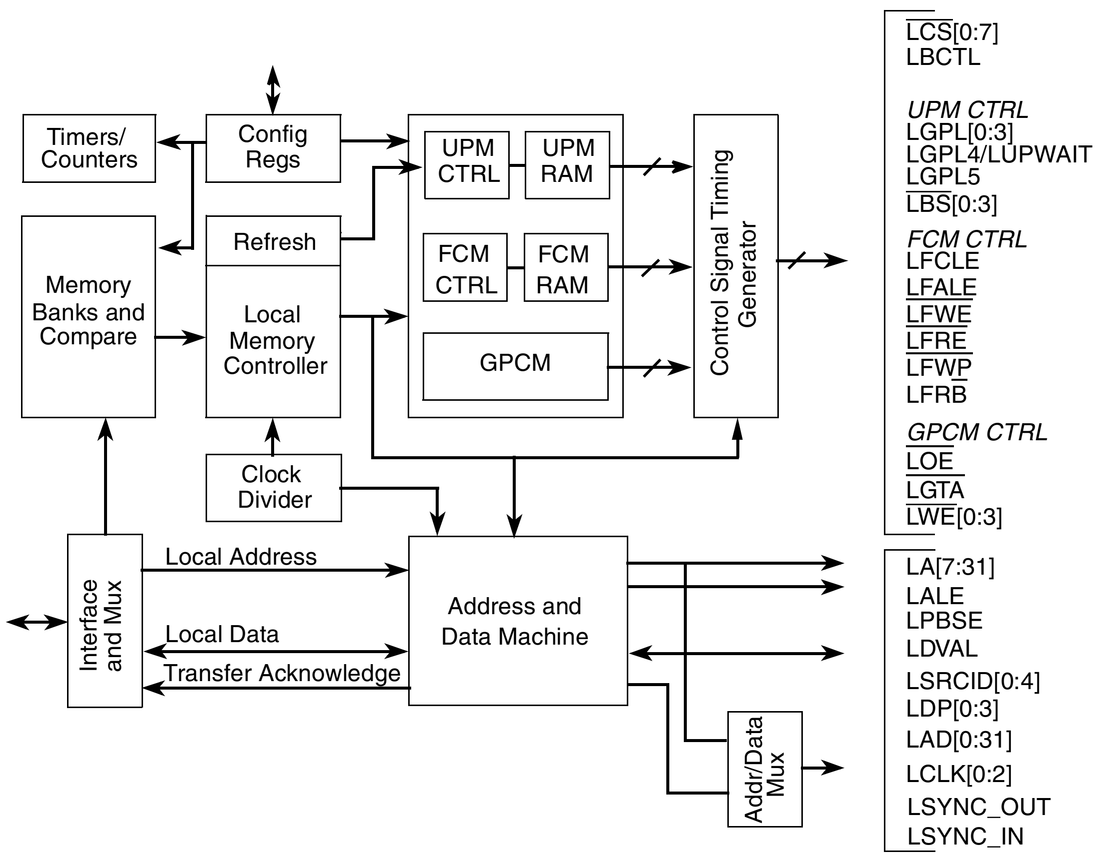


Figure 10-1. Enhanced Local Bus Controller Block Diagram

10.1.1 Overview

The main component of the eLBC is its memory controller, which provides a seamless interface to many types of memory devices and peripherals. The memory controller is responsible for controlling eight memory banks shared by a GPCM, an FCM, and up to three UPMs. As such, it supports a minimal glue logic interface to SRAM, EPROM, NOR Flash EEPROM, NAND Flash EEPROM, burstable RAM, regular DRAM devices, extended data output DRAM devices, and other peripherals. The external address latch signal (LALE) allows multiplexing of addresses with data signals to reduce the device pin count.The eLBC also includes a number of data checking and protection features such as data parity generation and checking, write protection and a bus monitor to ensure that each bus cycle is terminated within a user-specified period.

10.1.2 Features

The eLBC main features are as follows:

- Memory controller with eight memory banks
  - 32-bit address decoding with mask
  - Variable memory block sizes (32 Kbytes to 4 Gbytes)
  - Selection of control signal generation on a per-bank basis
  - Data buffer controls activated on a per-bank basis
  - Automatic segmentation of large transactions into memory accesses optimized for bus width and addressing capability
  - Odd/even parity checking including read-modify-write (RMW) parity for single accesses
  - Write-protection capability
  - Parity byte-select
- General-purpose chip-select machine (GPCM)
  - Compatible with SRAM, EPROM, NOR Flash EEPROM, and peripherals
  - Global (boot) chip-select available at system reset
  - Boot chip-select support for 8-, 16-, and 32-bit devices
  - Minimum three-clock access to external devices
  - Four byte-write-enable signals ($\overline{\text{LWE}}$[0:3])
  - Output enable signal ($\overline{\text{LOE}}$)
  - External access termination signal ($\overline{\text{LGTA}}$)
- NAND Flash control machine (FCM)
  - Compatible with small (512+16 bytes) and large (2048+64 bytes) page parallel NAND Flash EEPROM
  - Global (boot) chip-select available at system reset, with 4-Kbyte boot block buffer for execute-in-place boot loading
  - Boot chip-select support for 8-and 16-bit devices
  - Dual 2-Kbyte/eight 512-byte buffers allow simultaneous data transfer during flash reads and programming

- Interrupt-driven block transfer for reads and writes
- Programmable command and data transfer sequences of up to eight steps supported
- Generic command and address registers support proprietary flash interfaces
- Block write locking to ensure system security and integrity

- Three user-programmable machines (UPMs)
  - Programmable-array-based machine controls external signal timing with a granularity of up to one quarter of an external bus clock period
  - User-specified control-signal patterns run when an internal master requests a single-beat or burst read or write access.
  - UPM refresh timer runs a user-specified control signal pattern to support refresh
  - User-specified control-signal patterns can be initiated by software
  - Each UPM can be defined to support DRAM devices with depths of 64, 128, 256, and 512 Kbytes, and 1, 2, 4, 8, 16, 32, 64, 128, and 256 Mbytes
  - Support for 8-, 16-, and 32-bit devices
  - Page mode support for successive transfers within a burst
  - Internal address multiplexing supporting 64-, 128-, 256-, and 512-Kbyte, and 1-, 2-, 4-, 8-, 16-, 32-, 64-, 128-, and 256-Mbyte page banks
- Optional monitoring of transfers between local bus internal masters and local bus slaves (local bus error reporting on interrupt and status registers)
- Support for phase-locked loop (PLL) with software-configurable bypass for low frequency bus clocks

10.1.3 Modes of Operation

The eLBC provides one GPCM, one FCM, and three UPMs for the local bus, with no restriction on how many of the eight banks (chip selects) can be programmed to operate with any given machine. The internal transaction address is limited to 32 bits, so all chip selects must fall within the 4-Gbyte window addressed by the internal transaction address. When a memory transaction is dispatched to the eLBC, the internal transaction address is compared with the address information of each bank (chip select). The corresponding machine assigned to that bank (GPCM, FCM, or UPM) then takes ownership of the external signals that control the access and maintains control until the transaction ends. Thus, with the eLBC in GPCM or FCM, or UPM mode, only one of the eight chip selects is active at any time for the duration of the transaction except in the case of UPM refresh where all UPM machines that are enabled for refresh have concurrent chip select assertion.

10.1.3.1 eLBC Bus Clock and Clock Ratios

The eLBC supports ratios of 2, 4, and 8 between the faster internal (system) clock and slower external bus clock (LCLK[0–2]). This ratio is software programmable through the clock ratio register (LCRR[CLKDIV]). This ratio affects the resolution of signal timing shifts in GPCM and FCM modes and the interpretation of UPM array words in UPM mode. The bus clock is driven identically onto pins, LCLK[0–2], to allow the clock load to be shared equally across a set of signal nets, thereby enhancing the edge rates of the bus clock.

10.1.3.2 Source ID Debug Mode

The eLBC provides the ID of a transaction source on external device pins. When those pins are selected, the 5-bit internal ID of the current transaction source appears on LSRCID[0–4] whenever valid address or data is available on the eLBC external pins. The reserved value of 0x1F, which indicates invalid address or data, appears on the source ID pins at all other times. The combination of a valid source ID (any value except 0x1F) and the value of external address latch enable (LALE) and data valid (LDVAL) facilitate capturing useful debug data as follows:

- If a valid source ID is detected on LSRCID[0–4] and LALE is asserted, a valid full 32-bit address may be latched from LAD[0–26] and combined with LA[27–31].
- If a valid source ID is detected on LSRCID[0–4] and LDVAL is asserted, valid data may be latched from LAD.

The LSRCID[0–4] and LDVAL signals are multiplexed with other functions sharing the same external pins. Refer to Chapter 3, "Signal Descriptions," and Chapter 4, "Reset, Clocking, and Initialization," to learn how to enable the LSRCID/LDVAL pins.

10.2 External Signal Descriptions

Table 10-1 contains a list of external signals related to the eLBC and summarizes their function. Note that during assertion of $\overline{\text{HRESET}}$, the PLL is initially unlocked, so the LCLK and LSYNC_OUT values are likely to be unstable/jittery for several microseconds after the PLL locks, stable clock signals are driven on these signals.

Table 10-1. Signal Properties—Summary

Name	Alternate Function(s)	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
LALE	—	—	External address latch enable	1	O	Reset_cfg
$\overline{\text{LCS}}$[0]	—	—	Chip select 0	1	O	Reset_cfg
$\overline{\text{LCS}}$[1–7]	—	—	Chip selects [1–7]	7	O	All high
$\overline{\text{LWE0}}$/ $\overline{\text{LFWE}}$/ $\overline{\text{LBS0}}$	$\overline{\text{LWE0}}$	GPCM	Write enable 0	1	O	Reset_cfg
	$\overline{\text{LFWE}}$	FCM	Write enable	1		
	$\overline{\text{LBS0}}$	UPM	Byte (lane) select 0	1		
$\overline{\text{LWE}}$[1–3]/ $\overline{\text{LBS}}$[1–3]	$\overline{\text{LWE}}$	GPCM	Write enable 1–3	3	O	Reset_cfg
	$\overline{\text{LBS}}$	UPM	Byte (lane) select 1–3	3		
LGPL0/ LFCLE	LGPL0	UPM	General purpose line 0	1	O	Reset_cfg
	LFCLE	FCM	Flash command latch enable	1		

Table 10-1. Signal Properties—Summary (continued)

Name	Alternate Function(s)	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
LGPL1/ LFALE	LGPL1	UPM	General purpose line 1	1	O	Reset_cfg
	LFALE	FCM	Flash address latch enable	1		
$\overline{LOE}$/ LGPL2/ $\overline{LFRE}$	$\overline{LOE}$	GPCM	Output enable	1	O	
	$\overline{LFRE}$	FCM	Flash read enable	1		
	LGPL2	UPM	General purpose line 2	1		
LGPL3/ $\overline{LFWP}$	LGPL3	UPM	General purpose line 3	1	O	Reset_cfg
	$\overline{LFWP}$	FCM	Flash write protect	1		
$\overline{LGTA}$/ LFR$\overline{B}$/ LGPL4/ LUPWAIT/ LPBSE	$\overline{LGTA}$	GPCM	Transaction termination	1	I	High-Z
	LFR$\overline{B}$	FCM	Flash ready/$\overline{busy}$, open-drain shared pin	1	I	
	LGPL4	UPM	General purpose line 4	1	O	
	LUPWAIT	UPM	External device wait	1	I	
	LPBSE	—	Local bus parity byte select	1	O	
LGPL5	—	UPM	General purpose line 5	1	O	Reset_cfg
LBCTL	—	—	Data buffer control	1	O	
LA[7–31]	—	—	Non-multiplexed address bus	25	O	
LAD[0–31]	—	—	Multiplexed address/data bus	32	I/O	
LDP[0–3]	—	—	Local bus data parity	4	I/O	High-Z
LCLK[0–2]	—	—	Local bus clocks	3	O	Driven
LSYNC_IN	—	—	PLL synchronize input	1	I	—
LSYNC_OUT	—	—	PLL synchronize output	1	O	Driven
LDVAL	—	eLBC debug	Local bus data valid	1	O	
LSRCID[0–4]	—	eLBC debug	Local bus source ID	5	O	

Table 10-2 shows the detailed external signal descriptions for the eLBC.

Table 10-2. Enhanced Local Bus Controller Detailed Signal Descriptions

Signal	I/O	Description	
LALE	O	External address latch enable. The local bus memory controller provides control for an external address latch, which allows address and data to be multiplexed on the device pins.	
		State Meaning	Asserted/Negated—LALE is asserted with the address at the beginning of each memory controller transaction. The number of cycles for which it is asserted is governed by the ORn[EAD] and LCRR[EADC] fields. Note that no other control signals are asserted during the assertion of LALE.

Table 10-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description	
LCS[0–7]	O	Chip selects. Eight chip selects are provided that are mutually exclusive.	
		State Meaning	Asserted/Negated—Used to enable specific memory devices or peripherals connected to the eLBC. LCS[0–7] are provided on a per-bank basis with LCS0 corresponding to the chip select for memory bank 0, which has the memory type and attributes defined by BR0 and OR0.
LWE0/ LFWE/ LBS0, LWE[1–3]/ LBS[1–3]	O	GPCM write enable 0/FCM write enable/UPM byte select 0. These signals select or validate each byte lane of the data bus. For banks with port sizes of 32 bits (as set by BR*n*[PS]), all four signals are defined. For a 16-bit port size, only bits 0–1 are defined; and for an 8-bit port size, bit 0 is the only defined signal. The least-significant address bits of each access also determine which byte lanes are considered valid for a given data transfer.	
		State Meaning	Asserted/Negated—For GPCM operation, LWE[0–3] assert for each byte lane enabled for writing. LFWE enables command, address, and data writes to NAND Flash EEPROMs controlled by FCM. LBS[0:3] are programmable byte-select signals in UPM mode. See Section 10.4.4.4, “RAM Array,” for programming details about LBS[0:3].
		Timing	Assertion/Negation—See Section 10.4.2, “General-Purpose Chip-Select Machine (GPCM),” for details regarding the timing of LWE[0:3].
LGPL0/ LFCLE	O	General purpose line 0/FCM command latch enable.	
		State Meaning	Asserted/Negated—In UPM mode, LGPL0 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode, LFCLE enables command cycles to NAND Flash EEPROMs.
LGPL1/ LFALE	O	General-purpose line 1/FCM address latch enable.	
		State Meaning	Asserted/Negated—In UPM mode, LGPL1 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode, LFALE enables address cycles to NAND Flash EPROMs.
LOE/LGPL2/ LFRE	O	GPCM output enable/General-purpose line 2/FCM read enable.	
		State Meaning	Asserted/Negated—Controls the output buffer of memory when accessing memory/devices in GPCM mode. In UPM mode, LGPL2 is one of six general purpose signals; it is driven with a value programmed into the UPM array. LFRE enables data read cycles from NAND Flash EEPROMs controlled by FCM.
LGPL3/ LFWP	O	General-purpose line 3/FCM write protect.	
		State Meaning	Asserted/Negated—In UPM mode, LGPL3 is one of six general purpose signals; it is driven with a value programmed into the UPM array. In FCM mode LFWP protects NAND Flash EEPROMs from accidental erasure and programming when LFWP is asserted low—see Section 10.3.1.16, “Flash Mode Register (FMR),” for programming of FCM operations to control LFWP.

Table 10-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{LGTA}}$/LGPL4/ LFR$\overline{\text{B}}$/ LUPWAIT/ LPBSE	I/O	GPCM transfer acknowledge/General-purpose line 4/FCM Flash ready-busy/UPM wait/parity byte select.	
		State Meaning	Asserted/Negated—Input in GPCM or FCM modes used for transaction termination. It may also be configured as one of six general-purpose output signals when in UPM mode or as an input to force the UPM controller to wait for the memory/device. FCM uses LFR$\overline{\text{B}}$ to stall during long-latency read and programming operations, continuing once LFR$\overline{\text{B}}$ returns high. When configured as LPBSE, it disables any use in GPCM, FCM, or UPM modes. Because systems that use read-modify-write parity require an additional memory device, they must generate a byte-select like a normal data device. ANDing LBS[0:3] through external logic to achieve the logical function of this byte-select can affect memory access timing. The LBC provides this optional byte-select signal connection to RMW-parity devices.
LGPL5	O	General-purpose line 5	
		State Meaning	Asserted/Negated—One of six general purpose signals when in UPM mode, and drives a value programmed in the UPM array.
LBCTL	O	Data buffer control. The memory controller activates LBCTL for the local bus when a GPCM-, UPM-, or FCM-controlled bank is accessed. Buffer control is disabled by setting OR*n*[BCTLD].	
		State Meaning	Asserted/Negated—The LBCTL pin normally functions as a write/$\overline{\text{read}}$ control for a bus transceiver connected to the LAD lines. Note that an external data buffer must not drive the LAD lines in conflict with the eLBC when LBCTL is high, because LBCTL remains high after reset and during address phases.
LA[7–31]	O	Nonmultiplexed address bus. All bits driven on LA[7–31] are defined for 8-bit port sizes. For 32-bit port sizes, LA[30–31] are don't cares; for 16-bit port sizes LA[31] is a don't care.	
		State Meaning	Asserted/Negated—LA is the address bus used to transmit addresses to external RAM devices. Refer to Section 10.5, "Initialization/Application Information," for address signal multiplexing.
LAD[0:31]	I/O	Multiplexed address/data bus. For configuration of a port size in BR*n*[PS] as 32 bits, all of LAD[0–31] must be connected to the external RAM data bus, with LAD[0–7] occupying the most significant byte lane (at address offset 0). For a port size of 16 bits, LAD[0–7] connect to the most-significant byte lane (at address offset 0), while LAD[8–15] connect to the least-significant byte lane (at address offset 1); LAD[16–31] are unused for 16-bit port sizes. For a port size of 8 bits, only LAD[0–7] are connected to the external RAM.	
		State Meaning	Asserted/Negated—LAD is the shared 32-bit address/data bus through which external RAM devices transfer data and receive addresses.
		Timing	Assertion/Negation—During assertion of LALE, LAD are driven with the RAM address for the access to follow. External logic should propagate the address on LAD while LALE is asserted, and latch the address upon negation of LALE. After LALE is negated, LAD are either driven by write data or are made high-impedance by the eLBC in order to sample read data driven by an external device. Following the last data transfer of a write access, LAD are again taken into a high-impedance state.

Table 10-2. Enhanced Local Bus Controller Detailed Signal Descriptions (continued)

Signal	I/O	Description	
LDP[0–3]	I/O	Local bus data parity. Drives and receives the data parity corresponding with the data phases on LAD for GPCM and UPM controlled banks.	
		State Meaning	Asserted/Negated—During write accesses, a parity bit is generated for each 8 bits of LAD[0–31], such that LDP0 is even/odd parity for LAD[0–7], while LDP[3] is even/odd parity for LAD[24–31]. Unused byte lanes for port sizes less than 32 bits have undefined parity.
		Timing	Assertion/Negation—Drive and receive the data parity corresponding with the data phases on LAD. For read accesses, the parity bits for each byte lane are sampled on LDP[0–3] with the same timing that read data is sampled on LAD. LDP[0–3] change impedance in concert with LAD.
LCLK[0–2]	O	Local bus clocks	
		State Meaning	Asserted/Negated—LCLK[0–2] drive an identical bus clock signal for distributed loads. If the eLBC PLL is enabled (see LCRR[PBYP], Figure 10-18), the bus clock phase is shifted earlier than transitions on other eLBC signals (such as LAD*n* and $\overline{\text{LCS}n}$) by a time delay matching the delay of the PLL timing loop set up between LSYNC_OUT and LSYNC_IN.
LSYNC_OUT	O	PLL synchronization out.	
		State Meaning	Asserted/Negated—A replica of the bus clock, appearing on LSYNC_OUT, should be propagated through a passive timing loop and returned to LSYNC_IN for achieving correct PLL lock.
		Timing	Assertion/Negation—The time delay of the timing loop should be such that it compensates for the round-trip flight time of LCLK[0–2] and clocked drivers in the system. No load other than a timing loop should be placed on LSYNC_OUT.
LSYNC_IN	I	PLL synchronization in.	
		State Meaning	Asserted/Negated—See description of LSYNC_OUT.
LDVAL	O	Local bus data valid (eLBC debug mode only)	
		State Meaning	Asserted/Negated—For a read, LDVAL asserts for one bus cycle in the cycle immediately preceding the sampling of read data on LAD. For a write, LDVAL asserts for one bus cycle during the final cycle for which the current write data on LAD is valid. During burst transfers, LDVAL asserts for each data beat.
		Timing	Assertion/Negation—Valid only while the eLBC is in system debug mode. In debug mode, LDVAL asserts when the eLBC generates a data transfer acknowledge.
LSRCID[0–4]	O	Local bus source ID (eLBC debug mode only). In debug mode, all LSRCID[0–4] pins are driven high unless LSRCID[0–4] is driving a debug source ID for identifying the internal system device controlling the eLBC.	
		State Meaning	Asserted/Negated—Remain high until the last bus cycle of the assertion of LALE, in which case the source ID of the address is indicated, or until LDVAL is asserted, in which case the source ID relating to the data transfer is indicated. In case of address debug, LSRCID[0–4] is valid only when the address on LAD consists of all physical address bits—with optional padding—for reconstructing the system address presented to the eLBC.

10.3 Memory Map/Register Definition

Table 10-3 shows the memory mapped registers of the eLBC. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 10-3. Enhanced Local Bus Controller Registers

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	BR0—Base register 0	R/W	0x0000_*nnnn*	10.3.1.1/10-11
0x008	BR1—Base register 1	R/W	0x0000_0000	10.3.1.1/10-11
0x010	BR2—Base register 2	R/W	0x0000_0000	10.3.1.1/10-11
0x018	BR3—Base register 3	R/W	0x0000_0000	10.3.1.1/10-11
0x020	BR4—Base register 4	R/W	0x0000_0000	10.3.1.1/10-11
0x028	BR5—Base register 5	R/W	0x0000_0000	10.3.1.1/10-11
0x030	BR6—Base register 6	R/W	0x0000_0000	10.3.1.1/10-11
0x038	BR7—Base register 7	R/W	0x0000_0000	10.3.1.1/10-11
0x004	OR0—Options register 0	R/W	0x0000_0FF7	10.3.1.2/10-12
0x00C	OR1—Options register 1	R/W	0x0000_0000	10.3.1.2/10-12
0x014	OR2—Options register 2	R/W	0x0000_0000	10.3.1.2/10-12
0x01C	OR3—Options register 3	R/W	0x0000_0000	10.3.1.2/10-12
0x024	OR4—Options register 4	R/W	0x0000_0000	10.3.1.2/10-12
0x02C	OR5—Options register 5	R/W	0x0000_0000	10.3.1.2/10-12
0x034	OR6—Options register 6	R/W	0x0000_0000	10.3.1.2/10-12
0x03C	OR7—Options register 7	R/W	0x0000_0000	10.3.1.2/10-12
0x040–0x064	Reserved	—	—	—
0x068	MAR—UPM address register	R/W	0x0000_0000	10.3.1.3/10-20
0x06C	Reserved	—	—	—
0x070	MAMR—UPMA mode register	R/W	0x0000_0000	10.3.1.4/10-21
0x074	MBMR—UPMB mode register	R/W	0x0000_0000	10.3.1.4/10-21
0x078	MCMR—UPMC mode register	R/W	0x0000_0000	10.3.1.4/10-21
0x07C–0x080	Reserved	—	—	—
0x084	MRTPR—Memory refresh timer prescaler register	R/W	0x0000_0000	10.3.1.5/10-23
0x088	MDR—UPM/FCM data register	R/W	0x0000_0000	10.3.1.6/10-23
0x08C	Reserved	—	—	—

Table 10-3. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x090	LSOR—Special operation initiation register	R/W	0x0000_0000	10.3.1.7/10-24
0x094–0x09C	Reserved	—	—	—
0x0A0	LURT—UPM refresh timer	R/W	0x0000_0000	10.3.1.4/10-21
0x0A4–0x0AC	Reserved	—	—	—
0x0B0	LTESR—Transfer error status register	w1c	0x0000_0000	10.3.1.9/10-26
0x0B4	LTEDR—Transfer error disable register	R/W	0x0000_0000	10.3.1.10/10-28
0x0B8	LTEIR—Transfer error interrupt register	R/W	0x0000_0000	10.3.1.11/10-29
0x0BC	LTEATR—Transfer error attributes register	R/W	0x0000_0000	10.3.1.12/10-30
0x0C0	LTEAR—Transfer error address register	R/W	0x0000_0000	10.3.1.13/10-31
0x0C4–0x0CC	Reserved	—	—	—
0x0D0	LBCR—Configuration register	R/W	0x0000_0000	10.3.1.14/10-31
0x0D4	LCRR—Clock ratio register	R/W	0x8000_0008	10.3.1.15/10-33
0x0D8–0x0DC	Reserved	—	—	—
0x0E0	FMR—Flash mode register	R/W	0x0000_0*n*00	10.3.1.16/10-34
0x0E4	FIR—Flash instruction register	R/W	0x0000_0000	10.3.1.17/10-36
0x0E8	FCR—Flash command register	R/W	0x0000_0000	10.3.1.18/10-37
0x0EC	FBAR—Flash block address register	R/W	0x0000_0000	10.3.1.19/10-38
0x0F0	FPAR—Flash page address register	R/W	0x0000_0000	10.3.1.20/10-38
0x0F4	FBCR—Flash byte count register	R/W	0x0000_0000	10.3.1.21/10-40
0x0F8–0x0FC	Reserved	—	—	—

10.3.1 Register Descriptions

This section provides a detailed description of the eLBC configuration, status, and control registers with detailed bit and field descriptions.

Address offsets in the eLBC address range that are not defined in Table 10-3 should not be accessed for reading or writing. Similarly, only zero should be written to reserved bits of defined registers, as writing ones can have unpredictable results in some cases.

Bits designated as write-one-to-clear are cleared only by writing ones to them. Writing zeros to them has no effect.

10.3.1.1 Base Registers (BR0–BR7)

The base registers (BR*n*), shown in Figure 10-2, contain the base address and address types for each memory bank. The memory controller uses this information to compare the address bus value with the current address accessed. Each register (bank) includes a memory attribute and selects the machine for memory operation handling. Note that after system reset, BR0[V] is set, BR1[V]–BR7[V] are cleared, and the value of BR0[PS] reflects the initial port size configured by the boot ROM location .

Offset BR0: 0x0_5000
BR1: 0x0_5008
BR2: 0x0_5010
BR3: 0x0_5018
BR4: 0x0_5020
BR5: 0x0_5028
BR6: 0x0_5030
BR7: 0x0_5038

Access: Read/Write

Bits	0–16	17–18	19–20	21–22	23	24–26	27	28–29	30	31
R/W	BA	—	PS	DECC	WP	MSEL	—	—	—	V
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	P S	0 0	0	0 0 M	0	0 0	1	V[1]

[1] BR0 has its valid bit (V) set for RCWH[ROMLOC] = LBC. Thus bank 0 is valid with the port size (PS) configured from RCWH[ROMLOC] and RCWH[RLEXT] as loaded during reset. M = 0 for MSEL of GPCM, 1 for MSEL of FCM at boot. All other base registers have all bits cleared to zero during reset.

Figure 10-2. Base Registers (BR*n*)

Table 10-4 describes BR*n* fields.

Table 10-4. BR*n* Field Descriptions

Bits	Name	Description
0–16	BA	Base address. The upper 17 bits of each base register are compared to the address on the address bus to determine if the bus master is accessing a memory bank controlled by the memory controller. Used with the address mask bits OR*n*[AM].
17–18	—	Reserved
19–20	PS	Port size. Specifies the port size of this memory region. For BR0, PS is configured from the . For all other banks the value is reset to 00 (port size not defined). 00 Reserved 01 8-bit 10 16-bit (not supported for FCM) 11 32-bit (not supported for FCM)
21–22	DECC	Specifies the method for data error checking. 00 Data error checking disabled, but normal parity generation for GPCM and UPM. No ECC generation for FCM. 01 Normal parity generation and checking for GPCM and UPM. ECC checking is enabled, but ECC generation is disabled, for FCM on full-page transfers. 10 Read-modify-write parity generation and normal parity checking for GPCM and UPM. ECC checking and generation are enabled for FCM on full-page transfers. 11 Reserved

Table 10-4. BR*n* Field Descriptions (continued)

Bits	Name	Description
23	WP	Write protect. 0 Read and write accesses are allowed. 1 Only read accesses are allowed. The memory controller does not assert $\overline{LCSn}$ on write cycles to this memory bank. LTESR[WP] is set (if WP is set) if a write to this memory bank is attempted, and a local bus error interrupt is generated (if enabled), terminating the cycle.
24–26	MSEL	Machine select. Specifies the machine to use for handling memory operations. 000 GPCM (possible reset value) 001 FCM (possible reset value) 010 Reserved 011 Reserved 100 UPMA 101 UPMB 110 UPMC 111 Reserved
27	—	Reserved
28–29	—	Reserved
30	—	Reserved
31	V	Valid bit. Indicates that the contents of the BR*n* and OR*n* pair are valid. $\overline{LCSn}$ does not assert unless V is set (an access to a region that has no valid bit set may cause a bus time-out). After a system reset, only BR0[V] is set. 0 This bank is invalid. 1 This bank is valid.

10.3.1.2 Option Registers (OR0–OR7)

The OR*n* registers define the sizes of memory banks and access attributes. The OR*n* attribute bits support the following three modes of operation as defined by BR*n*[MSEL]:

- GPCM mode
- FCM mode
- UPM mode

The OR*n* registers are interpreted differently depending on which of the three machine types is selected for that bank. Because bank 0 can be used to boot, the reset value of OR0 may be different depending on power-on configuration options. Table 10-5 shows the reset values for OR0.

Table 10-5. Reset value of OR0 Register

Boot Source	OR0 Reset Value
FCM (small page NAND Flash)	0000_03AE
FCM (large page NAND Flash)	0000_07AE
GPCM	0000_0FF7
eLBC not used as a boot source	0000_0F07

10.3.1.2.1 Address Mask

The address mask field of the option registers (OR*n*[AM]) masks up to 17 corresponding BR*n*[BA] fields. The 15 LSBs of the 32-bit internal transaction address do not participate in bank address matching in selecting a bank for access. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. Table 10-6 shows memory bank sizes from 32 Kbytes to 4 Gbytes.

Table 10-6. Memory Bank Sizes in Relation to Address Mask

AM	Memory Bank Size
0000_0000_0000_0000_0	4 Gbytes
1000_0000_0000_0000_0	2 Gbytes
1100_0000_0000_0000_0	1 Gbyte
1110_0000_0000_0000_0	512 Mbytes
1111_0000_0000_0000_0	256 Mbytes
1111_1000_0000_0000_0	128 Mbytes
1111_1100_0000_0000_0	64 Mbytes
1111_1110_0000_0000_0	32 Mbytes
1111_1111_0000_0000_0	16 Mbytes
1111_1111_1000_0000_0	8 Mbytes
1111_1111_1100_0000_0	4 Mbytes
1111_1111_1110_0000_0	2 Mbytes
1111_1111_1111_0000_0	1 Mbyte
1111_1111_1111_1000_0	512 Kbytes
1111_1111_1111_1100_0	256 Kbytes
1111_1111_1111_1110_0	128 Kbytes
1111_1111_1111_1111_0	64 Kbytes
1111_1111_1111_1111_1	32 Kbytes

10.3.1.2.2 Option Registers (OR*n*)—GPCM Mode

Figure 10-3 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects the GPCM machine.

Offset OR0: 0x0_5004
OR1: 0x0_500c
OR2: 0x0_5014
OR3: 0x0_501c
OR4: 0x0_5024
OR5: 0x0_502c
OR6: 0x0_5034
OR7: 0x0_503c

Access: Read/Write

	0–15
R/W	AM
Reset	All zeros

	16	17–18	19	20	21–22	23	24–27	28	29	30	31
R/W	AM	—	BCTLD	CSNT	ACS	XACS	SCY	SETA	TRLX	EHTR	EAD
Reset	All zeros[1]										

[1] Refer to Table 10-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 10-3. Option Registers (OR*n*) in GPCM Mode

Table 10-7 describes OR*n* fields for GPCM mode.

Table 10-7. OR*n*—GPCM Field Descriptions

Bits	Name	Description
0–16	AM	GPCM address mask. Masks corresponding BR*n* bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. 0 Corresponding address bits are masked and therefore don't care for address checking. 1 Corresponding address bits are used in the comparison between base and transaction addresses.
17–18	—	Reserved
19	BCTLD	Buffer control disable. Disables assertion of LBCTL during access to the current memory bank. 0 LBCTL is asserted upon access to the current memory bank. 1 LBCTL is not asserted upon access to the current memory bank.

Table 10-7. OR*n*—GPCM Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>20</td><td>CSNT</td><td>Chip select negation time. Determines when LCSn and LWE are negated during an external memory write access handled by the GPCM, provided that ACS ≠ 00 (when ACS = 00, only LWE is affected by the setting of CSNT). This helps meet address/data hold times for slow memories and peripherals.
0 LCSn and LWE are negated normally.
1 LCSn and LWE are negated earlier depending on the value of LCRR[CLKDIV].
<table>
<tr><th>LCRR [CLKDIV]</th><th>CSNT</th><th>Meaning</th></tr>
<tr><td>x</td><td>0</td><td>LCSn and LWE are negated normally.</td></tr>
<tr><td>2</td><td>1</td><td>LCSn and LWE are negated normally.</td></tr>
<tr><td>4 or 8</td><td>1</td><td>LCSn and LWE are negated one quarter bus clock cycle earlier.</td></tr>
</table></td></tr>
<tr><td>21–22</td><td>ACS</td><td>Address to chip-select setup. Determines the delay of the LCSn assertion relative to the address change when the external memory access is handled by the GPCM. At system reset, OR0[ACS] = 11
<table>
<tr><th>LCRR [CLKDIV]</th><th>Value</th><th>Meaning</th></tr>
<tr><td rowspan="2">x</td><td>00</td><td>LCSn is output at the same time as the address lines. Note that this overrides the value of CSNT such that CSNT = 0.</td></tr>
<tr><td>01</td><td>Reserved.</td></tr>
<tr><td rowspan="2">2</td><td>10</td><td>LCSn is output one half bus clock cycle after the address lines.</td></tr>
<tr><td>11</td><td>LCSn is output one half bus clock cycle after the address lines.</td></tr>
<tr><td rowspan="2">4 or 8</td><td>10</td><td>LCSn is output one quarter bus clock cycle after the address lines.</td></tr>
<tr><td>11</td><td>LCSn is output one half bus clock cycle after the address lines.</td></tr>
</table></td></tr>
<tr><td>23</td><td>XACS</td><td>Extra address to chip-select setup. Setting this bit increases the delay of the LCSn assertion relative to the address change when the external memory access is handled by the GPCM. After a system reset, OR0[XACS] = 1.
0 Address to chip-select setup is determined by ORx[ACS] and LCRR[CLKDIV].
1 Address to chip-select setup is extended (see Table 10-30 and Table 10-31).</td></tr>
<tr><td>24–27</td><td>SCY</td><td>Cycle length in bus clocks. Determines the number of wait states inserted in the bus cycle, when the GPCM handles the external memory access. Thus it is the main parameter for determining cycle length. The total cycle length depends on other timing attribute settings. After a system reset, OR0[SCY] = 1111.
0000 No wait states
0001 1 bus clock cycle wait state
...
1111 15 bus clock cycle wait states</td></tr>
<tr><td>28</td><td>SETA</td><td>External address termination.
0 Access is terminated internally by the memory controller unless the external device asserts LGTA earlier to terminate the access.
1 Access is terminated externally by asserting the LGTA external pin. (Only LGTA can terminate the access).</td></tr>
</table>

Table 10-7. OR*n*—GPCM Field Descriptions (continued)

Bits	Name	Description
29	TRLX	Timing relaxed. Modifies the settings of timing parameters for slow memories or peripherals. 0 Normal timing is generated by the GPCM. 1 Relaxed timing on the following parameters: • Adds an additional cycle between the address and control signals (only if ACS is not equal to 00). • Doubles the number of wait states specified by SCY, providing up to 30 wait states. • Works in conjunction with EHTR to extend hold time on read accesses. • $\overline{\text{LCS}n}$ (only if ACS is not equal to 00) and $\overline{\text{LWE}}$ signals are negated one cycle earlier during writes.
30	EHTR	Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access. TRLX / EHTR / Meaning: 0 / 0 / The memory controller generates normal timing. No additional cycles are inserted. 0 / 1 / 1 idle clock cycle is inserted. 1 / 0 / 4 idle clock cycles are inserted. 1 / 1 / 8 idle clock cycles are inserted.
31	EAD	External address latch delay. Allow extra bus clock cycles when using external address latch (LALE). 0 No additional bus clock cycles (LALE asserted for one bus clock cycle only) 1 Extra bus clock cycles are added (LALE is asserted for the number of bus clock cycles specified by LCRR[EADC]).

10.3.1.2.3 Option Registers (OR*n*)—FCM Mode

Figure 10-4 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects the FCM machine.

Offset OR0: 0x0_5004
OR1: 0x0_500c
OR2: 0x0_5014
OR3: 0x0_501c
OR4: 0x0_5024
OR5: 0x0_502c
OR6: 0x0_5034
OR7: 0x0_503c

Access: Read/Write

	0–15
R/W	AM
Reset	All zeros

	16	17–18	19	20	21	22	23	24	25–27	28	29	30	31
R/W	AM	—	BCTLD	—	PGS	CSCT	CST	CHT	SCY	RST	TRLX	EHTR	—
Reset	All zeros[1]												

[1] Refer to Table 10-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 10-4. Option Registers (OR*n*) in FCM Mode

Table 10-8 describes OR*n* fields for FCM mode.

Table 10-8. OR*n*—FCM Field Descriptions

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0–16</td><td>AM</td><td>FCM address mask. Masks corresponding BRn bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map.
0 Corresponding address bits are masked.
1 Corresponding address bits are used in the comparison between base and transaction addresses.</td></tr>
<tr><td>17–18</td><td>—</td><td>Reserved</td></tr>
<tr><td>19</td><td>BCTLD</td><td>Buffer control disable. Disables assertion of LBCTL during access to the current memory bank.
0 LBCTL is asserted upon access to the current memory bank.
1 LBCTL is not asserted upon access to the current memory bank.</td></tr>
<tr><td>20</td><td>—</td><td>Reserved</td></tr>
<tr><td>21</td><td>PGS</td><td>NAND Flash EEPROM page size, buffer size, and block size.
0 Page size of 512 main area bytes plus 16 spare area bytes (small page devices); FCM RAM buffers are 1 Kbyte each; Flash block size of 16 Kbytes.
1 Page size of 2048 main area bytes plus 64 spare area bytes (large page devices); FCM RAM buffers are 4 Kbytes each; Flash block size of 128 Kbytes.</td></tr>
<tr><td>22</td><td>CSCT</td><td>Chip select to command time. Determines how far in advance $\overline{\text{LCS}n}$ is asserted prior to any bus activity during a NAND Flash access handled by the FCM. This helps meet chip-select setup times for slow memories.
<table>
<tr><th>TRLX</th><th>CSCT</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The chip-select is asserted 1 clock cycle before any command.</td></tr>
<tr><td>0</td><td>1</td><td>The chip-select is asserted 4 clock cycles before any command.</td></tr>
<tr><td>1</td><td>0</td><td>The chip-select is asserted 2 clock cycles before any command.</td></tr>
<tr><td>1</td><td>1</td><td>The chip-select is asserted 8 clock cycles before any command.</td></tr>
</table></td></tr>
<tr><td>23</td><td>CST</td><td>Command setup time. Determines the delay of $\overline{\text{LFWE}}$ assertion relative to the command, address, or data change when the external memory access is handled by the FCM.
<table>
<tr><th>TRLX</th><th>CST</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The write-enable is asserted coincident with any command.</td></tr>
<tr><td>0</td><td>1</td><td>The write-enable is asserted 0.25 clock cycles after any command, address, or data.</td></tr>
<tr><td>1</td><td>0</td><td>The write-enable is asserted 0.5 clock cycles after any command, address, or data.</td></tr>
<tr><td>1</td><td>1</td><td>The write-enable is asserted 1 clock cycle after any command, address, or data.</td></tr>
</table></td></tr>
</table>

Table 10-8. OR*n*—FCM Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>24</td><td>CHT</td><td>Command hold time. Determines the $\overline{\text{LFWE}}$ negation prior to the command, address, or data change when the external memory access is handled by the FCM.
<table>
<tr><th>TRLX</th><th>CHT</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The write-enable is negated 0.5 clock cycles before any command, address, or data change.</td></tr>
<tr><td>0</td><td>1</td><td>The write-enable is negated 1 clock cycle before any command, address, or data change.</td></tr>
<tr><td>1</td><td>0</td><td>The write-enable is negated 1.5 clock cycles before any command, address, or data change.</td></tr>
<tr><td>1</td><td>1</td><td>The write-enable is negated 2 clock cycles before any command, address, or data change.</td></tr>
</table></td></tr>
<tr><td>25–27</td><td>SCY</td><td>Cycle length in bus clocks. Determines:
• the number of wait states inserted in command, address, or data transfer bus cycles, when the FCM handles the external memory access. Thus it is the main parameter for determining cycle length. The total cycle length depends on other timing attribute settings.
• the delay between command/address writes and data write cycles, or the delay between write cycles and read cycles from NAND Flash EEPROM. A delay of 4×(2+SCY) clock cycles (TRLX = 0) or 8×(2+SCY) clock cycles (TRLX = 1) is inserted between the last write and the first data transfer to/from NAND Flash devices.
• the delay between a command write and the first sample point of the RDY/$\overline{\text{BSY}}$ pin (connected to LFR$\overline{\text{B}}$). LFR$\overline{\text{B}}$ is not sampled until 8×(2+SCY) clock cycles (TRLX = 0) or 16×(2+SCY) clock cycles (TRLX = 1) have elapsed following the command.
000 No extra wait states
001 1 bus clock cycle wait state
...
111 7 bus clock cycle wait states</td></tr>
<tr><td>28</td><td>RST</td><td>Read setup time. Determines the delay of $\overline{\text{LFRE}}$ assertion relative to sampling of read data when the external memory access is handled by the FCM.
<table>
<tr><th>TRLX</th><th>RST</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The read-enable is asserted 0.75 clock cycles prior to any wait states.</td></tr>
<tr><td>0</td><td>1</td><td>The read-enable is asserted 1 clock cycle prior to any wait states.</td></tr>
<tr><td>1</td><td>0</td><td>The read-enable is asserted 0.5 clock cycles prior to any wait states.</td></tr>
<tr><td>1</td><td>1</td><td>The read-enable is asserted 1 clock cycle prior to any wait states.</td></tr>
</table></td></tr>
<tr><td>29</td><td>TRLX</td><td>Timing relaxed. Modifies the settings of timing parameters for slow memories.
0 Normal timing is generated by the FCM.
1 Relaxed timing on the following parameters:
• Doubles the number of clock cycles between $\overline{\text{LCS}n}$ assertion and commands.
• Doubles the number of wait states specified by SCY, providing up to 14 wait states.
• Works in conjunction with CST and RST to extend command/address/data setup times.
• Adds one clock cycle to the command/address/data hold times.
• Works in conjunction with CBT to extend the wait time for read/busy status sampling by 16 clock cycles.
• Works in conjunction with EHTR to double hold time on read accesses.</td></tr>
</table>

Table 10-8. ORn—FCM Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>30</td><td>EHTR</td><td>Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access.
<table>
<tr><th>TRLX</th><th>EHTR</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>1 idle clock cycle is inserted.</td></tr>
<tr><td>0</td><td>1</td><td>2 idle clock cycles are inserted.</td></tr>
<tr><td>1</td><td>0</td><td>4 idle clock cycles are inserted.</td></tr>
<tr><td>1</td><td>1</td><td>8 idle clock cycles are inserted.</td></tr>
</table>
</td></tr>
<tr><td>31</td><td>—</td><td>Reserved</td></tr>
</table>

10.3.1.2.4 Option Registers (ORn)—UPM Mode

Figure 10-5 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects a UPM machine.

Offset OR0: 0x0_5004
OR1: 0x0_500c
OR2: 0x0_5014
OR3: 0x0_501c
OR4: 0x0_5024
OR5: 0x0_502c
OR6: 0x0_5034
OR7: 0x0_503c

Access: Read/Write

	0–15
R/W	AM
Reset	All zeros

	16	17–18	19	20–22	23	24–28	29	30	31
R/W	AM	—	BCTLD	—	BI	—	TRLX	EHTR	EAD
Reset	All zeros[1]								

[1] Refer to Table 10-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 10-5. Option Registers (ORn) in UPM Mode

Table 10-9 describes BR*n* fields for UPM mode.

Table 10-9. OR*n*—UPM Field Descriptions

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0–16</td><td>AM</td><td>UPM address mask. Masks corresponding BRn bits. Masking address bits independently allows external devices of different size address ranges to be used. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map.
0 Corresponding address bits are masked.
1 The corresponding address bits are used in the comparison with address pins.</td></tr>
<tr><td>17–18</td><td>—</td><td>Reserved</td></tr>
<tr><td>19</td><td>BCTLD</td><td>Buffer control disable. Disables assertion of LBCTL during access to the current memory bank.
0 LBCTL is asserted upon access to the current memory bank.
1 LBCTL is not asserted upon access to the current memory bank.</td></tr>
<tr><td>20–22</td><td>—</td><td>Reserved</td></tr>
<tr><td>23</td><td>BI</td><td>Burst inhibit. Indicates if this memory bank supports burst accesses.
0 The bank supports burst accesses.
1 The bank does not support burst accesses. The selected UPM executes burst accesses as a series of single accesses.</td></tr>
<tr><td>24–28</td><td>—</td><td>Reserved</td></tr>
<tr><td>29</td><td>TRLX</td><td>Timing relaxed. Works in conjunction with EHTR to extend hold time on read accesses.</td></tr>
<tr><td>30</td><td>EHTR</td><td>Extended hold time on read accesses. Indicates with TRLX how many cycles are inserted between a read access from the current bank and the next access.
<table>
<tr><th>TRLX</th><th>EHTR</th><th>Meaning</th></tr>
<tr><td>0</td><td>0</td><td>The memory controller generates normal timing. No additional cycles are inserted.</td></tr>
<tr><td>0</td><td>1</td><td>1 idle clock cycle is inserted.</td></tr>
<tr><td>1</td><td>0</td><td>4 idle clock cycles are inserted.</td></tr>
<tr><td>1</td><td>1</td><td>8 idle clock cycles are inserted.</td></tr>
</table></td></tr>
<tr><td>31</td><td>EAD</td><td>External address latch delay. Allow extra bus clock cycles when using external address latch (LALE).
0 No additional bus clock cycles (LALE asserted for one bus clock cycle only)
1 Extra bus clock cycles are added (LALE is asserted for the number of bus clock cycles specified by LCRR[EADC]).</td></tr>
</table>

10.3.1.3 UPM Memory Address Register (MAR)

Figure 10-6 shows the fields of the UPM memory address register (MAR).

Offset 0x0_5068 Access: Read/Write

	0–31
R	A
W	
Reset	All zeros

Figure 10-6. UPM Memory Address Register (MAR)

Table 10-10 describes the MAR fields.

Table 10-10. MAR Field Descriptions

Bits	Name	Description
0–31	A	Address that can be output to the address signals under control of the AMX bits in the UPM RAM word.

10.3.1.4 UPM Mode Registers (M*x*MR)

The UPM machine mode registers (MAMR, MBMR and MCMR), shown in Figure 10-7, contain the configuration for the three UPMs.

Offset MAMR: 0x0_5070
MBMR: 0x0_5074
MCMR: 0x0_5078

Access: Read/Write

	0	1	2–3	4	5–7	8–9	10–12	13	14–15
R/W	—	RFEN	OP	UWPL	AM	DS	G0CL	GPL4	RLF
Reset	All zeros								

	16–17	18–21	22–25	26–31
R/W	RLF	WLF	TLF	MAD
Reset	All zeros			

Figure 10-7. UPM Mode Registers (M*x*MR)

Table 10-11 describes UPM mode fields.

Table 10-11. M*x*MR Field Descriptions

Bits	Name	Description
0	—	Reserved
1	RFEN	Refresh enable. Indicates that the UPM needs refresh services. This bit must be set for UPMA (refresh executor) if refresh services are required on any UPM assigned chip selects. If MAMR[RFEN] = 0, no refresh services can be provided, even if UPMB and/or UPMC have their RFEN bit set. 0 Refresh services are not required 1 Refresh services are required
2–3	OP	Command opcode. Determines the command executed by the UPM*n* when a memory access hits a UPM assigned bank. 00 Normal operation 01 Write to UPM array. On the next memory access that hits a UPM assigned bank, write the contents of the MDR into the RAM location pointed to by MAD. After the access, MAD is automatically incremented. 10 Read from UPM array. On the next memory access that hits a UPM assigned bank, read the contents of the RAM location pointed to by MAD into the MDR. After the access, MAD is automatically incremented. 11 Run pattern. On the next memory access that hits a UPM assigned bank, run the pattern written in the RAM array. The pattern run starts at the location pointed to by MAD and continues until the LAST bit is set in the RAM word.
4	UWPL	LUPWAIT polarity active low. Sets the polarity of the LUPWAIT pin when in UPM mode. 0 LUPWAIT is active high. 1 LUPWAIT is active low.

Table 10-11. MxMR Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>5–7</td><td>AM</td><td>Address multiplex size. Determines how the address of the current memory cycle can be output on the address pins. This field is needed when interfacing with devices requiring row and column addresses multiplexed on the same pins. See Section 10.4.4.4.7, “Address Multiplexing (AMX)” for more information.
000 Internal transaction address a[8–23] driven on [10–25]; LAD[0–15] driven low.
001 Internal transaction address a[7–22] driven on [10–25]; LAD[0–15] driven low.
010 Internal transaction address a[6–21] driven on [10–25]; LAD[0–15] driven low.
011 Internal transaction address a[5–20] driven on [10–25]; LAD[0–15] driven low.
100 Internal transaction address a[4–19] driven on [10–25]; LAD[0–15] driven low.
101 Internal transaction address a[3–18] driven on [10–25]; LAD[0–15] driven low.
110 Reserved
111 Reserved</td></tr>
<tr><td>8–9</td><td>DS</td><td>Disable timer period. Guarantees a minimum time between accesses to the same memory bank controlled by UPMn. The disable timer is turned on by the TODT bit in the RAM array word, and when expired, the UPMn allows the machine access to handle a memory pattern to the same bank. Accesses to a different bank by the same UPMn is also allowed. To avoid conflicts between successive accesses to different banks, the minimum pattern in the RAM array for a request serviced, should not be shorter than the period established by DS.
00 1-bus clock cycle disable period
01 2-bus clock cycle disable period
10 3-bus clock cycle disable period
11 4-bus clock cycle disable period</td></tr>
<tr><td>10–12</td><td>G0CL</td><td>General line 0 control. Determines which logical address line can be output to the LGPL0 pin when the UPMn is selected to control the memory access.
000 A12
001 A11
010 A10
011 A9
100 A8
101 A7
110 A6
111 A5</td></tr>
<tr><td>13</td><td>GPL4</td><td>LGPL4 output line disable. Determines how the LGPL4/LUPWAIT pin is controlled by the corresponding bits in the UPMn array. See Table 10-38 on page 10-81.
<table>
<tr><th rowspan="2">Value</th><th rowspan="2">LGPL4/LUPWAIT Pin Function</th><th colspan="2">Interpretation of UPM Word Bits</th></tr>
<tr><th>G4T1/DLT3</th><th>G4T3/WAEN</th></tr>
<tr><td>0</td><td>LGPL4 (output)</td><td>G4T1</td><td>G4T3</td></tr>
<tr><td>1</td><td>LUPWAIT (input)</td><td>DLT3</td><td>WAEN</td></tr>
</table></td></tr>
<tr><td>14–17</td><td>RLF</td><td>Read loop field. Determines the number of times a loop defined in the UPMn will be executed for a burst- or single-beat read pattern or when MxMR[OP] = 11 (RUN command)
0000 16
0001 1
0010 2
0011 3
...
1110 14
1111 15</td></tr>
</table>

Table 10-11. MxMR Field Descriptions (continued)

Bits	Name	Description
18–21	WLF	Write loop field. Determines the number of times a loop defined in the UPM*n* will be executed for a burst- or single-beat write pattern. 0000 16 0001 1 0010 2 0011 3 ... 1110 14 1111 15
22–25	TLF	Refresh loop field. Determines the number of times a loop defined in the UPM*n* will be executed for a refresh service pattern. 0000 16 0001 1 0010 2 0011 3 ... 1110 14 1111 15
26–31	MAD	Machine address. RAM address pointer for the command executed. This field is incremented by 1, each time the UPM is accessed and the OP field is set to WRITE or READ. Address range is 64 words per UPM*n*.

10.3.1.5 Memory Refresh Timer Prescaler Register (MRTPR)

The refresh timer prescaler register (MRTPR), shown in Figure 10-8, is used to divide the system clock to provide the UPM refresh timers clock.

Offset 0x0_5084 Access: Read/Write

	0–7	8–31
R W	PTP	—
Reset	All zeros	

Figure 10-8. Memory Refresh Timer Prescaler Register (MRTPR)

Table 10-12 describes MRTPR fields.

Table 10-12. MRTPR Field Descriptions

Bits	Name	Description
0–7	PTP	Refresh timers prescaler. Determines the period of the refresh timers input clock. The system clock is divided by PTP except when the value is 00000_0000, which represents the maximum divider of 256.
8–31	—	Reserved

10.3.1.6 UPM/FCM Data Register (MDR)

The memory data register (MDR), shown in Figure 10-9 and Figure 10-10, contains data written to or read from the RAM array for UPM read or write commands. MDR also contains data written to or read from an external NAND Flash EEPROM for FCM write address, write data, and read status commands. MDR must be set up before issuing a write command to the UPM, or before issuing a FCM operation sequence that uses MDR to source address or data bytes.

Offset 0x0_5088 Access: Read/Write

	0–31
R	D
W	
Reset	All zeros

Figure 10-9. UPM Data Register in UPM Mode (MDR)

Offset 0x0_5088 Access: Read/Write

	0–7	8–15	16–23	24–31
R	AS3	AS2	AS1	AS0
W				
Reset	All zeros			

Figure 10-10. FCM Data Register in FCM Mode (MDR)

Table 10-13 describes MDR[D].

Table 10-13. MDR Field Description

Bits	Name	Description
0–31	D	In UPM mode, D is the data to be read or written into the RAM array when a write or read command is supplied to the UPM (M*x*MR[OP] = 01 or M*x*MR[OP] = 10).
0–7	AS3	In FCM mode, AS3 is the fourth byte of address sent by a custom address write operation, or the fourth byte of data read from a read status operation.
8–15	AS2	In FCM mode, AS2 is the third byte of address sent by a custom address write operation, or the third byte of data read from a read status operation.
16–23	AS1	In FCM mode, AS1 is the second byte of address sent by a custom address write operation, or the second byte of data read from a read status operation.
24–31	AS0	In FCM mode, AS0 is the first byte of address sent by a custom address write operation, or the first byte of data read from a read status operation.

10.3.1.7 Special Operation Initiation Register (LSOR)

The special operation initiation register (LSOR), shown in Figure 10-11, is used by software to trigger a special operation on the indicated bank. Writing to LSOR activates a special operation on bank LSOR[BANK] provided that the bank is valid and controlled by a memory controller whose mode OP field is set to a value other than 'normal operation.' If eLBC is currently busy with a memory transaction, writing LSOR completes immediately, but the special operation request is queued until eLBC can service it. To avoid race conditions between software and a busy eLBC, registers that affect currently running special operation and LSOR must not be re-written before a pending special operation has been completed. The UPM and FCM have different indications of when such special operations are completed. The behavior of eLBC is unpredictable if special operation modes are altered between LSOR being written and the relevant memory controller completing that access.

UPM special operation modes are set in registers M*x*MR[OP], see Section 10.3.1.4, "UPM Mode Registers (MxMR)." FCM special operation modes are set in FMR[OP], see Section 10.3.1.16, "Flash Mode Register (FMR)." Writing LSOR has the same effect as setting a special controller mode and performing a dummy access to a bank associated with the controller in question, but use of LSOR avoids changing settings for the address space occupied by the bank. More details of special operation sequences appear in Section 10.4.4.2.1, "UPM Programming Example (Two Sequential Writes to the RAM Array)."

Offset 0x0_5090 Access: Read/Write

	0–28	29–31
R/W	—	BANK
Reset	All zeros	

Figure 10-11. Special Operation Initiation Register (LSOR)

Table 10-14 describes LSOR.

Table 10-14. LSOR Field Description

Bits	Name	Description
0–28	—	Reserved
29–31	BANK	Bank on which a special operation is initiated. If the bank identified by BANK is marked valid (BR*n*[V] set) and the bank is controlled by a memory controller whose current mode OP is non-zero—or a special operation—eLBC will request the special operation to be activated on the selected bank when this field is written. Otherwise, writing this field has no effect. 000 Bank 0 is triggered for special operation ... 111 Bank 7 is triggered for special operation

10.3.1.8 UPM Refresh Timer (LURT)

The UPM refresh timer (LURT), shown in Figure 10-12, generates a refresh request for all valid banks that selected a UPM machine and are refresh-enabled (M*x*MR[RFEN] = 1). Each time the timer expires, a qualified bank generates a refresh request using the selected UPM. The qualified banks rotate their requests.

Offset 0x0_50A0 Access: Read/Write

	0–7	8–31
R/W	LURT	—
Reset	All zeros	

Figure 10-12. UPM Refresh Timer (LURT)

Table 10-15 describes LURT fields.

Table 10-15. LURT Field Descriptions

Bits	Name	Description
0–7	LURT	UPM refresh timer period. Determines, along with the timer prescaler (MRTPR), the timer period according to the following equation: $$\text{TimerPeriod} = \frac{\text{LURT}}{\left(\frac{\text{Fsystemclock}}{\text{MRTPR[PTP]}}\right)}$$ Example: For a 266-MHz system clock and a required service rate of 15.6 µs, given MRTPR[PTP] = 32, the LURT value should be 128 decimal. 128/(266 MHz/32) = 15.4 µs, which is less than the required service period of 15.6 µs. Note that the reset value (0x00) sets the maximum period to 256 x MRTPR[PTP] system clock cycles.
8–31	—	Reserved

10.3.1.9 Transfer Error Status Register (LTESR)

The transfer error status register (LTESR) indicates the cause of an error or event. LTESR, shown in Figure 10-13, is a write-1-to-clear register. Reading LTESR occurs normally; however, write operations can clear but not set bits. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear only the write protect error bit (LTESR[WP]) without affecting other LTESR bits, 0x0400_0000 should be written to the register. After any error/event reported by LTESR, LTEATR[V] must be cleared for LTESR to updated again.

Offset 0x0_50B0 Access: w1c

Bits	0	1	2	3–4	5	6–7	8	9	10–11	12	13–15
R W	BM	FCT	PAR	—	WP	—	—	—	—	CS	—
Reset	All zeros										

Bits	16–30	31
R W	—	CC
Reset	All zeros	

Figure 10-13. Transfer Error Status Register (LTESR)

Table 10-16 describes LTESR fields.

Table 10-16. LTESR Field Descriptions

Bits	Name	Description
0	BM	Bus monitor time-out 0 No local bus monitor time-out occurred. 1 Local bus monitor time-out occurred. No data beat was acknowledged on the bus within LBCR[BMT] x LBCR[BMTPS] bus clock cycles from the start of a transaction.
1	FCT	FCM command time-out 0 No FCM command time-out occurred. 1 A CW0, CW1, CW2, or CW3 command issued to FCM timed-out with respect to the timer configured by FMR[CWTO].
2	PAR	Parity or ECC error 0 No local bus parity error 1 Local bus parity error (GPCM or UPM), or uncorrectable ECC error (FCM). LTEATR[PB] indicates the byte lane that caused the error and LTEATR[BNK] indicates which memory controller bank was accessed.
3–4	—	Reserved
5	WP	Write protect error 0 No write protect error occurred. 1 A write was attempted to a local bus memory region that was defined as read-only in the memory controller. Usually, in this case, a bus monitor time-out will occur (as the cycle is not automatically terminated).
6–11	—	Reserved
12	CS	Chip select error 0 No chip select error occurred. 1 A transaction was sent to the eLBC that did not hit any memory bank.
13–30	—	Reserved
31	CC	FCM command completion event 0 No FCM operation in progress, or operation pending. 1 FCM operation has completed, allowing software to continue processing of results.

10.3.1.10 Transfer Error Check Disable Register (LTEDR)

The transfer error check disable register (LTEDR), shown in Figure 10-14, is used to disable error/event checking. Note that control of error/event checking is independent of control of reporting of errors/events (LTEIR) through the interrupt mechanism.

Offset 0x0_50B4 Access: Read/Write

	0	1	2	3–4	5	6–7	8	9	10–11	12	13–15
R/W	BMD	FCTD	PARD	—	WPD	—	—	—	—	CSD	—
Reset	All zeros										

	16–30	31
R/W	—	CCD
Reset	All zeros	

Figure 10-14. Transfer Error Check Disable Register (LTEDR)

Table 10-17 describes LTEDR fields.

Table 10-17. LTEDR Field Descriptions

Bits	Name	Description
0	BMD	Bus monitor disable 0 Bus monitor is enabled. 1 Bus monitor is disabled, but internal bus time-outs can still occur.
1	FCTD	FCM command time-out disable 0 FCM command timer is enabled. 1 FCM command time-out is disabled, but internal FCM command timer can terminate command waits.
2	PARD	Parity and ECC error checking disabled. 0 Parity and ECC error checking is enabled. 1 Parity and ECC error checking is disabled.
3–4	—	Reserved
5	WPD	Write protect error checking disable. 0 Write protect error checking is enabled. 1 Write protect error checking is disabled.
6–11	—	Reserved
12	CSD	Chip select error checking disable. 0 Chip select error checking is enabled. 1 Chip select error checking is disabled.
13–30	—	Reserved
31	CCD	FCM command completion checking disable. 0 Command completion checking is enabled. 1 Command completion checking is disabled.

10.3.1.11 Transfer Error Interrupt Enable Register (LTEIR)

The transfer error interrupt enable register (LTEIR), shown in Figure 10-15, is used to send or block error/event reporting through the eLBC internal interrupt mechanism. Software should clear pending errors/events in LTESR before enabling interrupts. After an interrupt has occurred, clearing relevant LTESR error/event bits negates the interrupt.

Offset 0x0_50B8 Access: Read/Write

	0	1	2	3–4	5	6–7	8	9	10–11	12	13–15
R/W	BMI	FCTI	PARI	—	WPI	—	—	—	—	CSI	—
Reset	All zeros										

	16–30	31
R/W	—	CCI
Reset	All zeros	

Figure 10-15. Transfer Error Interrupt Enable Register (LTEIR)

Table 10-18 describes LTEIR fields.

Table 10-18. LTEIR Field Descriptions

Bits	Name	Description
0	BMI	Bus monitor error interrupt enable. 0 Bus monitor error reporting is disabled. 1 Bus monitor error reporting is enabled.
1	FCTI	FCM command time-out interrupt enable. 0 FCM command time-out error reporting is disabled. 1 FCM command time-out error reporting is enabled.
2	PARI	Parity and ECC error interrupt enable. 0 Parity and ECC error reporting is disabled. 1 Parity and ECC error reporting is enabled.
3–4	—	Reserved
5	WPI	Write protect error interrupt enable. 0 Write protect error reporting is disabled. 1 Write protect error reporting is enabled.
6–11	—	Reserved
12	CSI	Chip select error interrupt enable. 0 Chip select error reporting is disabled. 1 Chip select error reporting is enabled.
13–30	—	Reserved
31	CCI	FCM command completion Event interrupt enable. 0 Command completion reporting is disabled. 1 Command completion reporting is enabled.

10.3.1.12 Transfer Error Attributes Register (LTEATR)

The transfer error attributes register (LTEATR) captures source attributes of an error/event. Figure 10-16 shows the LTEATR. After LTEATR[V] has been set, software must clear this bit to allow LTESR, LTEATR, and LTEAR to update following any subsequent events/errors.

Offset 0x0_50BC — Access: Read/Write

	0–2	3	4–10	11–15	16–19	20–27	28–30	31
R/W	—	RWB	—	SRCID	PB	BNK	—	V
Reset	All zeros							

Figure 10-16. Transfer Error Attributes Register (LTEATR

Table 10-19 describes LTEATR fields.

Table 10-19. LTEATR Field Descriptions

Bits	Name	Description
0–2	—	Reserved
3	RWB	Transaction type for the error: 0 The transaction for the error was a write transaction. 1 The transaction for the error was a read transaction.
4–10	—	Reserved
11–15	SRCID	Captures the source of the transaction when this information is provided on the internal interface to the eLBC.
16–19	PB	Parity error on byte or block. For GPCM and UPM, there are four parity error status bits, one per byte lane. A bit is set for the byte that had a parity error (bit 16 represents byte 0, the most significant byte lane). For FCM, there are at most four 512-byte page blocks (for a large page device) checked by ECC. A bit is set for the 512-byte block that had an uncorrectable ECC error on read (bit 16 represents block 0, the first 512 bytes of a page; if ORx[PGS] = 0, bits 17–19 are always 0).
20–27	BNK	Memory controller bank. There is one error status bit per memory controller bank (bit 20 represents bank 0). A bit is set for the local bus memory controller bank that had an error.
28–30	—	Reserved
31	V	Error attribute capture is valid. Indicates that the captured error information is valid. 0 Captured error attributes and address are not valid. 1 Captured error attributes and address are valid.

10.3.1.13 Transfer Error Address Register (LTEAR)

The transfer error address register (LTEAR) captures the address of a transaction that caused an error/event. The transfer error address register (LTEAR) is shown in Figure 10-17.

Offset 0x0_50C0 — Access: Read/Write

	0–31
R / W	A
Reset	All zeros

Figure 10-17. Transfer Error Address Register (LTEAR)

Table 10-20 describes LTEAR fields.

Table 10-20. LTEAR Field Descriptions

Bits	Name	Description
0–31	A	Transaction address for the error. For GPCM and UPM, holds the 32-bit address of the transaction resulting in an error. For FCM, this register is undefined.

10.3.1.14 Local Bus Configuration Register (LBCR)

The local bus configuration register (LBCR) is shown in Figure 10-18.

Offset 0x0_50D0 — Access: Read/Write

	0	1–7	8–9	10	11–13	14	15
R / W	LDIS	—	BCTLC	AHD	—	LPBSE	EPAR
Reset	All zeros						

	16–23	24–27	28–31
R / W	BMT	—	BMTPS
Reset	All zeros		

Figure 10-18. Local Bus Configuration Register

Table 10-21 describes LBCR fields.

Table 10-21. LBCR Field Descriptions

Bits	Name	Description
0	LDIS	Local bus disable 0 Local bus is enabled. 1 Local bus is disabled. No internal transactions will be acknowledged.
1–7	—	Reserved
8–9	BCTLC	Defines the use of LBCTL 00 LBCTL is used as W/R control for GPCM or UPM accesses (buffer control). 01 LBCTL is used as LOE for GPCM accesses only. 10 LBCTL is used as LWE for GPCM accesses only. 11 Reserved.
10	AHD	Address hold disable. Removes part of the hold time for LAD with respect to LALE in order to lengthen the LALE pulse. 0 During address phases on the local bus, the LALE signal negates one platform clock period prior to the address being invalidated. For instance, at 33.3 MHz, this provides 3 ns of additional address hold time at the external address latch. 1 During address phases on the local bus, the LALE signal negates 0.5 platform clock period prior to the address being invalidated. This halves the address hold time, but extends the latch enable duration. This may be necessary for very high frequency designs.
11–13	—	Reserved.
14	LPBSE	Enables parity byte select on LGTA/LFRB/LGPL4/LUPWAIT/LPBSE signal. 0 Parity byte select is disabled. LGTA/LGPL4/LPBSE signal is available for memory control as LGPL4 (output) or LGTA/LFRB/LUPWAIT (input). 1 Parity byte select is enabled. LPBSE signal is dedicated as the parity byte select output, and LGTA/LFRB/LUPWAIT is disabled.
15	EPAR	Determines odd or even parity. Writing GPCM or UPM controlled memory with EPAR = 1 and reading the memory with EPAR = 0 generates parity errors for testing. 0 Odd parity; normal, odd-parity ECC 1 Even parity; inverted, even-parity ECC
16–23	BMT	Bus monitor timing. Defines the bus monitor time-out period. Clearing BMT (reset value) selects the maximum count of bus clock cycles. For non-zero values of BMT, the number of LCLK clock cycles to count down before a time-out error is generated is given by: bus cycles = BMT × PS, where PS is set according to LBCR[BMTPS]. The value of BMT × PS must not be less than 40 bus cycles for reliable operation.

Table 10-21. LBCR Field Descriptions (continued)

Bits	Name	Description
24–27	—	Reserved
28–31	BMTPS	Bus monitor timer prescale. Defines the multiplier, PS, to scale LBCR[BMT] for determining bus time-outs. 0000 PS = 8 0001 PS = 16 0010 PS = 32 0011 PS = 64 0100 PS = 128 0101 PS = 256 0110 PS = 512 0111 PS = 1024 1000 PS = 2048 1001 PS = 4096 1010 PS = 8192 1011 PS = 16,384 1100 PS = 32,768 1101 PS = 65,536 1110 PS = 131,072 1111 PS = 262,144

10.3.1.15 Clock Ratio Register (LCRR)

The clock ratio register, shown in Figure 10-19, sets the system clock to eLBC bus frequency ratio. It also provides configuration bits for extra delay cycles for address and control signals.

NOTE

For proper operation of the system, it is required that this register setting will not be altered while local bus memories or devices are being accessed. Special care needs to be taken when running instructions from an eLBC memory.

Offset 0x0_50D4 Access: Read/Write

Bits	0	1–13	14–15	16–26	27–31
R/W	PBYP	—	EADC	—	CLKDIV
Reset	1	0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 0 0 0 0 0 0 0 0	0 *n* *n* 0 0

Figure 10-19. Clock Ratio Register (LCRR)

Table 10-22 describes LCRR fields.

Table 10-22. LCRR Field Descriptions

Bits	Name	Description
0	PBYP	PLL bypass. This bit should be set when using low bus clock frequencies (See device hardware specifications for applicable frequencies.). When in PLL bypass mode, incoming data is captured in the middle of the bus clock cycle. 0 The PLL is enabled. 1 The PLL is bypassed.
1–13	—	Reserved
14–15	EADC	External address delay cycles of LCLK. Defines the number of cycles for the assertion of LALE. 00 4 01 1 10 2 11 3
16–26	—	Reserved
27–31	CLKDIV	System clock divider. Sets the frequency ratio between the system clock and the local bus clock. The system clock is equivalent to csb_clk or twice csb_clk (if RCWL[LBIUCM] is set). Only the values shown below are allowed. **Note:** It is critical that no transactions are being executed via the local bus while CLKDIV is being modified. As such, prior to modification, the user must ensure that code is not executing out of the local bus. Once LCRR[CLKDIV] is written, the register should be read, and then an isync should be executed. 00000–00001 Reserved 00010 2 00011 Reserved 00100 4 00101–00111 Reserved 01000 8 01001–11111 Reserved

10.3.1.16 Flash Mode Register (FMR)

The local bus flash mode register (FMR), shown in Figure 10-20, controls global operation of the FCM.

Offset 0x0_50E0 Access: Read/Write

	0–15	16–19	20	21–22	23	24–25	26–27	28–29	30–31
R/W	—	CWTO	BOOT	—	ECCM	—	AL	—	OP
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0	R[1]	0 0	0	0 0	0 0	0 0	0 0

Figure 10-20. Flash Mode Register

[1] Bit R (field BOOT) is set if power-on-reset configuration selects FCM as the boot ROM target.

Table 10-23 describes FMR fields.

Table 10-23. FMR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–19	CWTO	Command wait time-out. For FCM commands that wait on LFRB being sampled high (CW0, CW1, RBW and RSW), FCM pauses execution of the instruction sequence until either LFRB is sampled high, or a timer controlled by CTO expires, whichever occurs first. The time-out in the latter case is: 0000 256 cycles of LCLK 0001 512 cycles of LCLK 0010 1024 cycles of LCLK 0011 2048 cycles of LCLK 0100 4096 cycles of LCLK 0101 8192 cycles of LCLK 0110 16,384 cycles of LCLK 0111 32,768 cycles of LCLK 1000 65,536 cycles of LCLK 1001 131,072 cycles of LCLK 1010 262,144 cycles of LCLK 1011 524,288 cycles of LCLK 1100 1,048,576 cycles of LCLK 1101 2,097,152 cycles of LCLK 1110 4,194,304 cycles of LCLK 1111 8,388,608 cycles of LCLK
20	BOOT	Flash auto-boot load mode. During system boot from NAND Flash EEPROM, this bit remains set to alter the use of the FCM buffer RAM. Software should clear BOOT once FCM is to be restored to normal operation. Setting BOOT without auto-boot in progress only alters the mapping of the buffer RAM. 0 FCM is operating in normal functional mode, with an 8 Kbyte FCM buffer RAM. 1 eLBC has been configured—either from reset or by a special operation OP = 01—to auto-load a 4-Kbyte boot block into the FCM buffer RAM, which maps only the 4 Kbytes of NAND flash main data region comprising the boot block. Any access to the buffer RAM is delayed until the entire boot block has been loaded.
21–22	—	Reserved
23	ECCM	ECC mode. When hardware checking and/or generation of error correcting codes (ECC) is enabled (that is, when BR*n*[DECC] is 01 or 10, and full page transfers are specified with FBCR[BC] = 0), ECCM sets the ECC block size and position of the ECC code word(s) in the NAND Flash spare region for both checking and generation functions. The format of the ECC code word conforms with the Samsung/Toshiba spare region assignment specifications. 0 ECC is checked/calculated over 512-Byte blocks. A 24-bit ECC is assigned to spare region bytes at offsets (N×16)+6 through (N×16)+8 for spare region N, N = 0–3. 1 ECC is checked/calculated over 512-Byte blocks. A 24-bit ECC is assigned to spare region bytes at offsets (N×16)+8 through (N×16)+10 for spare region N, N = 0–3.
24–25	—	Reserved

Table 10-23. FMR Field Descriptions (continued)

Bits	Name	Description
26–27	AL	Address length. AL sets the number of address bytes issued during page address (PA) operations. However, the number of address bytes issued for column address (CA) operations is determined by the device page size (for OR*n*[PGS] = 0, 1 CA byte is issued; for OR*n*[PGS] = 1, 2 CA bytes are issued). 00 2 bytes are issued for page addresses, thus a total of 3 (OR*n*[PGS] = 0) or 4 (OR*n*[PGS] = 1) address bytes are issued for a {CA,PA} sequence 01 3 bytes are issued for page addresses, thus a total of 4 (OR*n*[PGS] = 0) or 5 (OR*n*[PGS] = 1) address bytes are issued for a {CA,PA} sequence 10 4 bytes are issued for page addresses, thus a total of 5 (OR*n*[PGS] = 0) or 6 (OR*n*[PGS] = 1) address bytes are issued for a {CA,PA} sequence 11 —
28–29	—	Reserved
30–31	OP	Flash operation. For OP not equal to 00, a special operation is triggered on the next write to LSOR or dummy access to a bank controlled by FCM. Once a special operation has commenced, OP is automatically reset to 00 by FCM. Individual blocks may be temporarily unlocked for erase and reprogramming operations. 00 Normal operation. All read and write accesses to banks controlled by FCM access the shared FCM buffer RAM. No bus activity is caused by this operation. 01 Simulate auto-boot block loading, and set FMR[BOOT]. Boot block loading occurs from the bank triggered on the special operation, therefore the appropriate bank configuration must be initialized prior to issuing this operation. 10 Execute the command sequence contained in FIR, but with write protection enabled (pin $\overline{LFWP}$ asserted low) so that all Flash blocks are protected from accidental erasure and reprogramming. 11 Execute the command sequence contained in FIR, but permit the single block identified by FBAR[BLK] to be erased or reprogrammed, with pin $\overline{LFWP}$ remaining high during the access.

10.3.1.17 Flash Instruction Register (FIR)

The local bus flash instruction register (FIR), shown in Figure 10-21, holds a sequence of up to eight instructions for issue by the FCM. Setting FMR[OP] non-zero and writing LSOR or accessing a bank controlled by FCM causes FCM to read FIR 4 bits at a time, starting at bit 0 and continuing with adjacent 4-bit opcodes, until only NOP opcodes remain. The programmed instruction sequence of OP0, OP1,..., OP7 is performed on the activated bank, using the data buffer addressed by FPAR. If LTEIR[CCI] = 1 and LTEDR[CCD] = 0, eLBC will generate an interrupt once the entire sequence has completed, and software should examine LTEATR and clear its V bit.

Software must not alter the contents of the addressed FCM buffer, FIR, MDR, FCR, FBAR, FPAR, or FBCR while an operation is in progress—or eLBC will behave unpredictably—but software can freely modify the contents of any currently unused FCM RAM buffer in preparation for the next operation.

Offset 0x0_50E4 Access: Read/Write

	0–3	4–7	8–11	12–15	16–19	20–23	24–27	28–31
R/W	OP0	OP1	OP2	OP3	OP4	OP5	OP6	OP7
Reset	All zeros							

Figure 10-21. Flash Instruction Register

Table 10-24 describes FIR fields.

Table 10-24. FIR Field Descriptions

Bits	Name	Description
0–3	OP0	FCM operation codes. OP0 is executed first, followed by OP1, through to OP7. 0000 NOP—No-operation and end of operation sequence 0001 CA—Issue current column address as set in FPAR, with length set by ORx[PGS] 0010 PA—Issue current block+page address as set in FBAR and FPAR, with length set by FMR[AL] 0011 UA—Issue user-defined address byte from next AS field in MDR 0100 CM0—Issue command from FCR[CMD0] 0101 CM1—Issue command from FCR[CMD1] 0110 CM2—Issue command from FCR[CMD2] 0111 CM3—Issue command from FCR[CMD3] 1000 WB—Write FBCR bytes of data from current FCM buffer to Flash device 1001 WS—Write one byte (8b port) of data from next AS field of MDR to Flash device 1010 RB—Read FBCR bytes of data from Flash device into current FCM RAM buffer 1011 RS—Read one byte (8b port) of data from Flash device into next AS field of MDR 1100 CW0—Wait for LFR$\overline{\text{B}}$ to return high or time-out, then issue command from FCR[CMD0] 1101 CW1—Wait for LFR$\overline{\text{B}}$ to return high or time-out, then issue command from FCR[CMD1] 1110 RBW—Wait for LFR$\overline{\text{B}}$ to return high or time-out, then read FBCR bytes of data from Flash device into current FCM RAM buffer 1111 RSW—Wait for LFR$\overline{\text{B}}$ to return high or time-out, then read one byte (8b port) of data from Flash device into next AS field of MDR
4–7	OP1	
8–11	OP2	
12–15	OP3	
16–19	OP4	
20–23	OP5	
24–27	OP6	
28–31	OP7	

10.3.1.18 Flash Command Register (FCR)

The local bus flash command register (FCR), shown in Figure 10-22, holds up to four NAND Flash EEPROM command bytes that may be referenced by opcodes in FIR during FCM operation. The values of the commands should follow the manufacturer's datasheet for the relevant NAND Flash device.

Offset 0x0_50E8 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	CMD0	CMD1	CMD2	CMD3
Reset	All zeros			

Figure 10-22. Flash Command Register

Table 10-25 describes FCR fields.

Table 10-25. FCR Field Descriptions

Bits	Name	Description
0–7	CMD0	General purpose FCM Flash command byte 0. Opcodes in FIR that issue command index 0 write CMD0 to the NAND Flash command/data bus.
8–15	CMD1	General purpose FCM Flash command byte 1. Opcodes in FIR that issue command index 1 write CMD1 to the NAND Flash command/data bus.
16–23	CMD2	General purpose FCM Flash command byte 2. Opcodes in FIR that issue command index 2 write CMD2 to the NAND Flash command/data bus.
24–31	CMD3	General purpose FCM Flash command byte 3. Opcodes in FIR that issue command index 3 write CMD3 to the NAND Flash command/data bus.

10.3.1.19 Flash Block Address Register (FBAR)

The local bus flash block address register (FBAR), shown in Figure 10-23, locates the NAND Flash block index for the page currently accessed.

Offset 0x0_50EC — Access: Read/Write

Bits	0–7	8–31
R/W	—	BLK
Reset	All zeros	

Figure 10-23. Flash Block Address Register

Table 10-26 describes FBAR fields.

Table 10-26. FBAR Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–31	BLK	Flash block address. The size of the NAND Flash, as configured in OR*n*[PGS] and FMR[AL], determines the number of bits of BLK that are issued to the EEPROM during block address phases.

10.3.1.20 Flash Page Address Register (FPAR)

The local bus flash page address register (FPAR), shown in Figure 10-24 and Figure 10-25, locates the current NAND Flash page in both the external NAND Flash device and FCM buffer RAM.

Offset 0x0_50F0 — Access: Read/Write

Bits	0–16	17–21	22	23–31
R/W	—	PI	MS	CI
Reset	All zeros			

Figure 10-24. Flash Page Address Register, Small Page Device (ORx[PGS] = 0)

Offset 0x0_50F0 — Access: Read/Write

Bits	0–13	14–19	20	21–31
R/W	—	PI	MS	CI
Reset	All zeros			

Figure 10-25. Flash Page Address Register, Large Page Device (ORx[PGS] = 1)

Table 10-27 describes FPAR fields for small page devices.

Table 10-27. FPAR Field Descriptions, Small Page Device (ORx[PGS] = 0)

Bits	Name	Description
0–16	—	Reserved
17–21	PI	Page index. PI indexes the page in NAND Flash EEPROM at the current block defined by FBAR, and locates the corresponding transfer buffer in the FCM buffer RAM. The 3 LSBs of PI index one of the eight 1 Kbyte buffers in the FCM buffer RAM as follows: 000 The page is transferred to/from FCM buffer 0, address offsets 0x0000–0x03FF 001 The page is transferred to/from FCM buffer 1, address offsets 0x0400–0x07FF 010 The page is transferred to/from FCM buffer 2, address offsets 0x0800–0x0BFF 011 The page is transferred to/from FCM buffer 3, address offsets 0x0C00–0x0FFF 100 The page is transferred to/from FCM buffer 4, address offsets 0x1000–0x13FF 101 The page is transferred to/from FCM buffer 5, address offsets 0x1400–0x17FF 110 The page is transferred to/from FCM buffer 6, address offsets 0x1800–0x1BFF 111 The page is transferred to/from FCM buffer 7, address offsets 0x1C00–0x1FFF
22	MS	Main/spare region locator. In the case that FBCR[BC] = 0, MS is treated as 0. 0 Data is transferred to/from the main region of the FCM buffer; that is, the first 512 bytes of the buffer are used as the starting address. 1 Data is transferred to/from the spare region of the FCM buffer; that is, the second 512 bytes of the buffer are used as the starting address, but only an initial 16 bytes of spare region are defined.
23–31	CI	Column index. CI indexes the first byte to transfer to/from the main or spare region of the NAND Flash EEPROM and corresponding transfer buffer. In the case that FBCR[BC] = 0, CI is treated as 0. For MS = 0, CI can range 0x000–0x1FF; for MS = 1, CI can range 0x000–0x00F.

Table 10-28 describes FPAR fields for large page devices.

Table 10-28. FPAR Field Descriptions, Large Page Device (ORx[PGS] = 1)

Bits	Name	Description
0–13	—	Reserved
14–19	PI	Page index. PA indexes the page in NAND Flash EEPROM at the current block defined by FBAR, and locates the corresponding transfer buffer in the FCM buffer RAM. The LSB of PI indexes one of the two 4 Kbyte buffers in the FCM buffer RAM as follows: 0 The page is transferred to/from FCM buffer 0, address offsets 0x0000–0x0FFF 1 The page is transferred to/from FCM buffer 1, address offsets 0x1000–0x1FFF
20	MS	Main/spare region locator. In the case that FBCR[BC] = 0, MS is treated as 0. 0 Data is transferred to/from the main region of the FCM buffer; that is, the first 2048 bytes of the buffer are used as the starting address. 1 Data is transferred to/from the spare region of the FCM buffer; that is, the second 2048 bytes of the buffer are used as the starting address, but only an initial 64 bytes of spare region are defined.
21–31	CI	Column index. CI indexes the first byte to transfer to/from the main or spare region of the NAND Flash EEPROM and corresponding transfer buffer. In the case that FBCR[BC] = 0, CI is treated as 0. For MS = 0, CI can range 0x000–0x7FF; for MS = 1, CI can range 0x000–0x03F.

10.3.1.21 Flash Byte Count Register (FBCR)

The local bus flash byte count register (FBCR), shown in Figure 10-26, defines the size of FCM block transfers for reads and writes to the NAND Flash EEPROM.

Offset 0x0_50F4 Access: Read/Write

	0–19	20–31
R W	—	BC

Figure 10-26. Flash Byte Count Register

Table 10-29 describes FBCR fields.

Table 10-29. FBCR Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	BC	Byte count determines how many bytes are transferred by the FCM during data read (RB) or data write (WB) opcodes. The first byte accessed in the NAND Flash EEPROM is located by the FPAR register, and successive bytes are transferred until either BC bytes have been counted, or the end of the spare region of the currently addressed Flash page has been reached. If BC = 0, an entire Flash page and its spare region will be transferred by FCM, in which case FPAR[MS] and FPAR[CI] are treated as zero regardless of their values. BC = 0 is the only setting that permits FCM to generate and check ECC.

10.4 Functional Description

The eLBC allows the implementation of memory systems with very specific timing requirements.

- The GPCM provides interfacing for simpler, lower-performance memories and memory-mapped devices. It has inherently lower performance because it does not support bursting. For this reason, GPCM-controlled banks are used primarily for boot-loading from NVRAM or NOR Flash, and access to low-performance memory-mapped peripherals.
- The FCM interfaces the eLBC to NAND Flash EEPROMs with 8-bit data bus. The FCM has an automatic boot-loading feature that allows the CPU to boot from high density EEPROM, loading the boot block into 4 Kbytes of RAM for execution of the first level boot code. Following boot, FCM provides a flexible instruction sequencer that allows a user-defined command, address, and data transfer sequence of up to 8 steps to be executed against a memory-mapped buffer RAM. Programmable set-up time, hold time, and wait states permit the FCM to maximize the performance of NAND Flash block transfers, which can proceed in parallel with software processing of the multiple RAM buffers. A single-pass ECC engine in the FCM permits zero-overhead error checking, reporting, and correction in both boot blocks and page data transfers if enabled.
- The UPM supports refresh timers, address multiplexing of the external bus, and generation of programmable control signals for row address and column address strobes, to allow for a minimal glue logic interface to DRAMs, burstable SRAMs, and almost any other kind of peripheral with asynchronous timing or single data rate clocking. The UPM can be used to generate flexible,

user-defined timing patterns for control signals that govern a memory device. These patterns define how the external control signals behave during a read, write, burst-read, or burst-write access. Refresh timers are also available to periodically initiate user-defined refresh patterns.

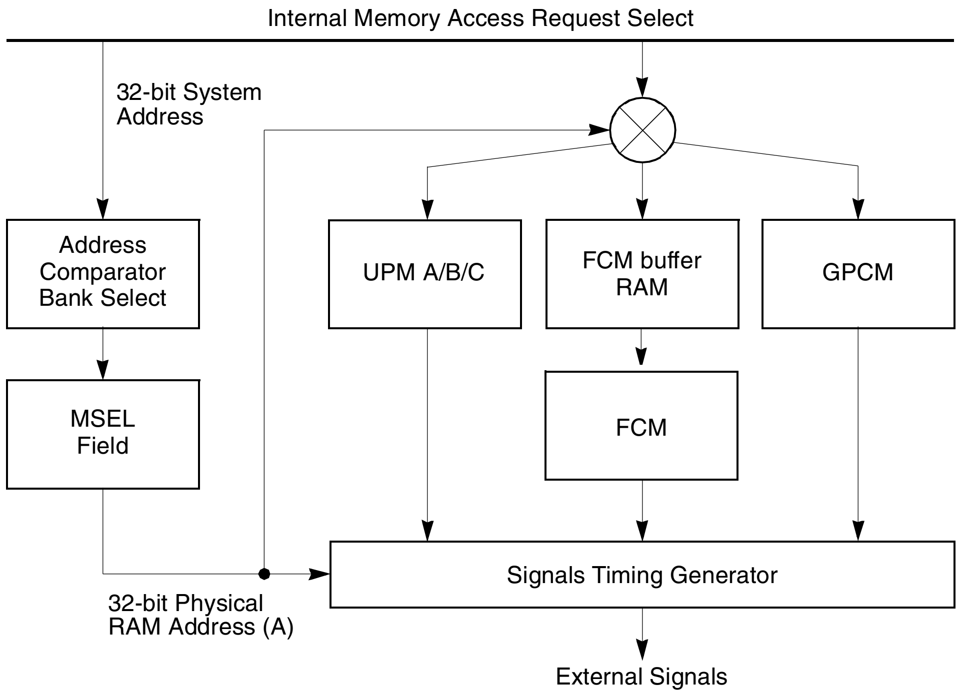


Figure 10-27. Basic Operation of Memory Controllers in the eLBC

Each memory bank (chip select) can be assigned to any of these three types of machines through the machine select bits of the base register for that bank (BR*n*[MSEL]), as illustrated in Figure 10-27. If a bank match occurs, the corresponding machine (GPCM, FCM, or UPM) then takes ownership of the external signals that control the access and maintains control until the transaction ends.

10.4.1 Basic Architecture

The following subsections describe the basic architecture of the eLBC.

10.4.1.1 Address and Address Space Checking

The defined base addresses are written to the BR*n* registers, while the corresponding address masks are written to the OR*n* registers. Each time a local bus access is requested, the internal transaction address is compared with each bank. Addresses are decoded by comparing the 17 MSBs of the address, masked by OR*n*[AM], with the base address for each bank (BR*n*[BA]). If a match is found on a memory controller bank, the attributes defined in the BR*n* and OR*n* for that bank are used to control the memory access. If a match is found in more than one bank, the lowest-numbered bank handles the memory access (that is, bank 0 has priority over bank 1).

10.4.1.2 External Address Latch Enable Signal (LALE)

The local bus uses a multiplexed address/data bus. Therefore the eLBC must distinguish between address and data phases, which take place on the same bus (LAD pins). The LALE signal, when asserted, signifies an address phase during which the eLBC drives the memory address on the LAD pins. An external address latch uses this signal to capture the address and provide it to the address pins of the memory or peripheral device. When LALE is negated, LAD then serves as the (bi-directional) data bus for the access. Any address phase initiates the assertion of LALE, which has a programmable duration of between 1 and 4 bus clock cycles.

To ensure adequate hold time on the external address latch, LALE negates earlier than the address changes on LAD during address phases. By default, LALE negates earlier by 1 platform clock period. For example, if the platform clock is operating at 33.3 MHz, then 1.83 ns of address hold time is introduced. However, at higher frequencies, the duration of the shortened LALE pulse may not meet the minimum latch enable pulse width specifications of some latches. In such cases, setting LBCR[AHD] = 1 increases the LALE pulse width by ½ platform clock cycle, but decreases the address hold time by the same amount. If both longer hold time and longer LALE pulse duration are needed, then the address phase can be extended using the ORn[EAD] and LCRR[EADC] fields, and the LBCR[AHD] bit can be left at 0. However, this will add latency to all address tenures.

The frequency of LALE assertion varies across the three memory controllers:

- For GPCM, every assertion of $\overline{\text{LCS}n}$ is considered an independent access, and accordingly, LALE asserts prior to each such access. For example, GPCM driving an 8-bit port would assert LALE and $\overline{\text{LCS}n}$ 32 times in order to satisfy a 32-byte cache line transfer.
- For FCM, LALE asserts prior to each multi-command operation sequence, but LALE can be ignored on NAND Flash EEPROM accesses as the signal does *not* enable address latching in such devices. The value on the LAD and LA pins during LALE assertion is driven low-impedance, but otherwise not defined for FCM banks.
- In the case of UPM, the frequency of LALE assertion depends on how the UPM RAM is programmed. UPM single accesses typically assert LALE once, upon commencement, but it is possible to program UPM to assert LALE several times, and to change the values of LA*n* with and without LALE being involved.

In general, when using the GPCM controller it is not necessary to use LA if a sufficiently wide latch is used to capture the entire address during LALE phases. The UPMs may require LA if the eLBC is generating its own burst address sequence.

To illustrate how a large transaction is handled by the eLBC, Figure 10-28 shows eLBC signals for the GPCM performing a 32-byte write starting at address 0x5420. Note that during each of the 32 assertions of LALE, LA[27–31] exactly mirror LAD[27–31], but during data phases, only LAD[0–7] and LDP[0] are driven with valid data and parity, respectively.

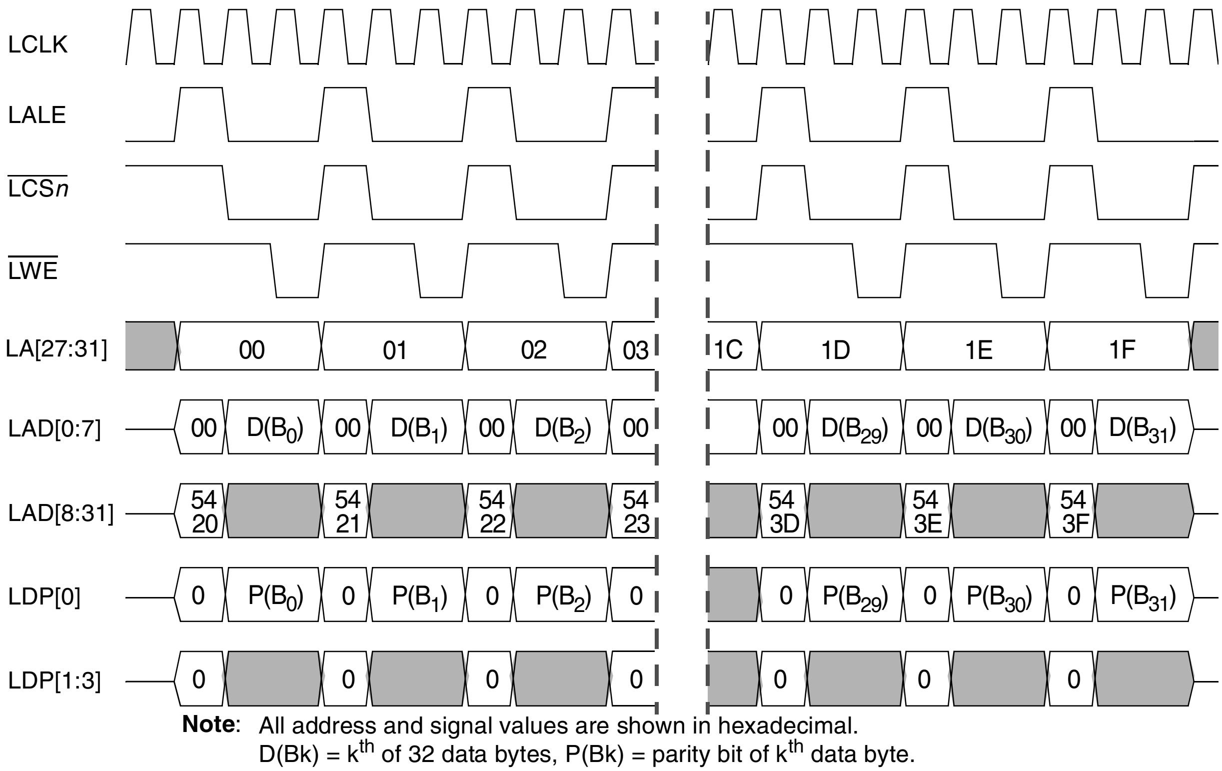


Figure 10-28. Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYP] = 0)

10.4.1.3 Data Transfer Acknowledge (TA)

The three memory controllers in the eLBC generate an internal transfer acknowledge signal, TA, to allow data on LAD to be either sampled (for reads) or changed (on writes). The data sampling/data change always occurs at the end of the bus cycle in which the eLBC asserts TA internally. In eLBC debug mode, TA is also visible externally on the LDVAL pin. The GPCM controller automatically generates TA according to the timing parameters programmed for them in the option and mode registers; FCM generates TA whenever data read and write instructions are executed out of register FIR; a UPM generates TA only when a UPM pattern has the UTA RAM word bit set. Figure 10-29 shows LALE, TA (internal), and $\overline{\text{LCS}n}$.

Note that TA and LALE are never asserted together, and that for the duration of LALE, $\overline{LCSn}$ (or any other control signal) remains negated or frozen.

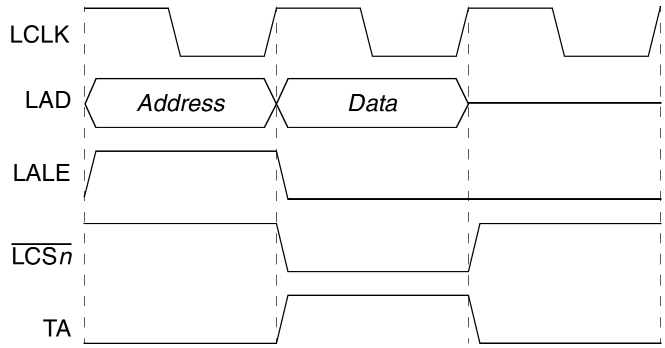


Figure 10-29. Basic eLBC Bus Cycle with LALE, TA, and $\overline{LCSn}$

10.4.1.4 Data Buffer Control (LBCTL)

The memory controller provides a data buffer control signal for the local bus (LBCTL). This signal is activated when a GPCM-, FCM-, or UPM-controlled bank is accessed. LBCTL can be disabled by setting OR*n*[BCTLD]. LBCTL can be further configured by LBCR[BCTLC] to act as an extra $\overline{LWE}$ or an extra $\overline{LOE}$ signal when in GPCM mode.

If LBCTL is configured as a data buffer control (LBCR[BCTLC] = 00), the signal is asserted (high) on the rising edge of the bus clock on the first cycle of the memory controller operation, coincident with LALE. If the access is a write, LBCTL remains high for the whole duration. However, if the access is a read, LBCTL is negated (low) with the negation of LALE so that the memory device is able to drive the bus. If back-to-back read accesses are pending, LBCTL is asserted (high) one bus clock cycle before the next transaction starts (that is, one bus clock cycle before LALE) to allow a whole bus cycle for the bus to turn around before the next address is driven.

10.4.1.5 Parity Generation and Checking (LDP)

Parity can be configured for any GPCM or UPM bank by programming BR*n*[DECC]. Parity is generated and checked on a per-byte basis using LDP[0–3] for the bank if BR*n*[DECC] = 01 (normal parity) or BR*n*[DECC] = 10 for read-modify-write (RMW) parity. Byte lane parity on LDP[0–3] is generated regardless of the BR*n*[DECC] setting. Note that RMW parity can be used only for 32-bit port size banks. LBCR[EPAR] determines the global type of parity (odd or even).

FCM calculates an ECC over 512-byte blocks, and hence does not use the LDP[0–3] pins. The setting of BR*n*[DECC] = 01 enables ECC checking only, while BR*n*[DECC] = 10 enables ECC generation and checking; in either case, LBCR[EPAR] determines the global type of block parity for ECC (odd or even).

10.4.1.6 Bus Monitor

A bus monitor is provided to ensure that each bus cycle is terminated within a reasonable (user defined) period. When a transaction starts, the bus monitor starts counting down from the time-out value (LBCR[BMT] × LBCR[BMTPS]) until a data beat is acknowledged on the bus. It then reloads the time-out value and resumes the countdown until the data tenure completes and then idles if there is no pending transaction. Setting LTEDR[BMD] disables bus monitor error checking (i.e. the LTESR[BM] bit is not set by a bus monitor time-out); however, the bus monitor is still active and can generate a UPM exception (as noted in Section 10.4.4.1.4, “Exception Requests,”) or terminate a GPCM access.

It is very important to ensure that the value of LBCR[BMT] is not set too low; otherwise spurious bus time-outs may occur during normal operation—resulting in incomplete data transfers. Accordingly, the time-out value represented by the LBCR[BMT], LBCR[BMTPS] pair must not be set below 40 bus cycles for time-out under any circumstances.

10.4.1.7 PLL Bypass Mode

At LCLK frequencies in excess of 66 MHz the local bus PLL is used to provide improved hold times at external receivers, and ease set-up margins for read data captured by eLBC. A wire loop between pins LSYNC_OUT and LSYNC_IN establishes the amount of LCLK skewing achieved by the PLL, which locks so as to produce edges on LCLK before the transition of other eLBC control and data signals.

At lower frequencies, the PLL may be unable to lock or provide sufficient hold time improvement for particularly slow devices. Accordingly, LCRR[PBYP] should be set to 1 to bypass the PLL at low frequencies, with the eLBC generating LCLK directly, while skewing it by half a bus clock cycle. An illustration of GPCM or UPM timing both with and without the PLL activated are shown in Figure 10-30 and Figure 10-31. When LCRR[PBYP] = 0, the skew, t_{LSKEW}, matches the round-trip propagation delay of the timing loop between LSYNC_OUT and LSYNC_IN, and data is generated or sampled on the next rising edge of LCLK. The timing diagrams shown normally in this chapter assume that LCRR[PBYP] = 0. When LCRR[PBYP] = 1, the skew equals half the period of LCLK to maximize hold time at the external receiver; in this bypass mode, eLBC drives new address, data, and control signals effectively on falling edges of LCLK, but continues to sample synchronous read data on rising edges of LCLK to maximize the set-up margin for reads.

NOTE

Since LCLK is not used for NAND Flash EEPROMs controlled by FCM, the eLBC drives and samples data on the same edge (rising edge when LCRR[PBYP] = 0 and falling edge when LCRR[PBYP] = 1) on FCM controlled banks.

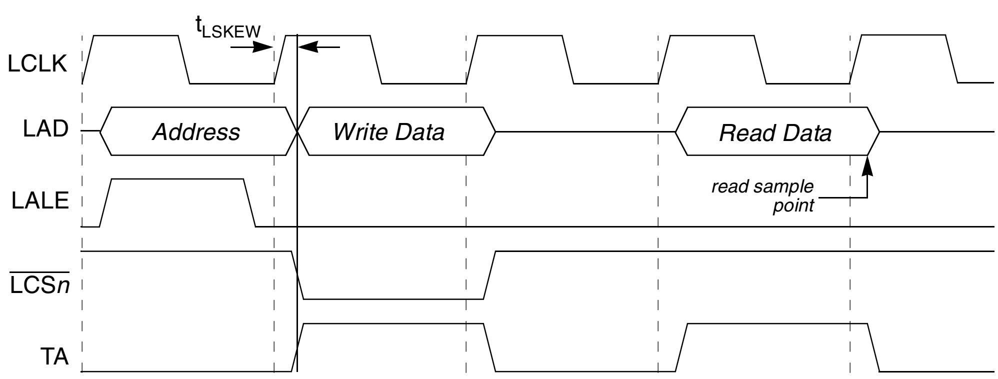


Figure 10-30. eLBC Bus Cycles in PLL Mode (GPCM and UPM only)

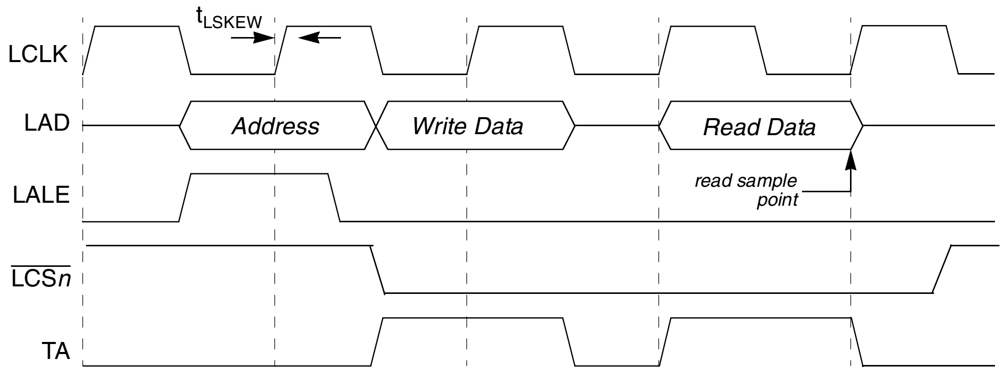


Figure 10-31. eLBC Bus Cycles in PLL-bypassed Mode (GPCM and UPM only)

10.4.2 General-Purpose Chip-Select Machine (GPCM)

The GPCM allows a minimal glue logic and flexible interface to SRAM, EPROM, FEPROM, ROM devices, and external peripherals. The GPCM contains two basic configuration register groups—BR*n* and OR*n*.

Figure 10-32 shows a simple connection between an 8-bit port size SRAM device and the eLBC in GPCM mode. Byte-write enable signals ($\overline{\text{LWE}}$) are available for each byte written to memory. Also, the output enable signal ($\overline{\text{LOE}}$) is provided to minimize external glue logic. On system reset, a global (boot) chip-select is available that provides a boot ROM chip-select ($\overline{\text{LCS0}}$) prior to the system being fully configured.

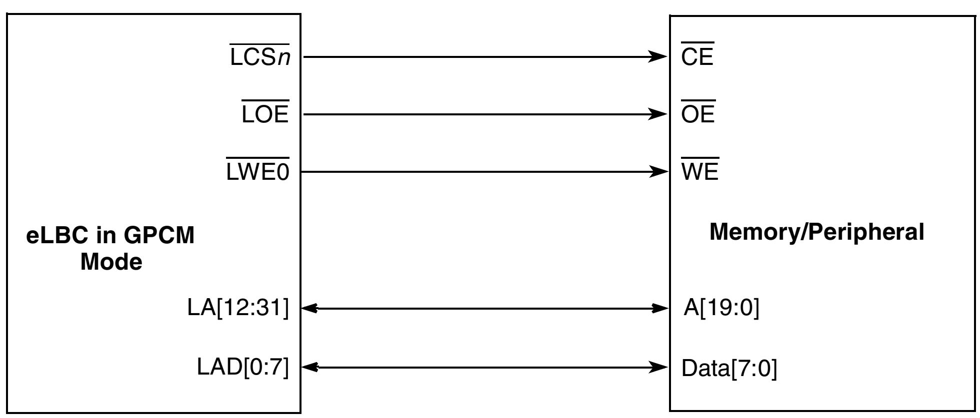


Figure 10-32. Enhanced Local Bus to GPCM Device Interface

Figure 10-33 shows $\overline{\text{LCS}}$ as defined by the setup time required between the address lines and $\overline{\text{CE}}$. The user can configure OR*n*[ACS] to specify $\overline{\text{LCS}}$ to meet this requirement. Generally, the attributes for the memory cycle are taken from OR*n*. These attributes include the CSNT, ACS, XACS, SCY, TRLX, EHTR and SETA fields.

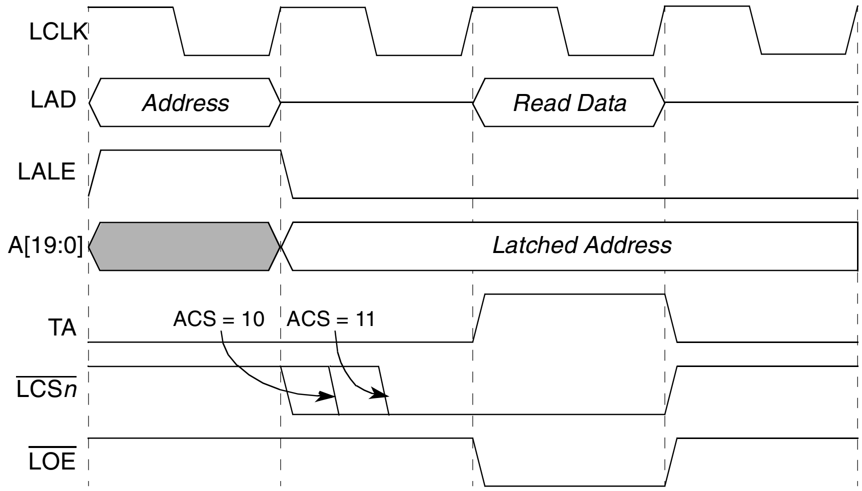
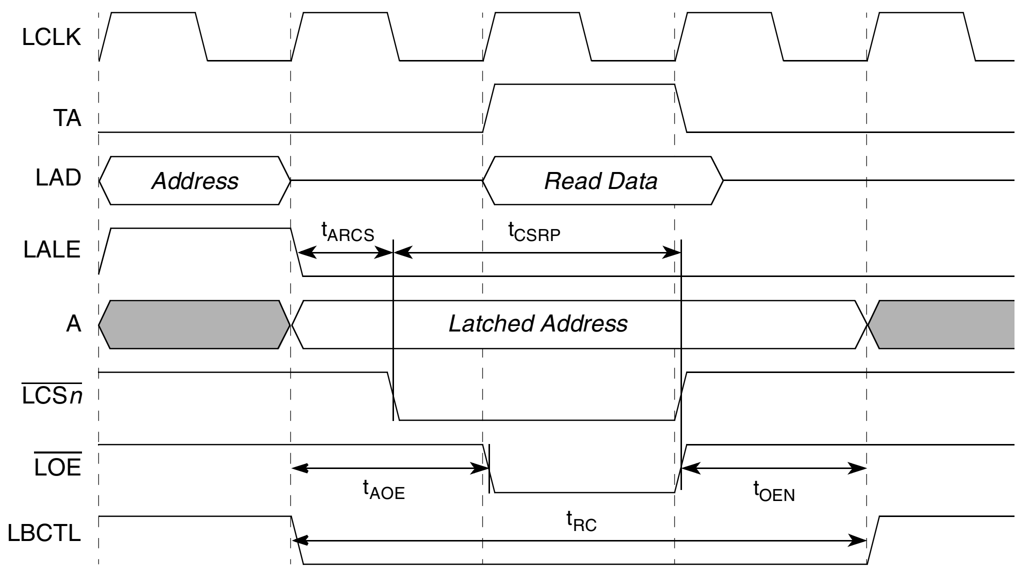


Figure 10-33. GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0, CLKDIV = 4,8)

10.4.2.1 GPCM Read Signal Timing

The basic GPCM read timing parameters that may be set by the OR*n* attributes are shown in Figure 10-34. The read access cycle commences upon latching of the memory address (LALE negated), and concludes when LBCTL returns high to turn the local bus around for a subsequent address phase. Read data is captured by eLBC on the falling edge of TA. $\overline{\text{LOE}}$ and $\overline{\text{LCS}n}$ negate high simultaneously, in some cases before the end of the read access to provide additional hold time for the external memory.



Notes:
t_{RC} = Read cycle time.
t_{ARCS} = Address valid to read chip-select time.
t_{AOE} = Address valid to output enable time.
t_{CSRP} = Read chip-select assertion period.
t_{OEN} = Output enable negated time.

Figure 10-34. GPCM General Read Timing Parameters

Table 10-30 lists the signal timing parameters for a GPCM read access as the option register attributes are varied.

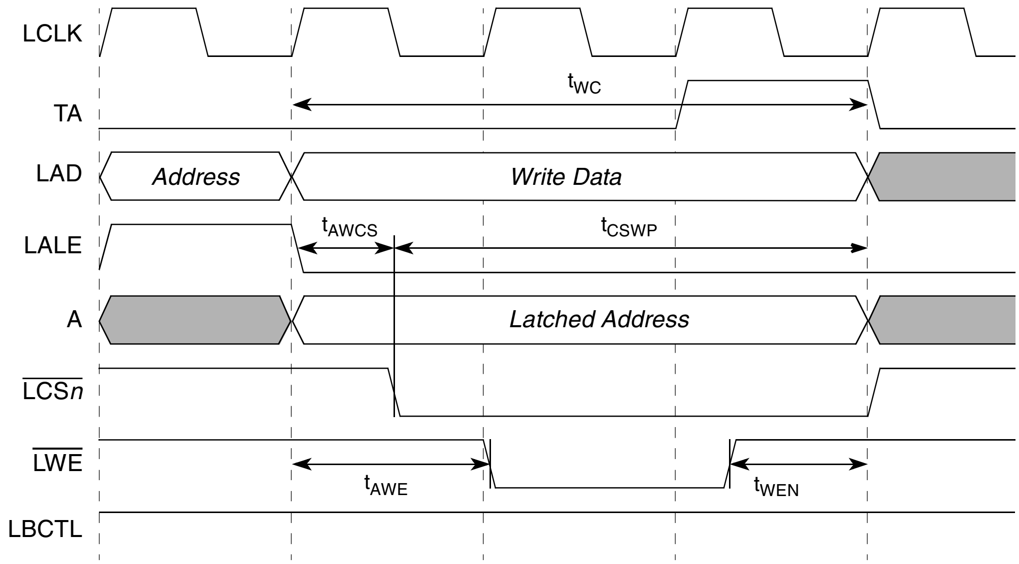
Table 10-30. GPCM Read Control Signal Timing

Option Register Attributes				Signal Timing (LCLK clock cycles)[1]				
TRLX	EHTR	XACS	ACS	t_{ARCS}	t_{CSRP}	t_{AOE}	t_{OEN}	t_{RC}
0	0	0	0X	0	2+SCY	1	0	2+SCY
0	0	0	10	¼ (½)	1¾+SCY (2+SCY)	1	0	2+SCY
0	0	0	11	½	1½+SCY	1	0	2+SCY
0	0	1	0X	0	2+SCY	1	0	2+SCY
0	0	1	10	1	1+SCY	1	0	2+SCY
0	0	1	11	2	1+SCY	2	0	3+SCY
0	1	0	0X	0	2+SCY	1	1	3+SCY
0	1	0	10	¼ (½)	1¾+SCY (1½+SCY)	1	1	3+SCY
0	1	0	11	½	1½+SCY	1	1	3+SCY
0	1	1	0X	0	2+SCY	1	1	3+SCY
0	1	1	10	1	1+SCY	1	1	3+SCY
0	1	1	11	2	1+SCY	2	1	4+SCY
1	0	0	0X	0	2+2×SCY	1	4	6+2×SCY
1	0	0	10	1¼ (1½)	1¾+2×SCY (1½+2×SCY)	2	4	7+2×SCY
1	0	0	11	1½	1½+2×SCY	2	4	7+2×SCY
1	0	1	0X	0	2+2×SCY	1	4	6+2×SCY
1	0	1	10	2	1+2×SCY	2	4	7+2×SCY
1	0	1	11	3	1+2×SCY	3	4	8+2×SCY
1	1	0	0X	0	2+2×SCY	1	8	10+2×SCY
1	1	0	10	1¼ (1½)	1¾+2×SCY (1½+2×SCY)	2	8	11+2×SCY
1	1	0	11	1½	1½+2×SCY	2	8	11+2×SCY
1	1	1	0X	0	2+2×SCY	1	8	10+2×SCY
1	1	1	10	2	1+2×SCY	2	8	11+2×SCY
1	1	1	11	3	1+2×SCY	3	8	12+2×SCY

1 Times in parentheses are specific for the case LCRR[CLKDIV] = 2; other times apply to all CLKDIV values.

10.4.2.2 GPCM Write Signal Timing

The basic GPCM write timing parameters that may be set by the OR*n* attributes are shown in Figure 10-35. The write access cycle commences upon latching of the memory address (LALE negated), and concludes when $\overline{LCSn}$ returns high. LBCTL remains stable for the entire cycle to drive data onto any secondary data bus. Write data becomes invalid following the falling edge of TA. $\overline{LWE}$ may, in some cases, negate high before the end of the write access to provide additional hold time for the external memory.



Notes:
t_{WC} = Write cycle time.
t_{AWCS} = Address valid to write chip-select time.
t_{AWE} = Address valid to write enable time.
t_{CSWP} = Write chip-select assertion period.
t_{WEN} = Write enable negated time wrt chip-sele

Figure 10-35. GPCM General Write Timing Parameters

Table 10-31 lists the signal timing parameters for a GPCM write access as the option register attributes are varied.

Table 10-31. GPCM Write Control Signal Timing

Option Register Attributes				Signal Timing (LCLK clock cycles)[1]				
TRLX	XACS	ACS	CSNT	t_{AWCS}	t_{CSWP}	t_{AWE}	t_{WEN}	t_{WC}
0	0	00	0	0	2+SCY	1	0	2+SCY
0	0	10	0	¼ (½)	1¾+SCY (2+SCY)	1	0	2+SCY
0	0	11	0	½	1½+SCY	1	0	2+SCY
0	1	00	0	0	2+SCY	1	0	2+SCY
0	1	10	0	1	1+SCY	1	0	2+SCY
0	1	11	0	2	1+SCY	2	0	3+SCY
0	0	00	1	0	2+SCY	1	¼ (0)	2+SCY

Table 10-31. GPCM Write Control Signal Timing (continued)

Option Register Attributes				Signal Timing (LCLK clock cycles)[1]				
TRLX	XACS	ACS	CSNT	t_{AWCS}	t_{CSWP}	t_{AWE}	t_{WEN}	t_{WC}
0	0	10	1	¼ (½)	1½+SCY	1	0	1¾+SCY (1½+SCY)
0	0	11	1	½	1¼+SCY (1+SCY)	1	0	1¾+SCY (1½+SCY)
0	1	00	1	0	2+SCY	1	¼ (0)	2+SCY
0	1	10	1	1	¾+SCY (½+SCY)	1	0	1¾+SCY (1½+SCY)
0	1	11	1	2	¾+SCY (½+SCY)	2	0	2¾+SCY (2½+SCY)
1	0	00	0	0	2+2×SCY	1	0	2+2×SCY
1	0	10	0	1¼ (1½)	1¾+2×SCY (2+2×SCY)	2	0	3+2×SCY
1	0	11	0	1½	1½+2×SCY	2	0	3+2×SCY
1	1	00	0	0	2+2×SCY	1	0	2+2×SCY
1	1	10	0	2	1+2×SCY	2	0	3+2×SCY
1	1	11	0	3	1+2×SCY	3	0	4+2×SCY
1	0	00	1	0	3+2×SCY	1	1¼ (1)	3+2×SCY
1	0	10	1	1¼ (1½)	1½+2×SCY	2	0	2¾+2×SCY (2½+2×SCY)
1	0	11	1	1½	1¼+2×SCY (1+2×SCY)	2	0	2¾+2×SCY (2½+2×SCY)
1	1	00	1	0	3+2×SCY	1	1¼ (1)	3+2×SCY
1	1	10	1	2	¾+2×SCY (½+2×SCY)	2	0	2¾+2×SCY (2½+2×SCY)
1	1	11	1	3	¾+2×SCY (½+2×SCY)	3	0	3¾+2×SCY (3½+2×SCY)

[1] Times in parentheses are specific for the case LCRR[CLKDIV] = 2; other times apply to all CLKDIV values.

10.4.2.3 Chip-Select Assertion Timing

The banks selected to work with the GPCM support an option to drive the $\overline{\text{LCS}n}$ signal with different timings (with respect to the external address/data bus). $\overline{\text{LCS}n}$ can be driven in any of the following ways:

- Simultaneous with the latched memory address. (This refers to the externally latched address and not the address timing on LAD. That is, the chip select does not assert during LALE).
- One quarter of a clock cycle later (for LCRR[CLKDIV] = 4, 8).

- One half of a clock cycle later (for LCRR[CLKDIV] = 2, 4, or 8).
- One clock cycle later (for LCRR[CLKDIV] = 4), when OR*n*[XACS] = 1.
- Two clock cycles later (for LCRR[CLKDIV] = 2, 4, or 8), when OR*n*[XACS] = 1.
- Three clock cycles later (for LCRR[CLKDIV] = 2, 4, or 8), when OR*n*[XACS] = 1 and OR*n*[TRLX] = 1.

The timing diagram in Figure 10-33 shows two chip-select assertion timings for the case LCRR[CLKDIV] = 4 or 8. If LCRR[CLKDIV] = 2, $\overline{\text{LCS}n}$ asserts identically for OR*n*[ACS] = 10 or 11.

10.4.2.3.1 Programmable Wait State Configuration

The GPCM supports internal generation of transfer acknowledge. It allows between zero and 30 wait states to be added to an access by programming OR*n*[SCY] and OR*n*[TRLX]. Internal generation of transfer acknowledge is enabled if OR*n*[SETA] = 0. If $\overline{\text{LGTA}}$ is asserted externally two bus clock cycles or more before the wait state counter has expired (to allow for synchronization latency), the current memory cycle is terminated by $\overline{\text{LGTA}}$; otherwise it is terminated by the expiration of the wait state counter. Regardless of the setting of OR*n*[SETA], wait states prolong the assertion duration of both $\overline{\text{LOE}}$ and $\overline{\text{LWE}n}$ in the same manner. When TRLX = 1, the number of wait states inserted by the memory controller is doubled from OR*n*[SCY] cycles to 2×OR*n*[SCY] cycles, allowing a maximum of 30 wait states.

10.4.2.3.2 Chip-Select and Write Enable Negation Timing

Figure 10-32 shows a basic connection between the local bus and a static memory device. In this case, $\overline{\text{LCS}n}$ is connected directly to $\overline{\text{CE}}$ of the memory device. The $\overline{\text{LWE}}$[0–3] signals are connected to the respective $\overline{\text{WE}}$[3–0] signals on the memory device where each $\overline{\text{LWE}}$[0–3] signal corresponds to a different data byte.

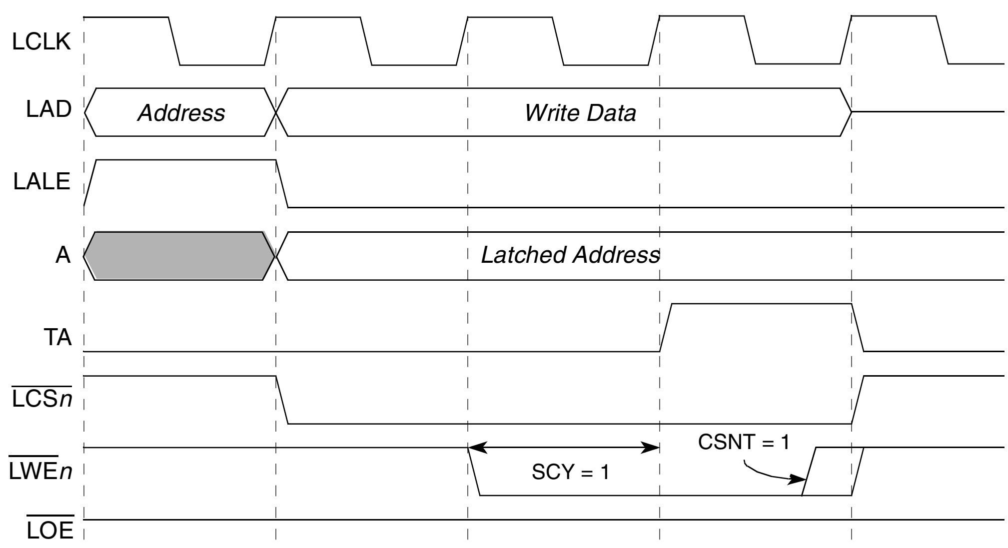


Figure 10-36. GPCM Basic Write Timing
(XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0, CLKDIV = 4, 8)

As Figure 10-36 shows, the timing for $\overline{\text{LCS}n}$ is the same as for the latched address. The strobes for the transaction are supplied by $\overline{\text{LOE}}$ or $\overline{\text{LWE}n}$, depending on the transaction direction—read or write (write case shown in the figure). OR*n*[CSNT], along with OR*n*[TRLX], control the timing for the appropriate strobe negation in write cycles. When this attribute is asserted, the strobe is negated one quarter of a clock before the normal case provided that LCRR[CLDIV] = 4 or 8. For example, when ACS = 00 and CSNT = 1, $\overline{\text{LWE}}$*n* is negated one quarter of a clock earlier, as shown in Figure 10-36. If LCRR[CLDIV] = 2, $\overline{\text{LWE}}$*n* is negated either coincident with $\overline{\text{LCS}}$*n* or one cycle earlier.

1. $\overline{\text{LCS}}$*n* is affected by CSNT and TRLX only if ACS[0] is non zero. However, $\overline{\text{LWE}}$*n* is affected independent of ACS.
2. When CSNT attribute is asserted, the strobe is negated one quarter of a clock before the normal case provided that LCRR[CLDIV] = 4 or 8.
3. TRLX = 1 in conjunction with CSNT = 1, negates the $\overline{\text{LCS}}$*n* and $\overline{\text{LWE}}$*n* 1+1/4 cycle earlier if LCRR[CLKDIV] = 4 or 8.

If LCRR[CLKDIV] = 2, $\overline{\text{LCS}}$*n* and $\overline{\text{LWE}}$*n* are negated either normally or one cycle earlier if TRLX = 1.

For example, when ACS = 00, CSNT = 1 and TRLX = 0, $\overline{\text{LWE}}$*n* is negated one quarter of a clock earlier and $\overline{\text{LCS}}$*n* is negated normally as shown in Figure 10-36.

10.4.2.3.3 Relaxed Timing

OR*x*[TRLX] is provided for memory systems that require more relaxed timing between signals. Setting TRLX = 1 has the following effect on timing:

- An additional bus cycle is added between the address and control signals (but only if ACS is not equal to 00).
- The number of wait states specified by SCY is doubled, providing up to 30 wait states.
- The extended hold time on read accesses (EHTR) is extended further.
- $\overline{\text{LCS}}$*n* signals are negated one cycle earlier during writes (but only if ACS is not equal to 00).
- $\overline{\text{LWE}}$[0–3] signals are negated one cycle earlier during writes.

Figure 10-37 and Figure 10-38 show relaxed timing read and write transactions. The effect of LCRR[CLKDIV] = 2 for these examples is only to delay the assertion of $\overline{\text{LCS}}$*n* in the ACS = 10 case to the ACS = 11 case. The example in Figure 10-38 also shows address and data multiplexing on LAD for a pair of writes issued consecutively.

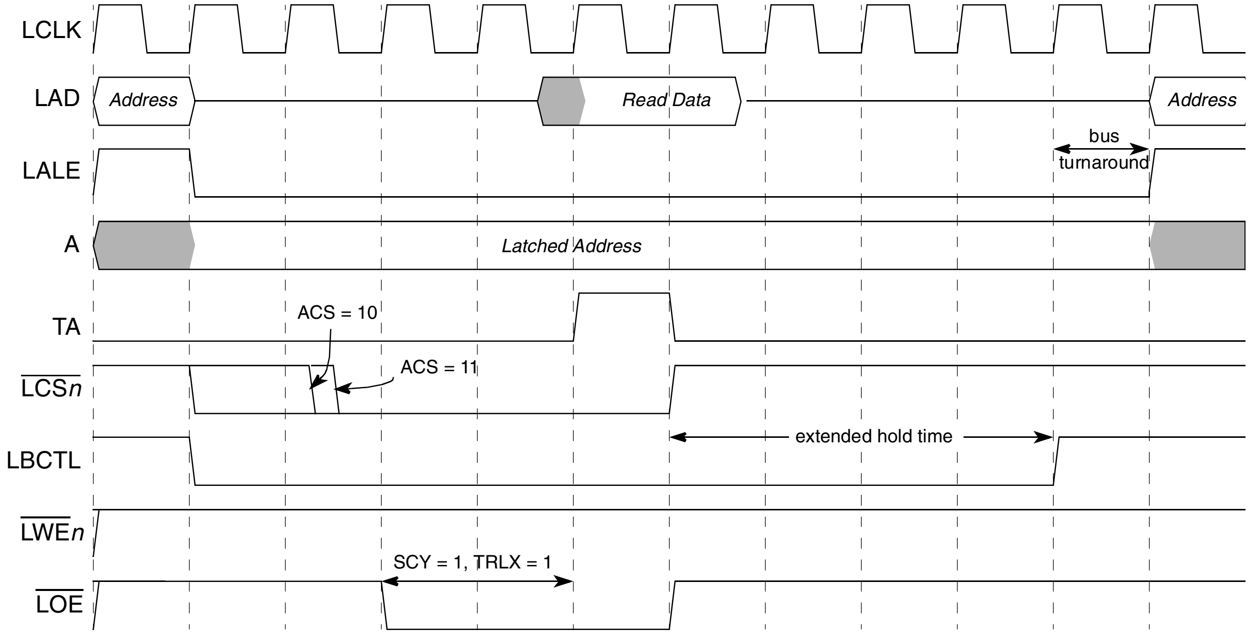


Figure 10-37. GPCM Relaxed Timing Back-to-Back Reads (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, EHTR = 0, CLKDIV = 4, 8)

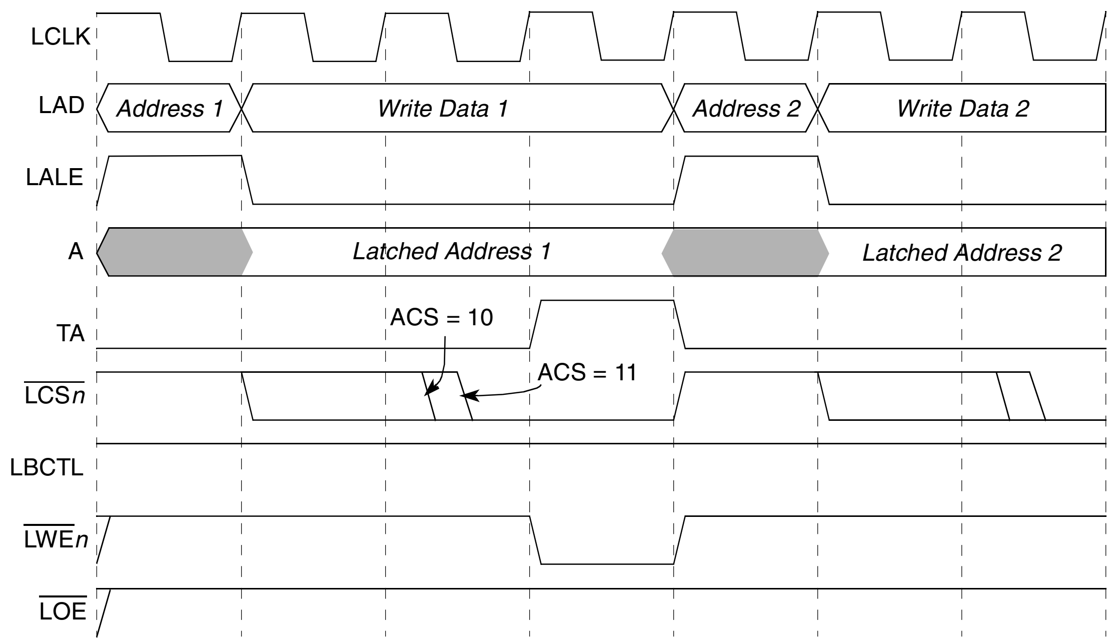


Figure 10-38. GPCM Relaxed Timing Back-to-Back Writes (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1, CLKDIV = 4, 8)

When TRLX and CSNT are set in a write access, the $\overline{\text{LWE}}$[0–3] strobe signals are negated one clock earlier than in the normal case, as shown in Figure 10-39 and Figure 10-40. If ACS ≠ 00, $\overline{\text{LCS}n}$ is also negated one clock earlier.

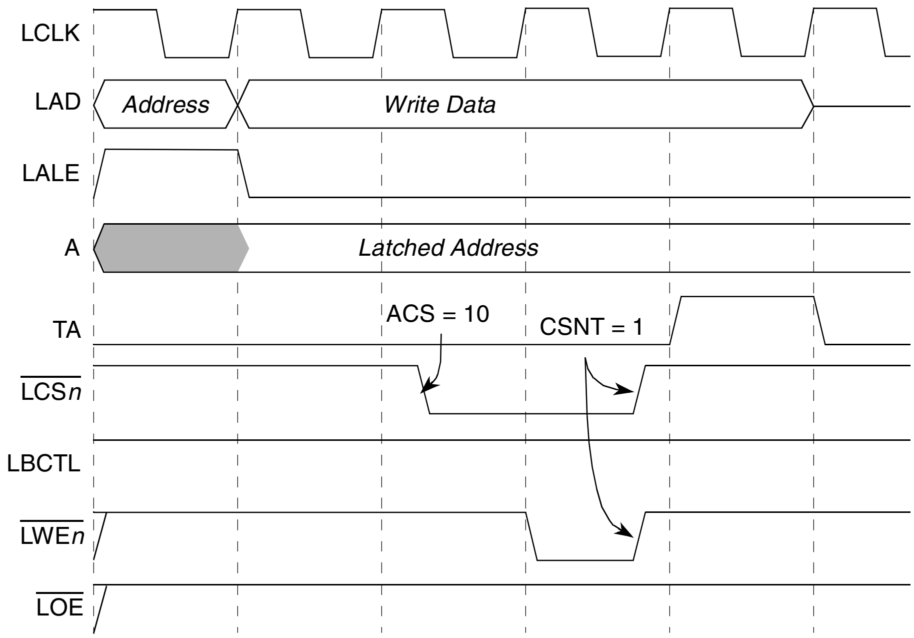


Figure 10-39. GPCM Relaxed Timing Write (XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1, CLKDIV = 4, 8)

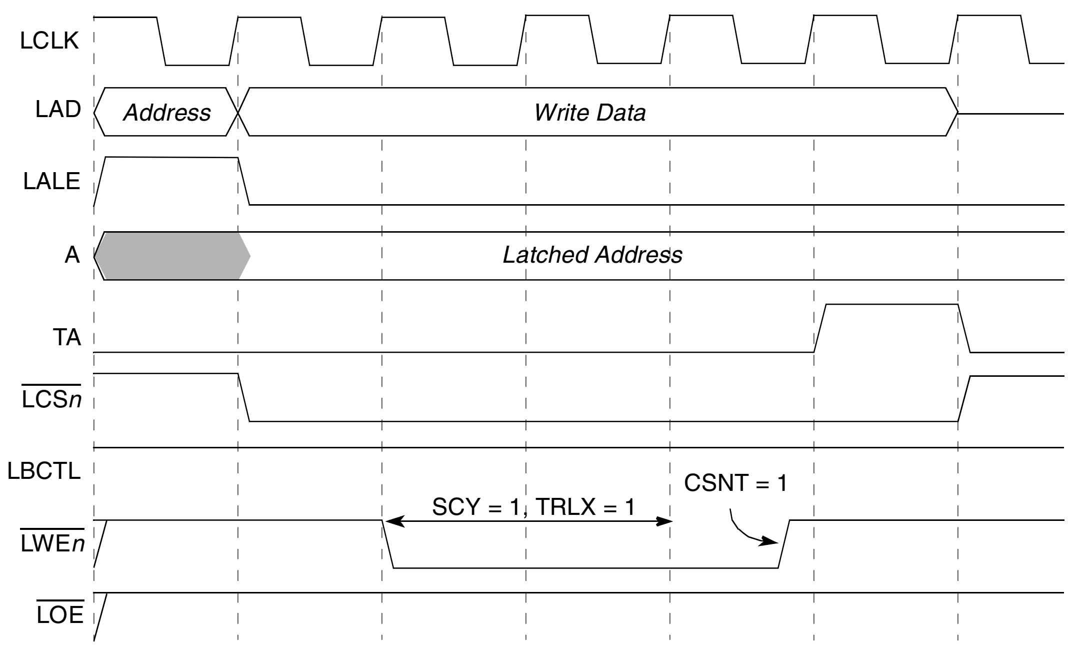


Figure 10-40. GPCM Relaxed Timing Write (XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1, CLKDIV = 4, 8)

10.4.2.3.4 Output Enable (LOE) Timing

The timing of the LOE is affected only by TRLX. It always asserts and negates on the rising edge of the bus clock. LOE asserts either on the rising edge of the bus clock after LCS*n* is asserted or coinciding with LCS*n* (if XACS = 1 and ACS = 10 or ACS = 11). Accordingly, assertion of LOE can be delayed (along with the assertion of LCS*n*) by programming TRLX = 1. LOE negates on the rising clock edge coinciding with LCS*n* negation

10.4.2.3.5 Extended Hold Time on Read Accesses

Slow memory devices that take a long time to disable their data bus drivers on read accesses should choose some combination of OR*n*[TRLX,EHTR]. Any access following a read access to the slower memory bank is delayed by the number of clock cycles specified in Table 10-7 in addition to any existing bus turnaround cycle. The final bus turnaround cycle is automatically inserted by the eLBC for reads, regardless of the setting of OR*n*[EHTR].

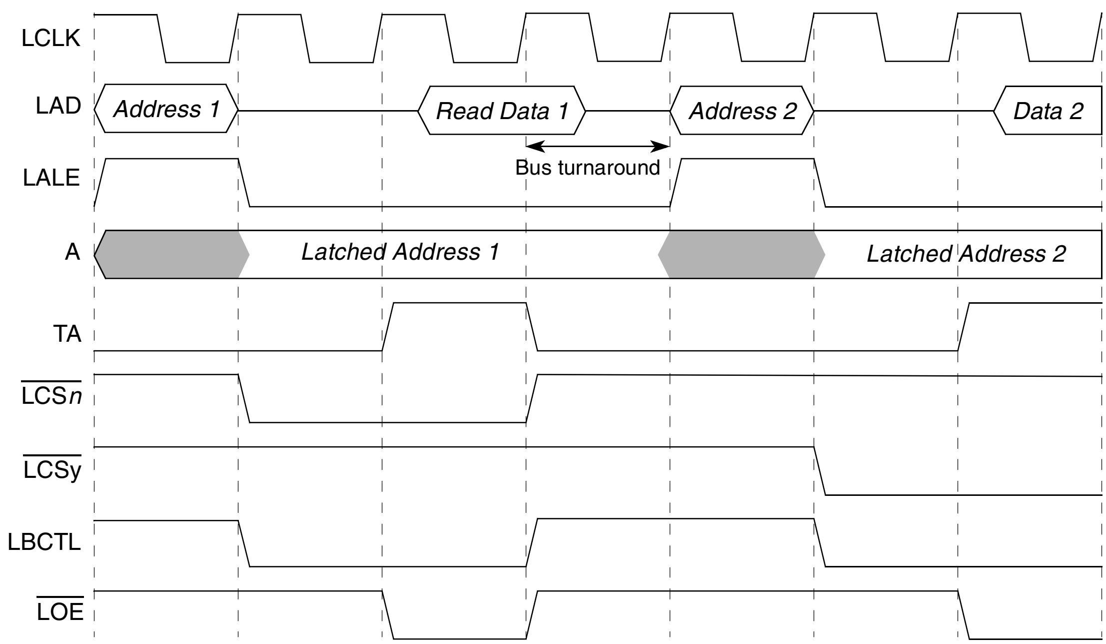


Figure 10-41. GPCM Read Followed by Read (TRLX = 0, EHTR = 0, Fastest Timing)

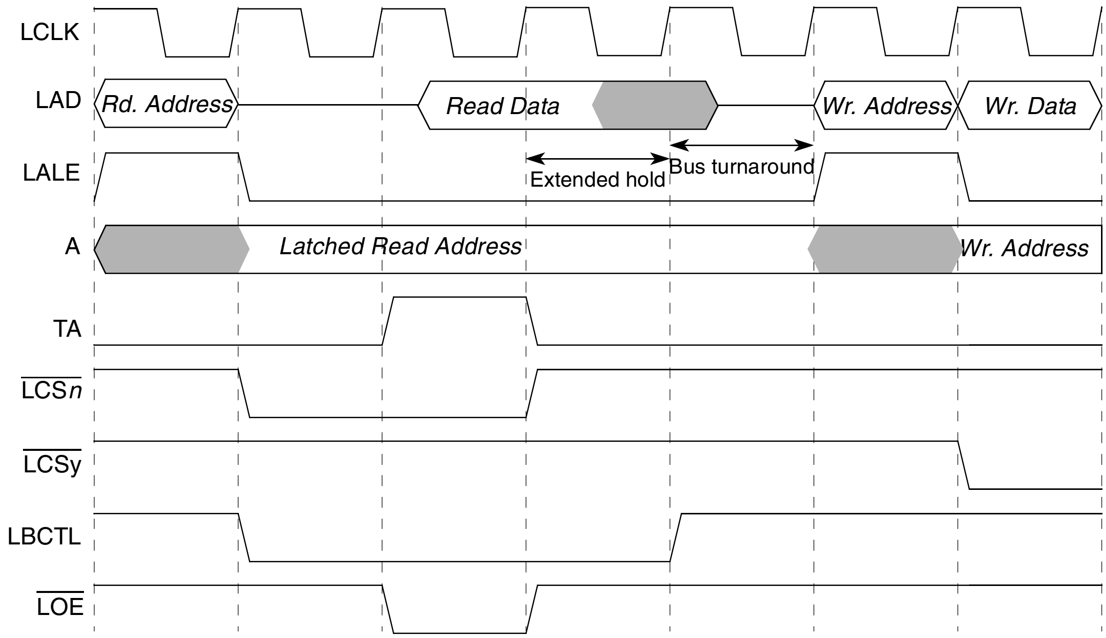


Figure 10-42. GPCM Read Followed by Write (TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads)

10.4.2.4 External Access Termination (LGTA)

External access termination is supported by the GPCM using the asynchronous LGTA input signal, which is synchronized and sampled internally by the local bus. If, during assertion of LCS*n*, the sampled LGTA signal is asserted, it is converted to an internal generation of transfer acknowledge, which terminates the current GPCM access (regardless of the setting of OR*n*[SETA]). LGTA should be asserted for at least one bus cycle to be effective. Note that because LGTA is synchronized, bus termination occurs two cycles after LGTA assertion, so in case of read cycle, the device still must drive data as long as LOE is asserted.

The user selects whether transfer acknowledge is generated internally or externally (LGTA) by programming OR*n*[SETA]. Asserting LGTA always terminates an access, even if OR*n*[SETA] = 0 (internal transfer acknowledge generation), but it is the only means by which an access can be terminated if OR*n*[SETA] = 1. The timing of $\overline{\text{LGTA}}$ is illustrated by the example in Figure 10-43.

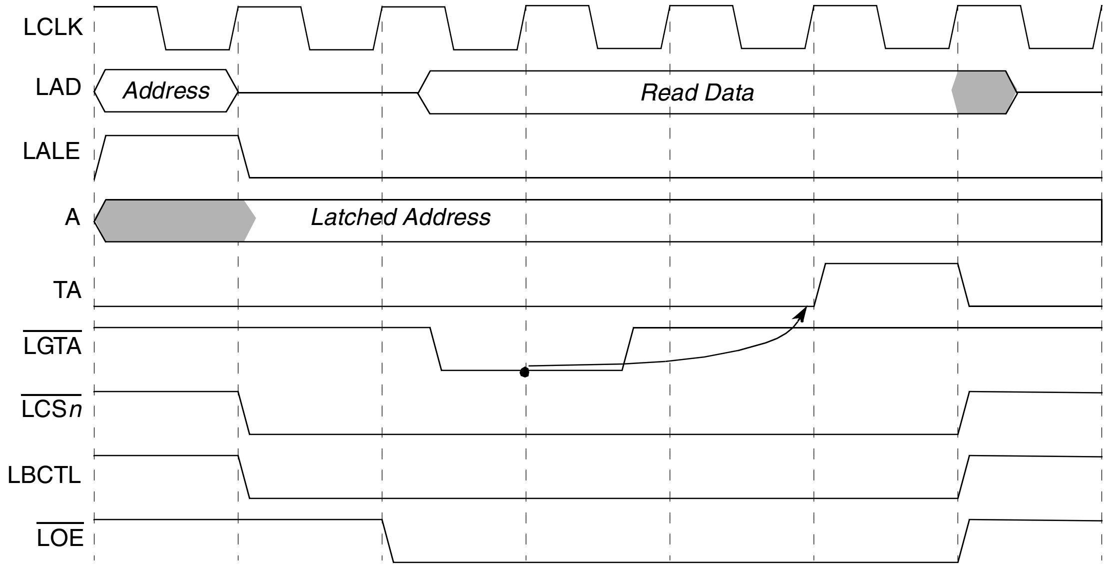


Figure 10-43. External Termination of GPCM Access

10.4.2.5 GPCM Boot Chip-Select Operation

Boot chip-select operation allows address decoding for a boot ROM before system initialization. $\overline{\text{LCS0}}$ is the boot chip-select output; its operation differs from other external chip-select outputs after a system reset. When the core begins accessing memory after system reset, $\overline{\text{LCS0}}$ is asserted for every local bus access until BR0 or OR0 is reconfigured.

The boot chip-select also provides a programmable port size, which is configured during reset. The boot chip-select does not provide write protection. $\overline{\text{LCS0}}$ operates this way until the first write to OR0 and it can be used as any other chip-select register after the preferred address range is loaded into BR0. After the first write to OR0, the boot chip-select can be restarted only with a hardware reset. Table 10-32 describes the initial values of the boot bank in the memory controller.

Table 10-32. Boot Bank Field Values after Reset for GPCM as Boot Controller

Register	Field	Setting
BR0	BA	0000_0000_0000_0000_0
	PS	*From* RCWH[ROMLOC]
	DECC	00
	WP	0
	MSEL	000
	—	—
	V	1

Table 10-32. Boot Bank Field Values after Reset for GPCM as Boot Controller (continued)

Register	Field	Setting
OR0	AM	0000_0000_0000_0000_0
	BCTLD	0
	CSNT	1
	ACS	11
	XACS	1
	SCY	1111
	SETA	0
	TRLX	1
	EHTR	1
	EAD	1

10.4.3 Flash Control Machine (FCM)

The FCM provides a glueless interface to parallel-bus NAND Flash EEPROM devices. The FCM contains three basic configuration register groups—BR*n*, OR*n*, and FMR.

Figure 10-44 shows a simple connection between an 8-bit port size NAND Flash EEPROM and the eLBC in FCM mode. Commands, address bytes, and data are all transferred on LAD[0–7][1], with $\overline{\text{LFWE}}$ asserted for transfers written to the device, or $\overline{\text{LFRE}}$ asserted for transfers read from the device. eLBC signals LFCLE and LFALE determine whether writes are of type command (only LFCLE asserted), address (only LFALE asserted), or write data (neither LFCLE nor LFALE asserted). The NAND Flash RDY/$\overline{\text{BSY}}$ pin is normally open-drain, and should be pulled high by a 4.7-KΩ resistor. On system reset, a global (boot)

1. Note bit numbering reversal: LAD[0] (msb) connects to Flash IO[7], while LAD[7] (lsb) connects to IO[0].

chip-select is available that provides a boot ROM chip-select ($\overline{LCS0}$) prior to the system being fully configured.

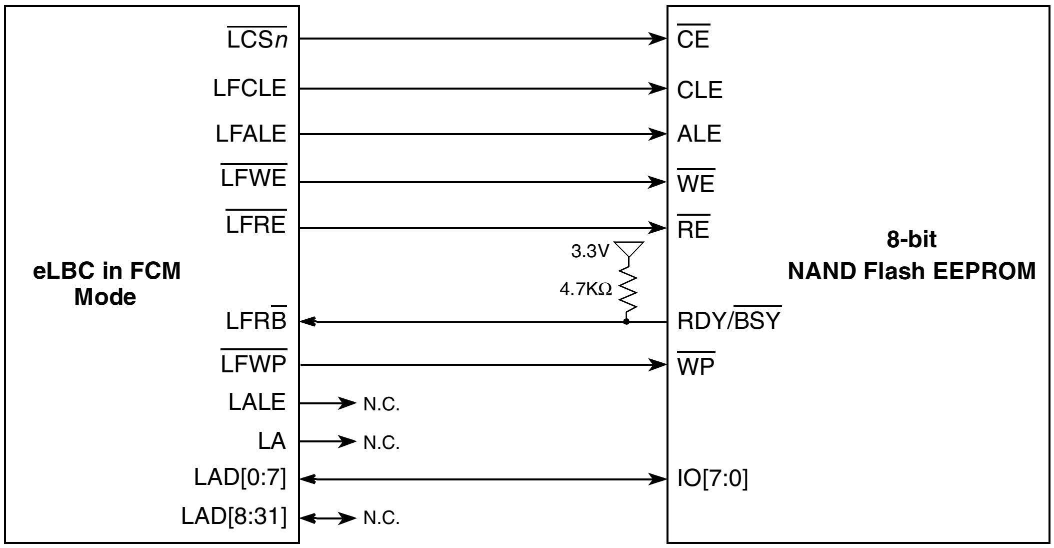


Figure 10-44. Local Bus to 8-Bit FCM Device Interface

Basic read access timing for FCM is shown in Figure 10-45. Although LCLK is shown for reference, NAND Flash EEPROMs do not make use of the clock.

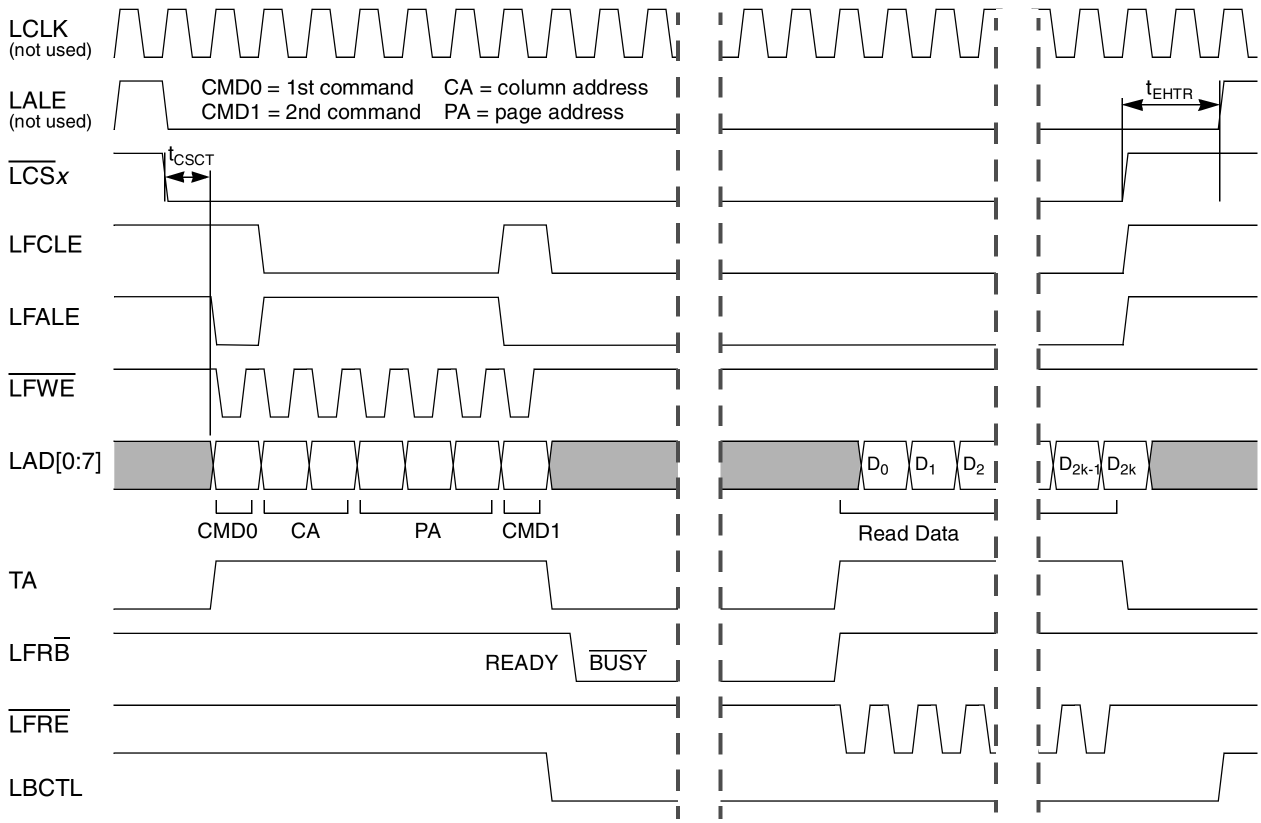


Figure 10-45. FCM Basic Page Read Timing
(PGS = 1, CSCT = 0, CST = 0, CHT = 1, RST = 1, SCY = 0, TRLX = 0, EHTR = 1)

Following the assertion of LALE, FCM asserts $\overline{\text{LCS}n}$ to commence a command sequence to the Flash device. After a delay of t_{CSCT}, the first command can be written to the device on assertion of $\overline{\text{LFWE}}$, followed by any parameters (typically address bytes and data), and concluded with a secondary command. In many cases, the second command initiates a long-running operation inside the Flash device, which pulls the wired-OR pin LFR$\overline{\text{B}}$ low to indicate that the device is busy. Since in Figure 10-45 FCM is now expecting a read response, it takes LBCTL low to turnaround any bus transceivers that are present. Upon LFR$\overline{\text{B}}$ indicating ready status, FCM asserts $\overline{\text{LFRE}}$ repeatedly to recover bytes of read data, and the bytes are stored in eLBC's FCM buffer RAM while an ECC is optionally computed on the bytes transferred. Finally, FCM negates $\overline{\text{LCS}n}$ and delays eLBC by t_{EHTR} before any subsequent memory access occurs.

10.4.3.1 FCM Buffer RAM

Read and write accesses to eLBC banks controlled by FCM do not access attached NAND Flash EEPROMs directly. Rather, these accesses read and write the FCM buffer RAM—a single, shared 8-Kbyte space internal to eLBC and mapped by the base address of every FCM bank. Even though each FCM-controlled bank will have a different base address to differentiate it, all accesses to such banks will access the same buffer space. External eLBC signals, such as LALE and $\overline{\text{LCS}n}$, will not assert upon accesses to the buffer RAM. The FCM buffer RAM is logically divided into two or more buffers, depending on the setting of OR*n*[PGS], with different buffers being accessible concurrently by software and FCM.

To perform a page read operation from a NAND Flash device, software initializes the FCM command, mode, and address registers, before issuing a special operation (FMR[OP] set non-zero) to a particular FCM-controlled bank. FCM will execute the sequence of op-codes held in FIR, reading data from the Flash device into the shared buffer RAM. While this read is taking place, software is free to access any data stored in other, currently inactive buffers of the FCM buffer RAM through reads or writes to any bank controlled by FCM. If command completion interrupts are enabled, an interrupt will be generated once FCM has completed the read. When FCM has completed its last command, software can switch to the newly read buffer and issue further commands.

To perform a page write operation, software first prepares data to be written in a fresh buffer. Then, the FCM command, mode, and address registers are initialized, and a special operation (FMR[OP] set non-zero) is issued to a particular FCM-controlled bank. FCM will execute the sequence of op-codes held in FIR, writing data from shared buffer RAM to the Flash device. To ensure that the device is enabled for programming, software must initialize FMR[OP] = 11, which prevents assertion of $\overline{\text{LFWP}}$ during the write. While this write is taking place, software is free to access any data stored in other, currently inactive buffers of the FCM buffer RAM through reads or writes to any bank controlled by FCM. When FCM has completed its last command, software can re-use the previously written buffer and issue further commands.

See Section 10.4.3.4.2, “Boot Block Loading into the FCM Buffer RAM,” for a description of the shared buffer RAM layout during boot.

10.4.3.1.1 Buffer Layout and Page Mapping for Small-Page NAND Flash Devices

The FCM buffer space is divided into eight 1-Kbyte buffers for small-page devices (OR*n*[PGS] = 0), mapped as shown in Figure 10-46. Each page in a small-page NAND Flash comprises 528 bytes, where 512 bytes appear as main region data, and 16 bytes appear as spare region data. The EEPROM’s page numbered *P* is associated with buffer number (*P* mod 8), where *P* = FPAR[PI]. Since the bank size set by OR*n*[AM] will be greater than 8 Kbytes, an identical image of the FCM buffer RAM appears replicated every 8 Kbytes throughout the bank address space. It is recommended that the bank size be set to 32 Kbytes, which covers a single NAND Flash block for small-page devices.

For FCM commands, register FPAR sets the page address and, therefore, also the buffer number.In the case that FBCR[BC] = 0, FCM transfers an entire page, comprising the 512-byte main region followed by the 16-byte spare region; the 496-byte reserved region is not accessed, and remains undefined for software. However, for commands given a specific byte count in FBCR[BC], FPAR[MS] locates the starting address in either the main region (MS = 0) or the spare region (MS = 1). Where different eLBC banks control both small and large-page devices, a large-page 4-Kbyte buffer must be assigned to either the first 4 or last 4 small-page buffers.

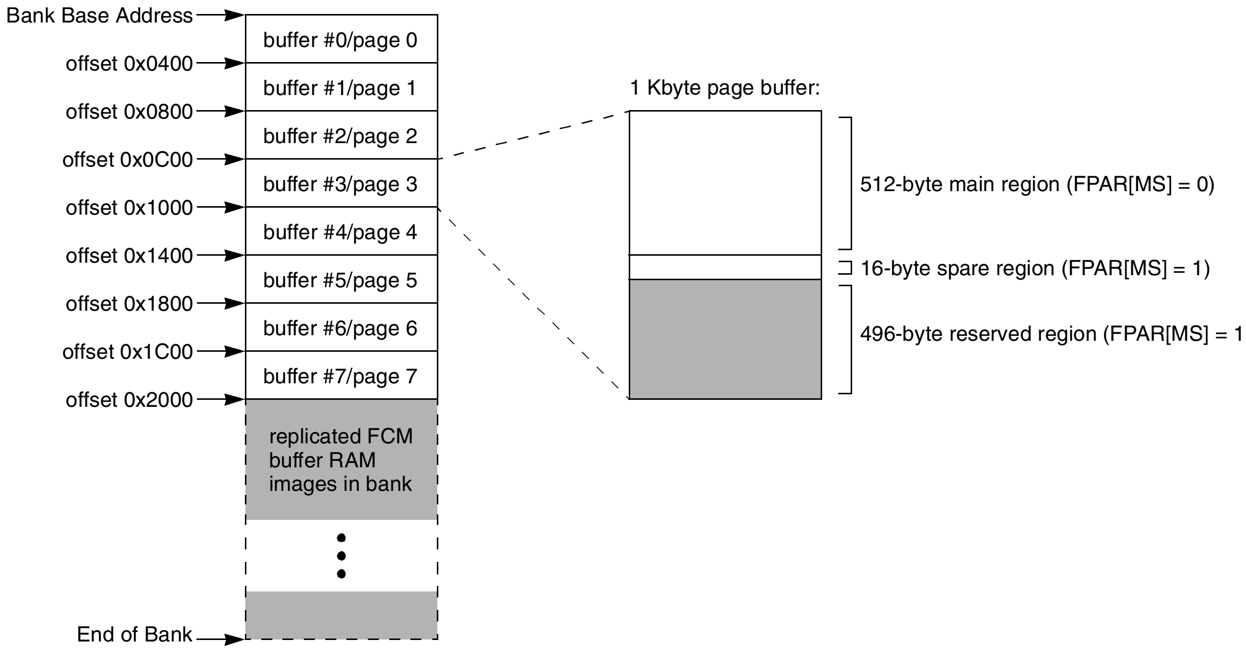


Figure 10-46. FCM Buffer RAM Memory Map for Small-Page (512-byte page) NAND Flash Devices

10.4.3.1.2 Buffer Layout and Page Mapping for Large-Page NAND Flash Devices

The FCM buffer space is divided into two 4 Kbyte buffers for large-page devices (OR*n*[PGS] = 1), mapped as shown in Figure 10-47. Each page in a large-page NAND Flash comprises 2112 bytes, where 2048 bytes appear as main region data, and 64 bytes appear as spare region data. The EEPROM's page numbered *P* is associated with buffer number (*P* mod 2), where *P* = FPAR[PI]. Since the bank size set by OR*n*[AM] will be greater than 8 Kbytes, an identical image of the FCM buffer RAM appears replicated every 8 Kbytes throughout the bank address space. It is recommended that the bank size be set to 256 Kbytes, which covers a single NAND Flash block for large-page devices.

For FCM commands, register FPAR sets the page address and, therefore, also the buffer number. In the case that FBCR[BC] = 0, FCM transfers an entire page, comprising the 2048-byte main region followed by the 64-byte spare region; the 1984-byte reserved region is not accessed, and remains undefined for software. However, for commands given a specific byte count in FBCR[BC], FPAR[MS] locates the starting address in either the main region (MS = 0) or the spare region (MS = 1). Where different eLBC banks control both small and large-page devices, a large-page 4 Kbyte buffer must be assigned to either the first 4 or last 4 small-page buffers.

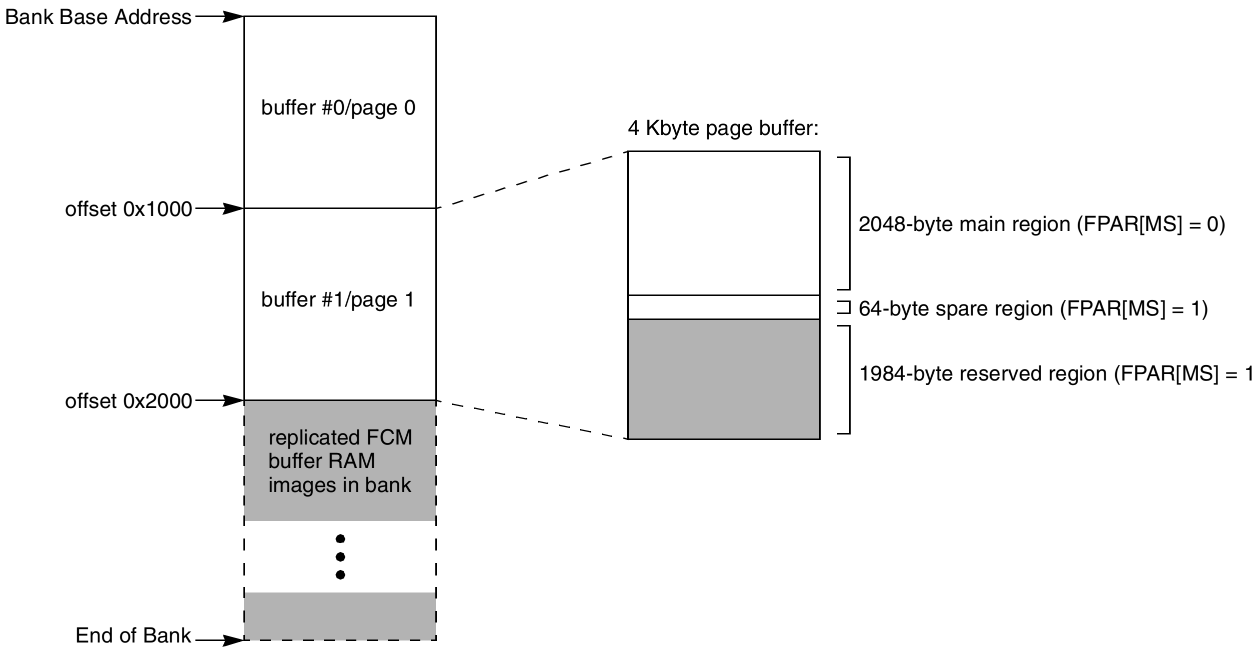


Figure 10-47. FCM Buffer RAM Memory Map for Large-Page (2-Kbyte page) NAND Flash Devices

10.4.3.1.3 Error Correcting Codes and the Spare Region

The FCM's ECC engine makes use of data in the NAND Flash spare region to store pre-computed ECC code words. ECC is calculated in a single pass over blocks of 512 bytes of data in the main region. The setting of FMR[ECCM] determines the location of the 24-bit ECC in the spare region.

The basic ECC algorithm is depicted in Figure 10-48. The stream of data bytes is considered to form a matrix having 8 columns (corresponding with the device bus IO[7–0] or IO[15–8]) and 512 rows (corresponding with each byte in the ECC block). Six bits of parity, {P_4, P_4', P_2, P_2', P_1, P_1'}, are calculated across the columns, and at most 18 bits of parity {P_{2048}, P_{2048}', ..., P_{16}, P_{16}', P_8, P_8'} are calculated across the rows to create a 24-bit Hamming code for the data block. In this calculation, parity bit P_N' is the exclusive-OR of every alternate *N*-bit group of bits positioned at even intervals (starting at

N-bit group 0, then continuing to group 2, 4, etc.), while parity bit P_N is the exclusive-OR of every alternate *N*-bit group of bits positioned at odd intervals (starting at *N*-bit group 1, then continuing to group 3, 5, etc.).

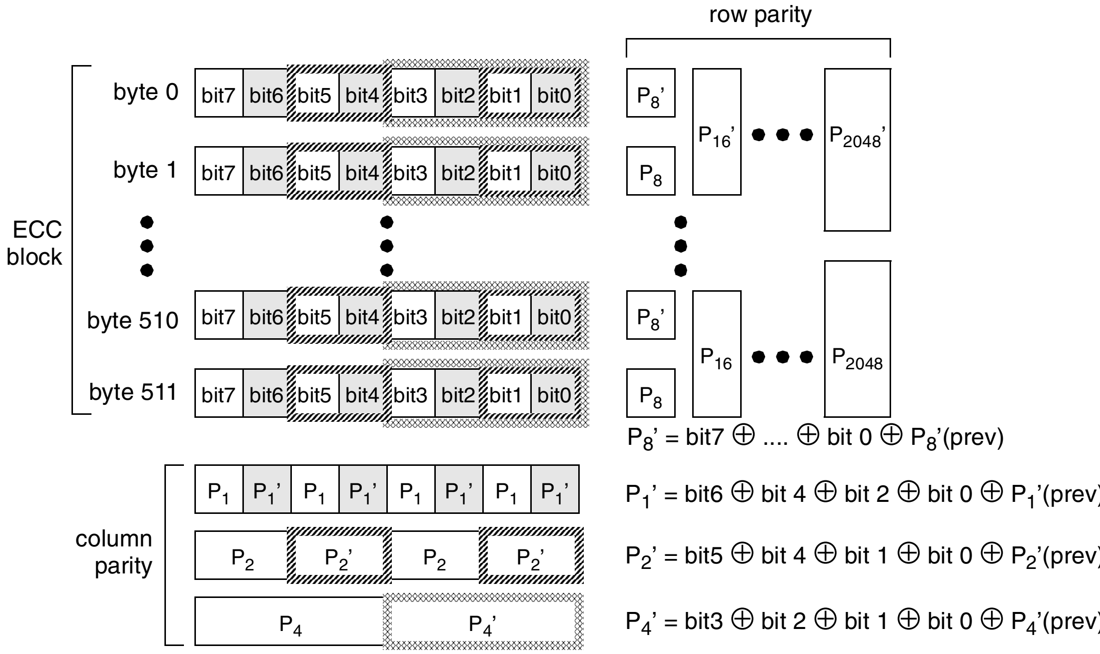


Figure 10-48. FCM ECC Calculation

The 24-bit ECC code word format is shown in Figure 10-49 for normal ECC polarity. Setting LBCR[EPAR] = 1 changes ECC polarity, and thus omits negation of each P_N and P_N' bit.

	0 (MSB)	1	2	3	4	5	6	7 (LSB)
EC0	$\sim P_{64}$	$\sim P_{64}'$	$\sim P_{32}$	$\sim P_{32}'$	$\sim P_{16}$	$\sim P_{16}'$	$\sim P_{8}$	$\sim P_{8}'$
EC1	$\sim P_{1024}$	$\sim P_{1024}'$	$\sim P_{512}$	$\sim P_{512}'$	$\sim P_{256}$	$\sim P_{256}'$	$\sim P_{128}$	$\sim P_{128}'$
EC2	$\sim P_{4}$	$\sim P_{4}'$	$\sim P_{2}$	$\sim P_{2}'$	$\sim P_{1}$	$\sim P_{1}'$	$\sim P_{2048}$	$\sim P_{2048}'$

Figure 10-49. ECC Layout for LBCR[EPAR] = 0 (~ represents logical negation)

The placement of ECC code words in relation to FMR[ECCM] is shown in Figure 10-50. For small-page devices, only a single 512-byte main region is ECC-protected. For large-page devices, there are four adjacent main regions, and each has a 16-byte spare region—of which only one is shown in the figure. If eLBC is configured to generate ECC (BR*n*[DECC] = 10), FCM will substitute on full-page write transfers the three code word bytes in place of the spare region data originally provided at the locations shown in Figure 10-50. Transfers shorter than a full page, however, require software to prepare the appropriate ECC in the spare region. Similarly, FCM can check and correct bit errors on full-page reads if BR*n*[DECC] = 01 or 10. A correctable error is a single bit error in any 512-byte block of main region data, as judged by comparison of a regenerated ECC with the ECC retrieved from the spare region, or a single bit error in the retrieved ECC only. Bit errors in the main region are corrected before FCM completes its final read transfer and signals an event in LTESR[CC]. Errors that appear more complex (two or more bits in error per 512-byte block) are not corrected, but are flagged as parity errors by FCM. The bit vector in LTEATR[PB] can be checked to determine which 512-byte blocks in a large-page NAND Flash main region were found to be uncorrectable.

ECCM	Byte 0 … Byte 511	Other Mains	Spare 0 … 5	6	7	8	9	10	11 … 15
0	Main Region		—	EC0	EC1	EC2	—		
1	Main Region		—			EC0	EC1	EC2	—

Figure 10-50. ECC Placement in NAND Flash Spare Regions in Relation to FMR[ECCM]

10.4.3.2 Programming FCM

FCM has a fully general command and data transfer sequencer that caters for both common and specific/proprietary NAND Flash command sequences. The command sequencer reads a program out of the FIR register, which can hold up to 8 instructions, each represented by a 4-bit op-code, as illustrated in Figure 10-51. The first instruction executed is read from FIR[OP0], the next is read from FIR[OP1], and likewise to subsequent instructions, ending at FIR[OP7] or until the only instructions remaining are NOPs. If FIR contains nothing but NOP instructions, FCM will not assert $\overline{LCSn}$, otherwise, $\overline{LCSn}$ is asserted prior to the first instruction and remains asserted until the last instruction has completed. If LTESR[CC] is enabled, completion of the last instruction will trigger a command completion event interrupt from eLBC.

Prior to executing a sequence, necessary operands for the instructions will need to be set in the FMR, FCR, MDR, FBCR, FBAR, and FPAR registers. The AS0–AS3 address and data pointers associated with FCM's use of MDR all reset to select AS0 at the start of the instruction sequence. A complete list of op-codes can be found in Section 10.3.1.17, "Flash Instruction Register (FIR)."

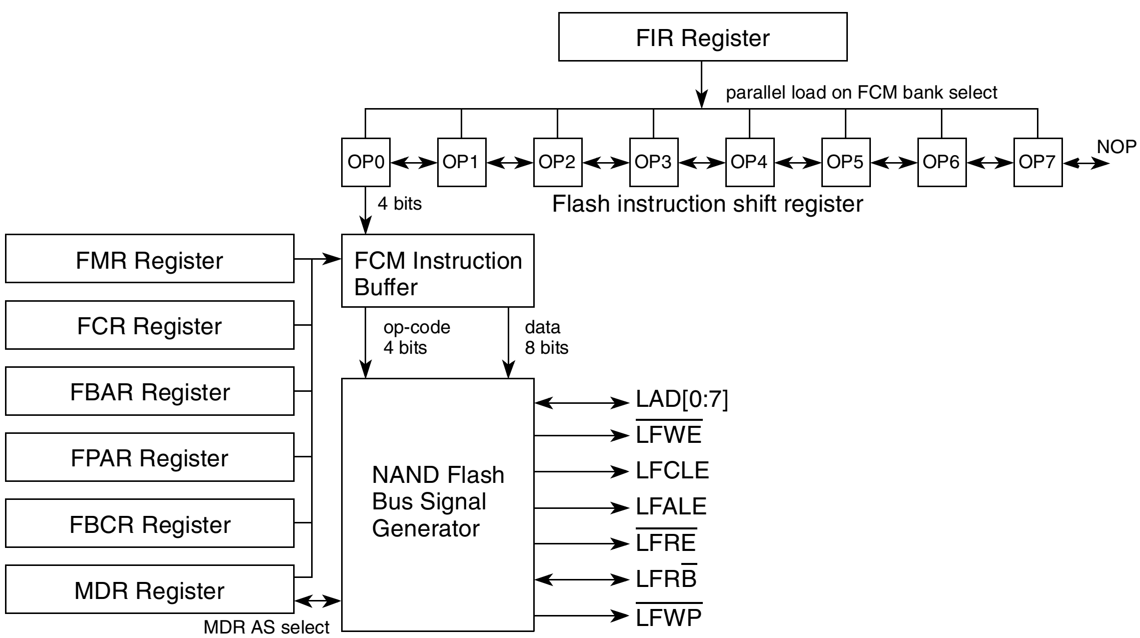


Figure 10-51. FCM Instruction Sequencer Mechanism

10.4.3.2.1 FCM Command Instructions

There are two kinds of command instruction:

- Commands that issue immediately—CM0, CM1, CM2, and CM3. These commands write a single command byte by asserting LFCLE and $\overline{\text{LFWE}}$ while driving an 8-bit command onto LAD[0–7]. Op-code CM*n* sources its command byte from field FCR[CMD*n*], therefore up to four different commands can be issued in any FCM instruction sequence.
- Commands that wait for LFR$\overline{\text{B}}$ to be sampled high (EEPROM in ready state) before issuing—CW0, and CW1. These commands first poll the LFR$\overline{\text{B}}$ pin, waiting for it to go high, before writing a single command byte onto LAD[0–7], sourced from FCR[CMD*n*] for op-code CW*n*. It is necessary to use CW*n* op-codes whenever the EEPROM is expected to be in a busy state (such as following a page read, block erase, or program operation) and therefore initially unresponsive to commands. To avoid deadlock in cases where the device is already available, FCM does not expect a transition on LFR$\overline{\text{B}}$. Rather, FCM waits for 8×(2+OR*n*[SCY]) clock cycles (when OR*n*[TRLX] = 0) or 16×(2+OR*n*[SCY]) clock cycles (when OR*n*[TRLX] = 1) before sampling the level of LFR$\overline{\text{B}}$. If the level of LFR$\overline{\text{B}}$ does not return high before a time-out set by FMR[CWTO] occurs, FCM proceeds to issue the command normally, and a FCT event is issued to LTESR.

The manufacturer's datasheet should be consulted to determine values for programming into the FCR register, and whether a given command in the sequence is expected to initiate busy device behavior.

10.4.3.2.2 FCM No-Operation Instruction

A NOP instruction that appears in FIR ahead of the last instruction is executed with the timing of a regular command instruction, but neither LFCLE nor $\overline{\text{LFWE}}$ are asserted. Thus a NOP instruction may be used to insert a pause matching the time taken for a regular command write.

10.4.3.2.3 FCM Address Instructions

Address instructions are used to issue addresses to the NAND Flash EEPROM. A complete device address is formed from a sequence of one or more bytes, each written onto LAD[0–7] with LFALE and $\overline{\text{LFWE}}$ asserted together. There are three kinds of address generation provided:

- Column address—CA. A column address comprises one byte (OR*n*[PGS] = 0) or two bytes (OR*n*[PGS] = 1) locating the starting byte or word to be transferred in the next page read or write sequence. FPAR[CI] sets the value of the column index provided that FBCR[BC] is non-zero. In the case that FBCR[BC] = 0, a column index of zero is issued to the device, regardless of the value in FPAR[CI].
- Page address—PA. A page address comprises 2, 3, or 4 bytes, depending on the setting of FMR[AL], and locates the data page in the NAND Flash address space. The complete page address is the concatenation of the block index, read from FBAR[BLK], with the page-in-block index, read from FPAR[PI]. The page address length set in FMR[AL] should correspond with the size of EEPROM being accessed. Similarly, the block index in FBAR[BLK] must not exceed the maximum block index for the device, as most devices require reserved address bits to be written as zero.

- User-defined address—UA. This instruction allows the FCM to write a user-defined address byte, which is read from the next AS field in MDR, starting at MDR[AS0]. Each subsequent UA instruction reads an adjacent AS field in MDR, until all four AS bytes (MDR[AS0], MDR[AS1], MDR[AS2], MDR[AS3]) have been sent; a fifth and any following UA instructions send zero as the address byte. Note that each UA instruction advances the MDR pointer for writes by one byte, and therefore a mix of UA and WS instructions can consume adjacent bytes from MDR.

10.4.3.2.4 FCM Data Read Instructions

Data read instructions assert $\overline{\text{LFRE}}$ repeatedly to transfer one or more bytes of read data from the NAND Flash EEPROM. Data read instructions are distinguished by their data destination:

- Read data to buffer RAM immediately—RB. This instruction reads FBCR[BC] bytes of data into the current FCM RAM buffer addressed by FPAR. If FBCR[BC] = 0, an entire page (including spare region) is transferred in a burst, starting at the page boundary, and the ECC calculation is checked against the ECC stored in the spare region. Correctable ECC errors are corrected; other errors may cause an interrupt if enabled. If the value of FBCR[BC] takes the read pointer beyond the end of the spare region in the buffer, FCM discards any excess bytes read.
- Read data/status to MDR immediately—RS. This instruction asserts $\overline{\text{LFRE}}$ exactly once to read one byte (8-bit port size) of data into the next AS field of MDR. Reads beyond the fourth byte of MDR are discarded. The MDR read pointer is independent of the MDR write pointer used by UA and WS instructions.
- Read data to buffer RAM once waited on ready—RBW. This instruction first polls the LFR$\overline{\text{B}}$ pin, waiting for it to go high, before proceeding with a read to buffer as described for the RB instruction. Sampling and time-outs for polling the LFR$\overline{\text{B}}$ pin follow the behavior of CW*n* instructions.
- Read data/status to MDR once waited on ready—RSW. This instruction first polls the LFR$\overline{\text{B}}$ pin, waiting for it to go high, before proceeding with a status read to MDR as described for the RS instruction. Sampling and time-outs for polling the LFR$\overline{\text{B}}$ pin follow the behavior of CW*n* instructions.

10.4.3.2.5 FCM Data Write Instructions

Data write instructions assert $\overline{\text{LFWE}}$ repeatedly (with LFCLE and LFALE both negated) to transfer one or more bytes of write data to the NAND Flash EEPROM. Data write instructions are distinguished by their data source:

- Write data from FCM buffer RAM—WB. This instruction writes FBCR[BC] bytes of data from the current FCM RAM buffer addressed by FPAR. If FBCR[BC] = 0, an entire page (including spare region) is transferred in a burst, starting at the page boundary, and the ECC calculation is stored in the and spare region in accordance with the setting of FMR[ECCM]. If the value of FBCR[BC] takes the write pointer beyond the end of the spare region in the buffer, the value of data written by FCM is undefined.
- Write data/status from MDR—WS. This instruction asserts $\overline{\text{LFWE}}$ exactly once to write one byte (8-bit port size) of data taken from the next AS field of MDR. Attempts to write beyond four bytes of MDR has the effect of writing zeros. The MDR write pointer is independent of the MDR read pointer used by RS and RSW instructions.

10.4.3.3 FCM Signal Timing

If BR*n*[MSEL] selects the FCM, the attributes for the memory cycle are taken from OR*n*. These attributes include the CSCT, CST, CHT, RST, SCY, TRLX, and EHTR fields.

10.4.3.3.1 FCM Chip-Select Timing

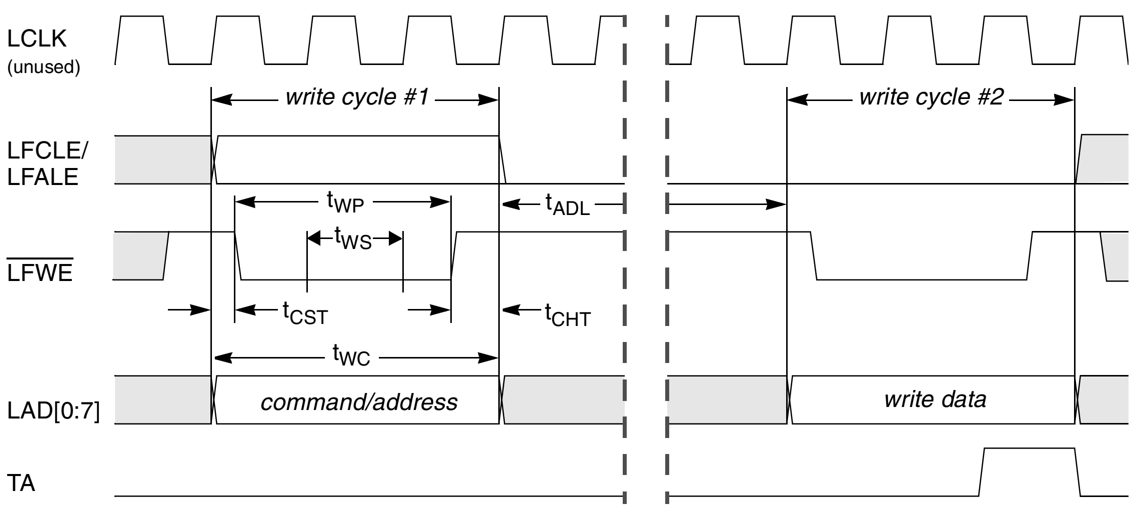
The timing of $\overline{LCSn}$ assertion in FCM mode is illustrated by the timing diagram in Figure 10-45. $\overline{LCSn}$ is asserted immediately following LALE negation, and remains asserted until the last instruction in FIR has completed. The delay, t_{CSCT}, between $\overline{LCSn}$ assertion and commencement of the first NAND Flash instruction is controlled by OR*n*[CSCT] and OR*n*[TRLX], as shown in Table 10-33. OR*n*[CSCT] should be set in accordance with the NAND Flash EEPROM chip-select to $\overline{WE}$ set-up time specification.

Table 10-33. FCM Chip-Select to First Command Timing

ORn[TRLX]	ORn[CSCT]	$\overline{LCSn}$ to First Command Delay
0	0	1 LCLK clock cycle
0	1	4 LCLK clock cycles
1	0	2 LCLK clock cycles
1	1	8 LCLK clock cycles

10.4.3.3.2 FCM Command, Address, and Write Data Timing

The FCM command (CM0–CM3, CW0, CW1), address (CA, PA, UA), and data write (WB, WS) instructions all share the same basic timing attributes. Assertion of $\overline{LFWE}$ initiates transfer via LAD[0–7], and the options in OR*n* for FCM mode establish the set-up, hold, and wait state timings with respect to $\overline{LFWE}$, as shown in Figure 10-52.



Notes: t_{CST} = Command to $\overline{LFWE}$ set-up time.
t_{CHT} = Command to $\overline{LFWE}$ hold time.
t_{ADL} = Command/address to write data delay.
t_{WP} = $\overline{LFWE}$ pulse time, driven low.
t_{WS} = Command wait state time.
t_{WC} = Command cycle time.

Figure 10-52. Timing of FCM Command/Address and Write Data Cycles (for TRLX = 0, CHT = 0, CST = 1, SCY = 1, CLKDIV = 4*N)

The timing parameters are summarized in Table 10-34.

Table 10-34. FCM Command, Address, and Write Data Timing Parameters

Option Register Attributes			Timing Parameter (LCLK Clock Cycles)[1]					
TRLX	CHT	CST	t_{CST}	t_{CHT}	t_{WS}	t_{WP}	t_{WC}	t_{ADL}
0	0	0	0	½	SCY	1½+SCY	2+SCY	4×(2+SCY)
0	0	1	¼	½	SCY	1¼+SCY	2+SCY	4×(2+SCY)
0	1	0	0	1	SCY	1+SCY	2+SCY	4×(2+SCY)
0	1	1	¼	1	SCY	¾+SCY	2+SCY	4×(2+SCY)
1	0	0	½	1½	2×SCY	1+2×SCY	3+2×SCY	8×(2+SCY)
1	0	1	1	1½	2×SCY	½+2×SCY	3+2×SCY	8×(2+SCY)
1	1	0	½	2	2×SCY	½+2×SCY	3+2×SCY	8×(2+SCY)
1	1	1	1	2	2×SCY	2×SCY	3+2×SCY	8×(2+SCY)

[1] In the parameters, SCY refers to a delay of OR*n*[SCY] clock cycles.

An example of minimum delay command timing appears in Figure 10-53. Note that the set-up, wait-state, and hold timing of command, address, and write data cycles with respect to $\overline{\text{LFWE}}$ assertion are all identical, and that the minimum cycle extends for two LCLK clock cycles.

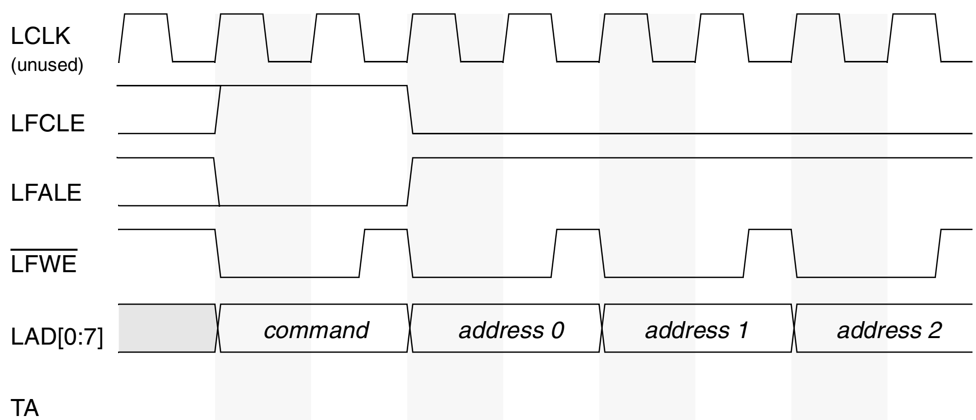


Figure 10-53. Example of FCM Command and Address Timing with Minimum Delay Parameters (for TRLX = 0, CHT = 0, CST = 0, SCY = 0, CLKDIV = 4*N)

An example of relaxed command timing is shown in Figure 10-54.

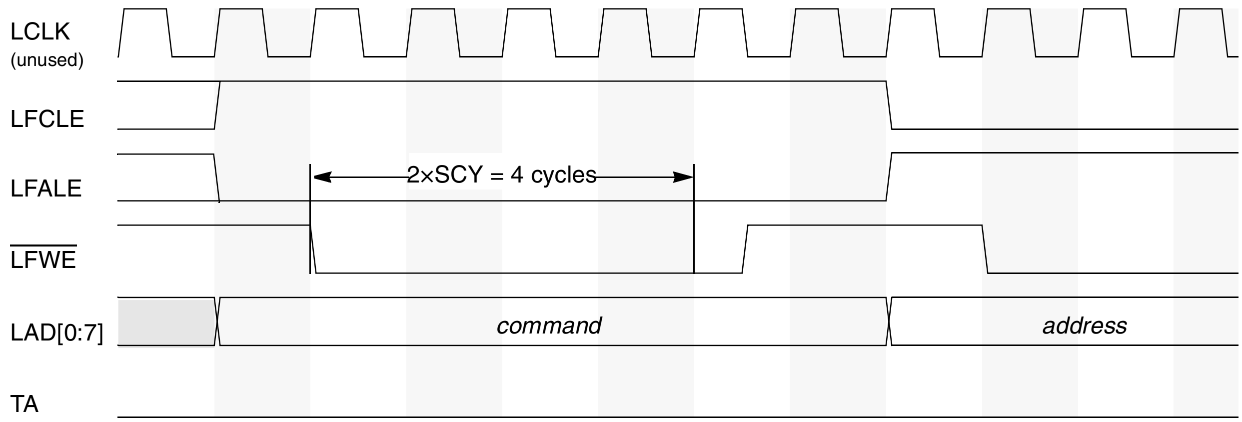


Figure 10-54. Example of FCM Command and Address Timing with Relaxed Parameters (for TRLX = 1, CHT = 0, CST = 1, SCY = 2, CLKDIV = 4*N)

10.4.3.3.3 FCM Ready/Busy Timing

Instructions CW0, CW1, RBW, and RSW force FCM to observe the state of the LFR$\overline{\text{B}}$ pin, which may be driven low by a long-latency NAND Flash operation, such as a page read. Following the issue of such commands, FCM waits as shown in Figure 10-55 before sampling the state of LFR$\overline{\text{B}}$. This guards against observing LFR$\overline{\text{B}}$ before it has been properly driven low by the device, but does not preclude LFR$\overline{\text{B}}$ from remaining high after a command. In addition, FCM samples and compares the state of LFR$\overline{\text{B}}$ on two consecutive cycles of LCLK to filter out noise on this signal as it rises to the ready state (LFR$\overline{\text{B}}$ = 1).

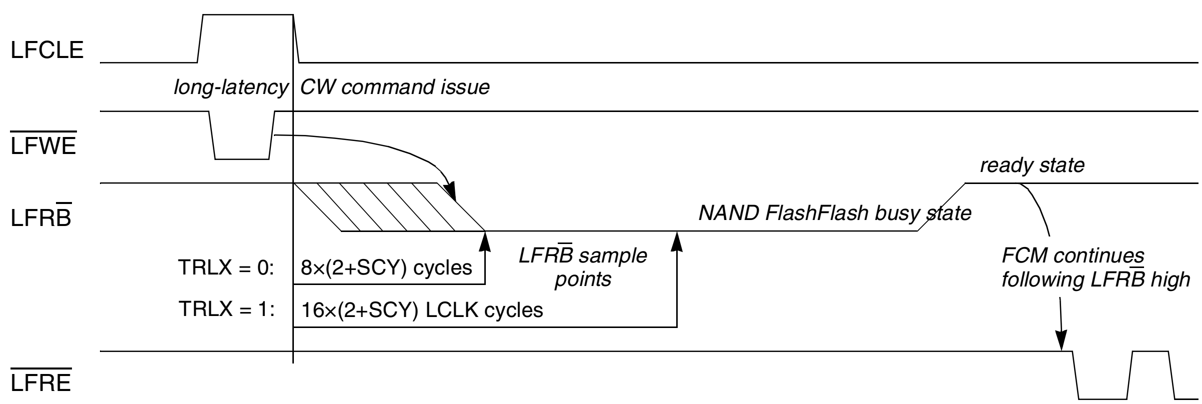
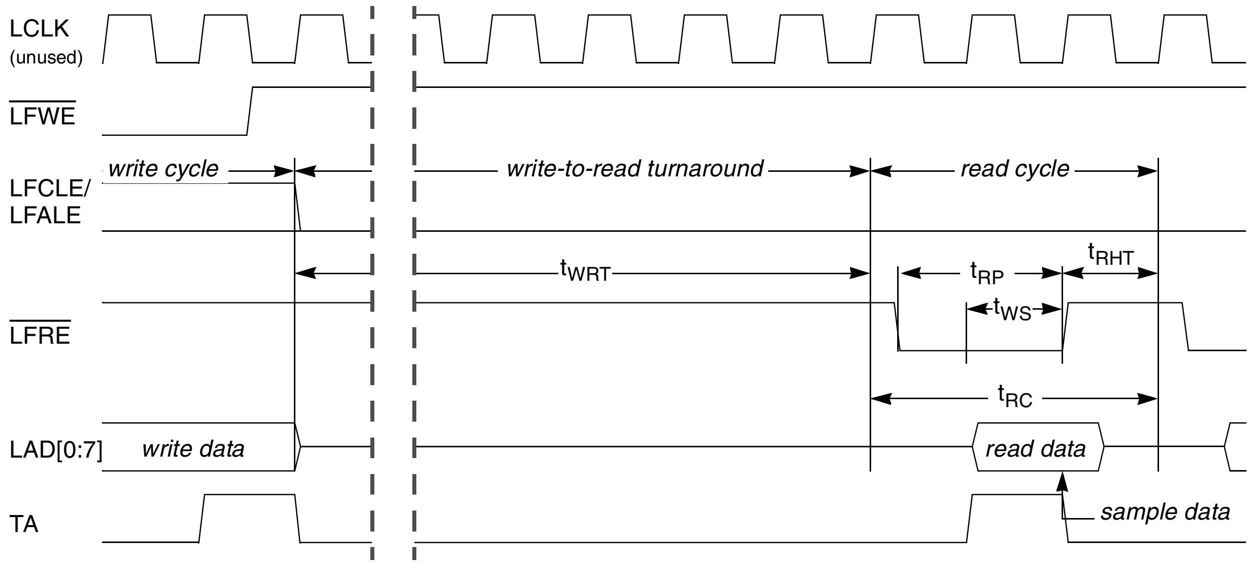


Figure 10-55. FCM Delay Prior to Sampling LFR$\overline{\text{B}}$ State

10.4.3.3.4 FCM Read Data Timing

The timing for read data transfers is shown in Figure 10-56. Upon assertion of $\overline{\text{LFRE}}$, the Flash device will enable its output drivers and drive valid read data while $\overline{\text{LFRE}}$ is held low. FCM samples read data on the rising edge of $\overline{\text{LFRE}}$, which follows an optional number of wait states. Note that FCM will delay the first read if a RBW or RSW instruction is issued, in which case $\overline{LFRB}$ sample timing takes effect (see Section 10.4.3.3.3, "FCM Ready/Busy Timing").



Notes: t_{RP} = $\overline{LFRE}$ pulse time, read period. t_{WS} = Read wait state time.
t_{RHT} = $\overline{LFRE}$ hold time. t_{RC} = Read data cycle time.
t_{WRT} = Write to read turnaround time.

Figure 10-56. FCM Read Data Timing (for TRLX = 0, RST = 0, SCY = 1, CLKDIV = 4*N)

The timing parameters are summarized in Table 10-35.

Table 10-35. FCM Read Data Timing Parameters

Option Register Attributes		Timing Parameter (LCLK Clock Cycles)[1]				
TRLX	RST	t_{RP}	t_{RHT}	t_{WS}	t_{RC}	t_{WRT}
0	0	¾+SCY	1	SCY	2+SCY	4×(2+SCY)
0	1	1+SCY	1	SCY	2+SCY	4×(2+SCY)
1	0	½+2×SCY	2	2×SCY	3+2×SCY	8×(2+SCY)
1	1	1+2×SCY	2	2×SCY	3+2×SCY	8×(2+SCY)

[1] In the parameters, SCY refers to a delay of OR*n*[SCY] clock cycles.

10.4.3.3.5 FCM Extended Read Hold Timing

Allowance for slow output driver turn-off when reading NAND Flash EEPROMs is made by setting OR*n*[EHTR] and OR*n*[TRLX]. The extended read data hold time, shown at t_{EHTR} in Figure 10-45 and Figure 10-57, is a delay inserted by FCM between the last data read and another eLBC memory access (requiring LALE assertion). $\overline{LCSn}$ is negated during t_{EHTR} to allow external devices and bus transceivers time to disable their drivers.

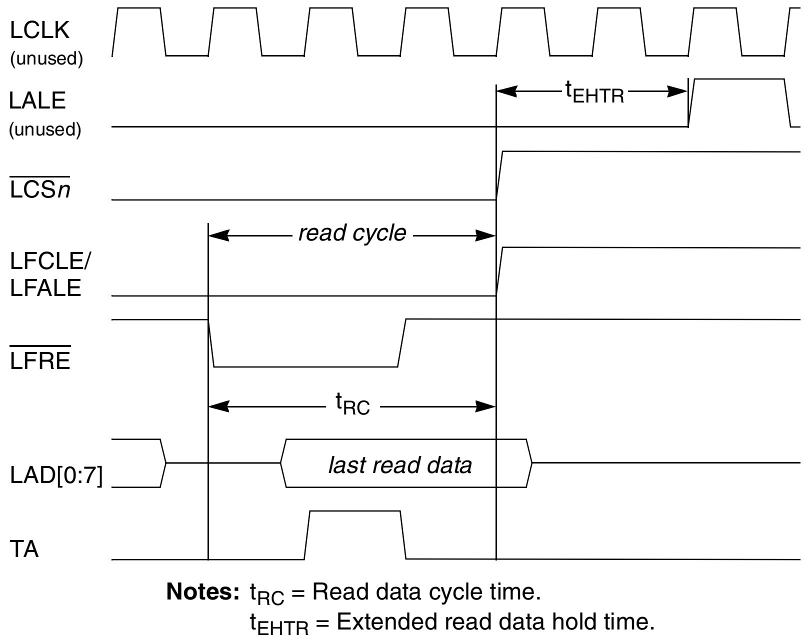


Figure 10-57. FCM Read Data Timing with Extended Hold Time (for TRLX = 0, EHTR = 1, RST = 1, SCY = 1, CLKDIV = 4*N)

10.4.3.4 FCM Boot Chip-Select Operation

Boot chip-select operation allows address decoding for a boot ROM before system initialization. $\overline{LCS0}$ is the boot chip-select output; its operation differs from other external chip-select outputs after a system reset. When the core begins accessing memory after system reset, $\overline{LCS0}$ is asserted initially to load a 4-Kbyte boot block into the FCM buffer RAM, but core instruction fetches occur from the buffer RAM.

10.4.3.4.1 FCM Bank 0 Reset Initialization

The boot chip-select also provides a programmable port size, which is configured during reset. The boot chip-select does not provide write protection. $\overline{LCS0}$ operates this way until the first write to OR0 and it can be used as any other chip-select register after the preferred address range is loaded into BR0. After the first write to OR0, the boot chip-select can be restarted only with a hardware reset. Table 10-36 describes the initial values of the boot bank in the memory controller.

Table 10-36. Boot Bank Field Values after Reset for FCM as Boot Controller

Register	Field	Setting
BR0	BA PS DECC WP MSEL V	0000_0000_0000_0000_0 From RCWH[ROMLOC] and RCWH[RLEXT] 00 0 001 0

Table 10-36. Boot Bank Field Values after Reset for FCM as Boot Controller (continued)

Register	Field	Setting
OR0	AM BCTLD PGS CSCT CST CHT RST SCY TRLX EHTR	0000_0000_0000_0000_0 0 From RCWH[ROMLOC] 1 1 1 1 From por_cfg_scy[1–3] 1 1

10.4.3.4.2 Boot Block Loading into the FCM Buffer RAM

If FCM is selected as the boot ROM controller from power-on-reset configuration, eLBC will automatically load from bank 0 a single 4 Kbyte page of boot code into the FCM buffer RAM during $\overline{\text{HRESET}}$ (See Section 4.3.2.2.4, “Boot ROM Location.”). The CPU can execute boot code directly from the FCM buffer RAM, but must ensure that any further data read from the NAND Flash EEPROM is transferred under software control in order to continue the bootstrap process.

Since OR0[AM] is initially cleared during reset, all CPU fetches to eLBC will access the FCM buffer RAM, which appears in the memory map as a 4-Kbyte RAM. No NAND Flash spare regions are mapped during boot, therefore only 4 Kbytes of contiguous, main region data, loaded from the first pages of the boot block, are accessible in eLBC bank 0, as indicated in Figure 10-58.

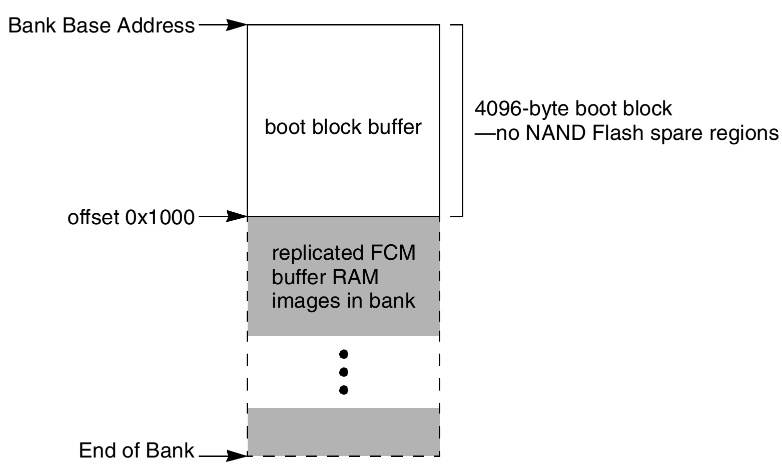


Figure 10-58. FCM Buffer RAM Memory Map During Boot Loading

The process for booting is as follows:

1. Following negation of $\overline{\text{HRESET}}$, eLBC is released from reset and commences automatic boot block loading if FCM is selected as the boot ROM location. Small-page or large-page, 8-bit NAND Flash devices can be used for boot loading when enabled with $\overline{\text{LCS0}}$. eLBC drives $\overline{\text{LFWP}}$ low during boot accesses to prevent accidental erasure of the NAND Flash boot ROM.
2. FCM starts searching for a valid boot block at block index 0.

3. FCM reads the spare regions of the first two pages of the current block, checking the bad block indication (BI) bytes to validate the block for reading. BI bytes must all hold the value 0xFF for the page to be considered readable.
   - For small-page devices, BI is a single byte read from spare region byte offset 5.
   - For large-page devices, BI is a single byte read from spare region byte offset 0.

   If either of the first two pages of the current block are marked invalid, then the boot block index is incremented by 1, and FCM repeats step 3. eLBC will continue searching for a bootable block indefinitely, therefore at least one block must be marked valid for boot loading to proceed. At the conclusion of the boot block search, the value of FBAR[BLK] points to the boot block.
4. If ECC checking is enabled, the FCM recovers from the spare region the stored ECC for each 512-byte block of boot data. The boot block must be prepared with ECC protection. During ECC generation, software should use FMR[ECCM] = 0 for small-page devices, and FMR[ECCM] = 1 for large-page devices.
5. FCM performs a sequence of random-access page reads, reading entire pages from the boot block until 4 Kbytes have been saved to the FCM buffer RAM. If ECC checking is enabled, the ECC of each 512-byte region is verified and single-bit errors are corrected if possible. If FCM is unable to correct ECC errors, eLBC halts the boot process and signals an unrecoverable error by asserting the $\overline{hreset\_req}$ signal.
6. The CPU now commences fetching instructions, in random order, from the FCM buffer RAM. This first-level boot loader typically copies a secondary boot loader into system memory, and continues booting from there. Boot software must clear FMR[BOOT] to enable normal operation of FCM.

10.4.4 User-Programmable Machines (UPMs)

UPMs are flexible interfaces that connect to a wide range of memory devices. At the heart of each UPM is an internal RAM array that specifies the logical value driven on the external memory control signals ($\overline{\text{LCS}n}$, $\overline{\text{LBS}}$[0–3] and LGPL[0–5]) for a given clock cycle. Each word in the RAM array provides bits that allow a memory access to be controlled with a resolution of up to one quarter of the external bus clock period on the byte-select and chip-select lines. A gap of 2 dead LCLK cycles is present on the UPM interface between UPM transactions.

NOTE

If the LGPL4/$\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/LUPWAIT/LPBSE signal is used as both an input and an output, a weak pull-up is required. Refer to the hardware specification for details regarding termination options.

Figure 10-59 shows the basic operation of each UPM.

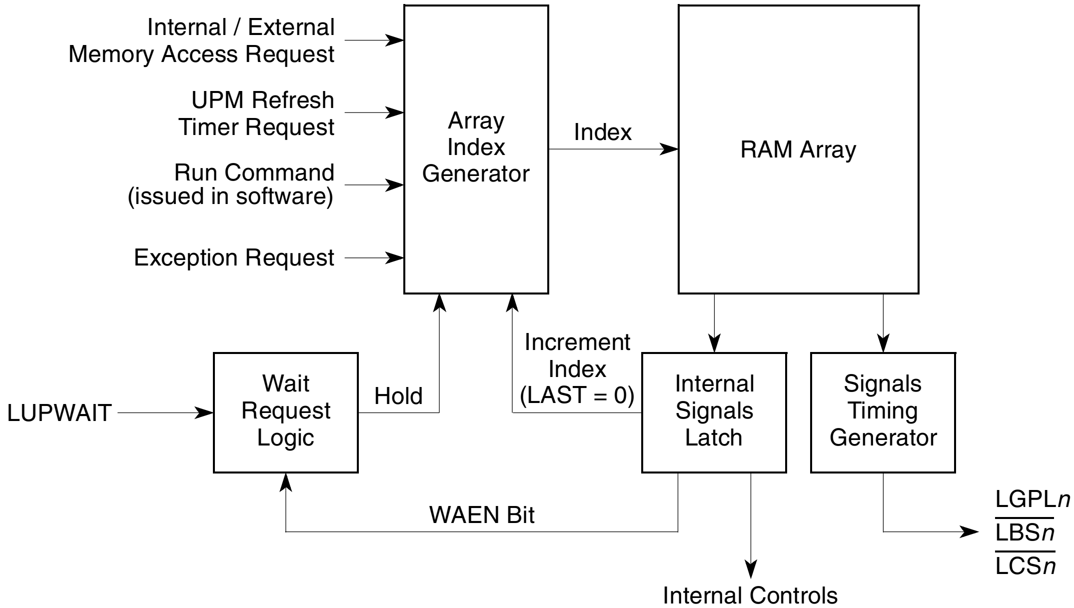


Figure 10-59. User-Programmable Machine Functional Block Diagram

The following events initiate a UPM cycle:

- Any internal device requests an external memory access to an address space mapped to a chip-select serviced by the UPM
- A UPM refresh timer expires and requests a transaction, such as a DRAM refresh
- A bus monitor time-out error during a normal UPM cycle redirects the UPM to execute an exception sequence

The RAM array contains 64 words of 32-bits each. The signal timing generator loads the RAM word from the RAM array to drive the general-purpose lines, byte-selects, and chip-selects. If the UPM reads a RAM word with WAEN set, the external LUPWAIT signal is sampled and synchronized by the memory controller and the current request is frozen.

10.4.4.1 UPM Requests

A special pattern location in the RAM array is associated with each of the possible UPM requests. An internal device's request for a memory access initiates one of the following patterns (M*x*MR[OP] = 00):

- Read single-beat pattern (RSS)
- Read burst cycle pattern (RBS)
- Write single-beat pattern (WSS)
- Write burst cycle pattern (WBS)

A UPM refresh timer request pattern initiates a refresh timer pattern (RTS).

An exception (caused by a bus monitor time-out error) occurring while another UPM pattern is running initiates an exception condition pattern (EXS).

Figure 10-60 and Table 10-37 show the start addresses of these patterns in the UPM RAM, according to cycle type. RUN commands (M*x*MR[OP] = 11), however, can initiate patterns starting at any of the 64 UPM RAM words.

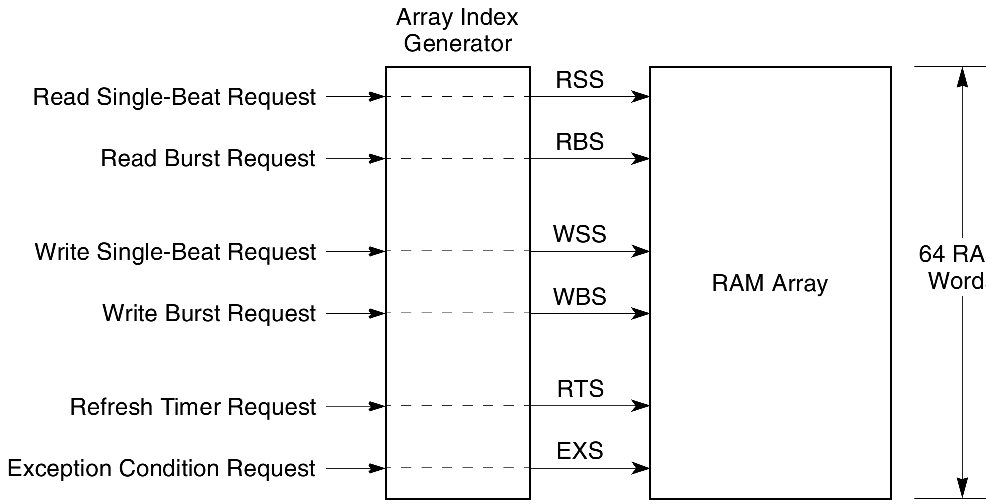


Figure 10-60. RAM Array Indexing

Table 10-37. UPM Routines Start Addresses

UPM Routine	Routine Start Address
Read single-beat (RSS)	0x00
Read burst (RBS)	0x08
Write single-beat (WSS)	0x18
Write burst (WBS)	0x20
Refresh timer (RTS)	0x30
Exception condition (EXS)	0x3C

10.4.4.1.1 Memory Access Requests

The user must ensure that the UPM is appropriately initialized before a request occurs.

The UPM supports two types of memory reads and writes:

- A single-beat transfer transfers one operand consisting of up to a single word (dependent on port size). A single-beat cycle starts with one transfer start and ends with one transfer acknowledge.
- A burst transfer transfers exactly 4 double words regardless of port size. For 32-bit accesses, the burst cycle starts with one transfer start but ends after eight transfer acknowledges, whereas an 8-bit device requires 32 transfer acknowledges.

The user must ensure that patterns for single-beat transfers contain one, and only one, transfer acknowledge (UTA bit in RAM word set high) and for a burst transfer, contain the exact number of transfer acknowledges required.

Any transfers that do not naturally fit single or burst transfers are synthesized as a series of single transfers. These accesses are treated by the UPM as back-to-back, single-beat transfers. Burst transfers can also be inhibited by setting OR*n*[BI]. Burst performance can be achieved by ensuring that UPM transactions are 32-byte aligned with a transaction size being some multiple of 32-bytes, which is a natural fit for cache-line transfers, for example.

10.4.4.1.2 UPM Refresh Timer Requests

Each UPM contains a refresh timer that can be programmed to generate refresh service requests of a particular pattern in the RAM array. Figure 10-61 shows the clock division hardware associated with memory refresh timer request generation. The UPM refresh timer register (LURT) defines the period for the timers associated with all three UPMs.

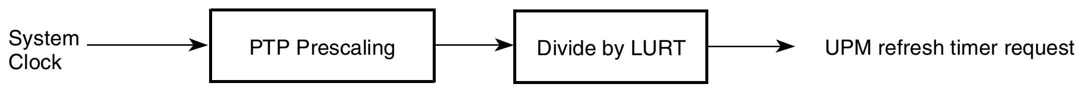


Figure 10-61. Memory Refresh Timer Request Block Diagram

By default, all local bus refreshes are performed using the refresh pattern of UPMA. This means that if refresh is required, MAMR[RFEN] must be set. It also means that only one refresh routine should be programmed and be placed in UPMA, which serves as the refresh executor. Any banks assigned to a UPM are provided with the common UPMA refresh pattern if the RFEN bit of the corresponding UPM is set, concurrently. UPMA assigned banks, therefore, always receive refresh services when MAMR[RFEN] is set, while UPMB and UPMC assigned banks also receive (the same) refresh services if the corresponding MxMR[RFEN] bits are set. In this scenario, more than one chip select may assert at the same time, as refresh pattern runs for all banks assigned to UPM with RFEN bit set.

10.4.4.1.3 Software Requests—RUN Command

Software can start a request to the UPM by issuing a RUN command to the UPM. Some memory devices have their own signal handshaking protocol to put them into special modes, such as self-refresh mode.

For these special cycles, the user creates a special RAM pattern that can be stored in any unused areas in the UPM RAM. Then a RUN command is used to run the cycle. The UPM runs the pattern beginning at the specified RAM location until it encounters a RAM word with its LAST bit set. The RUN command is issued by setting M*x*MR[OP] = 11 and accessing UPM*n* memory region with any write transaction that hits the corresponding UPM machine. M*x*MR[MAD] determines the starting address in the RAM array for the pattern.

Note that transfer acknowledges (UTA bit in the RAM word) are ignored for software (RUN command) requests, and hence the LAD signals remain high-impedance unless the normal initial LALE occurs or the RUN pattern causes assertion of LALE to occur on changes to the RAM word AMX field.

10.4.4.1.4 Exception Requests

When the eLBC under UPM control initiates an access to a memory device and an exception occurs (bus monitor time-out), the UPM provides a mechanism by which memory control signals can meet the device's timing requirements without losing data. The mechanism is the exception pattern that defines how the UPM negates its signals in a controlled manner.

10.4.4.2 Programming the UPMs

The UPM is a micro sequencer that requires microinstructions or RAM words to generate signal timings for different memory cycles. Follow these steps to program the UPMs:

1. Set up BR*n* and OR*n* registers.
2. Write patterns into the RAM array.
3. Program MRTPR, LURT and MAMR[RFEN] if refresh is required.
4. Program M*x*MR.

Patterns are written to the RAM array by setting M*x*MR[OP] = 01 and accessing the UPM with any write transaction that hits the relevant chip select. The entire array is thus programmed by an alternating series of writes: to MDR (RAM word to be written) each time followed by a read from MDR and then followed by a (dummy) write transaction to the relevant UPM assigned bank. A read from MDR is required to ensure that the MDR update has occurred prior to the (dummy) write transaction.

RAM array contents may also be read for debug purposes, for example, by alternating dummy read transactions, each time followed by reads of MDR (when M*x*MR[OP] = 10).

NOTE

M*x*MR / MDR registers should not be updated while dummy read/write access is still in progress. If the MxMR[MAD] is incremented then the previous dummy transaction is already completed.

In order to enforce proper ordering between updates to the M*x*MR/MDR register and the dummy accesses to the UPM memory region, two rules must be followed:

1. Since the result of any update to the M*x*MR/MDR register must be in effect before the dummy read or write to the UPM region, a write to M*x*MR/MDR should be followed immediately by a read of M*x*MR/MDR.
2. The UPM memory region should have the same MMU settings as the memory region containing the M*x*MR configuration register; both should be mapped by the MMU as cache-inhibited and guarded. This prevents the CPU from re-ordering a read of the UPM memory around the read of M*x*MR. Once the programming of the UPM array is complete the MMU setting for the associated address range can be set to the proper mode for normal operation, such as cacheable and copyback.

For proper signalling, the following guidelines must be followed while programming UPM RAM words:

- For UPM reads, program UTA and LAST in the same or consecutive RAM words.
- For UPM burst reads, program last UTA and LAST in the same or consecutive RAM words.
- For UPM writes, program UTA and LAST in the same RAM word.
- For UPM burst writes, program last UTA and LAST in the same RAM word.

10.4.4.2.1 UPM Programming Example (Two Sequential Writes to the RAM Array)

The following example further illustrates the steps required to perform two writes to the RAM array at non-sequential addresses assuming that the relevant BR*n* and OR*n* registers have been previously set up:

1. Program M*x*MR for the first write (with the desired RAM array address).
2. Write pattern/data to MDR to ensure that the MxMR has already been updated with the desired configuration.
3. Read MDR to ensure that the MDR has already been updated with the desired pattern. (Or, read MxMR register if step 2 is not performed.)
4. Perform a dummy write transaction.
5. Read/check M*x*MR[MAD]. If incremented, the previous dummy write transaction is completed; proceed to step 6. Repeat step 5 until incremented.
6. Program M*x*MR for the second write with the desired RAM array address.
7. Write pattern/data to MDR to ensure that the M*x*MR has already been updated with the desired configuration.
8. Read MDR to ensure that the MDR has already been updated with the desired pattern.
9. Perform a dummy write transaction.
10. Read/check M*x*MR[MAD]. If incremented, the previous dummy write transaction is completed.

Note that if step 1 (or 6) and 2 (or 7) are reversed, step 3 (or 8) is replaced by the following:

- Read M*x*MR to ensure that the M*x*MR has already been updated with the desired configuration.

10.4.4.2.2 UPM Programming Example (Two Sequential Reads from the RAM Array)

RAM array contents may also be read for debug purposes, for example, by alternating dummy read transactions, each time followed by reads of MDR (M*x*MR[OP] = 0b10). The following example further illustrates the steps required to perform two reads from the RAM array at non-sequential addresses assuming that the relevant BR*n* and OR*n* registers have been previously set up:

1. Program M*x*MR for the first read with the desired RAM array address.
2. Read M*x*MR to ensure that the M*x*MR has already been updated with the desired configuration, such as RAM array address.
3. Perform a dummy read transaction.
4. Read/check M*x*MR[MAD]. If incremented, the previous dummy read transaction is completed; proceed to step 5. Repeat step 4 until incremented.
5. Read MDR.
6. Program M*x*MR for the second read with the desired RAM array address.
7. Read M*x*MR to ensure that the M*x*MR has already been updated with the desired configuration, such as RAM array address.
8. Perform a dummy read transaction.
9. Read/check M*x*MR[MAD]. If incremented, the previous dummy read transaction is completed; proceed to step 10. Repeat step 9 until incremented.
10. Read MDR.

10.4.4.3 UPM Signal Timing

RAM word fields specify the value of the various external signals at a granularity of up to four values for each bus clock cycle. The signal timing generator causes external signals to behave according to timing specified in the current RAM word. For LCRR[CLKDIV] = 4 or 8, each bit in the RAM word relating to $\overline{\text{LCS}n}$ and $\overline{\text{LBS}}$ timing specifies the value of the corresponding external signal at each quarter phase of the bus clock. If LCRR[CLKDIV] = 2, the external signal can change value only on each half phase of the bus clock. If the RAM word in this case (LCRR[CLKDIV] = 2) specifies a quarter phase signal change, the signal timing generator interprets this as a half cycle change.

The division of UPM bus cycles into phases is shown in Figure 10-62 and Figure 10-63. If LCRR[CLKDIV] = 2, the bus cycle comprises only two active phases, T1 and T3, which correspond with the first and second halves of the bus clock cycle, respectively. However, if LCRR[CLKDIV] = 4 or 8, four phases, T1–T4, define four quarters of the bus clock cycle. Because T2 and T4 are inactive when LCRR[CLKDIV] = 2, UPM ignores signal timing programmed for assertion in either of these phases in the case LCRR[CLKDIV] = 2.

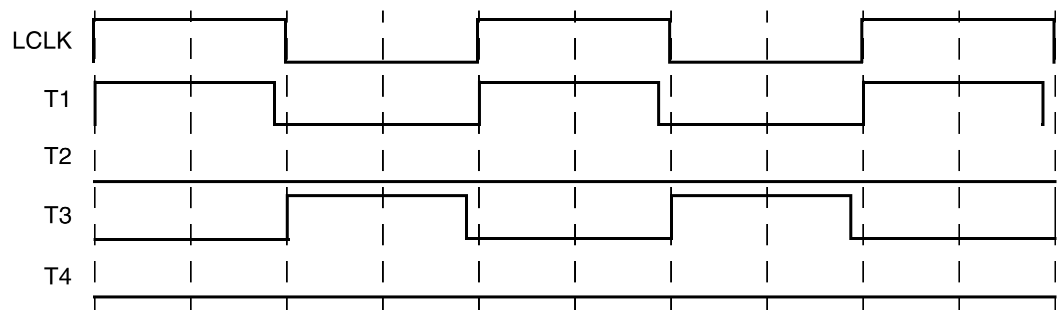


Figure 10-62. UPM Clock Scheme for LCRR[CLKDIV] = 2

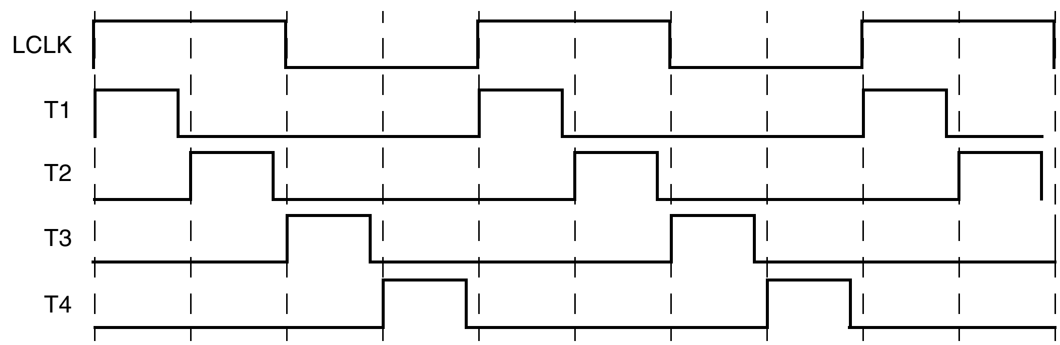


Figure 10-63. UPM Clock Scheme for LCRR[CLKDIV] = 4 or 8

10.4.4.4 RAM Array

The RAM array for each UPM is 64 locations deep and 32 bits wide, as shown in Figure 10-64. The signals at the bottom of the figure are UPM outputs. The selected $\overline{\text{LCS}n}$ is for the bank that matches the current address. The selected $\overline{\text{LBS}}$ is for the byte lanes read or written by the access.

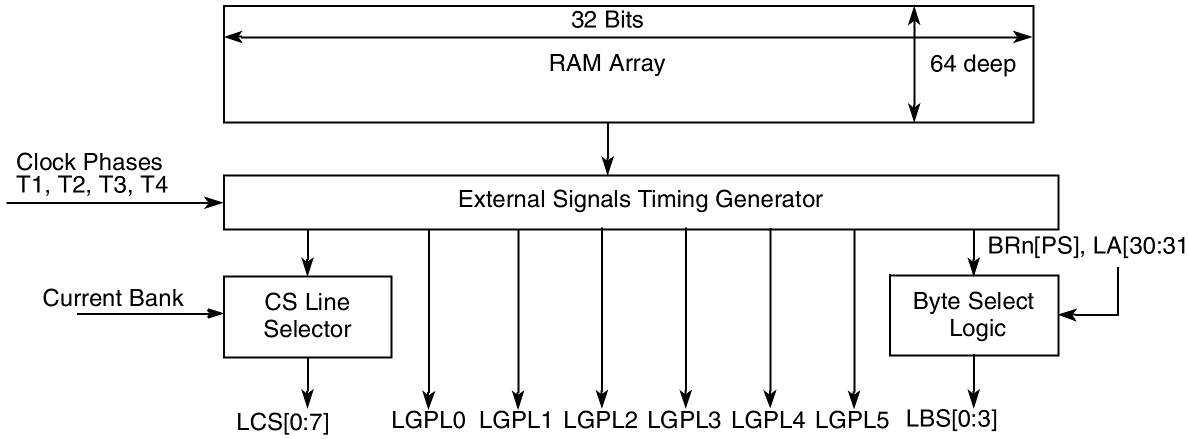


Figure 10-64. RAM Array and Signal Generation

10.4.4.4.1 RAM Words

The RAM word is a 32-bit microinstruction stored in one of 64 locations in the RAM array. It specifies timing for external signals controlled by the UPM. Figure 10-37 shows the RAM word fields. When LCRR[CLKDIV] = 4 or 8, the CST*n* and BST*n* bits determine the state of UPM signals $\overline{\text{LCS}n}$ and $\overline{\text{LBS}}$[0–3] at each quarter phase of the bus clock. When LCRR[CLKDIV] = 2, CST2 and CST4 are ignored and the external has the values defined by CST1 and CST3 but extended to half the clock cycle in duration. The same interpretation occurs for the BST*n* bits when LCRR[CLKDIV] = 2.

	0	1	2	3	4	5	6	7	8–9	10–11	12	13	14	15
R/W	CST1	CST2	CST3	CST4	BST1	BST2	BST3	BST4	G0L	G0H	G1T1	G1T3	G2T1	G2T3
Reset	All zeros													

	16	17	18	19	20	21	22–23	24	25	26–27	28	29	30	31
R/W	G3T1	G3T3	G4T1/ DLT3	G4T3/ WAEN	G5T1	G5T3	REDO	LOOP	EXEN	AMX	NA	UTA	TODT	LAST
Reset	All zeros													

Figure 10-65. RAM Word Fields

Table 10-38 contains descriptions of the RAM word fields.

Table 10-38. RAM Word Field Descriptions

Bits	Name	Description
0	CST1	Chip select timing 1. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock quarter phase 1 if LCRR[CLKDIV] = 4 or 8. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock half phase 1 if LCRR[CLKDIV] = 2.
1	CST2	Chip select timing 2. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock quarter phase 2 if LCRR[CLKDIV] = 4 or 8. Ignored when LCRR[CLKDIV] = 2.
2	CST3	Chip select timing 3. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock quarter phase 3 if LCRR[CLKDIV] = 4 or 8. Defines the state (0 or 1) of $\overline{\text{LCS}n}$ during bus clock half phase 2 if LCRR[CLKDIV] = 2.

Table 10-38. RAM Word Field Descriptions (continued)

Bits	Name	Description
3	CST4	Chip select timing 4. Defines the state (0 or 1) of $\overline{LCSn}$ during bus clock quarter phase 4 if LCRR[CLKDIV] = 4 or 8. Ignored when LCRR[CLKDIV] = 2.
4	BST1	Byte select timing1. Defines the state (0 or 1) of $\overline{LBS}$ during bus clock quarter phase 1 (LCRR[CLKDIV] = 4 or 8) or bus clock half phase 1 (LCRR[CLKDIV] = 2), in conjunction with BR*n*[PS] and LA[30–31].
5	BST2	Byte select timing 2. Defines the state (0 or 1) of $\overline{LBS}$ during bus clock quarter phase 2 (LCRR[CLKDIV] = 4 or 8), in conjunction with BR*n*[PS] and LA[30–31]. Ignored when LCRR[CLKDIV] = 2.
6	BST3	Byte select timing 3. Defines the state (0 or 1) of $\overline{LBS}$ during bus clock quarter phase 3 (LCRR[CLKDIV] = 4 or 8) or bus clock half phase 2 (LCRR[CLKDIV] = 2), in conjunction with BR*n*[PS] and LA[30–31].
7	BST4	Byte select timing 4. Defines the state (0 or 1) of $\overline{LBS}$ during bus clock quarter phase 4 (LCRR[CLKDIV] = 4 or 8), in conjunction with BR*n*[PS] and LA[30–31]. Ignored when LCRR[CLKDIV] = 2.
8–9	G0L	General purpose line 0 lower. Defines the state of LGPL0 during the bus clock quarter phases 1 and 2 (first half phase). 00 Value defined by M*x*MR[G0CL] 01 Reserved 10 0 11 1
10–11	G0H	General purpose line 0 higher. Defines the state of LGPL0 during the bus clock quarter phases 3 and 4 (second half phase). 00 Value defined by M*x*MR[G0CL] 01 Reserved 10 0 11 1
12	G1T1	General purpose line 1 timing 1. Defines the state (0 or 1) of LGPL1 during bus clock quarter phases 1 and 2 (first half phase).
13	G1T3	General purpose line 1 timing 3. Defines the state (0 or 1) of LGPL1 during bus clock quarter phases 3 and 4 (second half phase)
14	G2T1	General purpose line 2 timing 1. Defines state (0 or 1) of LGPL2 during bus clock quarter phases 1 and 2 (first half phase).
15	G2T3	General purpose line 2 timing 3. Defines the state (0 or 1) of LGPL2 during bus clock quarter phases 3 and 4 (second half phase).
16	G3T1	General purpose line 3 timing 1. Defines the state (0 or 1) of LGPL3 during bus clock quarter phases 1 and 2 (first half phase).
17	G3T3	General purpose line 3 timing 3. Defines the state (0 or 1) of LGPL3 during bus clock quarter phases 3 and 4 (second half phase).

Table 10-38. RAM Word Field Descriptions (continued)

Bits	Name	Description
18	G4T1/DLT3	General purpose line 4 timing 1/delay time 3. The function of this bit is determined by M*x*MR[GPL4]. If M*x*MR[GPL4] = 0 and LGPL4/LUPWAIT pin functions as an output (LGPL4), G4T1/DLT3 defines the state (0 or 1) of LGPL4 during bus clock quarter phases 1 and 2 (first half phase). If M*x*MR[GPL4] = 1 and LGPL4/LUPWAIT functions as an input (LUPWAIT), if a read burst or single read is executed, G4T1/DLT3 defines the sampling of the data bus as follows: 0 In the current word, the data bus should be sampled at the start of bus clock quarter phase 1 of the next bus clock cycle. 1 In the current word, the data bus should be sampled at the start of bus clock quarter phase 3 of the current bus clock cycle.
19	G4T3/WAEN	General purpose line 4 timing 3/wait enable. Bit function is determined by M*x*MR[GPL4]. If M*x*MR[GPL4] = 0 and LGPL4/LUPWAIT pin functions as an output (LGPL4), G4T3/WAEN defines the state (0 or 1) of LGPL4 during bus clock quarter phases 3 and 4 (second half phase). If M*x*MR[GPL4] = 1 and LGPL4/LUPWAIT functions as an input (LUPWAIT), G4T3/WAEN is used to enable the wait mechanism: 0 LUPWAIT detection is disabled. 1 LUPWAIT is enabled. If LUPWAIT is detected as being asserted, a freeze in the external signals logical values occurs until LUPWAIT is detected as being negated.
20	G5T1	General purpose line 5 timing 1. Defines the state (0 or 1) of LGPL5 during bus clock quarter phases 1 and 2 (first half phase).
21	G5T3	General purpose line 5 timing 3. Defines the state (0 or 1) of LGPL5 during bus clock quarter phases 3 and 4 (second half phase).
22–23	REDO	Redo current RAM word. Defines the number of times to execute the current RAM word. 00 Once (normal operation) 01 Twice 10 Three times 11 Four times
24	LOOP	Loop start/end. The first RAM word in the RAM array where LOOP is 1 is recognized as the loop start word. The next RAM word where LOOP is 1 is the loop end word. RAM words between, and including the start and end words, are defined as part of the loop. The number of times the UPM executes this loop is defined in the corresponding loop fields of the M*x*MR. 0 The current RAM word is not the loop start word or loop end word. 1 The current RAM word is the start or end of a loop. **Note:** AMX must not change values in any RAM word which begins a loop
25	EXEN	Exception enable. Allows branching to an exception pattern at the exception start address (EXS). When an internal bus monitor time-out exception is recognized and EXEN in the RAM word is set, the UPM branches to the special exception start address (EXS) and begins operating as the pattern defined there specifies. The user should provide an exception pattern to negate signals controlled by the UPM in a controlled fashion. For DRAM control, a handler should negate RAS and CAS to prevent data corruption. If EXEN = 0, exceptions are ignored by UPM (but not by local bus) and execution continues. After the UPM branches to the exception start address, it continues reading until the LAST bit is set in the RAM word. 0 The UPM continues executing the remaining RAM words, ignoring any internal bus monitor time-out. 1 The current RAM word allows a branch to the exception pattern after the current cycle if an exception condition is detected.

Table 10-38. RAM Word Field Descriptions (continued)

Bits	Name	Description
26–27	AMX	Address multiplexing. Determines the source of LAD during an LALE phase. Any change in the AMX field initiates a new LALE (address) phase. 00 LAD (and/or in conjunction with LA) is the non-multiplexed address. For example, column address. 01 Reserved 10 LAD (and/or in conjunction with LA) is driven with the multiplexed address according to M*x*MR[AM]. For example, row address. See Section 10.4.4.4.7, "Address Multiplexing (AMX)" for more information. 11 LAD (and/or in conjunction with LA) is driven with the contents of MAR. Used, for example, to initialize a mode. **Note:** AMX must not change values in any RAM word which begins a loop. **Note:** Source ID debug mode is only supported for the AMX = 00 setting.
28	NA	Next burst address. Determines when the address is incremented during a burst access. 0 The address increment function is disabled. 1 The address is incremented in the next cycle. In conjunction with the BR*n*[PS], the increment value of LA*n* is 1, 2, or 4 for port sizes of 8 bits, 16 bits, and 32 bits, respectively.
29	UTA	UPM transfer acknowledge. Indicates assertion of transfer acknowledge in the current cycle. 0 Transfer acknowledge is not asserted in the current cycle. 1 Transfer acknowledge is asserted in the current cycle. In case of UPM writes, program UTA and LAST in same RAM word. In case of UPM reads, program UTA and LAST in consecutive or same RAM words.
30	TODT	Turn-on disable timer. The disable timer associated with each UPM allows a minimum time to be guaranteed between two successive accesses to the same memory bank. This feature is critical when DRAM requires a RAS precharge time. TODT turns the timer on to prevent another UPM access to the same bank until the timer expires.The disable timer period is determined in M*x*MR[DS*n*]. The disable timer does not affect memory accesses to different banks. Note that TODT must be set together with LAST, otherwise it is ignored. 0 The disable timer is turned off. 1 The disable timer for the current bank is activated preventing a new access to the same bank (when controlled by the UPMs) until the disable timer expires. For example, precharge time.
31	LAST	Last word. When LAST is read in a RAM word, the current UPM pattern terminates and control signal timing set in the RAM word is applied to the current (and last) cycle. However, if the disable timer is activated and the next access is to the same bank, execution of the next UPM pattern is held off and the control signal values specified in the last word are extended in duration for the number of clock cycles specified in M*x*MR[DS*n*]. 0 The UPM continues executing RAM words. 1 Indicates the last RAM word in the program. The service to the UPM request is done after this cycle concludes. In case of UPM writes, program UTA and LAST in same RAM word. In case of UPM reads, program UTA and LAST in consecutive or same RAM words.

10.4.4.4.2 Chip-Select Signal Timing (CST*n*)

If BR*n*[MSEL] of the accessed bank selects a UPM on the currently requested cycle, the UPM manipulates the $\overline{\text{LCS}n}$ for that bank with timing as specified in the UPM RAM word CST*n* fields. The selected UPM affects only the assertion and negation of the appropriate $\overline{\text{LCS}n}$ signal. The state of the selected $\overline{\text{LCS}n}$

signal of the corresponding bank depends on the value of each CST*n* bit. Figure 10-66 shows how UPMs control $\overline{\mathrm{LCS}n}$ signals.

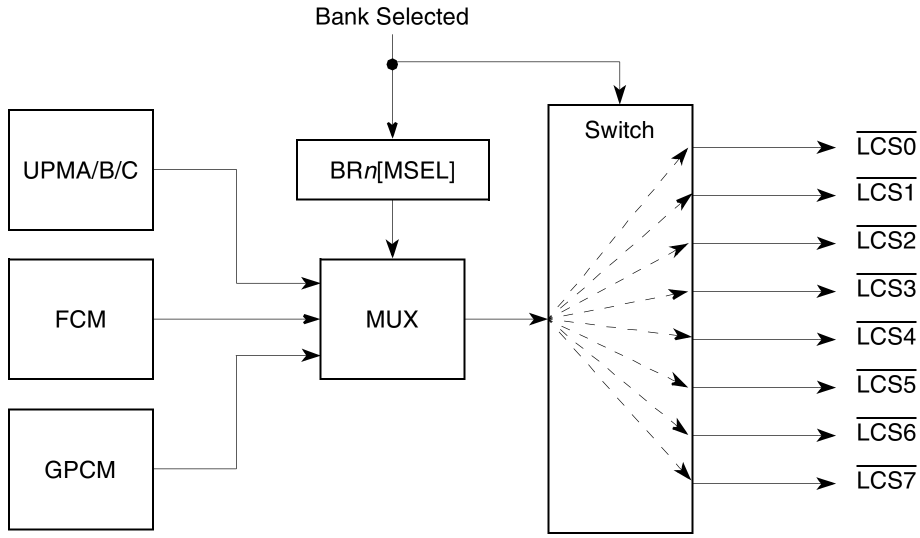


Figure 10-66. $\overline{\mathrm{LCS}n}$ Signal Selection

10.4.4.4.3 Byte Select Signal Timing (BST*n*)

If BR*n*[MSEL] of the accessed memory bank selects a UPM on the currently requested cycle, the selected UPM affects the assertion and negation of the appropriate $\overline{\mathrm{LBS}}$[0–3] signal. The timing of all four byte-select signals is specified in the RAM word. However, $\overline{\mathrm{LBS}}$[0–3] are also controlled by the port size of the accessed bank, the number of bytes to transfer, and the address accessed. Figure 10-67 shows how UPMs control $\overline{\mathrm{LBS}}$[0–3].

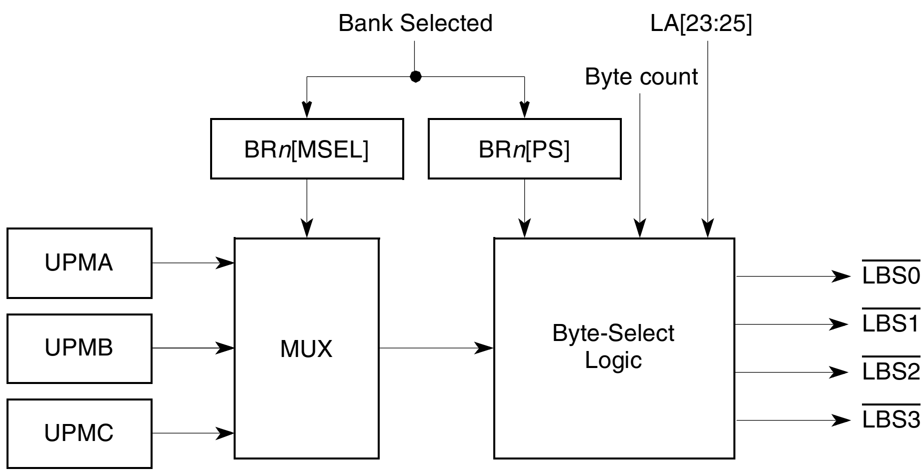


Figure 10-67. $\overline{\mathrm{LBS}}$ Signal Selection

The uppermost byte select ($\overline{\mathrm{LBS0}}$), when asserted, indicates that LAD[0–7] contains valid data during a cycle. Likewise, $\overline{\mathrm{LBS1}}$ indicates that LAD[8–15] contain valid data, $\overline{\mathrm{LBS2}}$ indicates that LAD[16–23] contains valid data, and $\overline{LBS3}$ indicates that LAD[24–31] contain valid data. For a UPM refresh timer request, all $\overline{LBS}$[0–3] signals are asserted/negated by the UPM according to the refresh pattern only. Following any internal bus monitor exception, the $\overline{LBS}$[0–3] signals are negated regardless of the exception handling provided by any UPM exception pattern to prevent spurious writes to external RAM.

10.4.4.4.4 General-Purpose Signals (G*n*T*n*, GO*n*)

The general-purpose signals (LGPL[0–5]) each have two bits in the RAM word that define the logical value of the signal to be changed at the rising edge of the bus clock and/or at the falling edge of the bus clock. LGPL0 offers enhancements beyond the other LGPL*n* lines.

LGPL0 can be controlled by an address line specified in M*x*MR[G0CL]. To use this feature, G0H and G0L should be set in the RAM word. For example, for a SIMM with multiple banks, this address line can be used to switch between internal memory device banks.

10.4.4.4.5 Loop Control (LOOP)

The LOOP bit in the RAM word specifies the beginning and end of a set of UPM RAM words that are to be repeated. The first time LOOP = 1, the memory controller recognizes it as a loop start word and loads the memory loop counter with the corresponding contents of the loop field shown in Table 10-39. The next RAM word for which LOOP = 1 is recognized as a loop end word. When it is reached, the loop counter is decremented by one.

Continued loop execution depends on the loop counter. If the counter is not zero, the next RAM word executed is the loop start word. Otherwise, the next RAM word executed is the one after the loop end word. Loops can be executed sequentially but cannot be nested. Also, special care must be taken:

- LAST and LOOP must not be set together.
- Loop start word should not have an AMX change with regard to the previous word.

Table 10-39. M*x*MR Loop Field Use

Request Serviced	Loop Field
Read single-beat cycle	RLF
Read burst cycle	RLF
Write single-beat cycle	WLF
Write burst cycle	WLF
Refresh timer expired	TLF
RUN command	RLF

10.4.4.4.6 Repeat Execution of Current RAM Word (REDO)

The REDO function is useful for wait-state insertion in a long UPM routine that would otherwise need too many RAM words. Setting the REDO bits of the RAM word to a nonzero value causes the UPM to re-execute the current RAM word up to three more times, as defined in the REDO field of the current RAM word.

Special care must be taken in the following cases:

- When UTA and REDO are set together, TA is asserted the number of times specified by the REDO function.
- When NA and REDO are set together, the address is incremented the number of times specified by the REDO function.
- When LOOP and REDO are set together, the loop mechanism works as usual and the line is repeated according to the REDO function.
- LAST and REDO must not be set together.
- REDO should not be used within the exception routine.

10.4.4.4.7 Address Multiplexing (AMX)

Address lines can be controlled by the user-provided pattern in the UPM. The address multiplex (AMX) bits in the RAM word can choose between driving the transaction address (AMX = 00), driving it according to the multiplexing specified by the M*x*MR[AM] field (AMX = 10), or driving the contents of MAR (AMX = 11) on the address signals. The next address (NA) bit of the RAM word does not affect LA signals, unless AMX = 00 and chooses the column address for NA = 1.

In all cases, LA[27–31] of the eLBC are driven by the five lsbs of the address selected by AMX, regardless of whether the next address (NA) bit of the RAM word is used to increment the current address. The effect of NA = 1 is visible only when AMX = 00 chooses the column address.

Table 10-40 shows how the RAM word AMX bits and M*x*MR[AM] settings can be used to affect row × column address multiplexing on the LA[16–31] signals. When AMX = 10, LAD[0–15] are driven low during an address phase.

Table 10-40. UPM Address Multiplexing

	msb	Internal Transaction Address																													lsb
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
AMX = 10 MxMR[AM] = 000 (Row)									16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31								
AMX = 00 (Col)																									24	25	26	27	28	29	30	31
AMX = 10 MxMR[AM] = 001 (Row)								16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31									
AMX = 00 (Col)																								23	24	25	26	27	28	29	30	31
AMX = 10 MxMR[AM] = 010 (Row)							16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31										
AMX = 00 (Col)																							22	23	24	25	26	27	28	29	30	31
AMX = 10 MxMR[AM] = 011 (Row)						16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31											
AMX = 00 (Col)																						21	22	23	24	25	26	27	28	29	30	31
AMX = 10 MxMR[AM] = 100 (Row)					16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31												

Table 10-40. UPM Address Multiplexing (continued)

	msb	Internal Transaction Address																													lsb
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
AMX = 00 (Col)																					20	21	22	23	24	25	26	27	28	29	30	31
AMX = 10 MxMR[AM] = 101 (Row)				16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31													
AMX = 00 (Col)																				19	20	21	22	23	24	25	26	27	28	29	30	31
AMX = 10 MxMR[AM] = 110	Reserved																															
AMX = 10 MxMR[AM] = 111	Reserved																															

Note that any change to the AMX field from one RAM word to the next RAM word executed results in an address phase on the {LAD*n*, LA*n*} bus with the assertion of LALE for the number of cycles set for LALE in the OR*n* and LCRR registers. The LGPL[0–5] signals maintain the value specified in the RAM word during the LALE phase.

NOTE

AMX must not change values in any RAM word which begins a loop.

10.4.4.4.8 Data Valid and Data Sample Control (UTA)

When a read access is handled by the UPM, and the UTA bit is 1 (data is to be sampled by the eLBC), the value of the DLT3 bit in the same RAM word, in conjunction with M*x*MR[GPL4], determines when the data input is sampled by the eLBC as follows:

- If M*x*MR[GPL4] = 1 (G4T4/DLT3 functions as DLT3) and DLT3 = 1 in the RAM word, data is latched on the falling edge of the bus clock instead of the rising edge. The eLBC samples the data on the next falling edge of the bus clock, which is during the middle of the current bus cycle. This feature should be used only in systems without external synchronous bus devices that require mid-cycle sampling.
- If M*x*MR[GPL4] = 0 (G4T4/DLT3 functions as G4T4), or if M*x*MR[GPL4] = 1 but DLT3 = 0 in the RAM word, data is latched on the rising edge of the bus clock, which occurs at the end of the current bus clock cycle (normal operation).

Figure 10-68 shows how data sampling is controlled by the UPM.

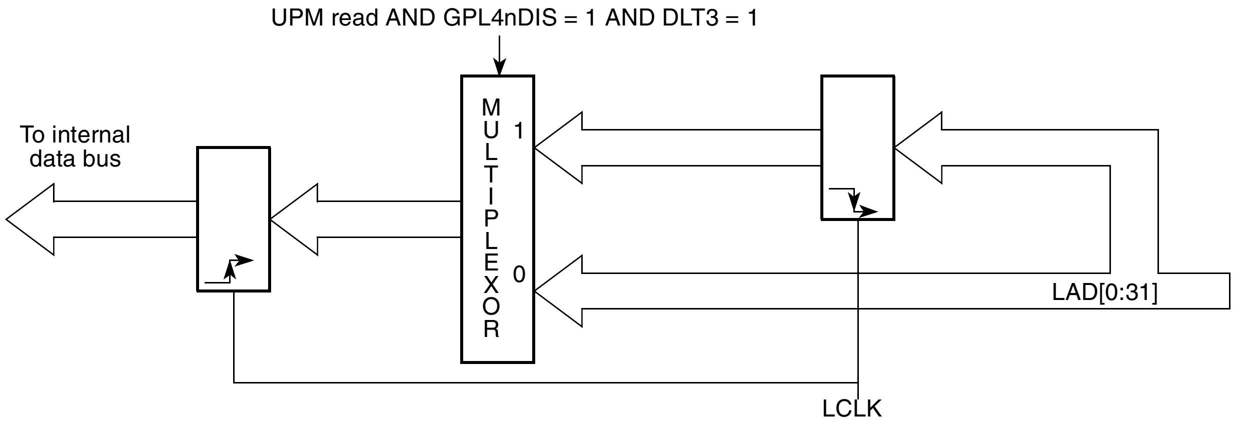


Figure 10-68. UPM Read Access Data Sampling

10.4.4.4.9 LGPL[0–5] Signal Negation (LAST)

When the LAST bit is read in a RAM word, the current UPM pattern is terminated at the end of the current cycle. On the next cycle (following LAST) all the UPM signals are negated unconditionally (driven to logic 1), unless there is a back-to-back UPM request pending. In this case, the signal values for the cycle following the one in which the LAST bit was set are taken from the first RAM word of the pending UPM routine.

In case of UPM writes, program UTA and LAST in same RAM word. In case of UPM reads, program UTA and LAST in consecutive or same RAM words.

10.4.4.4.10 Wait Mechanism (WAEN)

The WAEN bit in the RAM array word can be used to enable the UPM wait mechanism in selected UPM RAM words. If the UPM reads a RAM word with WAEN set, the external LUPWAIT signal is sampled and synchronized by the memory controller as if it were an asynchronous signal. The WAEN bit is ignored if LAST = 1 in the same RAM word.

Synchronization of LUPWAIT starts at the rising edge of the bus clock and takes at least 1 bus cycle to complete. If LUPWAIT is asserted and WAEN = 1 in the current UPM word, the UPM is frozen until LUPWAIT is negated. The value of external signals driven by the UPM remains as indicated in the previous RAM word. When LUPWAIT is negated, the UPM continues normal functions. Note that during WAIT cycles, the UPM does not handle data.

Figure 10-69 shows how the WAEN bit in the word read by the UPM and the LUPWAIT signal are used to hold the UPM in a particular state until LUPWAIT is negated. As the example shows, the $\overline{\text{LCS}n}$ and LGPL1 states and the WAEN value are frozen until LUPWAIT is recognized as negated. WAEN is typically set before the line that contains UTA = 1. Note that if WAEN and NA are both set in the same

RAM word, NA causes the burst address to increment once as normal regardless of whether the UPM freezes.

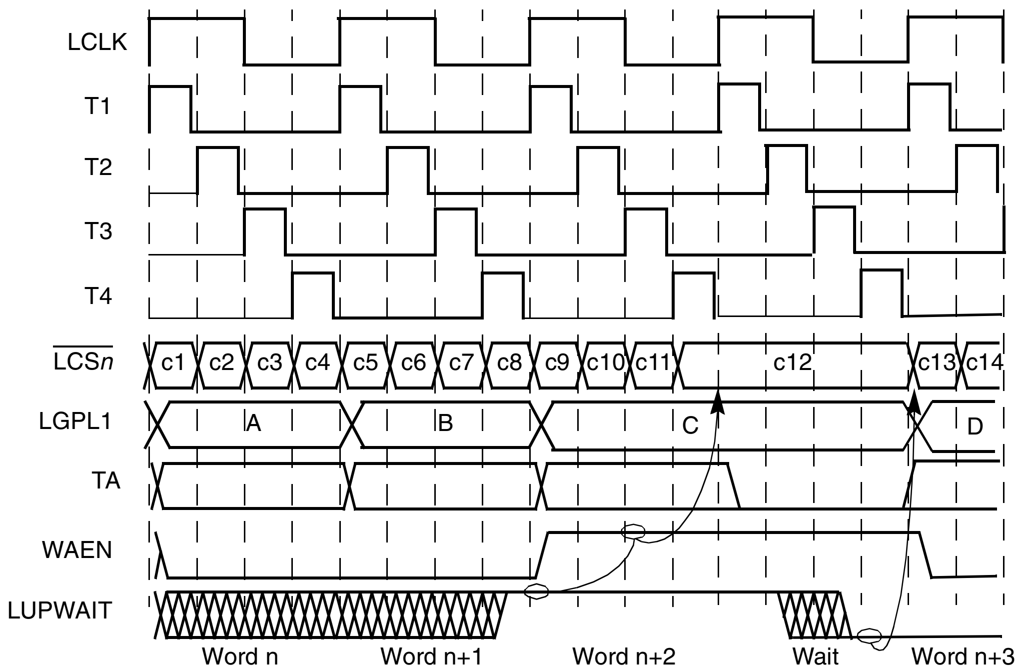


Figure 10-69. Effect of LUPWAIT Signal

10.4.4.5 Synchronous Sampling of LUPWAIT for Early Transfer Acknowledge

If LUPWAIT is to be considered an asynchronous signal, which can be asserted/negated at any time, no UPM RAM word must contain both WAEN = 1 and UTA = 1 simultaneously.

However, programming WAEN = 1 and UTA = 1 in the same RAM word, under certain conditions, allows the UPM to treat LUPWAIT as a synchronous signal, which must meet set-up and hold times in relation to the rising edge of the bus clock. The conditions are as follows:

- The PLL must be enabled, that is, LCRR[PBYP] = 0.
- DLT3 bit must be cleared in the same RAM word to avoid mid-sampling of read data.
- LBCR[LPBSE] = 0 and MXMR[GPL4] = 1
- The combination WAEN=1 and UTA=1 should be in the RAM word next to the word which gets frozen by LUPWAIT assertion. This condition limits the use of this mode to cases where the exact cycle of LUPWAIT assertion is predictable.

In this mode, as soon as UPM samples LUPWAIT negated on the rising edge of the bus clock, it immediately generates an internal transfer acknowledge, which allows a data transfer one bus clock cycle later. The generation of transfer acknowledge is early because LUPWAIT is not re-synchronized. The acknowledge occurs early or normally depending on whether the UPM was already frozen inWAIT cycles or not. This feature allows the synchronous negation of LUPWAIT to affect a data transfer, even if UTA, WAEN, and LAST are set simultaneously.

10.4.4.6 Extended Hold Time on Read Accesses

Slow memory devices that take a long time to turn off their data bus drivers on read accesses should choose some non-zero combination of OR*n*[TRLX] and OR*n*[EHTR]. The next accesses after a read access to the slow memory device is delayed by the number of clock cycles specified in the OR*n* register in addition to any existing bus turnaround cycle.

10.5 Initialization/Application Information

10.5.1 Interfacing to Peripherals in Different Address Modes

This section provides guidelines for interfacing to peripherals.

10.5.1.1 Multiplexed Address/Data Bus for 32-Bit Addressing

In order to reduce pins on the local bus, address and data signals are multiplexed. To build the address, an external latch is used to demultiplex and reconstruct the original address. Since the LALE signal provides the correct timing to control a standard logic latch, no external intelligence is needed. To pass data, the LAD signals can be directly connected to the data signals of the memory/peripheral.

Transactions on the local bus begin with an address phase. The eLBC drives the transaction address on the LAD signals and asserts the LALE signal to latch the address. This assertion causes address bits A[0–31] to appear on LAD[0–31]. The eLBC can then continue on into the data phase.

The eLBC supports port sizes of 8, 16, and 32 bits. When there is an access larger than the port size, the eLBC breaks up the access into smaller transactions using the non-multiplexed address signals LA*n*. For 32-bit devices, LA[30–31] are irrelevant since these address bits are implicit in the byte lanes which carry data. Similarly, for 16-bit devices, LA[30] is used and LA[31] is irrelevant; however, for 8-bit devices, LA[30–31] are necessary.

In addition, the eLBC supports burst transfers in the UPM machine. To minimize the amount of address phases needed on the local bus and to optimize the throughput, LA*n* are driven separatelyand should be used whenever a device requires the five least-significant addresses. The five least-significant address bits should not be used from LAD[27–31]. All other address bits, A[0–26], must be reconstructed through the latch, as shown in Figure 10-70.

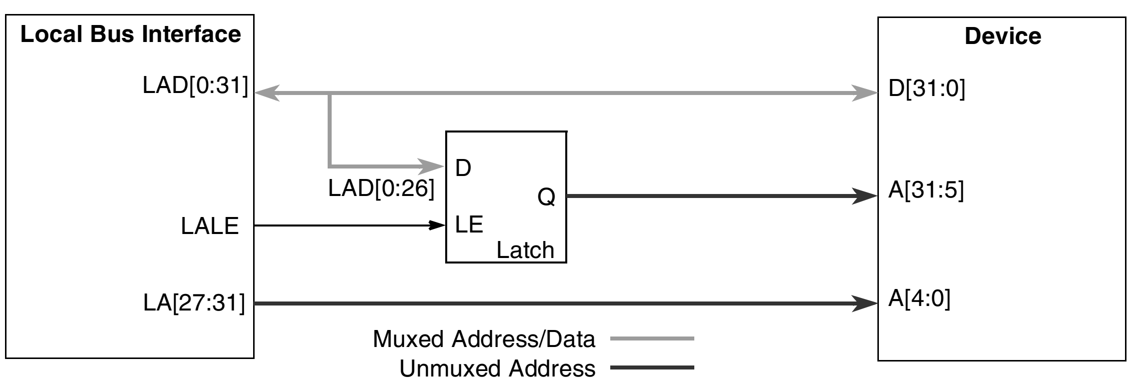


Figure 10-70. Multiplexed Address/Data Bus for 32-Bit Addressing

10.5.1.2 Non-Multiplexed Address and Data Buses

For small address space applications the address latch may be eliminated entirely if the local bus address is taken entirely from LA[10–31], in which case addresses driven onto LAD during address phases are simply ignored. The connection is illustrated in Figure 10-71. In non-multiplexed mode, waveforms etc remain the same except that few things need not be taken care of like ASHIFT parameter, LAD bus turnaround time, LALE timings etc.

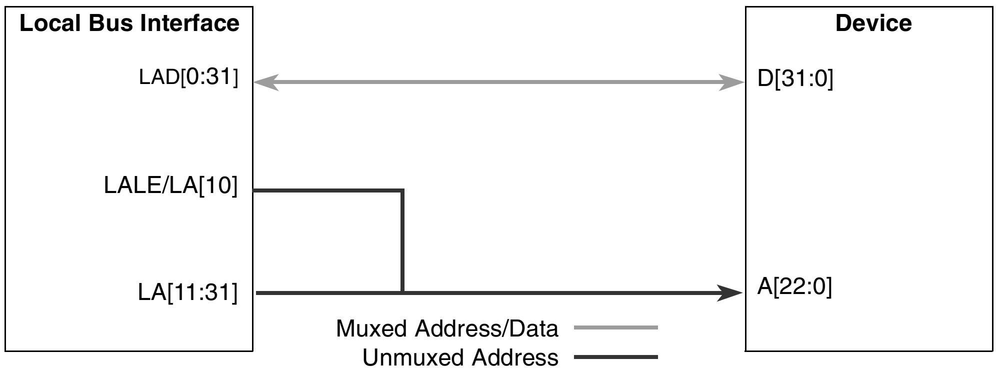


Figure 10-71. Non-Multiplexed Address and Data Buses

10.5.1.3 Multiplexed Address and Data To Save Maximum Pins In 8- to 16-Bit Addressing

With the use of a feature called address byte swap by setting LBCR[ABSWP], data and address muxing can be swapped from the default available. Currently, LAD[0–31] carries A[0–31]. In case of 8-bit interface with 8-bit addressing we do not get benefit of pin reduction with the available muxing. This is because,the MSB of data is muxed with MSB of address. While 8-bit data required is LAD[0–7] and address required is lower order bits A[24–31] we need to pull out all the 16 bits out of the device. For pin limited devices, this feature can be used where LAD[0–7] is mapped to the lsb's of A[24–31] and LAD[8–15] carries **lsb+1** [16–23]. As a result while interfacing with 8 bit address and 8-bit data only LAD[0–7] is suffice with LALE pin. The only drawback of this feature is that it does not support burst as all the address is latched from LAD bus.

10.5.1.4 Peripheral Hierarchy on the Local Bus for High Bus Speeds

To achieve the highest possible bus speeds on the local bus, it is recommended to reduce the number of devices connected directly to the bus. For best results, only one bank of synchronous SRAMs should have a direct connection, and a bus demultiplexor should be used to replace separate latch and separate bus transceiver combinations. Figure 10-72 shows an example of such a hierarchy. This section is only a guideline, and the board designer must simulate the electric characteristics of the scenario to determine the maximum operating frequency.

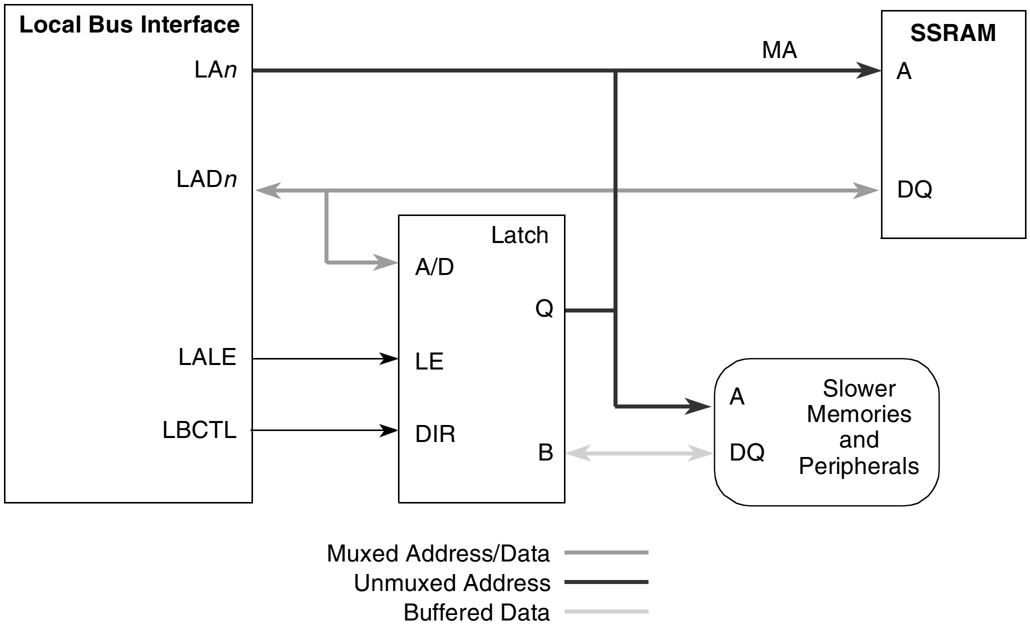


Figure 10-72. Local Bus Peripheral Hierarchy for High Bus Speeds

10.5.1.5 GPCM Timings

In case a system contains a memory hierarchy with high speed synchronous memories (synchronous SRAM) and lower speed asynchronous memories (for example, FLASH EPROM and peripherals) the GPCM-controlled memories should be decoupled by buffers to reduce capacitive loading on the bus. Those buffers have to be taken into account for the timing calculations.

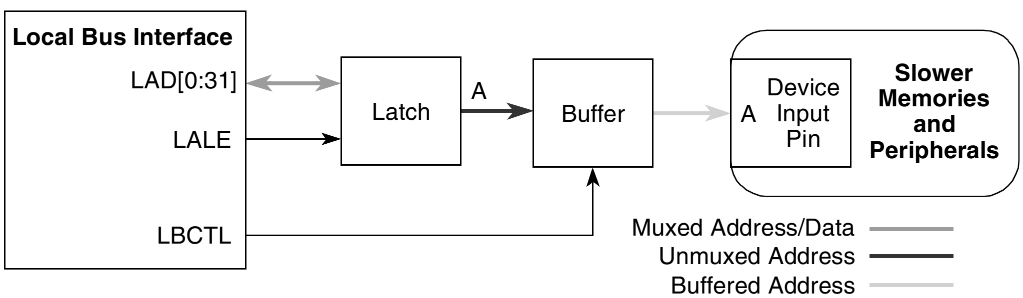


Figure 10-73. GPCM Address Timings

To calculate address setup timing for a slower peripheral/memory device, several parameters have to be added: propagation delay for the address latch, propagation delay for the buffer and the address setup for the actual peripheral. Typical values for the two propagation delays are in the order of 3–6 ns, so for a 133-MHz bus frequency, $\overline{\text{LCS}}$ should arrive on the order of 3 bus clocks later.

For data timings, only the propagation delay of one buffer plus the actual data setup time has to be considered.

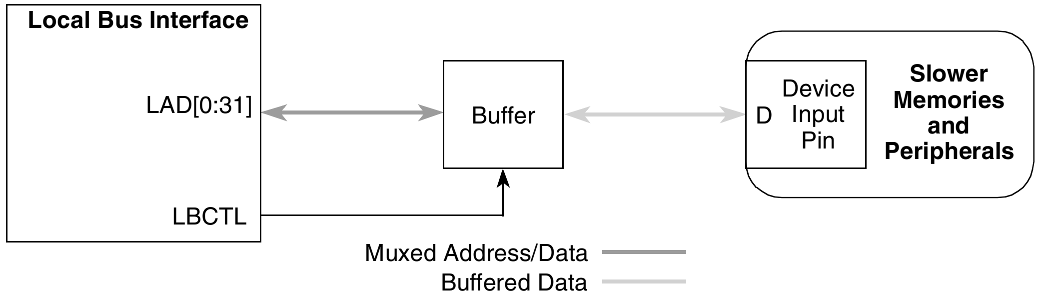


Figure 10-74. GPCM Data Timings

10.5.2 Bus Turnaround

Because the local bus uses multiplexed address and data, special consideration must be given to avoid bus contention at bus turnaround. The following cases must be examined:

- Address phase after previous read
- Read data phase after address phase
- Read-modify-write cycle for parity protected memory banks
- UPM cycles with additional address phases

The bus does not change direction for the following cases so they need no special attention:

- Continued burst after the first beat
- Write data phase after address phase
- Address phase after previous write

10.5.2.1 Address Phase after Previous Read

During a read cycle, the memory/peripheral drives the bus and the bus transceiver drives LAD. After the data has been sampled, the output drivers of the external device must be disabled. This can take some time; for slow devices the EHTR feature of the GPCM or the programmability of the UPM should be used to guarantee that those devices have stopped driving the bus when the eLBC memory controller ends the bus cycle.

In this case, after the previous cycle ends, LBCTL goes high and changes the direction of the bus transceiver. The eLBC then inserts a bus turnaround cycle to avoid contention. The external device has now already placed its data signals in high impedance and no bus contention will occur.

10.5.2.2 Read Data Phase after Address Phase

During the address phase, LAD actively drives the address and LBCTL is high, driving the bus transceivers in the same direction as during a write. After the end of the address phase, LBCTL goes low and changes the direction of the bus transceiver. The eLBC places the LAD signals in high impedance after its t_{dis}(LB). The LBCTL will have its new state after t_{en}(LB) and, because this is an asynchronous input, the transceiver starts to drive those signals after its t_{en}(transceiver) time. The system designer has to ensure, that [t_{en}(LB) + t_{en}(transceiver)] is larger than t_{dis} (LB) to avoid bus contention.

10.5.2.3 Read-Modify-Write Cycle for Parity Protected Memory Banks

Principally, a read-modify-write cycle is a read cycle immediately followed by a write cycle. Because the write cycle will have a new address phase in any case, this basically is the same case as an address phase after a previous read.

10.5.2.4 UPM Cycles with Additional Address Phases

The flexibility of the UPM allows the user to insert additional address phases during read cycles by changing the AMX field, therefore turning around the bus during one pattern. The eLBC automatically inserts a single bus turnaround cycle if the bus (LAD) was previously high impedance for any reason, such as a read, before LALE is driven and LAD is driven with the new address. The turnaround cycle is not inserted on a write, because the bus was already driven to begin with.

However, bus contention could potentially still occur on the far side of a bus transceiver. It is the responsibility of the designer of the UPM pattern to guarantee that enough idle cycles are inserted in the UPM pattern to avoid this.

10.5.3 Interface to Different Port-Size Devices

The eLBC supports 8-, 16-, and 32-bit data port sizes. However, the bus requires that the portion of the data bus used for a transfer to or from a particular port size be fixed. A 32-bit port must reside on LAD[0–31], a 16-bit port must reside on LAD[0–15], and an 8-bit port must reside on LAD[0–7]. The local bus always tries to transfer the maximum amount of data on all bus cycles. Figure 10-75 shows the device connections on the data bus.

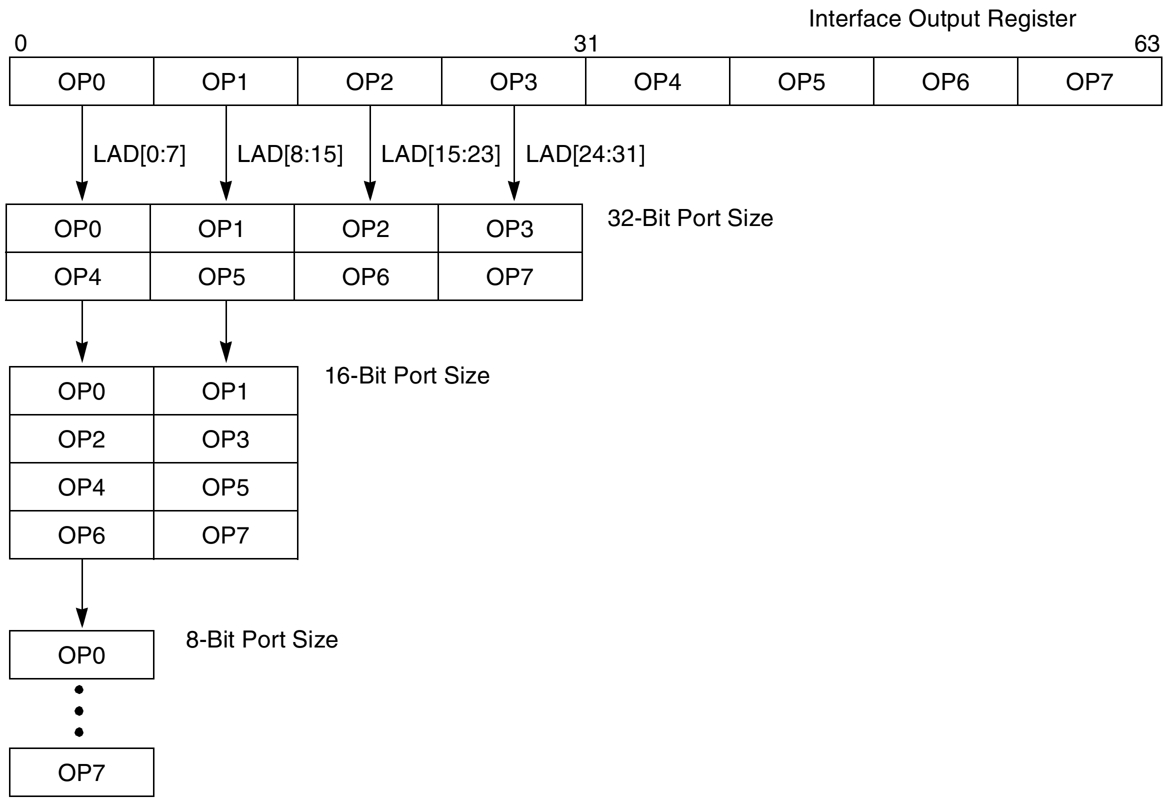


Figure 10-75. Interface to Different Port-Size Devices

Table 10-41 lists the bytes required on the data bus for read cycles.

Table 10-41. Data Bus Drive Requirements For Read Cycles

Transfer Size	Address State [1] 3 lsbs	Port Size/LAD Data Bus Assignments											
		32-Bit				16-Bit				8-Bit			
		0–7	8–15	16–23	24–31	0–7	8–15	16–23	24–31	0–7	8–15	16–23	24–31
Byte	000	OP0[2]	—[3]	—	—	OP0	—			OP0			
	001	—	OP1	—	—	—	OP1			OP1			
	010	—	—	OP2	—	OP2	—			OP2			
	011	—	—	—	OP3	—	OP3			OP3			
	100	OP4	—	—	—	OP4	—			OP4			
	101	—	OP5	—	—	—	OP5			OP5			
	110	—	—	OP6	—	OP6	—			OP6			
	111	—	—	—	OP7	—	OP7			OP7			

Table 10-41. Data Bus Drive Requirements For Read Cycles (continued)

Transfer Size	Address State [1] 3 lsbs	Port Size/LAD Data Bus Assignments											
		32-Bit				16-Bit				8-Bit			
		0–7	8–15	16–23	24–31	0–7	8–15	16–23	24–31	0–7	8–15	16–23	24–31
Half Word	000	OP0	OP1	—	—	OP0	OP1			OP0			
	001	—	OP1	OP2	—	—	OP1			OP1			
	010	—	—	OP2	OP3	OP2	OP3			OP2			
	100	OP4	OP5	—	—	OP4	OP5			OP4			
	101	—	OP5	OP6	—	—	OP5			OP5			
	110	—	—	OP6	OP7	OP6	OP7			OP6			
Word	000	OP0	OP1	OP2	OP3	OP0	OP1			OP0			
	100	OP4	OP5	OP6	OP7	OP4	OP5			OP4			

1 Address state is the calculated address for port size.

2 OP*n*: These lanes are read or written during that bus transaction. OP0 is the most-significant byte of a doubleword operand and OP7 is the least-significant byte.

3 — Denotes a byte not driven during that read cycle.

10.5.4 Command Sequence Examples for NAND Flash EEPROM

In order to program the eLBC and FCM for executing NAND Flash command sequences, command codes and pause states should be obtained from the relevant NAND Flash device data sheet and programmed into FCM configuration registers. This section illustrates some common sequences for large-page, multi-gigabit NAND Flash EEPROMs; however, details should be verified against manufacturers' specific programming data.

Throughout these examples it is assumed that one or more banks of eLBC has been configured under FCM control (BR*n*[MSEL] = 001), with base address, port size, ECC mode, and timing parameters configured in accordance with the device's hardware specifications.

10.5.4.1 NAND Flash Soft Reset Command Sequence Example

An example of configuring FCM to execute a soft reset command to large-page NAND Flash is shown in Table 10-42. This sequence does not require use of the shared FCM buffer RAM. The sequence is initiated by writing FMR[OP] = 10, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Table 10-42. FCM Register Settings for Soft Reset (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0xFF000000	CMD0 = 0xFF = reset command; other commands unused
FBAR	—	unused

Table 10-42. FCM Register Settings for Soft Reset (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FPAR	—	unused
FBCR	—	unused
MDR	—	unused
FIR	0x40000000	OP0 = CM0 = command 0; OP1–OP7 = NOP

10.5.4.2 NAND Flash Read Status Command Sequence Example

An example of configuring FCM to execute a status read command to large-page NAND Flash is shown in Table 10-43. This sequence does not require use of the shared FCM buffer RAM, but reads the NAND Flash status into register MDR[AS0]. The sequence is initiated by writing FMR[OP] = 10 and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Table 10-43. FCM Register Settings for Status Read (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0x70000000	CMD0 = 0x70 = read status command; other commands unused
FBAR	—	unused
FPAR	—	unused
FBCR	—	unused
MDR	—	Status returned in AS0
FIR	0x4B000000	OP0 = CM0 = command 0; OP1 = RS = read status to MDR; OP2–OP7 = NOP

10.5.4.3 NAND Flash Read Identification Command Sequence Example

An example of configuring FCM to execute a status ID command to large-page NAND Flash is shown in Table 10-44. This sequence does not require use of the shared FCM buffer RAM, but uses MDR to set up a dummy address prior to the sequence, and then to receive the first 4 bytes of ID during the sequence. The sequence is initiated by writing FMR[OP] = 10, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled. MDR[AS3–AS0] then can be read to obtain the first 4 bytes of NAND Flash ID.

Table 10-44. FCM Register Settings for ID Read (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0x90000000	CMD0 = 0x90 = read ID command; other commands unused
FBAR	—	unused
FPAR	—	unused

Table 10-44. FCM Register Settings for ID Read (ORn[PGS] = 1)

Register	Initial Contents	Description
FBCR	—	unused
MDR	0x00000000	AS0 = 0x00 = dummy address for read ID command; AS0–AS3 return with first 4 bytes of ID code
FIR	0x43BBBBB0	OP0 = CM0 = command 0; OP1 = UA = user address from MDR; OP2–OP6 = RS = read 4 bytes ID into MDR[AS3–AS0]; OP7 = NOP

10.5.4.4 NAND Flash Page Read Command Sequence Example

An example of configuring FCM to execute a random page read command to large-page NAND Flash is shown in Table 10-45. This sequence reads an entire page (main and spare region) into the shared FCM buffer RAM, checking ECC as it proceeds. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Table 10-45. FCM Register Settings for Page Read (ORn[PGS] = 1)

Register	Initial Contents	Description
FCR	0x00300000	CMD0 = 0x00 = random read address entry; CMD1 = 0x30 = read page
FBAR	block index (e.g. block 0x00010ab4)	BLK locates index of 128-Kbyte block
FPAR	page offset (e.g. 0x00005000 locates page 5, buffer 1)	PI locates page index in BLK; PI mod 2 indexes FCM buffer RAM; MS = 0 and CI = 0
FBCR	0x00000000	BC = 0 to read entire 2112-byte page with ECC check
MDR	—	unused
FIR	0x4125E000	OP0 = CM0 = command 0; OP1 = CA = column address; OP2 = PA = page address; OP3 = CM1 = command 1; OP4 = RBW = wait on Flash ready and read data into FCM buffer; OP5–OP7 = NOP

10.5.4.5 NAND Flash Block Erase Command Sequence Example

An example of configuring FCM to execute a block erase command to large-page NAND Flash is shown in Table 10-46. This sequence does not require use of the shared FCM buffer RAM, but returns with the erase status in MDR[AS0]. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled.

Note that operations specified by OP3 and OP4 (status read) should never be skipped while erasing a NAND Flash device, because, in case that happens, contention may arise on LGPL4. A possible case is that the next transaction from eLBC may try to use that pin as an output and since the NAND Flash device might already be driving it, contention will occur. In case OP3 and OP4 operations are skipped, it may also happen that a new command is issued to the NAND Flash device even when the device has not yet finished processing the previous request. This may also result in unpredictable behavior.

Table 10-46. FCM Register Settings for Block Erase (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0x6070D000	CMD0 = 0x60 = block address entry; CMD1 = 0x70 = read status CMD2 = 0xD0 = erase block;
FBAR	block index (e.g. block 0x00010AB4)	BLK locates index of 128-Kbyte block
FPAR	0x00000000	PI = 0 to locate block boundary
FBCR	—	unused
MDR	—	returns with AS0 holding erase status
FIR	0x426DB000	OP0 = CM0 = command 0; OP1 = PA = page address; OP2 = CM2 = command 2; OP3 = CW1 = wait on Flash ready and issue command 1; OP4 = RS = read erase status into MDR[AS0]; OP5–OP7 = NOP

10.5.4.6 NAND Flash Program Command Sequence Example

An example of configuring FCM to execute a program command to large-page NAND Flash is shown in Table 10-47. This sequence writes an entire page (main and spare region) from the shared FCM buffer RAM, generating ECC as it proceeds. The shared buffer (buffer 1 for page index 5) must be initialized by software prior to starting the sequence. The sequence is initiated by writing FMR[OP] = 11, and issuing a special operation to the bank. At the conclusion of the sequence, eLBC will issue a command complete interrupt (LTESR[CC]) if interrupts are enabled. The status of the programming operation is returned in MDR[AS0].

Note that operations specified by OP5 and OP6 (status read) should never be skipped while programming a NAND Flash device, because, in case that happens, contention may arise on LGPL4. A possible case is that the next transaction from eLBC may try to use that pin as an output and since the NAND Flash device might already be driving it, contention will occur. In case OP5 and OP6 operations are skipped, it may also happen that a new command is issued to the NAND Flash device even when the device has not yet finished processing the previous request. This may also result in unpredictable behavior.

Table 10-47. FCM Register Settings for Page Program (OR*n*[PGS] = 1)

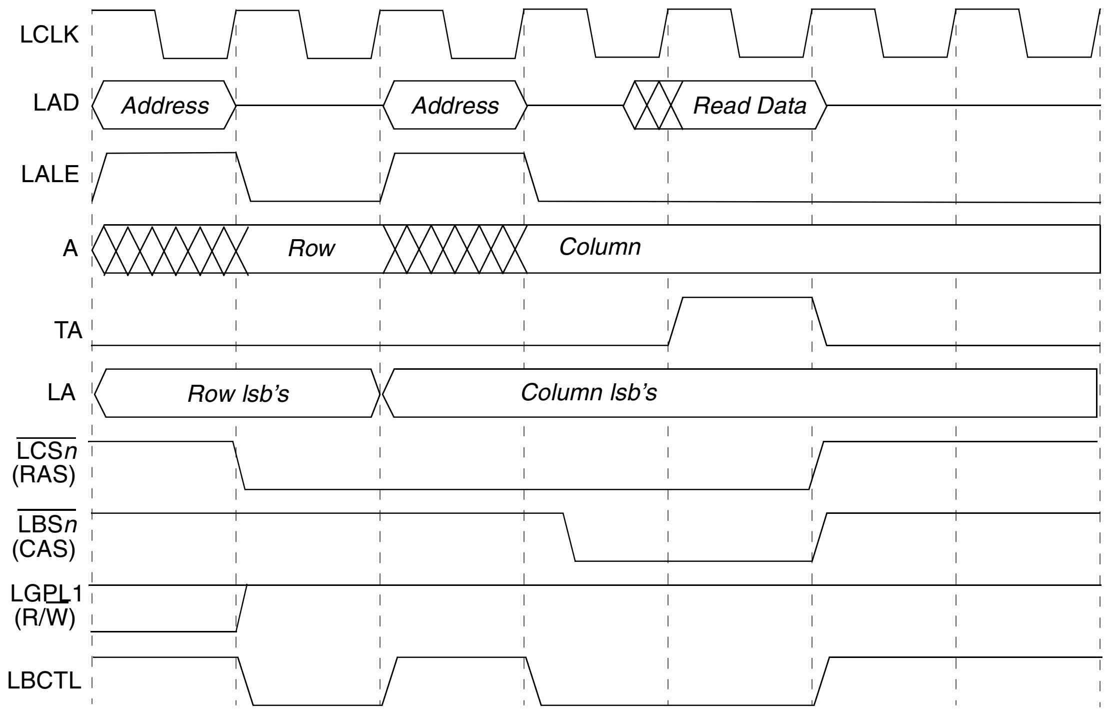
Register	Initial Contents	Description
FCR	0x80701000	CMD0 = 0x80 = page address and data entry; CMD1 = 0x70 = read status CMD2 = 0x10 = program page;
FBAR	block index (e.g. block 0x00010AB4)	BLK locates index of 128-Kbyte block

Table 10-47. FCM Register Settings for Page Program (OR*n*[PGS] = 1) (continued)

Register	Initial Contents	Description
FPAR	page offset (e.g. 0x00005000 locates page 5, buffer 1)	PI locates page index in BLK; PI mod 2 indexes FCM buffer RAM; MS = 0 and CI = 0
FBCR	0x00000000	BC = 0 to write entire 2112-Byte page with ECC generation
MDR	—	returns with AS0 holding program status
FIR	0x41286DB0	OP0 = CM0 = command 0; OP1 = CA = column address; OP2 = PA = page address; OP3 = WB = write data from buffer; OP4 = CM2 = command 2; OP5 = CW1 = wait on Flash ready and issue command 1; OP6 = RS = read erase status into MDR[AS0]; OP7 = NOP

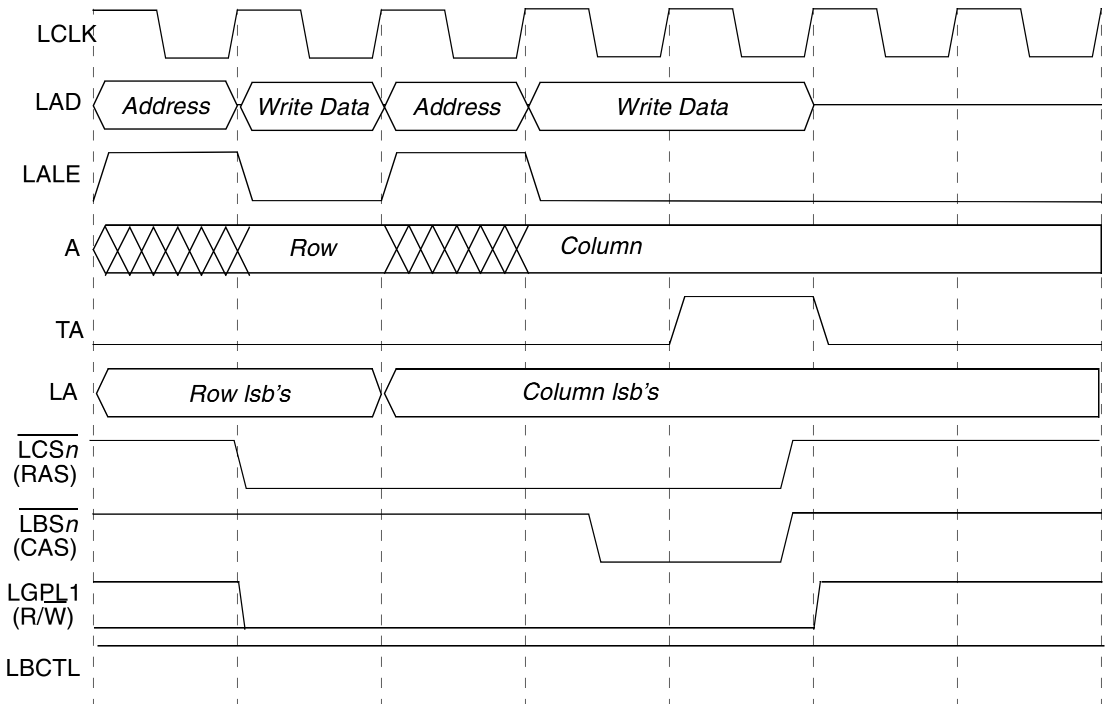
10.5.5 Interfacing to Fast-Page Mode DRAM Using UPM

Connecting the local bus UPM controller to a DRAM device requires a detailed examination of the timing diagrams representing the possible memory cycles that must be performed when accessing this device. This section shows timing diagrams for various UPM configurations for fast-page mode DRAM, with LCRR[CLKDIV] = 4 or 8. These illustrative examples may not represent the timing necessary for any specific device used with the eLBC. Here, LGPL1 is programmed to drive R/$\overline{W}$ of the DRAM, although any LGPL*n* signal may be used for this purpose.



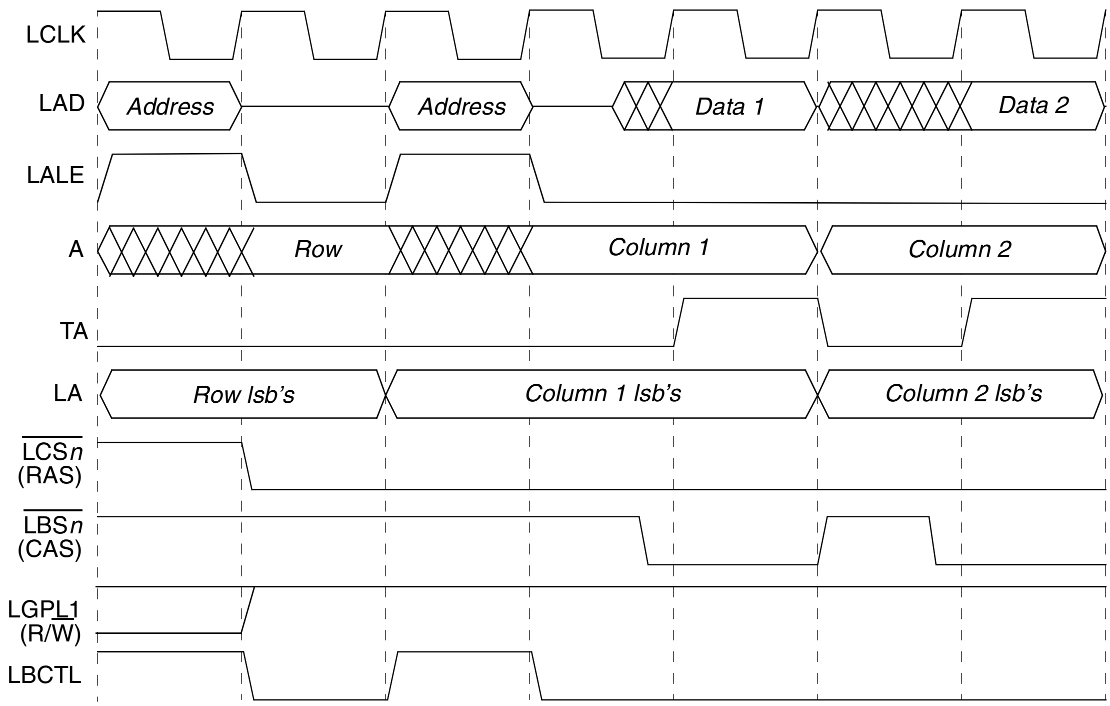
cst1	0	LALE pause (due to change in AMX)	0	0	Bit 0
cst2	0		0	0	Bit 1
cst3	0		0	0	Bit 2
cst4	0		0	0	Bit 3
bst1	1		1	0	Bit 4
bst2	1		0	0	Bit 5
bst3	1		0	0	Bit 6
bst4	1		0	0	Bit 7
g0l0					Bit 8
g0l1					Bit 9
g0h0					Bit 10
g0h1					Bit 11
g1t1	1		1	1	Bit 12
g1t3	1		1	1	Bit 13
g2t1					Bit 14
g2t3					Bit 15
g3t1					Bit 16
g3t3					Bit 17
g4t1					Bit 18
g4t3					Bit 19
g5t1					Bit 20
g5t3					Bit 21
redo[0]					Bit 22
redo[1]					Bit 23
loop	0		0	0	Bit 24
exen	0		0	0	Bit 25
amx0	1		0	0	Bit 26
amx1	0		0	0	Bit 27
na	0		0	0	Bit 28
uta	0		0	1	Bit 29
todt	0		0	1	Bit 30
last	0		0	1	Bit 31
	RSS	**RSS+1**	**RSS+1**	**RSS+2**	

Figure 10-76. Single-Beat Read Access to FPM DRAM



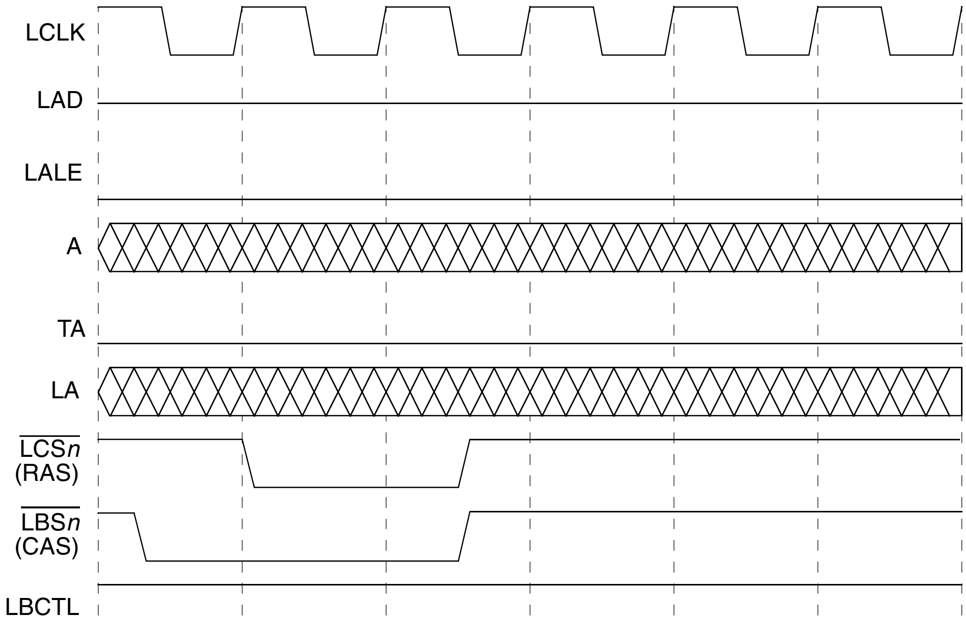
cst1	0	LALE pause (due to change in AMX)	0	0	Bit 0
cst2	0		0	0	Bit 1
cst3	0		0	0	Bit 2
cst4	0		0	1	Bit 3
bst1	1		1	0	Bit 4
bst2	1		1	0	Bit 5
bst3	1		0	0	Bit 6
bst4	1		0	1	Bit 7
g0l0					Bit 8
g0l1					Bit 9
g0h0					Bit 10
g0h1					Bit 11
g1t1	0		0	0	Bit 12
g1t3	0		0	0	Bit 13
g2t1					Bit 14
g2t3					Bit 15
g3t1					Bit 16
g3t3					Bit 17
g4t1					Bit 18
g4t3					Bit 19
g5t1					Bit 20
g5t3					Bit 21
redo[0]					Bit 22
redo[1]					Bit 23
loop	0		0	0	Bit 24
exen	0		0	0	Bit 25
amx0	1		0	0	Bit 26
amx1	0		0	0	Bit 27
na	0		0	0	Bit 28
uta	0		0	1	Bit 29
todt	0		0	1	Bit 30
last	0		0	1	Bit 31
	WSS	**WSS+1**	**WSS+1**	**WSS+2**	

Figure 10-77. Single-Beat Write Access to FPM DRAM



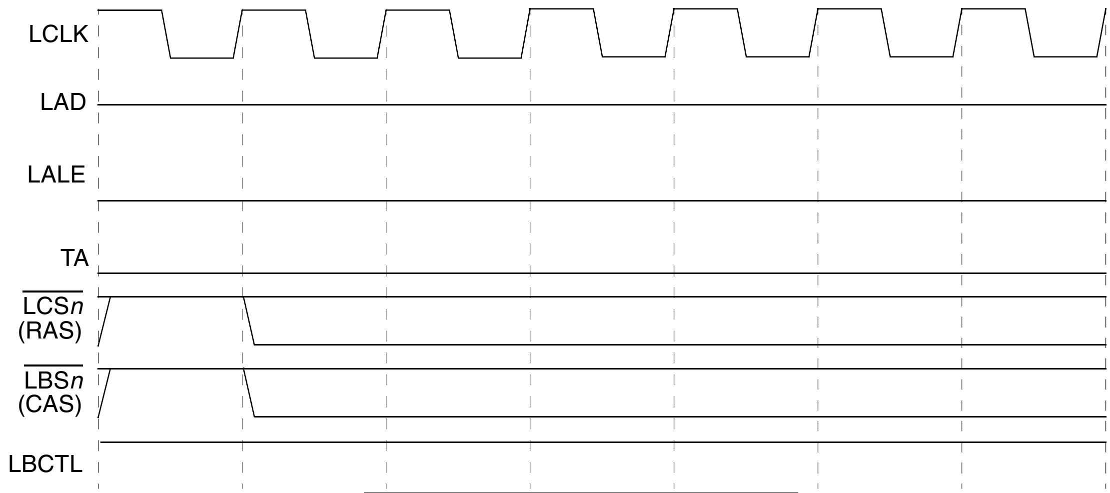
cst1	0	LALE pause (due to change in AMX)	0	0	1	Bit 0
cst2	0		0	0	1	Bit 1
cst3	0		0	0	1	Bit 2
cst4	0		0	0	1	Bit 3
bst1	1		1	0	1	Bit 4
bst2	1		1	0	1	Bit 5
bst3	1		1	0	1	Bit 6
bst4	1		0	0	1	Bit 7
g0l0						Bit 8
g0l1						Bit 9
g0h0						Bit 10
g0h1						Bit 11
g1t1	1		1	1	1	Bit 12
g1t3	1		1	1	1	Bit 13
g2t1						Bit 14
g2t3						Bit 15
g3t1						Bit 16
g3t3						Bit 17
g4t1						Bit 18
g4t3						Bit 19
g5t1						Bit 20
g5t3						Bit 21
redo[0]						Bit 22
redo[1]						Bit 23
loop	0		1	1	0	Bit 24
exen	0		0	1	0	Bit 25
amx0	1		0	0	0	Bit 26
amx1	0		0	0	0	Bit 27
na	0		0	1	0	Bit 28
uta	0		0	1	0	Bit 29
todt	0		0	0	1	Bit 30
last	0		0	0	1	Bit 31
	RBS		**RBS+1**	**RBS+2**	**RBS+3**	

Figure 10-78. Burst Read Access to FPM DRAM Using LOOP (Two Beats Shown)



	PTS	PTS+1	PTS+2	
cst1	1	0	0	Bit 0
cst2	1	0	0	Bit 1
cst3	1	0	1	Bit 2
cst4	1	0	1	Bit 3
bst1	1	0	0	Bit 4
bst2	0	0	0	Bit 5
bst3	0	0	1	Bit 6
bst4	0	0	1	Bit 7
g0l0				Bit 8
g0l1				Bit 9
g0h0				Bit 10
g0h1				Bit 11
g1t1				Bit 12
g1t3				Bit 13
g2t1				Bit 14
g2t3				Bit 15
g3t1				Bit 16
g3t3				Bit 17
g4t1				Bit 18
g4t3				Bit 19
g5t1				Bit 20
g5t3				Bit 21
redo[0]				Bit 22
redo[1]				Bit 23
loop	0	0	0	Bit 24
exen	0	0	0	Bit 25
amx0	0	0	0	Bit 26
amx1	0	0	0	Bit 27
na	0	0	0	Bit 28
uta	0	0	0	Bit 29
todt	0	0	1	Bit 30
last	0	0	1	Bit 31

Figure 10-79. Refresh Cycle (CBR) to FPM DRAM



cst1	1	Bit 0
cst2	1	Bit 1
cst3	1	Bit 2
cst4	1	Bit 3
bst1	1	Bit 4
bst2	1	Bit 5
bst3	1	Bit 6
bst4	1	Bit 7
g0l0		Bit 8
g0l1		Bit 9
g0h0		Bit 10
g0h1		Bit 11
g1t1		Bit 12
g1t3		Bit 13
g2t1		Bit 14
g2t3		Bit 15
g3t1		Bit 16
g3t3		Bit 17
g4t1		Bit 18
g4t3		Bit 19
g5t1		Bit 20
g5t3		Bit 21
redo[0]		Bit 22
redo[1]		Bit 23
loop	0	Bit 24
exen	0	Bit 25
amx0	0	Bit 26
amx1	0	Bit 27
na	0	Bit 28
uta	0	Bit 29
todt	1	Bit 30
last	1	Bit 31
	EXS	

Figure 10-80. Exception Cycle

10.5.6 Interfacing to ZBT SRAM Using UPM

ZBT SRAMs have been designed to optimize the performance of table access in networking applications. This section describes how to interface to ZBT SRAMs. Figure 10-81 shows the connections. The UPM is used to generate control signals. The same interfacing is used for pipelined and flow-through versions of ZBT SRAMs. However different UPM patterns must be generated for those cases. Because ZBT

SRAMs will mostly be used by performance-critical applications, we assume here that, typically, the maximum width of the local bus of 32 bits will be used.

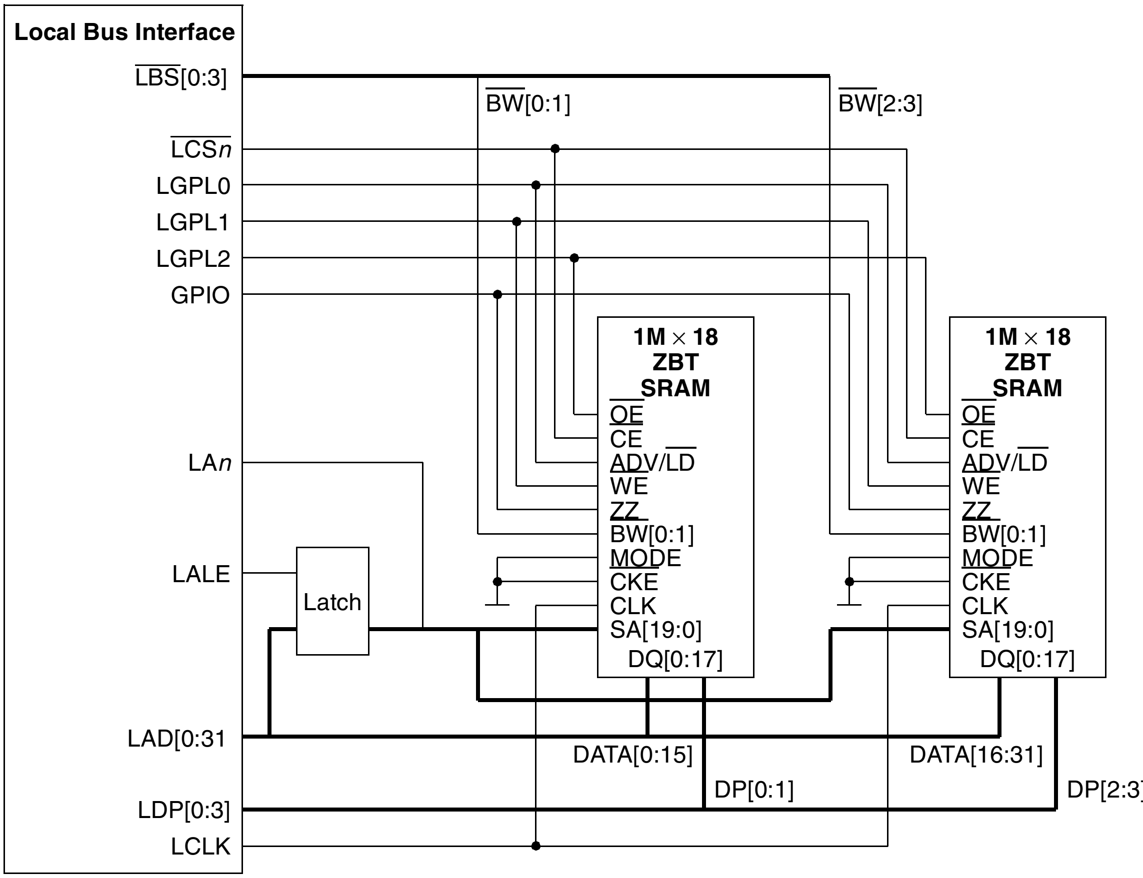


Figure 10-81. Interface to ZBT SRAM

ZBT SRAMs allow different configurations. For the local bus, the burst order should be set to linear burst order by tying the mode pin to GND. CKE should also be tied to ground.

ZBT SRAMs perform four-beat bursts. Because the eLBC generates eight-beat transactions (for 32-bit ports) the UPM breaks down each burst into two consecutive four-beat bursts. The internal address generator of the eLBC generates the new LA bits for each burst. In other words, because linear burst is used on the SRAM, the device itself bursts with the burst addresses of [0:1:2:3]. The local bus always generates linear bursts and expects [0:1:2:3:4:5:6:7]. Therefore, two consecutive linear bursts of the ZBT SRAM with {A27, A28} = {0,0} for the first burst, and {A27, A28} = {1,0} for the second burst give the desired burst pattern.

The UPM also supports single beat accesses. Because the ZBT SRAM does not support this and always responds with a burst, the UPM pattern has to take care that data for the critical beat is provided (for write) or sampled (for read), and that the rest of the burst is ignored (by negating WE). The UPM controller basically has to wait for the end of the SRAM burst to avoid bus contention with further bus activities.

Chapter 11
Enhanced Secure Digital Host Controller

11.1 Overview

The enhanced secure digital host controller (eSDHC) provides an interface between the host system and these types of memory cards:

- MultiMediaCard (MMC)

  MMC is a universal low-cost data storage and communication medium designed to cover a wide area of applications including mobile video and gaming, which are available from either pre-loaded MMC cards or downloadable from cellular phones, WLAN, or other wireless networks. Old MMC cards are based on a seven-pin serial bus with a single data pin, while the new high-speed MMC communication is based on an advanced 11-pin serial bus designed to operate in a low voltage range.

- Secure digital (SD) card

  The secure digital (SD) card is an evolution of old MMC technology. It is specifically designed to meet the security, capacity, performance, and environment requirements inherent in the emerging audio and video consumer electronic devices. The physical form factor, pin assignments, and data transfer protocol are forward-compatible with the old MMC.

The eSDHC acts as a bridge, passing host bus transactions to SD/MMC cards by sending commands and performing data accesses to or from the cards. It handles the SD/MMC protocol at the transmission level. Figure 11-1 shows connection of the eSDHC.

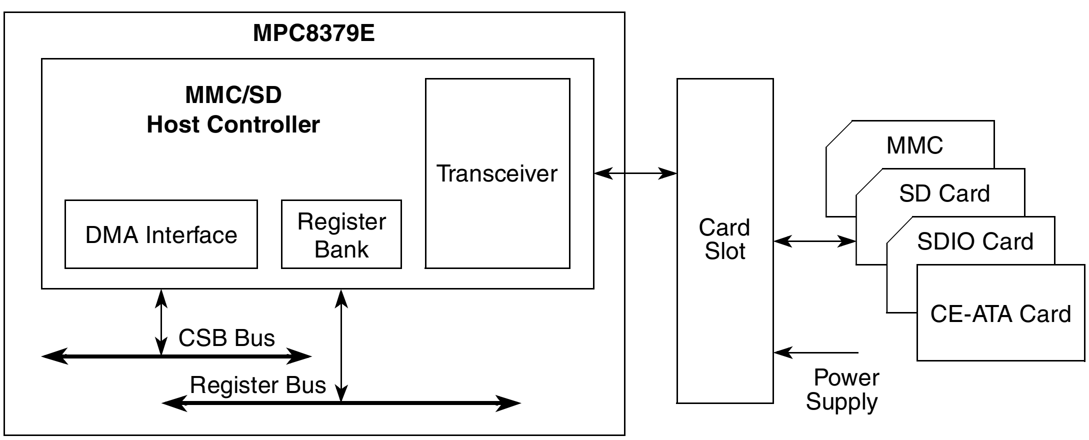


Figure 11-1. System Connection of the eSDHC

Figure 11-2 is a block diagram of the eSDHC.

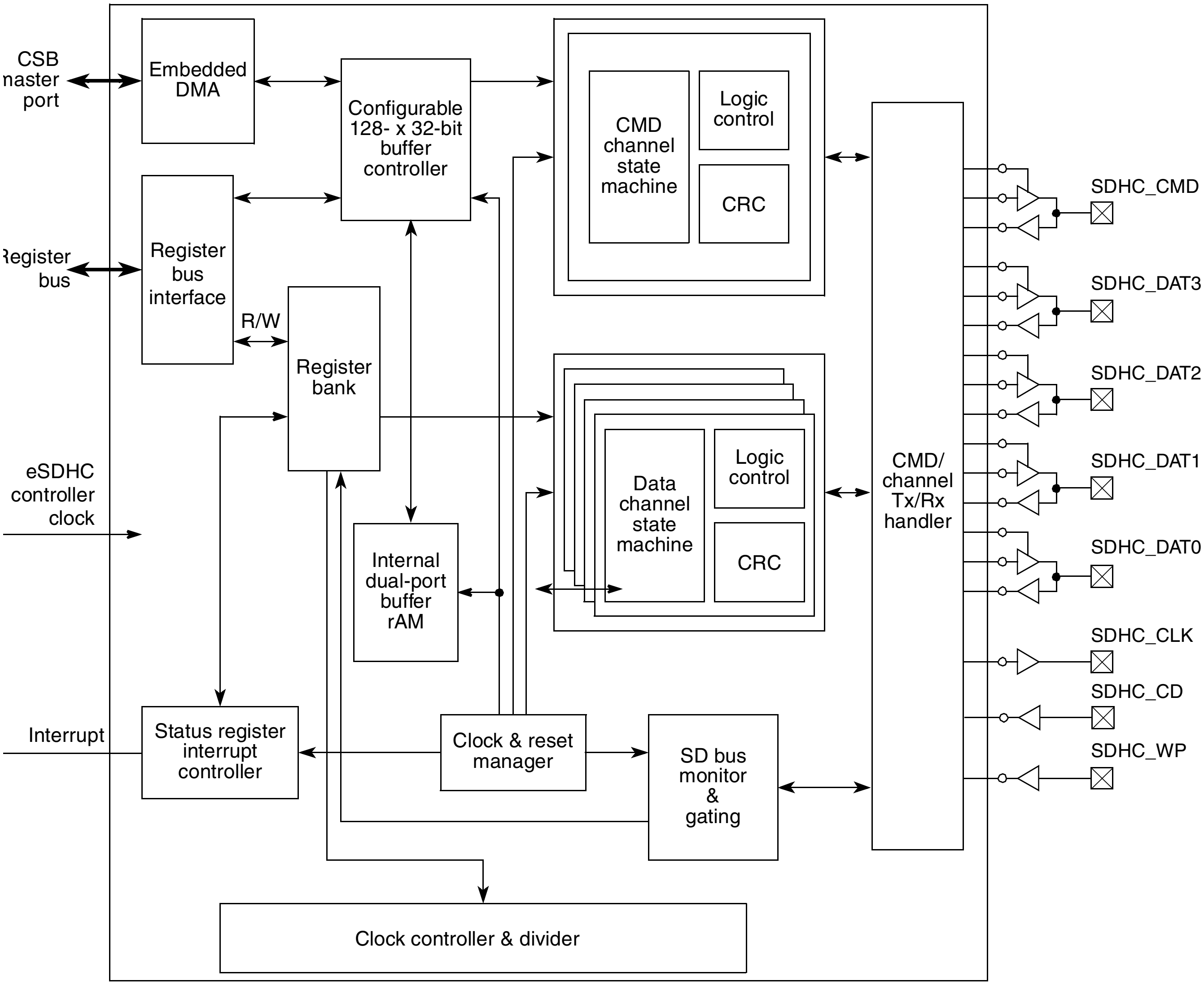


Figure 11-2. eSDHC Block Diagram

11.2 Features

The eSDHC includes the following features:

- Compatible with the following specifications:
  - — *SD Host Controller Standard Specification, Version 2.0* (http://www.sdcard.org) with test event register support
  - — *MultiMediaCard System Specification, Version 4.0* (http://www.mmca.org)
  - — *SD Memory Card Specification, Version 2.0* (http://www.sdcard.org)
- Designed to work with SD Memory, miniSD Memory, SD Combo, MMC, MMC*plus*, and RS-MMC cards

- Card bus clock frequency up to 50 MHz
- Supports 1-/4-bit SD mode, 1-/4-bit MMC modes
  - Up to 200 Mbps data transfer for SD/MMC cards using four parallel data lines
- Single- and multi-block read and write
- Write-protection switch for write operations
- Synchronous and asynchronous abort
- Pause during the data transfer at a block gap
- Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer commands while the data transfer is in progress
- Fully configurable 128 × 32-bit FIFO for read/write data
- Internal DMA capabilities

11.2.1 Data Transfer Modes

The eSDHC can select the following modes for data transfer:

- SD 1-bit
- SD 4-bit
- MMC 1-bit
- MMC 4-bit
- Full-speed mode (up to 25 MHz) or high-speed mode (up to 50 MHz)

11.3 External Signal Description

The eSDHC has eight chip I/O signals.

- SDHC_CLK is the internally generated clock signal that drives the MMC, or SD card.
- SDHC_CMD I/O sends commands and receive responses from the card.
- SDHC_DAT3–SDHC_DAT0 perform data transfers between the eSDHC and the card.
- SDHC_CD and SDHC_WP are card detection and write protection signals from the socket.
  - Signals SDHC_CD and SDHC_WP are optional for system implementation.

Table 11-1 shows the properties of the eSDHC I/O signals.

Table 11-1. Signal Properties

Name	Port	Function	Reset State	Pull up/Pull down Required
SDHC_CLK	O	Clock for MMC/SD card	0	N/A
SDHC_CMD	I/O	Command line to card	High impedance	Pull up
SDHC_DAT3	I/O	**4-bit mode:** DAT3 line or configured as card detection pin **1-bit mode:** May be configured as card detection pin	High impedance	Board should have 100K pull down. The card drives 50K pull up as required by the SD card specification.

Table 11-1. Signal Properties (continued)

Name	Port	Function	Reset State	Pull up/Pull down Required
SDHC_DAT2	I/O	**4-bit mode:** DAT2 line or read wait **1-bit mode:** Read wait	High impedance	Pull up
SDHC_DAT1	I/O	**4-bit mode:** DAT1 line or interrupt detect **1-bit mode:** Interrupt detect	High impedance	Pull up
SDHC_DAT0	I/O	DAT0 line or busy-state detect	High impedance	Pull up
SDHC_CD	I	Card detection pin By default CDP = 0, and so the SDHC_CD signal should be tied high (meaning card present). If CDP is programmed to 1, the SDHC_CD signal should be tied low (card present, according to the standard). If not implemented, tie to a card present value according to the polarity set by the SCR[CDP].	N/A	Board should have 100K pull down. The card drives 50K pull up as required by the SD card specification.
SDHC_WP	I	Card write protect detect; if not used, tied low	N/A	N/A

11.4 Memory Map/Register Definition

Table 11-2 shows the memory mapped registers of the eSDHC module and their offsets. It lists the offset, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of IMMRBAR together with the eSDHC block base address and offset listed in Table 11-2. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

NOTE

All eSDHC registers must be accessed as aligned 4-byte quantities. Accesses to the eSDHC registers that are less than 4-bytes are not supported.

Table 11-2. eSDHC Memory Map

eSDHC Registers—Block Base Address 0x2_E000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	DMA system address (DSADDR)	R/W	0x0000_0008	11.4.1/11-5
0x004	Block attributes (BLKATTR)	R/W	0x0000_0008	11.4.2/11-6
0x008	Command argument (CMDARG)	R/W	0x0000_0000	11.4.3/11-7
0x00C	Command transfer type (XFERTYP)	R/W	0x0000_0000	11.4.4/11-7
0x010	Command response0 (CMDRSP0)	R	0x0000_0000	11.4.5/11-10
0x014	Command response1 (CMDRSP1)	R	0x0000_0000	11.4.5/11-10
0x018	Command response2 (CMDRSP2)	R	0x0000_0000	11.4.5/11-10
0x01C	Command response3 (CMDRSP3)	R	0x0000_0000	11.4.5/11-10
0x020	Data buffer access port (DATPORT)	R/W	0x0000_0000	11.4.6/11-11
0x024	Present state (PRSSTAT)	R	0xFF80_0000	11.4.7/11-12

Table 11-2. eSDHC Memory Map (continued)

eSDHC Registers—Block Base Address 0x2_E000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x028	Protocol control (PROCTL)	R/W	0x0000_0000	11.4.8/11-16
0x02C	System control (SYSCTL)	Mixed	0x0000_8000	11.4.9/11-18
0x030	Interrupt status (IRQSTAT)	w1c	0x0000_0000	11.4.10/11-21
0x034	Interrupt status enable (IRQSTATEN)	R/W	0x117F_013F	11.4.11/11-25
0x038	Interrupt signal enable (IRQSIGEN)	R/W	0x0000_0000	11.4.12/11-27
0x03C	Auto CMD12 status (AUTOC12ERR)	R	0x0000_0000	11.4.13/11-29
0x040	Host controller capabilities (HOSTCAPBLT)	R	0x01E3_0000	11.4.14/11-31
0x044[1]	Watermark level (WML)	R/W	0x0010_0010	11.4.15/11-32
0x050	Force event (FEVT)	W	0x0000_0000	11.4.16/11-32
0x0FC	Host controller version (HOSTVER)	R	0x0000_0001	11.4.17/11-34
0x40C	DMA control register (DCR)	R/W	0x0000_0000	11.4.18/11-35

[1] The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

11.4.1 DMA System Address Register (DSADDR)

The DMA system address register contains the system memory address used for DMA transfers. Only access this register when no transactions are executing (after transactions have stopped). The host driver should wait until PRSSTAT[DLA] is cleared.

Offset: 0x000 — Access: Read/Write

	0–31
R/W	DS_ADDR
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

Figure 11-3. DMA System Address Register (DSADDR)

Table 11-3. DSADDR Field Descriptions

Field	Description
0–31 DS_ADDR	DMA system address. When the eSDHC stops a DMA transfer, this register points to the system address of the next contiguous data position. **Note:** The DS_ADDR must be aligned to a four-byte boundary; the two least-significant bits must be cleared.

11.4.2 Block Attributes Register (BLKATTR)

The block attributes register configures the number of data blocks and the number of bytes in each block. Only access this register when no transactions are executing (after transactions have stopped). The host driver should wait until PRSSTAT[DLA] is cleared. During a data transfer,

- Reading this register may return an invalid value.
- Writing this register is ignored.

Offset: 0x004 (BLKATTR) Access: Read/Write

	0–15	16–18	19–31
R/W	BLKCNT	—	BLKSZE
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 1 0 0 0

Figure 11-4. Block Attributes Register (BLKATTR)

Table 11-4. BLKATTR Field Descriptions

Field	Description
0–15 BLKCNT	Block count for current transfer. This field is enabled when XFERTYP[BCEN] is set and is valid only for multiple block transfers. The host driver should set this field to a value between 1 and the maximum block count. The eSDHC decrements the block count after each block transfer and stops when the count reaches zero. Clearing this field results in no data blocks being transferred. When saving transfer context as a result of a suspend command, this field indicates the number of blocks yet to be transferred. When restoring transfer context prior to issuing a resume command, the host driver should write the previously saved block count. 0000 Stop count 0001 1 block 0002 2 blocks ... FFFF 65,535 blocks
16–18	Reserved
19–31 BLKSIZE	Transfer block size. Specifies the block size for block data transfers. Values can range from one byte up to the maximum buffer size. The DMA always writes at least four bytes to memory. Thus, software should allocate a buffer space rounded up to a 4-byte alighted size in order to avoid data corruption. 0000 No data transfer 0001 1 byte 0002 2 bytes 0003 3 bytes 0004 4 bytes ... 01FF 511 bytes 0200 512 bytes ... 0800 2048 bytes 1000 4096 bytes

11.4.3 Command Argument Register (CMDARG)

The command argument register contains the SD/MMC command argument.

Offset: 0x008 (CMDARG) Access: Read/Write

	0–31
R W	CMDARG
Reset	All zeros

Figure 11-5. Command Argument Register (CMDARG)

Table 11-5. CMDARG Field Descriptions

Field	Description
0–31 CMDARG	Command argument. The SD/MMC command argument is specified as bits 39–8 of the command format in the SD or MMC Specification. If PRSSTAT[CMD] is set, this register is write-protected.

11.4.4 Transfer Type Register (XFERTYP)

The transfer type register controls the operation of data transfers. The host driver should set this register before issuing a command followed by a data transfer, or before issuing a resume command. To prevent data loss, the eSDHC prevents a write to the bits that are involved in the data transfer of this register while the data transfer is active.

The host driver should check PRSSTAT[CDIHB] and PRSSTAT[CIHB] before writing to this register.

- If PRSSTAT[CDIHB] is set, any attempt to send a command with data by writing to this register is ignored.
- If PRSSTAT[CIHB] is set, any write to this register is ignored.

Offset: 0x00C (XFERTYP) Access: Read/Write

	0–1	2–7	8–9	10	11	12	13	14–15	16–25	26	27	28	29	30	31
R W	—	CMDINX	CMDTYP	DPSEL	CICEN	CCCEN	—	RSPTYP	—	MSBSEL	DTDSEL	—	AC12EN	BCEN	DMAEN
Reset	All zeros														

Figure 11-6. Transfer Type Register (XFERTYP)

Table 11-6. XFERTYP Field Descriptions

Field	Description
0–1	Reserved
2–7 CMDINX	Command index. These bits should be set to the command number (CMD0–63, ACMD0–63) that is specified in bits 45–40 of the command format in the *SD Memory Card Physical Layer Specification*.

Table 11-6. XFERTYP Field Descriptions (continued)

Field	Description
8–9 CMDTYP	Command type. There are three types of special commands: suspend, resume, and abort. Clear this bit field for all other commands. • Suspend command. If the suspend command succeeds, the eSDHC assumes the SD bus has been released and it is possible to issue the next command which uses the SDHC_DAT line. The eSDHC de-asserts read wait for read transactions and stops checking busy for write transactions. In 4-bit mode, the interrupt cycle starts. If the suspend command fails, the eSDHC maintains its current state, and the host driver should restart the transfer by setting PROCTL[CREQ]. The eSDHC does not check if the suspend command succeeds or not. It is the host driver's responsibility to issue a normal CMD52 marked as suspend command when the suspend request is accepted by the card, so that eSDHC can be informed that the SD bus is released and de-assert read wait during read operation. • Resume command. The host driver restarts the data transfer by restoring the registers saved before sending the suspend command and sends the resume command. The eSDHC checks for pending busy state before starting write transfers. • Abort command. If this command is set when executing a read transfer, the eSDHC stops reads to the buffer. If this command is set when executing a write transfer, the eSDHC stops driving the SDHC_DAT line. After issuing the abort command, the host driver should issue a software reset. (Abort transaction) 00 Normal—other commands 01 Suspend—CMD52 for writing bus suspend in the common card control register (CCCR) 10 Resume—CMD52 for writing function select in CCCR 11 Abort—CMD12, CMD52 for writing I/O abort in CCCR
10 DPSEL	Data present select. Set to indicate that data is present and should be transferred using the SDHC_DAT line. It is cleared for the following: • Commands using only the SDHC_CMD line (e.g. CMD52) • Commands with no data transfer but using busy signal on the SDHC_DAT[0] line (R1b or R5b, e.g. CMD38) **Note:** In resume command, this bit should be set while the other bits in this register should be set the same as when the transfer initially launched. 0 No data present 1 Data present
11 CICEN	Command index check enable. 0 Disable. The index field is not checked. 1 Enable. The eSDHC checks the index field in the response to see if it has the same value as the command index. If it is not, it is reported as a command index error.
12 CCCEN	Command CRC check enable. The number of bits checked by the CRC field value changes according to the length of the response. (Refer to RSPTYP[1:0] and Table 11-8.) 0 Disable. The CRC field is not checked. 1 Enable. The eSDHC checks the CRC field in the response if it contains the CRC field. If an error is detected, it is reported as a command CRC error.
13	Reserved
14–15 RSPTYP	Response type select. 00 No response 01 Response length 136 10 Response length 48 11 Response length 48 check busy after response
16–25	Reserved

Table 11-6. XFERTYP Field Descriptions (continued)

Field	Description
26 MSBSEL	Multi/single block select. Enables multiple block SDHC_DAT line data transfers. For any other commands, this bit should be cleared. If this bit is cleared, it is not necessary to set the block count register. (Refer to Table 11-7.) 0 Single block 1 Multiple blocks
27 DTDSEL	Data transfer direction select. Defines the direction of SDHC_DAT line data transfers. The bit is set by the host driver to transfer data from the SD card to the eSDHC and it is cleared for all other commands. 0 Write (host to card) 1 Read (card to host)
28	Reserved
29 AC12EN	Auto CMD12 enable. Multiple block transfers for memory require CMD12 to stop the transaction. If this bit is set, the eSDHC issues CMD12 automatically when the last block transfer is completed. The host driver should not set this bit to issue commands that do not require CMD12 to stop a multiple block data transfer. In particular, secure commands defined in the Part 3 File Security specification do not require CMD12. In a single block transfer, the eSDHC ignores this bit. 0 Disable 1 Enable
30 BCEN	Block count enable. Enables the block attributes register, which is only relevant for multiple block transfers. When this bit is cleared, the block attributes register is disabled, which is useful in executing an infinite transfer. 0 Disable 1 Enable
31 DMAEN	DMA enable. Enables DMA functionality as described in Section 11.5.2, “DMA CSB Interface.” If this bit is set, a DMA operation should begin when the host driver writes to the CMDINX field of the transfer type register. 0 Disable 1 Enable

Table 11-7 shows how register settings determine types of data transfers.

Table 11-7. Determination of Transfer Type

Multi/Single Block Select XFERTYP[MSBSEL]	Block Count Enable XFERTYP[BCEN]	Block Count BLKATTR[BLKCNT]	Function
0	Don’t Care	Don’t Care	Single Transfer
1	0	Don’t Care	Infinite Transfer
1	1	Positive Number	Multiple Transfer
1	1	Zero	No Data Transfer

Table 11-8 shows how the response type can be determined by the command index check enable, command CRC check enable, and response type bits.

Table 11-8. Relation Between Parameters and Name of Response Type

Response Type XFERTYP[RSPTYP]	Index Check Enable XFERTYP[CICEN]	CRC Check Enable XFERTYP[CCCEN]	Response Type
00	0	0	No Response
01	0	1	R2
10	0	0	R3, R4
10	1	1	R1, R5, R6
11	1	1	R1b, R5b

NOTE

The CRC field for R3 and R4 is expected to be all 1s. The CRC check should be disabled for these response types.

11.4.5 Command Response 0–3 (CMDRSP0–3)

The command response registers stores the four parts of the response bits from the card.

Offset: 0x010 (CMDRSP0)
0x014 (CMDRSP1)
0x018 (CMDRSP2)
0x01C (CMDRSP3)

Access: Read

	0 … 31
R	CMDRSP
W	
Reset	All zeros

Figure 11-7. Command Response 0–3 Register (CMDRSP*n*)

Table 11-9 describes the mapping of command responses from the SD bus to the command response registers for each response type. In the table, R[ ] refers to a bit range within the response data as transmitted on the SD bus.

Table 11-9. Response Bit Definition for Each Response Type

Response Type	Meaning of Response	Response Field	Response Register
R1,R1b (normal response)	Card status	R[39:8]	CMDRSP0
R1b (Auto CMD12 response)	Card status for Auto CMD12	R[39:8]	CMDRSP3
R2 (CID, CSD register)	CID/CSD register [127:8]	R[127:8]	{CMDRSP3[23:0], CMDRSP2, CMDRSP1, CMDRSP0}
R3 (OCR register)	OCR register for memory	R[39:8]	CMDRSP0

Table 11-9. Response Bit Definition for Each Response Type (continued)

Response Type	Meaning of Response	Response Field	Response Register
R4 (OCR register)	OCR register for I/O etc.	R[39:8]	CMDRSP0
R6 (publish RCA)	New published RCA[31:16] and card status[15:0]	R[39:8]	CMDRSP0

This table shows that:

- Most responses with a length of 48 (R[47:0]) have 32 bits of the response data (R[39:8]) stored in the CMDRSP0 register.
- Responses of type R1b (Auto CMD12 responses) have response data bits R[39:8] stored in the CMDRSP3 register.
- Responses with length 136 (R[135:0]) have 120 bits of the response data (R[127:8]) stored in the CMDRSP0, 1, 2, and 3 registers.

To be able to read the response status efficiently, the eSDHC only stores part of the response data in the command response registers. This enables the host driver to efficiently read 32 bits of response data in one read cycle on a 32-bit bus system. Parts of the response, the index field, and the CRC are checked by the eSDHC (as specified by XFERTYP[CICEN, CCCEN]) and generate an error interrupt if any error is detected. The bit range for the CRC check depends on the response length. If the response length is 48, the eSDHC checks R[47:1], and if the response length is 136, the eSDHC checks R[119:1].

Since the eSDHC may have a multiple block data transfer executing concurrently with a CMD_wo_DAT command, the eSDHC stores the Auto CMD12 response in the CMDRSP3 register and the CMD_wo_DAT response is stored in CMDRSP0. This allows the eSDHC to avoid overwriting the Auto CMD12 response with the CMD_wo_DAT and vice versa. When the eSDHC modifies part of the command response registers it preserves the unmodified bits.

11.4.6 Buffer Data Port Register (DATPORT)

The buffer data port register is a 32-bit data port register used to access the internal buffer.

NOTE

When the internal DMA is not enabled and a write transaction is in operation, DATPORT must not be read.

Offset: 0x020 (DATPORT) Access: Read/Write

	0–31
R / W	DATCONT
Reset	All zeros

Figure 11-8. Buffer Data Port Register (DATPORT)

Table 11-10. DATPORT Field Descriptions

Field	Description
0–31 DATCONT	Data content. The buffer data port register is for 32-bit data access by the CPU or an external DMA. When the internal DMA is enabled, any write to this register is ignored, and a read from this register always yields 0.

11.4.7 Present State Register (PRSSTAT)

PRSSTAT indicates the status of the eSDHC to the host driver.

Offset: 0x024 (PRSSTAT) Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	—				DLSL				CLSL	—			WPSPL	CDPL	—	CINS
W																
Reset	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	—				BREN	BWEN	RTA	WTA	SD OFF	PER OFF	HCK OFF	IPG OFF	—	DLA	CDIHB	CIHB
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-9. Present State Register (PRSSTAT)

Table 11-11. PRSSTAT Field Descriptions

Field	Description
0–3	Reserved
4–7 DLSL	SDHC_DAT[3:0] line signal level. These bits are used to check the SDHC_DAT line level to recover from errors, and for debugging.This is especially useful in detecting the busy signal level from SDHC_DAT[0]. The reset value is affected by the external pull resistors. By default, read value of this bit field after reset is 0111, when SDHC_DAT[3] is pull-down and other lines are pull-up. PRSSTAT Bit / SDHC_DAT*n*: 4 / 3; 5 / 2; 6 / 1; 7 / 0
8 CLSL	SDHC_CMD line signal level. This status is used to check the SDHC_CMD line level to recover from errors, and for debugging. The reset value is affected by the external pull resistor, by default, read value of this bit after reset is 1, when the command line is pull-up.
9–11	Reserved

Nested table within DLSL description:

PRSSTAT Bit	SDHC_DAT*n*
4	3
5	2
6	1
7	0

Table 11-11. PRSSTAT Field Descriptions (continued)

Field	Description
12 WPSPL	Write protect switch pin level. The write protect switch is supported for memory and combo cards.This bit reflects the SDHC_WP pin of the card socket. A software reset does not affect this bit. The reset value is affected by the external write protect switch. If the SDHC_WP pin is not used, it should be tied to 0 so that the reset value of this bit is 0 and write is enabled. 0 Write enabled (SDHC_WP = 0) 1 Write protected (SDHC_WP = 1)
13 CDPL	Card detect pin level. This bit reflects the inverse value of the SDHC_CD pin for the card socket. Debouncing is not performed on this bit. This bit may be valid, but it is not guaranteed because of a propagation delay. Use of this bit is limited to testing since it must be debounced by software. A software reset does not affect this bit. Write to the force event register does not affect this bit. The reset value is affected by the external card detection pin. If this bit is not used, it should be tied to 0. 0 No card present (SDHC_CD = 0) 1 Card present (SDHC_CD = 1)
14	Reserved
15 CINS	Card inserted. Indicates if a card has been inserted. The eSDHC debounces this signal so that the host driver does not need to wait for it to stabilize. Changing from 0 to 1 generates a card-insertion interrupt in the interrupt status register and changing from 1 to 0 generates a card removal interrupt in the interrupt status register. A write to the force event register does not affect this bit. The software reset for all in the system control register does not affect this bit. A software reset does not affect this bit. 0 Power-on-reset or no card 1 Card inserted
16–19	Reserved
20 BREN	Buffer read enable. This status is used for non-DMA read transfers. The eSDHC may implement multiple buffers to transfer data efficiently. This read-only flag indicates that a burst-length of valid data exists in the host-side buffer. When the buffer is read, this bit is cleared. When a burst length of data is ready in the buffer, this bit is set and a buffer read ready interrupt is generated (if the interrupt is enabled). 0 Buffer read disable 1 Buffer read enable
21 BWEN	Buffer write enable. This status is used for non-DMA write transfers. The eSDHC can implement multiple buffers to transfer data efficiently. This read-only flag indicates if space is available for a burst length of write data. When the buffer is written, this bit is cleared. When a burst length of data is written to the buffer, this bit is set and a buffer write ready interrupt is generated (if the interrupt is enabled). 0 Buffer write disable 1 Buffer write enable
22 RTA	Read transfer active. This status is used for detecting completion of a read transfer. This bit is set for either of the following conditions: • After the end bit of the read command • When writing a 1 to PROCTL[CREQ] to restart a read transfer This bit is cleared for either of the following conditions: • When the last data block as specified by block length is transferred to the system • When all valid data blocks have been transferred to the system and no current block transfers are being sent as a result of PROCTL[SABGREQ] being set. A transfer complete interrupt is generated when this bit changes to 0. 0 No valid data 1 Transferring data

Table 11-11. PRSSTAT Field Descriptions (continued)

Field	Description
23 WTA	Write transfer active. This status indicates a write transfer is active. If this bit is 0, it means no valid write data exists in eSDHC. This bit is set in either of the following cases: • After the end bit of the write command. • When writing a 1 to PROCTL[CREQ] to restart a write transfer. This bit is cleared in either of the following cases: • After getting the CRC status of the last data block, as specified by the transfer count (single and multiple) • After getting the CRC status of any block where data transmission is about to be stopped by a stop-at-block-gap request. During a write transaction, a IRQSTAT[BGE] interrupt is generated when this bit is changed to 0, as result of PROCTL[SABGREQ] being set. This status is useful for the host driver in determining when to issue commands during write busy. 0 No valid data 1 Transferring data
24 SDOFF	SD clock gated off internally. Indicates the SD clock is internally gated off because of a buffer overrun, buffer underrun, or a read pause without read-wait assertion. This bit is for the host driver to debug data transaction on SD bus.
25 PEROFF	The internal bus clock gated off internally. This status bit indicates the internal bus clock is internally gated off. This bit is for the host driver to debug a transaction on SD bus.
26 HCKOFF	Master clock gated off internally. This status bit indicates master clock is internally gated off. This bit is for the host driver to debug a data transfer.
27 IPGOFF	Controller clock gated off internally. Indicates that the controller clock is internally gated off. This bit is for the host driver to debug. The controller clock runs at csb_clk / SCCR[ESDHCCM].
28	Reserved

Table 11-11. PRSSTAT Field Descriptions (continued)

Field	Description
29 DLA	Data line active. Indicates whether one of the SDHC_DAT line on SD bus is in use. For read transactions, this bit indicates if a read transfer is executing on the SD bus. Clearing this bit from 1 to 0 between data blocks generates a block gap event interrupt. This bit is set in either of the following cases: • After the end bit of the read command • When writing a 1 to PROCTL[CREQ] to restart a read transfer This bit is cleared in either of the following cases: • When the end bit of the last data block is sent from the SD bus to the eSDHC • When beginning a read wait transfer initiated by a stop at block gap request The eSDHC waits at the next block gap by driving read wait at the start of the interrupt cycle. If the read-wait signal is already driven (data buffer cannot receive data), the eSDHC can wait for current block gap by continuing to drive the read-wait signal. It is necessary to support read wait in order to use the suspend/resume function. For write transactions, this bit indicates that a write transfer is executing on the SD bus. Clearing this bit from 1 to 0 generates a transfer complete interrupt. This bit is set in any of the following cases: • After the end bit of the write command • When writing a 1 to PROCTL[CREQ] to continue a write transfer This bit is cleared in any of the following cases: • When the SD card releases write-busy of the last data block, the eSDHC also detects if output is not busy. If the SD card does not drive the busy signal after CRC status is received, the eSDHC should consider the card drive not busy. • When the SD card releases write-busy prior to waiting for write transfer as a result of a stop at block gap request 0 SDHC_DAT line inactive 1 SDHC_DAT line active
30 CDIHB	Command inhibit (SDHC_DAT). This bit is set if the SDHC_DAT line is active, the read transfer active is set, or read wait is asserted. If this bit is cleared, it indicates the eSDHC can issue the next SD/MMC command. Commands with busy signal belong to command inhibit (SDHC_DAT) (e.g. R1b and R5b type). Clearing from 1 to 0 generates a transfer complete interrupt. **Note:** The SD host driver can save registers for a suspend transaction after this bit has cleared from 1 to 0. 0 Can issue command which uses the SDHC_DAT line 1 Cannot issue command which uses the SDHC_DAT line
31 CIHB	Command inhibit (SDHC_CMD). This bit is cleared, if the SDHC_CMD line is not in use and the eSDHC can issue a SD/MMC command using the SDHC_CMD line. This bit is set immediately after the XFERTYP register is written. This bit is cleared when the command response is received. Even if the CDIHB bit is set, commands using only the SDHC_CMD line can be issued if this bit is cleared. Clearing from 1 to 0 generates a command complete interrupt. If the eSDHC cannot issue the command because of a command conflict error (refer to command CRC error) or because of command not issued by Auto CMD12 error, this bit remains set and IRQSTAT[CC] is not set. Status issuing Auto CMD12 is not read from this bit. 0 Can issue command using only SDHC_CMD line 1 Cannot issue command

NOTE

The host driver can issue CMD0, CMD12, CMD13 (for memory) when the SDHC_DAT lines are busy during a data transfer. These commands can be issued when PRSSTAT[CIHB] is cleared. Other commands should be issued when PRSSTAT[CDIHB] is cleared. Possible changes to the SD physical specification may add other commands to this list in the future.

11.4.8 Protocol Control Register (PROCTL)

The protocol control register is shown in Figure 11-10.

Offset: 0x028 (PROCTL) Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	—					WECRM	WECINS	WECINT	—					RWCTL	CREQ	SABGREQ
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	—								CDSS	CDTL	EMODE		D3CD	DTW		—
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Figure 11-10. Protocol Control Register (PROCTL)

Table 11-12. PROCTL Field Descriptions

Field	Description
0–4	Reserved
5 WECRM	Wake-up event enable on SD card removal. This bit enables wakeup event via card removal assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit. 0 Disable 1 Enable
6 WECINS	Wake-up event enable on SD card insertion. This bit enables wakeup event via card insertion assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit. 0 Disable 1 Enable
7 WECINT	Wake-up event enable on card interrupt. This bit enables wakeup event via card interrupt assertion in the IRQSTAT register. This bit can be set to 1 if FN_WUS (wake-up support) in CIS is set to 1. 0 Disable 1 Enable
8–12	Reserved

Table 11-12. PROCTL Field Descriptions (continued)

Field	Description
13 RWCTL	Read wait control. If the card supports read wait, set this bit to enable the read wait protocol to stop read data using the SDHC_DAT[2] line. Otherwise, the eSDHC has to stop the SD clock to hold read data, which restricts command generation. If the card does not support read wait, this bit should never be set otherwise an SDHC_DAT line conflict may occur. If this bit is cleared, a stop-at-block-gap-during-read operation is also supported, but the eSDHC stops the SD clock to pause the reading operation. 0 Disable read-wait control, and stop SD clock at block gap when the SABGREQ bit is set 1 Enable read-wait control, and assert read wait without stopping the SD clock at block gap when PROCTL[SABGREQ] is set
14 CREQ	Continue request. Restarts a transaction which was stopped using the stop-at-block-gap request. To cancel the request, clear SABGREQ and set this bit to restart the transfer. The eSDHC automatically clears this bit in either of the following cases: • For a read transaction, the PRSSTAT[DLA] bit changes from 0 to 1 as a read transaction restarts. • For a write transaction, the PRSSTAT[WTA] bit changes from 0 to 1 as the write transaction restarts. Therefore, it is not necessary for the host driver to clear. If SABGREQ and this bit are set, the continue request is ignored. 0 No effect 1 Restart
15 SABGREQ	Stop at block gap request. Stops executing a transaction at the next block gap for both DMA and non-DMA transfers. Until the TC bit is set, indicating a transfer completion, the host driver should leave this bit set. Clearing SABGREQ and CREQ does not cause the transaction to restart. Read wait is used to stop the read transaction at the block gap. The eSDHC honors stop-at-block-gap request for write transfers. Otherwise, the eSDHC stops the SD bus clock to pause the read operation during the block gap. For write transfers in which the host driver writes data to the data port register, the host driver should set this bit after all block data is written. If this bit is set, the host driver should not write data to the DATPORT register after a block is sent. When this bit is set, the host driver should not clear this bit before IRQSTAT[TC] is set. Otherwise, the eSDHC behavior is undefined. Confirm that IRQSTAT[TC] is enabled. This bit affects PRSSTAT[RTA, WTA, DLA, CIHB]. 0 Transfer 1 Stop or not resume yet
16–23	Reserved
24 CDSS	Card detect signal selection. Selects the source for card detection. 0 Card detection level is selected (for normal purpose) 1 Card detection test level is selected (for test purpose)
25 CDTL	Card detect test level. Determines card insertion status when CDSS is set. 0 No card in the slot 1 Card is inserted
26–27 EMODE	Endian mode. eSDHC supports only address-invariant mode in data transfer. 00 Reserved 01 Reserved 10 Address-invariant mode. Each byte location in the main memory is mapped to the same byte location in the MMC/SD card. 11 Reserved

Table 11-12. PROCTL Field Descriptions (continued)

Field	Description
28 D3CD	SDHC_DAT3 as card detection pin. If this bit is set, SDHC_DAT3 should be pulled down to act as a card detection pin. Be cautious when using this feature, because SDHC_DAT3 is chip-select for SPI mode, and a pull-down on this pin and CMD0 may set the card into SPI mode, which the eSDHC does not support. 0 SDHC_DAT3 does not monitor card insertion 1 SDHC_DAT3 is card detection pin
29–30 DTW	Data transfer width. Selects the data width of the SD bus. The host driver should set it to match the data width of the card. 00 1-bit mode 01 4-bit mode 10 Reserved 11 Reserved
31	Reserved

There are three ways to restart the transfer after a stop at the block gap. The appropriate method depends on whether the eSDHC issues a suspend command or the SD card accepts the suspend command:

- If the host driver does not issue a suspend command, the continue request should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card accepts it, a resume command should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card does not accept it, PROCTL[CREQ] should be used to restart the transfer.

Any time PROCTL[SABGREQ] stops the data transfer, the host driver should wait for IRQSTAT[TC] before attempting to restart the transfer. When restarting the data transfer by continue request, the host driver should clear PROCTL[SABGREQ] before or simultaneously.

11.4.9 System Control Register (SYSCTL)

The system control register is shown in Figure 11-11.

Offset: 0x02C (SYSCTL) Access: Mixed

Bits	0–3	4	5	6	7	8–11	12–15
R	—	INITA				—	DTOCV
W			RSTD	RSTC	RSTA		
Reset	0 0 0 0	0	0	0	0	0 0 0 0	0 0 0 0

Bits	16–23	24–27	28	29	30	31
R	SDCLKFS	DVS	—	PEREN	HCKEN	IPGEN
W						
Reset	1 0 0 0 0 0 0 0	0 0 0 0	0	0	0	0

Figure 11-11. System Control Register (SYSCTL)

Table 11-13. SYSCTL Field Descriptions

Field	Description
0–3	Reserved
4 INITA	Initialization active. When this bit is written ‘1’, 80 SD clocks are sent to the card. After the 80 clocks are sent, this bit is self-cleared. This bit is very useful during the card power-up period when 74 SD clocks are needed and clock auto-gating feature is enabled. Writing one to this bit when it is already set has no effect. Clearing this bit at any time does not affect it. When PRSSTAT[CIHB] or PRSSTAT[CDIHB] is set, writing a one to this bit is ignored. That is, when the command line or data line is active, writing to this bit is not allowed.
5 RSTD	Software reset for SDHC_DAT line. The DMA and part of the data circuit are reset. The following registers and bits are cleared by this bit: • DATPORT register • Buffer is cleared and initialized; PRSSTAT register • PRSSTAT[BREN, BWEN, RTA, WTA, DLA, CDIHB] • PROCTL[CREQ, SABGREQ] • IRQSTAT[BRR, BWR, DINT, BGE, TC] 0 Work 1 Reset
6 RSTC	Software reset for SDHC_CMD line. Only part of the command circuit is reset. The following bits are cleared by this bit: • PRSSTAT[CIHB] • IRQSTAT[CC] 0 Work 1 Reset
7 RSTA	Software reset for all. This reset affects the entire host controller except for the card-detection circuit. Register bits of type ROC, RW, RW1C, and RWAC are cleared. During its initialization, the host driver should set this bit to reset the eSDHC. The eSDHC should clear this bit when capabilities registers are valid and the host driver can read them. Additional use of the this bit does not affect the value of the capabilities registers. After this bit is set, it is recommended the host driver reset the external card and re-initialize it. 0 Work 1 Reset
8–11	Reserved
12–15 DTOCV	Data timeout counter value. Determines the interval by which SDHC_DAT line timeouts are detected. Refer to the data timeout error Section 11.4.10, “Interrupt Status Register (IRQSTAT)”, for information on factors that dictate timeout generation. Timeout clock frequency is generated by dividing the base clock SDHC_CLK value by this value. When setting this register, prevent inadvertent timeout events by clearing IRQSTATEN[DTOESEN]. 0000 SDHC_CLK x 2^{13} 0001 SDHC_CLK x 2^{14} ... 1110 SDHC_CLK x 2^{27} 1111 Reserved

Table 11-13. SYSCTL Field Descriptions (continued)

Field	Description
16–23 SDCLKFS	SDHC_CLK frequency select. This field, together with DVS, selects the frequency of SDHC_CLK pin. This bit holds the prescaler of the base clock frequency. Only the following settings are allowed: 0x01 Base clock divided by 2 0x02 Base clock divided by 4 0x04 Base clock divided by 8 0x08 Base clock divided by 16 0x10 Base clock divided by 32 0x20 Base clock divided by 64 0x40 Base clock divided by 128 0x80 Base clock divided by 256 Multiple bits must not be set or the behavior of this prescaler is undefined. According to the SD Physical Specification version 1.1, the maximum SD clock frequency is 50 MHz, and should never exceed this limit. The frequency of SDHC_CLK is set by the following formula: $$\text{clock frequency} = (\text{base clock}) / [(\text{SDCLKFS} \times 2) \times (\text{DVS} + 1)] \qquad \text{Eqn. 11-1}$$ For example, if the base clock frequency is 96 MHz, and the target frequency is 25 MHz, then choosing the prescaler value of 0x1 and divisor value of 0x1 yields 24 MHz, which is the nearest frequency less than or equal to the target. Similarly, to approach a clock value of 400 KHz, the prescaler value of 0x04 and divisor value of 0xE yields the exact clock value of 400 KHz. The reset value of this bit field is 0x80. So, if the input base clock is about 96 MHz, the default SD clock after reset is 375 KHz. **Note:** The base clock frequency equals the csb_clk / SCCR[SDHCCM].
24–27 DVS	Divisor. Provides a more exact divisor to generate the desired SD clock frequency. The settings are as follows: 0x0 Divide by 1 0x1 Divide by 2 ... 0xE Divide by 15 0xF Divide by 16
28	Reserved
29 PEREN	Peripheral clock enable. If set, the peripheral clock is always active and no automatic gating is applied, thus SDHC_CLK is active only except auto gating-off during buffer danger. If cleared, the peripheral clock is automatically off when no transaction is on the SD bus. Clearing this bit does not stop SDHC_CLK immediately. The peripheral clock will be internally gated off, if none of the following factors are met: • Command part is reset • Data part is reset • Soft reset • Command is about to send • Clock divisor is just updated • Continue request is just set • This bit is set • Card insertion is detected • Card removal is detected • Card external interrupt is detected • 80 clocks for initialization phase is ongoing 0 The peripheral clock is internally gated off 1 The peripheral clock is not automatically gated off

Table 11-13. SYSCTL Field Descriptions (continued)

Field	Description
30 HCKEN	Master clock enable. If set, master clock is always active and no automatic gating is applied. If cleared, master clock is automatically off when no data transfer is on SD bus. **Note:** Master clock is the clock to the DMA engine and to the CSB interface logic. 0) Master clock is internally gated off 1) Master clock is not automatically gated off
31 IPGEN	Controller clock enable. If this bit is set, the controller clock is always active and no automatic gating is applied. The controller clock is internally gated off, if neither the following factors is met: • Command part is reset • Data part is reset • Soft reset • Command is about to send • Clock divisor is just updated • Continue request is just set • This bit is set • Card insertion is detected • Card removal is detected • Card external interrupt is detected • The controller clock is not gated off **Note:** The controller clock is not auto-gated off if the peripheral clock is not gated off. So, clearing this bit only does not take effect if SYSCTL[PEREN] is not cleared. 0 The controller clock is internally gated off 1 The controller clock is not automatically gated off

11.4.10 Interrupt Status Register (IRQSTAT)

An interrupt is generated when one of the status bits and its corresponding interrupt enable bit are set. For all bits, writing one to a bit clears it, while writing zero keeps the bit unchanged. More than one status can be cleared with a single register write. For a card interrupt (IRQSTAT[CINT]), the card must stop asserting the interrupt before writing one to clear. Otherwise, the CINT bit is set again.

Offset: 0x030 (IRQSTAT) Access: w1c

	0–2	3	4–6	7	8	9	10	11	12	13	14	15
R	—	DMAE	—	AC12E	—	DEBE	DCE	DTOE	CIE	CEBE	CCE	CTOE
W		w1c		w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros											

	16–22	23	24	25	26	27	28	29	30	31
R	—	CINT	CRM	CINS	BRR	BWR	DINT	BGE	TC	CC
W		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros									

Figure 11-12. Interrupt Status Register (IRQSTAT)

Table 11-14. IRQSTAT Field Descriptions

Field	Description
0–2	Reserved
3 DMAE	DMA error. Occurs when internal DMA transfer failed. This bit is set when some error occurs in the data transfer. The value in the DMA system address register is the next fetch address where the error occurs. Since any error corrupts the entire data block, the host driver should restart the transfer from the corrupted block boundary. The address of the block boundary can be calculated from the current DS_ADDR value or the remaining number of blocks and the block size. 0 No Error 1 Error
4–6	Reserved
7 AC12E	Auto CMD12 error. Occurs when one of the bits in AUTOC12ERR is set. This bit is also set when Auto CMD12 is not executed due to a previous command error. 0 No Error 1 Error
8	Reserved
9 DEBE	Data end bit error. Occurs when detecting 0 at the end bit position of read data on the SDHC_DAT line or at the end bit position of the CRC. 0 No Error 1 Error **Note:** When DEBE and CINT are set, the software should ignore DEBE. But, it must not ignore the other status bits. The software should also clear this bit by writing 1 to it. It is highly recommended to clear this bit before the next transfer.
10 DCE	Data CRC error. Occurs when detecting CRC error when transferring read data on the SDHC_DAT line or when detecting the write CRC status having a value other than 0b010. 0 No Error 1 Error
11 DTOE	Data timeout error. Occurs during one of following timeout conditions: • Busy timeout for R1b and R5b types • Busy timeout after write CRC status • Read data timeout 0 No error 1 Timeout
12 CIE	Command index error. Occurs if a command index error occurs in the command response. 0 No error 1 Timeout
13 CEBE	Command end bit error. Occurs when the end bit of a command response is 0. 0 No error 1 End bit error generated
14 CCE	Command CRC error. A command CRC error is generated in two cases: • If a response is returned and IRQSTAT[CTOE] is cleared (indicating no timeout), this bit is set when detecting a CRC error in the command response. • The eSDHC detects a SDHC_CMD line conflict by monitoring the SDHC_CMD line when a command is issued. If the eSDHC drives the SDHC_CMD line to 1, but detects 0 on the SDHC_CMD line at the next SDHC_CLK edge, then the eSDHC aborts the command (stop driving SDHC_CMD line) and sets this bit. The CTOE bit is also set to distinguish the SDHC_CMD line conflict. 0 No error 1 CRC error generated

Table 11-14. IRQSTAT Field Descriptions (continued)

Field	Description
15 CTOE	Command timeout error. Occurs if no response is returned within 64 SDHC_CLK cycles from the end bit of the command. Also, if eSDHC detects a SDHC_CMD line conflict, this bit is set along with IRQSTAT[CCE] as shown in Table 11-27. 0 No error 1 Time out
16–22	Reserved
23 CINT	Card interrupt. • In 1-bit mode, the eSDHC detects the card interrupt without the SD clock to support wakeup. • In 4-bit mode, the card interrupt signal is sampled during the interrupt cycle. So, there are some sample delays between the interrupt signal from the SD card and the interrupt to the host system. Writing 1 clears this bit. But, if the interrupt source from the SD card is not cleared, this bit is set again. To clear this bit, the SD card interrupt source must be cleared followed by writing 1 to this bit. When this bit is set and the host driver needs to start the interrupt service, IRQSIGEN[CINTIEN] should be cleared to stop driving the interrupt signal to the host system. After completing the card interrupt service, write 1 to clear this bit, set IRQSIGEN[CINTIEN], and start sampling the interrupt signal again. 0 No card interrupt 1 Generate card interrupt
24 CRM	Card removal. This bit is set if PRSSTAT[CINS] changes from 1 to 0. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated. When this bit is cleared, it is set again if no card is inserted. To leave it cleared, clear IRQSTATEN[CRMSEN]. 0 Card state unstable or inserted 1 Card removed
25 CINS	Card insertion. This bit is set if PRSSTAT[CINS] changes from 0 to 1. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated. When this bit is cleared, it is set again if a card has been inserted. To leave it cleared, clear IRQSTATEN[CINSEN]. 0 Card state unstable or removed 1 Card inserted
26 BRR	Buffer read ready. This bit is set if PRSSTAT[BREN] changes from 0 to 1. 0 Not ready to read buffer 1 Ready to read buffer
27 BWR	Buffer write ready. This bit is set if PRSSTAT[BWEN] changes from 0 to 1. 0 Not ready to write buffer 1 Ready to write buffer
28 DINT	DMA interrupt. Occurs when the internal DMA finishes the data transfer successfully. If errors occur during data transfer, this bit is not set. Instead, the DMAE bit is set. 0 No DMA interrupt 1 DMA interrupt is generated

Table 11-14. IRQSTAT Field Descriptions (continued)

Field	Description
29 BGE	Block gap event. If PROCTL[SABGREQ] is set, this bit is set when a read or write transaction is stopped at a block gap. If PROCTL[SABGREQ] is cleared, this bit is not set. During a read transaction, this bit is set at the falling edge of the SDHC_DAT line active status (when the transaction is stopped at SD bus timing). Read wait must be supported to use this function. During a write transaction, this bit is set at the falling edge of PRSSTAT[WTA] (after reading the CRC status at SD bus timing). 0 No block gap event 1 Transaction stopped at block gap
30 TC	Transfer complete. This bit is set when a read or write transfer is completed. For a read transaction, this bit is set at the falling edge of PRSSTAT[WTA]. There are two cases in which this interrupt is generated: • When a data transfer is completed, as specified by data length (after the last data has been read to the host system). • When data has stopped at the block gap and completed the data transfer by setting PROCTL[SABGREQ] (after valid data has been read to the host system). For a write transaction, this bit is set at the falling edge of PRSSTAT[DLA]. There are two cases in which this interrupt is generated: • When the last data is written to the SD card, as specified by data length and the busy signal is released. • When data transfers are stopped at the block gap by setting PROCTL[SABGREQ] and data transfers have completed (after valid data is written to the SD card and the busy signal is released).
31 CC	Command complete. This bit is set when the end bit of the command response is received (except Auto CMD12). Refer to PRSSTAT[CIHB]. 0 No command complete 1 Command complete

Table 11-15 below shows that command timeout error has higher priority than command complete. If both bits are set, it can be assumed that the response was not received correctly.

Table 11-15. Relation Between Command Timeout Error and Command Complete Status

Command Complete	Command Timeout Error	Meaning of the Status
0	0	—
Don’t Care	1	Response not received within 64 SDHC_CLK cycles
1	0	Response received

Table 11-16 below shows that transfer complete has higher priority than data timeout error. If both bits are set, the data transfer can be considered complete.

Table 11-16. Relation Between Data Timeout Error and Transfer Complete Status

Transfer Complete	Data Timeout Error	Meaning of the Status
0	0	—
0	1	Timeout occur during transfer
1	X	Data transfer complete

The relation between command CRC error and command timeout error is shown in Table 11-17 below.

Table 11-17. Relation Between Command CRC Error and Command Timeout Error

Command CRC Error	Command Timeout Error	Meaning of the Status
0	0	No error
0	1	Response Timeout Error
1	0	Response CRC Error
1	1	SDHC_CMD line conflict

11.4.11 Interrupt Status Enable Register (IRQSTATEN)

Setting the bits of IRQSTATEN enables the corresponding interrupt status bit to be set by the specified event. If any bit is cleared, the corresponding IRQSTAT bit is also cleared and is never set.

Offset: 0x034 (IRQSTATEN) Access: Read/Write

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	—			DMAESEN	—			AC12ESEN	—	DEBESEN	DCESEN	DTOESEN	CIESEN	CEBESEN	CCESEN	CTOESEN
Reset	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	1

Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	—							CINTSEN	CRMSEN	CINSSEN	BRRSEN	BWRSEN	DINTSEN	BGESEN	TCSEN	CCSEN
Reset	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1

Figure 11-13. Interrupt Status Enable Register (IRQSTATEN)

Table 11-18. IRQSTATEN Field Descriptions

Field	Description
0–2	Reserved
3 DMAESEN	DMA error status enable 0 Masked 1 Enabled
4–6	Reserved
7 AC12ESEN	Auto CMD12 error status enable 0 Masked 1 Enabled
8	Reserved
9 DEBESEN	Data end bit error status enable 0 Masked 1 Enabled

Table 11-18. IRQSTATEN Field Descriptions (continued)

Field	Description
10 DCESEN	Data CRC error status enable 0 Masked 1 Enabled
11 DTOESEN	Data timeout error status enable 0 Masked 1 Enabled
12 CIESEN	Command index error status enable 0 Masked 1 Enabled
13 CEBESEN	Command end bit error status enable 0 Masked 1 Enabled
14 CCESEN	Command CRC error status enable 0 Masked 1 Enabled
15 CTOESEN	Command timeout error status enable 0 Masked 1 Enabled
16–22	Reserved
23 CINTSEN	Card interrupt status enable. If this bit is cleared, the eSDHC clears the interrupt request to the system. The card interrupt detection is stopped when this bit is cleared and restarted when this bit is set. To prevent inadvertent interrupts, the host driver should clear this bit before servicing the card interrupt and should set this bit again after all interrupt requests from the card are cleared. 0 Masked 1 Enabled
24 CRMSEN	Card removal status enable 0 Masked 1 Enabled
25 CINSEN	Card insertion status enable 0 Masked 1 Enabled
26 BRRSEN	Buffer read ready status enable 0 Masked 1 Enabled
27 BWRSEN	Buffer write ready status enable 0 Masked 1 Enabled
28 DINTSEN	DMA interrupt status enable 0 Masked 1 Enabled
29 BGESEN	Block gap event status enable 0 Masked 1 Enabled

Table 11-18. IRQSTATEN Field Descriptions (continued)

Field	Description
30 TCSEN	Transfer complete status enable 0 Masked 1 Enabled
31 CCSEN	Command complete status enable 0 Masked 1 Enabled

NOTE

The eSDHC may sample the card interrupt signal during the interrupt period and hold its value in the flip-flop. As a result of synchronization, there is a delay in the card interrupt (which is asserted from the card) to the time the host system is informed.

To detect a SDHC_CMD line conflict, the host driver must set both CTOESEN and CCESEN bits.

11.4.12 Interrupt Signal Enable Register (IRQSIGEN)

IRQSIGEN selects which interrupt status is indicated to the host system as the interrupt. These status bits all share the same interrupt line. Setting any of these bits enables an interrupt generation. The corresponding status register bit generates an interrupt when the corresponding interrupt signal enable bit is set.

Offset: 0x038 (IRQSIGEN) Access: Read/Write

Bits	0–2	3	4–6	7	8	9	10	11	12	13	14	15
R/W	—	DMAE IEN	—	AC12E IEN	—	DEBE IEN	DCE IEN	DTOE IEN	CIE IEN	CEBE IEN	CCE IEN	CTOE IEN
Reset	All zeros											

Bits	16–22	23	24	25	26	27	28	29	30	31
R/W	—	CINT IEN	CRM IEN	CINS IEN	BRR IEN	BWR IEN	DINT IEN	BGE IEN	TC IEN	CC IEN
Reset	All zeros									

Figure 11-14. Interrupt Signal Enable Register (IRQSIGEN)

Table 11-19. IRQSIGEN Field Descriptions

Field	Description
0–2	Reserved
3 DMAEIEN	DMA error interrupt enable 0 Masked 1 Enabled
4–6	Reserved
7 AC12EIEN	Auto CMD12 error interrupt enable 0 Masked 1 Enabled
8	Reserved
9 DEBEIEN	Data end bit error interrupt enable 0 Masked 1 Enabled
10 DCEIEN	Data CRC error interrupt enable 0 Masked 1 Enabled
11 DTOEIEN	Data timeout error interrupt enable 0 Masked 1 Enabled
12 CIEIEN	Command index error interrupt enable 0 Masked 1 Enabled
13 CEBEIEN	Command end bit error interrupt enable 0 Masked 1 Enabled
14 CCEIEN	Command CRC error interrupt enable 0 Masked 1 Enabled
15 CTOEIEN	Command timeout error interrupt enable 0 Masked 1 Enabled
16–22	Reserved
23 CINTIEN	Card interrupt signal enable 0 Masked 1 Enabled
24 CRMIEN	Card removal interrupt enable 0 Masked 1 Enabled
25 CINSIEN	Card insertion interrupt enable 0 Masked 1 Enabled
26 BRRIEN	Buffer read ready interrupt enable 0 Masked 1 Enabled

Table 11-19. IRQSIGEN Field Descriptions (continued)

Field	Description
27 BWRIEN	Buffer write ready interrupt enable 0 Masked 1 Enabled
28 DINTIEN	DMA interrupt enable 0 Masked 1 Enabled
29 BGEIEN	Block gap event interrupt enable 0 Masked 1 Enabled
30 TCIEN	Transfer complete interrupt enable 0 Masked 1 Enabled
31 CCIEN	Command complete interrupt enable 0 Masked 1 Enabled

11.4.13 Auto CMD12 Error Status Register (AUTOC12ERR)

When IRQSTAT[AC12E] is set, the host driver checks this register to identify what kind of error Auto CMD12 indicated. This register is valid only when IRQSTAT[AC12E] is set.

Offset: 0x03C (AUTOC12ERR) Access: Read

	0–23	24	25–26	27	28	29	30	31
R	—	CNIB AC12E	—	AC12 IE	AC12 CE	AC12 EBE	AC12 TOE	AC12 NE
W								
Reset	All zeros							

Figure 11-15. Auto CMD12 Error Status Register (AUTOC12ERR)

Table 11-20. AUTOC12ERR Field Descriptions

Field	Description
0–23	Reserved
24 CNIBAC12E	Command not issued by Auto CMD12 error. This bit is set when CMD_wo_DAT is not executed due to an Auto CMD12 error (D04–D01). 0 No error 1 Not Issued
25–26	Reserved
27 AC12IE	Auto CMD12 index error. Occurs if the command index error occurs in response to a command. 0 No error 1 Error, the CMD index in response is not CMD12

Table 11-20. AUTOC12ERR Field Descriptions (continued)

Field	Description
28 AC12CE	Auto CMD12 CRC error. Occurs when detecting a CRC error in the command response. 0 No CRC error 1 CRC error met in Auto CMD12 response
29 AC12EBE	Auto CMD12 end bit error. Occurs when detecting that the end bit of command response is 0 when it should be 1. 0 No error 1 End bit error generated
30 AC12TOE	Auto CMD12 timeout error. Occurs if no response is returned within 64 SDHC_CLK cycles from the end bit of the command. If this bit is set, the other error status bits (2–4) are meaningless. 0 No error 1 Time out
31 AC12NE	Auto CMD12 not executed. If a memory multiple block data transfer is not started due to command error, this bit is not set because it is not necessary to issue Auto CMD12. Setting this bit means eSDHC cannot issue Auto CMD12 to stop the memory multiple block data transfer due to some error. If this bit is set, the other error status bits (1–4) are meaningless. 0 Executed 1 Not executed

Table 11-21. Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12

Auto CMD12 CRC Error	Auto CMD12 Timeout Error	Types of Error
0	0	No error
0	1	Response timeout error
1	0	Response CRC error
1	1	SDHC_CMD line conflict

There are three scenarios when AUTOC12ERR can be changed:

1. When eSDHC is going to issue Auto CMD12
   — Set AC12NE if Auto CMD12 cannot be issued due to an error in the previous command.
   — Clear AC12NE if Auto CMD12 is issued.
2. At the end bit of an Auto CMD12 response
   — Check received responses by checking the error bits 1–4.
   — Set if error is detected.
   — Clear if error is not detected.
3. Before reading AUTOC12ERR[CNIBAC12E]
   — Set CNIBAC12E if there is a command that cannot be issued
   — Clear CNIBAC12E if there is no command to issue

The timing of generating the Auto CMD12 error and writing to the command register is asynchronous. The command may be blocked by any Auto CMD12 error causing CNIBAC12E to be set. Therefore, it is suggested to read this register only when IRQSTAT[AC12E] is set. An Auto CMD12 error interrupt is generated when one of the error bits 0–4 is set1. The CNIBAC12E error bit does not generate an interrupt.

11.4.14 Host Controller Capabilities (HOSTCAPBLT)

The host controller capabilities provides the host driver with information specific to the eSDHC implementation. The value in this register does not change in a software reset, and any write to this register is ignored.

Offset: 0x040 (HOSTCAPBLT) Access: Read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	—					VS18	VS30	VS33	SRS	DMAS	HSS	—		MBL		
W																
Reset	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	—															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11-16. Host Capabilities Register (HOSTCAPBLT)

Table 11-22. HOSTCAPBLT Field Descriptions

Field	Description
0–4	Reserved
5 VS18	Voltage support 1.8 V. This bit depends on the host system ability. 0 1.8 V not supported 1 1.8 V supported
6 VS30	Voltage support 3.0 V. This bit depends on the host system ability. 0 3.0 V not supported 1 3.0 V supported
7 VS33	Voltage support 3.3 V. This bit depends on the host system ability. 0 3.3 V not supported 1 3.3 V supported
8 SRS	Suspend/resume support. Indicates if eSDHC supports suspend/resume functionality. If this bit is 0, the suspend and resume mechanism, as well as the read wait, are not supported and the host driver should not issue suspend or resume commands. 0 Not supported 1 Supported
9 DMAS	DMA support. Indicates if eSDHC is capable of using internal DMA to transfer data between system memory and the data buffer directly. 0 DMA not supported 1 DMA supported

Table 11-22. HOSTCAPBLT Field Descriptions (continued)

Field	Description
10 HSS	High speed support. Indicates if the eSDHC supports high speed mode and the host system can supply the SD clock frequency from 25 to 50 MHz. 0 High speed supported 1 High speed supported
11–12	Reserved
13–15 MBL	Max block length. Indicates the maximum block size that the host driver can read and write to the buffer in the eSDHC. The buffer should transfer block size without wait cycles. 000 512 bytes 001 1024 bytes 010 2048 bytes 011 4096 bytes
16–31	Reserved

11.4.15 Watermark Level Register (WML)

Both write and read watermark levels are configurable. The value can be any number from 1–128 words.

Offset: 0x044 (WML) Access: Read/Write

	0–7	8–15	16–23	24–31
R W	—	WR_WML	—	RD_WML
Reset	0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0

Figure 11-17. Watermark Level Register (WML)

Table 11-23. WML Field Descriptions

Field	Description
0–7	Reserved
8–15 WR_WML	Write watermark level. Number of 32-bit words of watermark level in DMA write operation. Also, the number of words of write burst length. **Note:** The minimum value is 0x02, which represents 2 words (8 bytes).
16–23	Reserved
24–31 RD_WML	Read watermark level. Number of 32-bit words of watermark level in DMA read operation. Also, the number of words of read burst length. **Note:** The minimum value for RD_WML is 0x02, which means 2 words (8 bytes), and the maximum value for RD_WML is 0x10, which means 16 words (64 bytes). Setting RD_WML to values outside this range results in non-predicted behavior.

11.4.16 Force Event Register (FEVT)

The force event register is not a physically implemented register. Rather, it is an address to which the IRQSTAT register can be written if the corresponding bit of IRQSTATEN is set. Therefore, this register is a write-only register and writing zero has no effect. Writing 1 to this register sets the corresponding bit of IRQSTAT. Reading from this register always returns zeroes.

Forcing a card interrupt generates a short pulse on the SDHC_DAT[1] line, and the driver may treat this interrupt as normal. The interrupt service routine may skip polling the card-interrupt source as the interrupt is self-cleared.

Offset: 0x050 (FEVT) Access: Write

Bits	0	1–2	3	4–6	7	8	9	10	11	12	13	14	15
R	0	—	0	—	0	—	0	0	0	0	0	0	0
W	FEVTCINT		FEVTDMAE		FEVTAC12E		FEVTDEBE	FEVTDCE	FEVTDTOE	FEVTCIE	FEVTCEBE	FEVTCCE	FEVTCTOE
Reset	All zeros												

Bits	16–23	24	25–26	27	28	29	30	31
R	—	0	—	0	0	0	0	0
W		FEVTCNIBAC12E		FEVTAC12IE	FEVTAC12EBE	FEVTAC12CE	FEVTAC12TOE	FEVTAC12NE
Reset	All zeros							

Figure 11-18. Force Event Register (FEVT)

Table 11-24. FEVT Field Descriptions

Field	Description
0 FEVTCINT	Force event card interrupt. Writing 1 to this bit generates a low-level short pulse on the internal SDHC_DAT[1] line, which imitates a self-clearing interrupt from the external card. If enabled, IRQSTAT[CINT] is set and the interrupt service routine may treat this interrupt as a normal interrupt from the external card.
1–2	Reserved
3 FEVTDMAE	Force event DMA error. Forces IRQSTAT[DMAE] to set.
4–6	Reserved
7 FEVTAC12E	Force event Auto CMD12 error. Forces IRQSTAT[AC12E] to set.
8	Reserved
9 FEVTDEBE	Force event data end bit error. Forces IRQSTAT[DEBE] to set.
10 FEVTDCE	Force event data CRC error. Forces IRQSTAT[DCE] to set.
11 FEVTDTOE	Force event data time out error. Forces IRQSTAT[DTOE] to set.
12 FEVTCIE	Force event command index error. Forces IRQSTAT[CCE] to set.

Table 11-24. FEVT Field Descriptions (continued)

Field	Description
13 FEVTCEBE	Force event command end bit error. Forces IRQSTAT[CEBE] to set.
14 FEVTCCE	Force event command CRC error. Forces IRQSTAT[CCE] to set.
15 FEVTCCE	Force event command time out error. Forces IRQSTAT[CTOE] to set.
16–23	Reserved
24 FEVTCNIBAC12E	Force event command not executed by Auto CMD12 error. Forces AUTOC12ERR[CNIBAC12E] to set.
25–26	Reserved
27 FEVTAC12IE	Force event Auto CMD12 index error. Forces AUTOC12ERR[AC12IE] to set.
28 FEVTAC12EBE	Force event Auto CMD12 end bit error. Forces AUTOC12ERR[AC12EBE] to set.
29 FEVTAC12CE	Force event Auto CMD12 CRC error. Forces AUTOC12ERR[AC12CE] to set.
30 FEVTAC12TOE	Force event Auto CMD12 time out error. Forces AUTOC12ERR[AC12TOE] to set.
31 FEVTAC12NE	Force event Auto CMD12 not executed. Forces AUTOC12ERR[AC12NE] to set.

11.4.17 Host Controller Version Register (HOSTVER)

The host controller version register contains the version for the vendor and the host controller. All the bits are read-only.

Offset: 0x0FC (HOSTVER) Access: Read

	0–15	16–23	24–31
R	—	VVN	SVN
W			
Reset	0000 0000 0000 0000	0000 0000	0000 0001

Figure 11-19. Host Controller Version Register (HOSTVER)

Table 11-25. HOSTVER Field Descriptions

Field	Description
0–15	Reserved
16–23 VVN	Vendor version number. The host driver should not use this status. The upper and the lower 4-bits indicate the version. 0x00 Freescale eSDHC version 1.0 0x01 Freescale eSDHC version 2.0 others Reserved
24–31 SVN	Specification version number. Indicates for the host controller specification version. The upper and the lower 4-bits indicate the version. 0x00 SD Host Specification Version 1.0 0x01 SD Host Specification Version 2.0, supports the test event register. others Reserved

11.4.18 DMA Control Register (DCR)

The DMA control register controls, shown in Figure 11-20, various settings that affect the system response to DMA operations.

Offset: 0x40C (DCR) Access: Read/Write

	0–15
R W	—
Reset	All zeros

	16–17	18	19–24	25	26–28	29	30	31
R W	PRI	CDP	—	SNOOP		RD_SAFE	RD_PFE	RD_PF_SIZE
Reset	All zeros							

Figure 11-20. DMA Control Register (DCR)

Table 11-26. DCR Field Descriptions

Field	Description
0–15	Reserved.
16–17 PRI	Priority. This field is used to present priority level for CSB arbitration for eSDHC DMA requests. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
18 CDP	Card detect polarity. 0 SDHC_CD level 1 = card present; PRSSTAT[CINS] = 1 and PRSSTAT[CDPL] = 1 SDHC_CD level 0 = no card present; PRSSTAT[CINS] = 0 and PRSSTAT[CDPL] = 0 1 SDHC_CD level 1 = no card present; PRSSTAT[CINS] = 0 and PRSSTAT[CDPL] = 0 SDHC_CD level 0 = card present; PRSSTAT[CINS] = 1 and PRSSTAT[CDPL] = 1
19–24	Reserved.

Table 11-26. DCR Field Descriptions (continued)

Field	Description
25 SNOOP	Snoop attribute. 0 DMA transactions are not snooped by the CPU data cache 1 DMA transactions are snooped by the CPU data cache
26–28	Reserved.
29 RD_SAFE	Read safe. This bit should be set only if the target of a read-DMA operation is a well-behaved memory that is not affected by the read operation and returns the same data if read again from the same location. This means that unaligned reading operation can be rounded up to enable more efficient read operations. 0 It is not safe to read more bytes that were intended 1 It is safe to read more bytes that were intended
30 RD_PFE	Read prefetch enable. This bit should be set if the target of read-DMA operation is a well-behaved memory that is not affected by the read operation and returns the same data if read again from the same location. This means that prefetch of data can be done by the internal bus units and it results in faster read completion. 0 It is not allowed to prefetch data on DMA read operation 1 It is allowed to prefetch data on DMA read operation
31 RD_PF_SIZE	Read prefetch size. Determines the prefetch byte count to be used if RD_PFE is set. 0 64 bytes prefetch 1 32 bytes prefetch

11.5 Functional Description

The following sections provide a brief functional description of the major system blocks, including the data buffer, DMA CSB interface, register bank, register bus interface, dual-port memory wrapper, data/command controller, clock and reset manager, and clock generator.

11.5.1 Data Buffer

The eSDHC uses one configurable data buffer so that data can be transferred between the internal system bus (register bus or CSB bus) and the SD card in an optimized manner to maximize throughput between the two clock domains (the IP peripheral clock and the master clock). See Figure 11-21 for an illustration of the buffer scheme.

The buffer is used as temporary storage for data being transferred between the host system and the card. The burst lengths for read and write are both configurable and can be any value between 1 and 128 words.

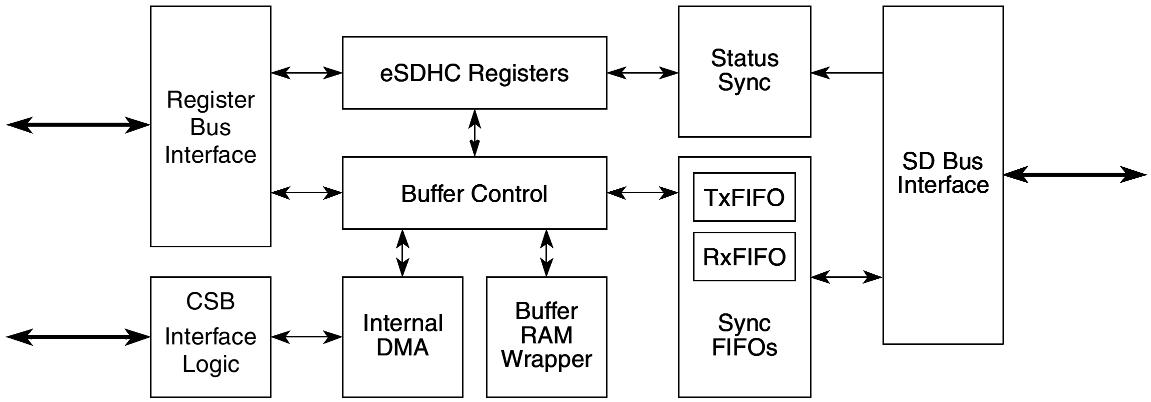


Figure 11-21. eSDHC Buffer Scheme

For a host read operation, when the amount of data exceeds the RD_WML value, the eSDHC sets PRSSTAT[BREN] and either:

- Issues a DMA request to inform the system to read the data
- Issues a DMA interrupt to inform the system to read the data
- When granted CSB access permission, the internal DMA burst-reads RD_WML number of words

Conversely, for a host write operation, when the amount of buffer spaces exceeds the WR_WML value, the eSDHC sets PRSSTAT[BWEN] and either:

- Issues a DMA request to inform the system to write data to the buffer
- Issues a DMA interrupt to inform the system to write data to the buffer
- When granted CSB access permission, the internal DMA burst-writess WR_WML number of words into the buffer

11.5.1.1 Write Operation Sequence

There are two ways to write data into the buffer when the user transfers data to the card:

- Processor core polling IRQSTAT[BWR] (interrupt or polling)
- Internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), the eSDHC asserts an external DMA request when more than WML[WR_WML] number of empty buffer word slots are available and ready for receiving new data. At the same time, the eSDHC sets IRQSTAT[BWR]. The buffer write ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferredand no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the write operation from that address. It is recommended that a software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[WR_WML] number of words of data can be held in the buffer. If the buffer is empty and the host system does not write data in time, the eSDHC stops the SDHC_CLK to avoid a data buffer underrun situation.

11.5.1.2 Read Operation Sequence

There are two ways to read data from the buffer when transferring data to the card:

- Processor core polling IRQSTAT[BRR] (interrupt or polling)
- Internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), the eSDHC asserts a DMA request when more than WML[RD_WML] number of words are available and ready for the system to fetch the data. At the same time, the eSDHC sets the IRQSTAT[BRR] bit. The buffer read ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferred and no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the read operation from that address. It is recommended that a software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[RD_WML] number of words of data are in the buffer. If the buffer is full and the host system does not read the data in time, the eSDHC stops the SDHC_CLK to avoid a data buffer overrun situation.

11.5.1.3 Data Buffer Size

To use the buffer in the most optimized way, the buffer size must be known. In the eSDHC the data buffer can hold up to 128 32-bit words, and the read and write watermark levels are each configurable from 1–128 words. The host driver may configure the values according to the system situation and requirements.

During multi-block data transfer, the maximum block length is 4096 bytes, which can satisfy all the requirements from MMC and SD cards. Any block length less than this value is also allowed. The only restriction is from the external card since it may not support such a large block or a partial block access that is not an integer multiple of 512 bytes.

11.5.2 DMA CSB Interface

The internal DMA implements a DMA engine and CSB master. When the internal DMA is enabled (XFERTYP[DMAEN] is set), the buffer interrupt status bits are still set if they are enabled. To avoid setting them, clear IRQSTATEN[BWRSEN, BRRSEN]. See Figure 11-22 for illustration of the DMA CSB interface block.

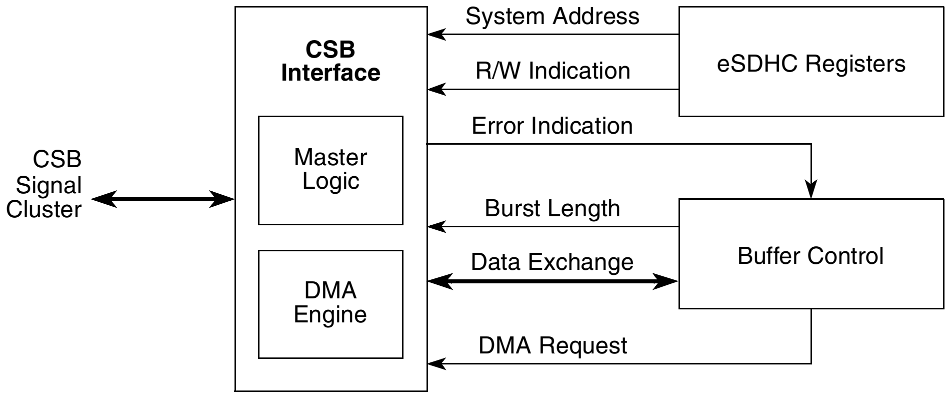


Figure 11-22. DMA CSB Interface Block

11.5.2.1 Internal DMA Request

If the watermark level is met in the data transfer and the internal DMA is enabled, the data buffer block sends a DMA request to this block. Meanwhile, the external DMA request is disabled. The delay of response from the internal DMA engine depends on the system bus loading and the priority assigned to eSDHC. The DMA engine does not respond to the request during its burst transfer, and is available as soon as the burst is over. The data buffer deasserts the request once an access on the buffer is made. Upon access to the buffer by the internal DMA, the data buffer updates its internal buffer pointer and when the watermark level is satisfied, another DMA request is sent.

The data transfer is in the block unit and the last watermark level is always set to the remaining number of words. For instance, for a multi-block data read with each block size of 31 bytes, the burst length is set at six words. After the first burst transfer, if there are more than seven bytes in the buffer, which might be partly some data of the next block, another DMA read request is sent because the remaining number of words to send for the current block is $(31 - 6 \times 4) \div 4 = 2$, and eSDHC reads two words out of the buffer, with seven valid bytes and one stuff byte automatically added by eSDHC.

11.5.2.2 DMA Burst Length

Just like the CPU polling access, the DMA burst length for the internal DMA engine does not a restriction other than the maximum size. The burst length for read or write can be 1–128 words. The actual burst length for the DMA depends on which is smaller: configured watermark level or the remaining words of current block.

Take the example in Section 11.5.2.1, “Internal DMA Request,” again. After six words are read, the burst length is two words to complete the 31-byte block. The burst length then changes back to six words to prepare for the next 31-byte block. The host driver writer may take this variable burst length into account. It is also acceptable to configure the burst length as the divisor of block size so that each time the burst length is the same.

11.5.2.3 CSB Master Interface

It is possible that the internal DMA engine fails during the data transfer. When an error occurs, the DMA engine stops the transfer and goes to the idle state, while the internal data buffer stops working, too. IRQSTAT[DMAE] is set to inform the driver.

Once the IRQSTAT[DMAE] interrupt is received, software should send CMD12 to abort the current transfer and read DSADDR[DS_ADDR] to obtain the start address of the corrupted block. After the DMA error is fixed, the software should apply a data reset and restart the transfer from this address to recover the corrupted block.

11.5.3 SD Protocol Unit

The SD protocol unit deals with all SD protocol affairs and performs the following:

- Acts as the bridge between internal buffer and the SD bus
- Sends the command data and its argument serially
- Stores the serial response bit stream into corresponding registers
- Detects bus state on SDHC_DAT[0] line
- Asserts read wait signal
- Gates off SD clock when the buffer announces danger status
- Detects write-protect state
- And other functions

It consists of four submodules: SD transceiver, SD clock and monitor, command agent and data agent.

11.5.3.1 SD Transceiver

In the SD protocol unit, the transceiver is the main control module. It consists of a FSM and the control module, from which the control signals for all other three modules are generated.

11.5.3.2 SD Clock and Monitor

This module monitors the signal level on all four data lines and the command lines, directly route the level values into the register bank for the driver to debug with.

The transceiver reports the card insertion state according to the SDHC_CD state, or signal level on SDHC_DAT[3] line when PROCTL[D3CD] is set.

The module detects the SDHC_WP (write protect) line. With the information of SDHC_WP state, the register bank ignores the command accompanied by write operation, when the SD_WP switch is on.

If the internal data buffer is in danger and the SD clock must be gated off to avoid buffer over/underrun, this module asserts the gate of output SD clock to shut the clock off. When the buffer danger is eliminated when system access of the buffer catches up, the clock gate of this module is open and the SD clock is active again.

11.5.3.3 Command Agent

The command agent deals with the transactions on SDHC_CMD line. See Figure 11-23 for illustration of the structure for the command CRC shift register.

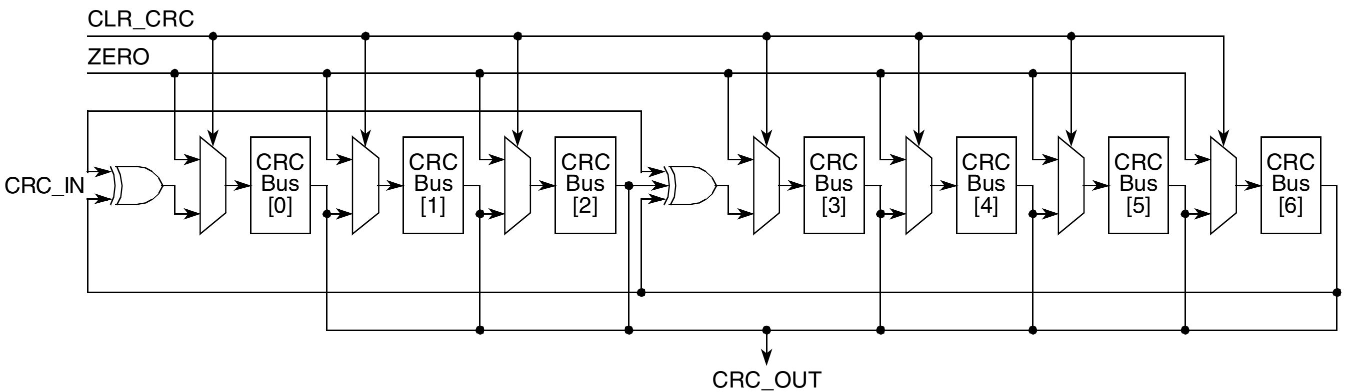


Figure 11-23. Command CRC Shift Register

The CRC polynomials for the SDHC_CMD are as follows:

Generator polynomial: $G(x) = x^7 + x^3 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + \ldots + (\text{last bit}) \times x^0$

$CRC[6:0] = \text{Remainder } [(M(x) \times x^7) \div G(x)]$

11.5.3.4 Data Agent

The data agent handles the transactions on the four data lines. Moreover, this module also detects the busy state from on SDHC_DAT[0] line, and generates read wait state by the request from the transceiver. The CRC polynomials for the SDHC_DAT are as follows:

Generator polynomial: $G(x) = x^{16} + x^{12} + x^5 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + \ldots + (\text{last bit}) \times x^0$

$CRC[15:0] = \text{Remainder } [(M(x) \times x^{16}) \div G(x)]$

11.5.4 Clock & Reset Manager

This module controls all the reset signals within the eSDHC. There are four types of reset signals within eSDHC: hardware reset, software reset for all, software reset for data, and software reset for command. All these signals are fed into this module and stable signals are generated inside the module to reset all other modules.

This module also gates off all the inside signals. The module monitors the activities of all other modules, supplies the clocks for them, and when enabled, automatically gates off the corresponding clocks.

11.5.5 Clock Generator

The clock generator generates the SDHC_CLK by dividing the internal bus clock into two stages. Refer to Figure 11-24 for the structure of the divider, in which the term base represents the frequency of the internal bus clock. Refer to SYSCTL[SDCLKFS] and SYSCTL[DVS] (see Section 11.4.9, “System Control Register (SYSCTL)”) to select the divisor values.

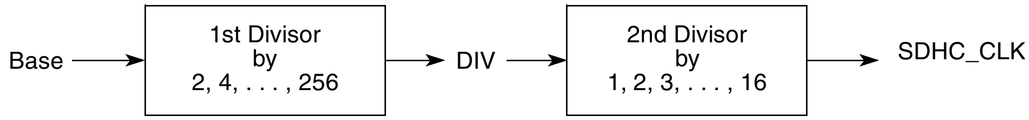


Figure 11-24. Two Stages of Clock Divider

The first stage is a prescaler. The frequency of clock output from this stage, DIV, can be base, base/2, base/4, ..., or base/256.

The second stage outputs the actual clock, SDHC_CLK, as the driving clock for all sub-modules of SD protocol unit, and the sync FIFOs in Figure 11-21 to synchronize with the data rate from the internal data buffer. It can be div, div/2, div/3,..., or div/16. Thus, the highest frequency of SDHC_CLK generated by the internal bus clock is base while the lowest frequency is base/4096.

11.5.6 Card Insertion and Removal Detection

The eSDHC uses the SDHC_DAT[3] pin or the SDHC_CD pin to detect card insertion or removal. When SDHC_DAT[3] pin is used for card detection, user needs to pull-down this pad as a default state. When there is no card on the MMC/SD bus, the SDHC_DAT[3] is pulled to a low voltage level by default. When any card is inserted to or removed from the socket, the eSDHC detects the logic value changes on the SDHC_DAT[3] pin and generates an interrupt.

When SDHC_DAT[3] pin is not used for card detection, SDHC_CD must be connected for card detection. It may be implemented by a GPIO. Whether SDHC_DAT[3] is configured for card detection or not, SDHC_CD is always a reference for card detection, either SDHC_DAT[3] or SDHC_CD reports card inserted, the eSDHC informs the host system that a card is inserted, and the interrupt is sent if it is enabled.

11.5.7 Power Management and Wake-Up Events

When there is no operation between eSDHC and the card through SD bus, you can completely disable the internal clocks in the chip level clock control module to save power. When you need to use the eSDHC to communicate with the card, it can enable the clock and start the operation. This can be done by clearing the SCCR[SDHCCM] bits.

In some circumstances, when the clocks to eSDHC are disabled, or when system is in low power mode, there are some events when you need to enable the clock and handle the event. These events are called wakeup interrupts. The eSDHC can generate these interrupts even there are no clocks enabled. The three interrupts which can be used as wake-up events are:

- Card removal interrupt
- Card insertion interrupt

These three wake-up events (or wake-up interrupts) can be also used to wake up the system from low-power modes.

NOTE

To make the interrupt as wakeup event when all the clocks to eSDHC are disabled or when whole system is in low power mode, the corresponding wakeup enable bit need to be set. Refer to Section 11.4.8, “Protocol Control Register (PROCTL),” for more information on the eSDHC PROCTL register.

11.5.7.1 Setting Wake Up Events

For the eSDHC to respond to a wake up event, the software must set the respective wake up enable bit before the CPU enters sleep mode. Refer to Section 11.4.8, “Protocol Control Register (PROCTL),” for more information on the wakeup enable bits.

Before the software disables the host clock, it should ensure that all of the following conditions have been met:

- No read or write transfer is active
- Data and command lines are not active
- No interrupts are pending
- Internal data buffer is empty

11.6 Initialization/Application Information

All communication between system and cards are controlled by the host. The host sends commands of two types: broadcast and addressed (point-to-point) commands.

Broadcast commands are intended for all cards, such as GO_IDLE_STATE, SEND_OP_COND, ALL_SEND_CID, etc. In broadcast mode, all cards are in the open-drain mode to avoid bus contention. Refer to Section 11.6.5, “Commands for MMC/SD,” for the commands of bc and bcr categories.

After the broadcast command CMD3 is issued, the cards enter standby mode. Addressed type commands are used from this point. In this mode, the SDHC_CMD/SDHC_DAT I/O pads turn to push-pull mode, to have the driving capability for maximum frequency operation. Refer to Section 11.6.5, “Commands for MMC/SD,” for the commands of ac and adtc categories.

11.6.1 Command Send and Response Receive Basic Operation

Assuming data type WORD is an unsigned 32-bit integer, the below flow is a guideline for sending a command to the card(s):

```
send_command(cmd_index, cmd_arg, other requirements)
{
WORD wCmd; // 32-bit integer to make up the data to write into the XFERTYP register, it is
          // recommended to implement in a bit-field manner
wCmd = (<cmd_index> & 0x3f) << 24; // set the first 8 bits as '00'+<cmd_index>
set CMDTYP, DPSEL, CICEN, CCCEN, RSTTYP, and DTDSEL according to the command index;
         // XFERTYP register bits
if (internal DMA is used) wCmd |= 0x1;
```

```
if (multi-block transfer) {
        set XFERTYP[MSBSEL] bit;
        if (finite block number) {
                set XFERTYP[BCEN] bit;
                if (auto12 command is to use) set XFERTYP[AC12EN] bit;
        }
}
write_reg(CMDARG, <cmd_arg>); // configure the command argument
write_reg(XFERTYP, wCmd); // set XFERTYP register as wCmd value to issue the command
}
wait_for_response(cmd_index)
{
while (IRQSTAT[CC] is not set); // wait until command complete bit is set
read IRQSTAT register and check if any error bits about command are set;
if (any error bits are set) report error;
write 1 to clear IRQSTAT[CC] and all command error bits;
}
```

For the sake of simplicity, the function wait_for_response is implemented here by means of polling. For an effective and formal way, the response is usually checked after the command complete interrupt is received. By doing this, ensure the corresponding interrupt status bits are enabled.

For some scenarios, the response timeout is expected. For instance, after all cards respond to CMD3 and go to the standby state, no response to the host when CMD2 is sent. The host driver should manage false errors similar to this with caution.

11.6.2 Card Identification Mode

When a card is inserted to the socket or the card was reset by the host, the host needs to validate the operation voltage range, identify the cards, and request the cards to publish the relative card address (RCA) or to set the RCA for the MMCs.

11.6.2.1 Card Detect

See Figure 11-25 for a flow diagram showing the detection of MMC and SD cards using the eSDHC.

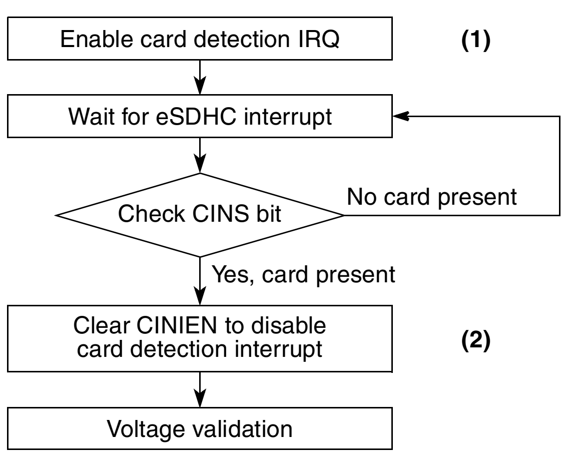


Figure 11-25. Flow Diagram for Card Detection

- Set IRQSIGEN[CINIEN] to enable card detection interrupt.

- When an interrupt from eSDHC is received, check IRQSTAT[CINS] to see if it is caused by card insertion.
- Clear the IRQSIGEN[CINIEN] to disable card detection interrupt and ignore all card insertion interrupt afterwards.

11.6.2.2 Reset

The host consists of three types of reset:

- Hardware reset (card and host) which is driven by POR (power on reset).
- Software reset (host only) is proceeded by the write operation on the SYSCTL[RSTD], SYSCTL[RSTC], or SYSCTL[RSTA] bits to reset the data part, command part, or all parts of the host controller, respectively.
- Card reset (card only). The command CMD0, GO_IDLE_STATE, is the software reset command for all types of MMCs and SD memory cards. This command sets each card into idle state regardless of the current card state. The cards are initialized with a default relative card address (RCA = 0x0000) and with a default driver stage register setting (lowest speed, highest driving current capability).

After the card is reset, the host needs to validate the voltage range of the card. See Figure 11-26 for the software flow to reset the eSDHC and card.

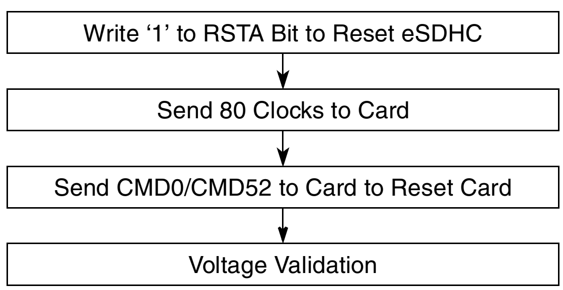


Figure 11-26. Flow Chart for Reset of eSDHC and SD I/O Card

```
software_reset()
{
        set_bit(SYSCTL, RSTA);              // software reset the host
        set SYSCTL[DTOCV and SDCLKFS];      // get the SDHC_CLK of frequency around 400 KHz
        configure I/O pad;                  // set the voltage of external card to around 3.0 V
        poll PRSSTAT[CIHB and CDIHB];       // wait until both bits are cleared
        set_bit(SYSCTRL, INTIA);            // send 80 clock ticks for card to power-up
        send_command(CMD_GO_IDLE_STATE, <other parameters>); // reset the card with CMD0
        or send_command(CMD_IO_RW_DIRECT, <other parameters>);
}
```

11.6.2.3 Voltage Validation

All cards should be able to establish communication with the host using any operation voltage in the maximum allowed voltage range specified in this standard. However, the supported minimum and maximum values for V_{DD} are defined in the operation conditions register (OCR) and may not cover the whole range. Cards that store the CID (card identification) and CSD data in the preloaded memory are only able to communicate these information under data transfer V_{DD} conditions. This means that if the host and card have different V_{DD} ranges, the card is not able to complete the identification cycle, nor is it able to send CSD data.

Therefore, a special command is available:

- SEND_OP_CONT (CMD1 for MMC),
- SD_SEND_OP_CONT (ACMD41 for SD Memory)

The voltage validation procedure is designed to provide a mechanism to identify and reject cards which do not match the V_{DD} range(s) desired by the host. This is accomplished by the host sending the desired V_{DD} voltage window as the operand of this command. Cards that can not perform data transfer in the specified range must discontinue any further bus operations and enter the inactive state. By omitting the voltage range in the command, the host can query each card and determine the common voltage range before sending out-of-range cards into the inactive state. This query should be used if the host is able to select a common voltage range or if a notification should be sent to the system when a non-usable cards in the stack is detected.

11.6.2.4 Card Registry

Card registry on MMC and SD/SD Combo cards are different.

For the SD card, the identification process starts at a clock rate lower than 400 KHz and the power voltage higher than 2.7 V, as defined by the card specification. At this time, the SDHC_CMD line output drives are push-pull drivers instead of open-drain. After the bus is activated, the host requests the card to send their valid operation conditions. The response to ACMD41 is the operation condition register of the card. The same command should be sent to all of the new cards in the system. Incompatible cards are placed into the inactive state. The host then issues the command, ALL_SEND_CID (CMD2), to each card to get its CID. Cards that are currently unidentified (that is, in ready state), send their CID number as the response. After the CID is sent by the card, the card goes into the identification state.

The host then issues Send_Relative_Addr (CMD3), requesting the card to publish a new relative card address (RCA) that is shorter than CID. This RCA is used to address the card for future data transfer operations. Once the RCA is received, the card changes its state to the standby state. At this point, if the host wants the card to have an alternative RCA number, it may ask the card to publish a new number by sending another Send_Relative_Addr command to the card. The last published RCA is the actual RCA of the card.

The host repeats the identification process with CMD2 and CMD3 for each card in the system until the last CMD2 gets no response from any of the cards in system.

For MMC operation, the host starts the card identification process in open-drain mode with the identification clock rate lower than 400 KHz, the power voltage higher than 2.7 V. The open-drain driver stages on the SDHC_CMD line allow parallel card operation during card identification. After the bus is activated the host requests the cards to send their valid operation conditions (CMD1). The response to CMD1 is the wired-OR operation on the condition restrictions of all cards in the system. Incompatible cards are sent into inactive state. The host then issues the broadcast command All_Send_CID (CMD2), asking all cards for their unique CID number. All unidentified cards (the cards in ready state) simultaneously start sending their CID numbers serially, while bit-wise monitoring their outgoing bit stream. Those cards, whose outgoing CID bits do not match the corresponding bits on the command line in any one of the bit periods, stop sending their CID immediately and must wait for the next identification cycle. Since the CID is unique for each card, only one card can successfully send its full CID to the host. This card then goes into identification state. Thereafter, the host issues Set_Relative_Addr (CMD3) to assign to this card a relative card address (RCA). Once the RCA is received, the card state changes to the stand-by state, and the card does not react in further identification cycles, and its output driver switches from open-drain to push-pull. The host repeats the process, namely CMD2 and CMD3, until the host receives a time-out condition to recognize completion of the identification process.

11.6.3 Card Access

These sections describe the supported access modes with external cards.

11.6.3.1 Block Write

This section describes the process of writing data to external cards in block mode.

11.6.3.1.1 Normal Write

During block write (CMD24–27), one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. If the CRC fails, the card should indicate the failure on the SDHC_DAT line (see below). The transferred data is discarded and not written, and all further transmitted blocks (in multi-block write mode) are ignored.

If the host uses partial blocks whose accumulated length is not block-aligned and block misalignment is not allowed (CSD parameter WRITE_BLK_MISALIGN is not set), the card detects the block misalignment error and aborts programming before the beginning of the first misaligned block. The card sets the ADDRESS_ERROR error bit in the status register, defined in the MMC/SD Specification, and then waits in the receive-data state for a stop command while ignoring all further data transfers. The write operation is also aborted if the host attempts to write over a write-protected area.

For MMC and SD cards, programming the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in the ROM, this unchangeable section must match the corresponding section of the receive buffer. If this match fails, then the card reports an error and does not change any register contents.

Some cards may require a long and unpredictable period of time to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing. If its write buffer is full and unable to accept new data from a new WRITE_BLOCK command, the card holds the SDHC_DAT line low. The host may poll the status of the card with a SEND_STATUS command (CMD13) cards, at any time and the card responds with its status. The card status indicates whether the card can accept new data or if the write process is still in progress. The host may deselect the card by issuing CMD7 (to select a different card) to change the card into the standby state and release the SDHC_DAT line without interrupting the write operation. When re-selecting the card, it reactivates the busy indication by pulling SDHC_DAT low if programming is still in progress and the write buffer is unavailable.

For simplicity, the software flow described below incorporates the internal DMA, and the write operation is a multi-block write with Auto CMD12 enabled. For the other method (CPU polling status) and different transfer nature, the internal DMA part of the procedure should be removed and alternative steps inserted.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

11.6.3.1.2 Write with Pause

The write operation can be paused during the transfer. Instead of stopping the SDHC_CLK at any time to pause all the operations which is also inaccessible to the host driver, the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Since there is no timeout condition in a write operation during the data blocks, a write operation to the cards can be paused in this way and if line SDHC_DAT0 is not required to de-assert to release busy state, no suspend command is needed.

Similar to the flow described in Section 11.6.3.1.1, “Normal Write,” the write with pause is shown with the same type of write operations:

1. Check the card status and wait until card is ready for data.
2. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
3. Set the eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Set PROCTL[SABGREQ].
7. Wait for the transfer complete interrupt.
8. Clear PROCTL[SABGREQ].
9. Check the status bit to see if a read CRC error occurred.
10. Set PROCTL[CREQ] to continue the read operation.
11. Wait for the transfer complete interrupt.
12. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

The number of blocks left during the data transfer is accessible by reading the content of BLKATTR[BLKCNT]. Due to the data transfers and setting PROCTL[SABGREQ] are concurrent, along with the delay of register read and the register setting, the actual number of blocks left may not be the same as the value read earlier. The driver should read the value of BLKATTR[BLKCNT] after the transfer is paused and the transfer complete interrupt is received.

It is also possible that the transfer of the last block begins when the stop-at-block-gap request is sent to the buffer. In this case, the next block gap is the actual end of the transfer, and therefore, the request is ignored. The driver should treat this as a non-pause transfer and a common write operation.

When the write operation is paused, the data transfer inside the host system does not stop and the transfer remains active until the data buffer is full. The eSDHC reads the resume command as a normal command with a data transfer, and it is the driver's responsibility to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN, AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of multi-block transfer.

11.6.3.2 Block Read

11.6.3.2.1 Normal Read

For block reads, the basic unit of a data transfer is a block whose maximum size is stored in areas defined in corresponding card specifications. A CRC is appended to the end of each block, ensuring data transfer integrity. CMD17, CMD18, CMD53, and so on, can initiate a block read. After completing the transfer, the card returns to the transfer state.

For multi-block reads, data blocks are continuously transferred until a stop command is issued. If the host uses partial blocks whose accumulated length is not block aligned and block misalignment is not allowed, the card which does not support partial block length, should detect the block misalignment at the beginning of the first misaligned block and report the error, depending on its card type.

For simplicity, the software flow described below incorporates the internal DMA, and the read operation is a multi-block read with Auto CMD12 enabled. For the other method (CPU polling status) and different transfer nature, the internal DMA part should be removed and the alternative steps are straightforward.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

11.6.3.2.2 Read with Pause

In general, the read operation is not able to pause.

Similar to the flow described in Section 11.6.3.2.1, “Normal Read,” the read with pause is shown with the same type of read operations:

1. Set PROCTL[RWCTL].
2. Check the card status and wait until the card is ready for data.
3. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
4. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in Step 2.
5. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
6. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
7. Set PROCTL[SABGREQ].
8. Wait for the transfer complete interrupt.
9. Clear PROCTL[SABGREQ].
10. Check the status bit to see if a read CRC error occurred.
11. Set PROCTL[CREQ] to continue the read operation.
12. Wait for the transfer complete interrupt.
13. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

Similar to the write operation, it is possible to meet the ending block of the transfer when paused. In this case, the eSDHC ignores the stop-at-block-gap request and treats it as a command read operation.

Unlike the write operation, there is no remaining data inside the buffer when the transfer is paused. All data received before the pause is transferred to the host system. Whether or not a suspend command is sent, the internal data buffer is not flushed.

If the suspend command is sent and the transfer is later resumed by means of the resume command, the eSDHC takes the command as a normal one accompanied with data transfer, and it is left for the driver to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN] and IRQSTT[AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of a multi-block transfer.

11.6.3.3 Transfer Error

11.6.3.3.1 CRC Error

At the end of a block transfer, a write CRC status error or read CRC error may occur. For this type of error, the last block received should be discarded because the integrity of the data block is not guaranteed. It is recommended to discard the following data blocks and re-transfer the block from the corrupted one. For a multi-block transfer, the host driver should issue CMD12 to abort the current process and start the transfer by a new data command. In this scenario, even when the XFERTYP[AC12EN, BCEN] are set, the eSDHC does not automatically send CMD12 because the last block is not transferred. On the other hand, if it is within the last block that CRC error occurs, Auto CMD12 is sent by the eSDHC. In this case, the driver should resend or re-obtain the last block with a single block transfer.

11.6.3.3.2 Internal DMA Error

During the data transfer with the internal DMA, if the DMA engine encounters an error on the CSB bus, the DMA operation is aborted and a DMA error interrupt is sent to the host system. When acknowledged by such an interrupt, the driver should calculate the start address of the data block where the error occurred. The start address can be calculated by either of the following methods:

- Read the DSADDR[DSADDR] field. The error occurs during the previous burst. Therefore, by taking the block size, the previous burst length, and the start address of the next burst transfer into account, one can obtain the start address of the corrupted block.
- Read the BLKATTR[BLKCNT] field. The start address of the corrupted block can be calculated by the number of blocks left, the total number to transfer, the start address of transfer, and the size of each block. However, if BCEN is not set, the contents of the block attribute register does not change and this method does not work.

When a DMA error occurs, it is recommended to abort the current transfer by means of CMD12 (for multi-block transfer), apply a reset for data, and restart the transfer from the corrupted block to recover the error.

11.6.3.3.3 Auto CMD12 Error

After the last block of a multi-block transfer is sent or received and XFERTYP[AC12EN] is set when the data transfer is initiated by the data command, the eSDHC automatically sends CMD12 to the card to stop the transfer. When an error occurs at this point, it is recommended that the host driver responds by:

1. Auto CMD12 response timeout. It is not certain whether the command has been accepted by the card or not. The driver should clear the Auto CMD12 error status bits and resend CMD12 until it is accepted by the card.
2. Auto CMD12 response CRC error. Since CMD12 has been received by the card, the card aborts the transfer. The driver may ignore the error and clear the error status bit.
3. Auto CMD12 conflict error or not sent. The command was not sent. Therefore, the driver should send CMD12 manually.

11.6.3.4 Card Interrupt

The external cards can inform the host controller through the use of special signals.

11.6.4 Switch Function

MMCs transferring data with a bus width other than one-bit wide is a new feature added to the MMC specification. The high-speed timing mode for all card devices is also newly-defined in recent various card specifications. To enable these new features, a type of switch command should be issued by the host driver.

For SD cards, the high-speed mode is queried and enabled by CMD6 (with the mnemonic symbol as SWICH_FUNC); for MMCs, the high-speed mode is queried by CMD8 and enabled by CMD6 (with the mnemonic symbol as SWITCH).

The 4-bit and 8-bit bus width of MMC is also enabled by the SWITCH command, but with a different argument.

These new functions can also be disabled by software reset , but such manner of restoring to normal mode is not recommended because a complete identification process is needed before the card is ready for data transfer.

For simplicity, the following flowcharts do not show a current capability check, which is recommended in the function switch process.

11.6.4.1 Query, Enable and Disable SD High Speed Mode

```
enable_sd_high_speed_mode(void)
{
        set BLKATTR[BLKCNT] to 1 (block), set BLKATTR[BLKSIZE] to 64 (bytes);
        send CMD6, with argument 0xFFFFF1 and read 64 bytes of data accompanying the R1
                response;
        wait data transfer done bit is set;
        check if the bit 401 of received 512 bit is set;
        if (bit 401 is '0') report the SD card does not support high speed mode and return;
        send CMD6, with argument 0x80FFFFF1 and read 64 bytes of data accompanying the R1
                response;
        check if the bit field 379~376 is 0xF;
        if (the bit field is 0xF) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                the card_clk of around 50MHz;
        (data transactions like normal peers)
}
disable_sd_high_speed_mode(void)
{
        set BLKCNT field to 1 (block), set BLKSIZE field to 64 (bytes);
        send CMD6, with argument 0x80FFFFF0 and read 64 bytes of data accompanying the R1
                response;
        check if the bit field 379~376 is 0xF;
        if (the bit field is 0xF) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                the card_clk of the desired value below 25MHz;
        (data transactions like normal peers)
}
```

11.6.4.2 Query, Enable and Disable MMC High Speed Mode

```
enable_mmc_high_speed_mode(void)
{
        send CMD9 to get CSD value of MMC;
        check if the value of SPEC_VER field is 4 or above;
        if (SPEC_VER value is less than 4) report the MMC does not support high speed mode and
                return;
        set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
        send CMD8 to get EXT_CSD value of MMC;
        extract the value of CARD_TYPE field to check the 'high speed mode' in this MMC is
                26MHz or 52MHz;
        send CMD6 with argument 0x1B90100;
        send CMD13 to wait card ready (busy line released);
        send CMD8 to get EXT_CSD value of MMC;
        check if HS_TIMING byte (byte number 185) is 1;
        if (HS_TIMING is not 1) report MMC switching to high speed mode failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                the card_clk of around 26MHz or 52MHz according to the CARD_TYPE;
```

```
        (data transactions like normal peers)
}

disable_mmc_high_speed_mode(void)
{
        send CMD6 with argument 0x2B90100;
        set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
        send CMD8 to get EXT_CSD value of MMC;
        check if HS_TIMING byte (byte number 185) is 0;
        if (HS_TIMING is not 0) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                the card_clk of the desired value below 20MHz;
        (data transactions like normal peers)
}
```

11.6.4.3 Set MMC Bus Width

```
change_mmc_bus_width(void)
{
        send CMD9 to get CSD value of MMC;
        check if the value of SPEC_VER field is 4 or above;
        if (SPEC_VER value is less than 4) report the MMC does not support multiple bit width
                and return;
        send CMD6 with argument 0x3B70x00; (8-bit, x=2; 4-bit, x=1; 1-bit, x=0)
        send CMD13 to wait card ready (busy line released);
        (data transactions like normal peers)
}
```

11.6.5 Commands for MMC/SD

See Table 11-27 for the list of commands for the MMC/SD cards. Refer to the corresponding specifications for details about the command information.

Four kinds of commands control the MMC:

1. Broadcast commands (bc)—no response
2. Broadcast commands with response (bcr)—response from all cards simultaneously
3. Addressed (point-to-point) commands (ac)—no data transfer on SDHC_DAT
4. Addressed (point-to-point) data transfer commands (ADTC)

Table 11-27. Commands for MMC/SD

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD0	bc	[31:0] stuff bits	—	GO_IDLE_STATE	Resets all MMC and SD memory cards to idle state.
CMD1	bcr	[31:0] OCR without busy	R3	SEND_OP_COND	Asks all MMCs and SD memory cards in idle state to send their operation conditions register contents in the response on the SDHC_CMD line.
CMD2	bcr	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the SDHC_CMD line.

Table 11-27. Commands for MMC/SD (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD3[(1)]	ac	[31:6] RCA [15:0] stuff bits	R1	SET/SEND_RELATIVE_ADDR	Assigns relative address to the card.
CMD4	bc	[31:0] DSR [15:0] stuff bits	—	SET_DSR	Programs the DSR of all cards.
CMD6[(2)]	adtc	[31] Mode 0: Check function 1: Switch function [30:8] Reserved for function groups 6 ~ 3 (All 0 or 0xFFFF) [7:4] Function group1 for command system [3:0] Function group2 for access mode	R1	SWITCH_FUNC	Checks switch ability (mode 0) and switch card function (mode 1). Refer to SD Physical Specification version 1.1 for details.
CMD6[(3)]	ac	[31:26] Set to 0 [25:24] Access [23:16] Index [15:8] Value [7:3] Set to 0 [2:0] Cmd Set	R1b	SWITCH	Switches the mode of operation of the selected card or modifies the EXT_CSD registers. Refer to the MultiMediaCard System Specification version 4.0 final draft 2 for details.
CMD7	ac	[31:6] RCA [15:0] stuff bits	R1b	SELECT/DESELECT_CARD	Command toggles a card between the stand-by and transfer states or between the programming and disconnect states. In both cases, the card is selected by its own relative address and gets deselected by any other address; address 0 deselects all.
CMD8	adtc	[31:0] stuff bits	R1	SEND_EXT_CSD	The card sends its EXT_CSD register as a block of data, with block size of 512 bytes.
CMD9	ac	[31:6] RCA [15:0] stuff bits	R2	SEND_CSD	Addressed card sends its card-specific data (CSD) on the SDHC_CMD line.
CMD10	ac	[31:6] RCA [15:0] stuff bits	R2	SEND_CID	Addressed card sends its card-identification (CID) on the SDHC_CMD line.
CMD11	adtc	[31:0] data address	R1	READ_DAT_UNTIL_STOP	Reads data stream from the card starting at the given address until STOP_TRANSMISSION is received.
CMD12	ac	[31:0] stuff bits	R1b	STOP_TRANSMISSION	Forces the card to stop transmission.
CMD13	ac	[31:6] RCA [15:0] stuff bits	R1	SEND_STATUS	Addressed card sends its status register.
CMD14	Reserved				
CMD15	ac	[31:6] RCA [15:0] stuff bits	—	GO_INACTIVE_STATE	Sets the card to inactive state in order to protect the card stack against communication breakdowns.

Table 11-27. Commands for MMC/SD (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD16	ac	[31:0] block length	R1	SET_BLOCKLEN	Sets the block length (in bytes) for all following block commands (read and write). Default block length is specified in the CSD.
CMD17	adtc	[31:0] data address	R1	READ_SINGLE_BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.
CMD18	adtc	[31:0] data address	R1	READ_MULTIPLE_BLOCK	Continuously transfers data blocks from card to host until interrupted by a stop command.
CMD19	Reserved				
CMD20	adtc	[31:0] data address	R1	WRITE_DAT_UNTIL_STOP	Writes data stream from the host starting at the given address until the STOP_TRANSMISION command is received.
CMD21–23	Reserved				
CMD24	adtc	[31:0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.
CMD25	adtc	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until the STOP_TRANSMISSION command is received.
CMD26	adtc	[31:0] stuff bits	R1	PROGRAM_CID	Programming of the card identification register. This command should be issued only once per card. The card contains hardware to prevent this operation after the first programming. Normally this command is reserved for the manufacturer.
CMD27	adtc	[31:0] stuff bits	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28	ac	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write-protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card-specific data (WP_GRP_SIZE).
CMD29	ac	[31:0] data address	R1b	CLR_WRITE_PROT	If the card provides write-protection features, this command clears the write protection bit of the addressed group.
CMD30	adtc	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card provides write-protection features, this command asks the card to send the status of the write-protection bits.
CMD31	Reserved				

Table 11-27. Commands for MMC/SD (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD32	ac	[31:0] data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group.
CMD33	ac	[31:0] data address	R1	TAG_SECTOR_END	Sets the address of the last write block of the continuous range to be erased.
CMD34	ac	[31:0] data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection.
CMD35	ac	[31:0] data address	R1	TAG_ERASE_GROUP_START	Sets the address of the first erase group within a range to be selected for erase.
CMD36	ac	[31:0] data address	R1	TAG_ERASE_GROUP_END	Sets the address of the last erase group within a continuous range to be selected for erase.
CMD37	ac	[31:0] data address	R1	UNTAG_ERASE_GROUP	Removes one previously selected erase group from the erase selection.
CMD38	ac	[31:0] stuff bits	R1b	ERASE	Erase all previously selected sectors.
CMD39	ac	[31:0] RCA [15] register write flag [14:8] register address [7:0] register data	R4	FAST_IO	Used to write and read 8-bit (register) data fields. The command address a card and a register and provides the data for writing if the write flag is set. The R4 response contains data read from the address register. This command accesses application dependent registers which are not defined in the MMC standard.
CMD40	bcr	[31:0] stuff bits	R5	GO_IRQ_STATE	Sets the system into interrupt mode.
CMD41	Reserved				
CDM42	adtc	[31:0] stuff bits	R1b	LOCK_UNLOCK	Used to set/reset the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
CMD43–51	Reserved				
CMD52	ac	[31:0] stuff bits	R5	IO_RW_DIRECT	Access a single register within the total 128 Kbytes of register space in any I/O function.
CMD53	ac	[31:0] stuff bits	R5	IO_RW_EXTENDED	Access a multiple I/O register with a single command, it allows the reading or writing of a large number of I/O registers.
CMD54	Reserved				
CMD55	ac	[31:16] RCA [15:0] stuff bits	R1	APP_CMD	Indicates to the card that the next command is an application specific command rather that a standard command.

Table 11-27. Commands for MMC/SD (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD56	adtc	[31:1] stuff bits [0]: RD/WR	R1b	GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general-purpose or application-specific commands. The size of the data block is set by the SET_BLOCK_LEN command.
ACMDs should be preceded with the APP_CMD command (Commands listed below are for SD cards only. Other SD commands not listed below are not supported by this module)					
ACMD6	ac	[31:2] stuff bits [1:0] bus width	R1	SET_BUS_WIDTH	Defines the data bus width (00 = 1 bit or 10 = 4 bit bus) to be used for data transfer. The allowed data bus widths are given in DCR register.
ACMD13	adtc	[31:0] stuff bits	R1	SD_STATUS	Send the SD memory card status.
ACMD22	adtc	[31:0] stuff bits	R1	SEND_NUM_WR_ SECTORS	Send the number of the written (without errors) sectors. Responds with 32 bit + CRC data block.
ACMD23	ac	[31:23] stuff bits [22:0] number of blocks	R1	SET_WR_BLK_ERASE_ COUNT	—
ACMD41	bcr	[31:0] OCR	R3	SD_APP_OP_COND	Asks the accessed card to send its operating condition register (OCR) content in the response on the SDHC_CMD line.
ACMD42	ac	—	R1	SET_CLR_CARD_DETECT	—
ACMD51	adtc	[31:0] stuff bits	R1	SEND_SCR	Reads the SD Configuration Register (SCR)

1 Registers mentioned in this table are SD card registers.

NOTE

- CMD3 differs for MMC and SD cards
  For MMC cards, CMD3 is referred to as SET_RELATIVE_ADDR and has a response type R1
  For SD cards, CMD3 is referred to as SEND_RELATIVE_ADDR and has a response type R6, with RCA inside
- CMD6 differs completely between high-speed MMC cards and high-speed SD cards. Command SWITCH_FUNC is used for high speed SD cards.
- Command SWITCH is for high-speed MMC cards. The index field can contain any value from 0–255, but only values 0–191 are valid. If the index value is in the 192–255 range, the card does not perform any modification and the status bit EXT_CSD[SWITCH_ERROR] is set. The access bits are shown in Table 11-28:

Table 11-28. EXT_CSD Access Modes

Bits	Access Name	Operation
00	Command set	The command set is changed according to the command set field of the argument
01	Set bits	The bits in the pointed byte are set, according to the set bits in the value field.
10	Clear bits	The bits in the pointed byte are cleared, according to the set bits in the value field.
11	Write byte	The value field is written into the pointed byte.

11.6.6 Software Restrictions

When polling read or write, once the software begins a buffer read or write, it must access exactly the number of times as set in the watermark level register, as if a DMA burst occurred.

When the internal DMA is not enabled and a write transaction is in operation, DATPORT (described in Section 11.4.6, “Buffer Data Port Register (DATPORT)”) must not be read.

Chapter 12
Sequencer

12.1 Overview

The I/O sequencer switches transactions among its ports, using a buffer pool to minimize blocking. Figure 12-1 is a block diagram of the I/O sequencer (IOS).

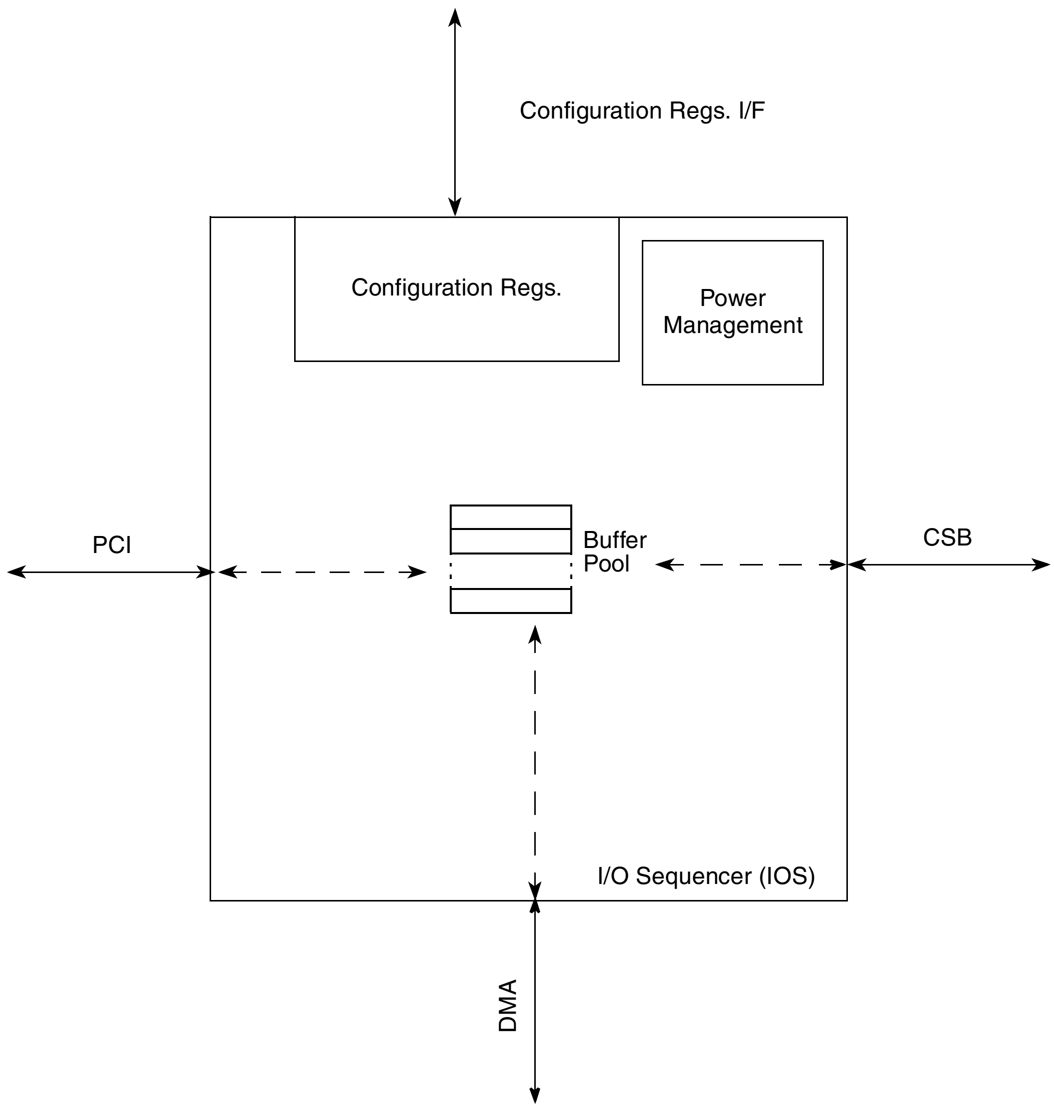


Figure 12-1. I/O Sequencer Block Diagram

12.1.1 Features

The I/O sequencer includes the following features:

- Switches transactions among its ports
- Contains 8 cache-line (32-byte) buffers to allow streaming of PCI transactions
- Performs address translation on outbound PCI transactions

Note that the number of CSB masters allowed to access to PCI outbound windows is restricted to no more than four non-CPU masters or no more than two non-CPU plus the CPU. If this number is exceeded, deadlock can occur on CSB arbitration.

12.2 External Signal Description

The I/O sequencer has no external signals.

12.3 Memory Map/Register Definition

Table 12-1 shows the I/O sequencer memory map.

Table 12-1. Sequencer Memory Map

Offset	Register	Access	Reset	Section/Page
0x00	POTAR0—PCI outbound translation address register 0	R/W	0x0000_0000	12.4.1/12-3
0x08	POBAR0—PCI outbound base address register 0	R/W	0x0000_0000	12.4.2/12-3
0x10	POCMR0—PCI outbound comparison mask register 0	R/W	0x0000_0000	12.4.3/12-4
0x18	POTAR1—PCI outbound translation address register 1	R/W	0x0000_0000	12.4.1/12-3
0x20	POBAR1—PCI outbound base address register 1	R/W	0x0000_0000	12.4.2/12-3
0x28	POCMR1—PCI outbound comparison mask register 1	R/W	0x0000_0000	12.4.3/12-4
0x30	POTAR2—PCI outbound translation address register 2	R/W	0x0000_0000	12.4.1/12-3
0x38	POBAR2—PCI outbound base address register 2	R/W	0x0000_0000	12.4.2/12-3
0x40	POCMR2—PCI outbound comparison mask register 2	R/W	0x0000_0000	12.4.3/12-4
0x48	POTAR3—PCI outbound translation address register 3	R/W	0x0000_0000	12.4.1/12-3
0x50	POBAR3—PCI outbound base address register 3	R/W	0x0000_0000	12.4.2/12-3
0x58	POCMR3—PCI outbound comparison mask register 3	R/W	0x0000_0000	12.4.3/12-4
0x60	POTAR4—PCI outbound translation address register 4	R/W	0x0000_0000	12.4.1/12-3
0x68	POBAR4—PCI outbound base address register 4	R/W	0x0000_0000	12.4.2/12-3
0x70	POCMR4—PCI outbound comparison mask register 4	R/W	0x0000_0000	12.4.3/12-4
0x78	POTAR5—PCI outbound translation address register 5	R/W	0x0000_0000	12.4.1/12-3
0x80	POBAR5—PCI outbound base address register 5	R/W	0x0000_0000	12.4.2/12-3
0x88	POCMR5—PCI outbound comparison mask register 5	R/W	0x0000_0000	12.4.3/12-4

Table 12-1. Sequencer Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0xF0	PMCR—Power management control register	R/W	0x0000_0000	12.4.4/12-5
0xF8	DTCR—Discard timer control register	R/W	0x0000_0000	12.4.5/12-6

12.4 Register Descriptions

This section describes the PCI registers.

12.4.1 PCI Outbound Translation Address Registers (POTAR*n*)

The PCI outbound translation address register defines the location of the outbound translation window in the PCI (translated) address space. Figure 12-2 shows the POTAR*n* register fields.

Offset 0x00, 0x18, 0x30
0x48, 0x60, 0x78

Access: Read/Write

	0–11	12–31
R W	—	TA
Reset	All zeros	

Figure 12-2. PCI Outbound Translation Address Registers (POTAR*n*)

Table 12-2 describes POTAR*n* fields.

Table 12-2. POTAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	TA	Translation address. Contains the starting address of the outbound translated address. It also corresponds to the most-significant 20 bits of a 32-bit address. The translation address must be aligned based on the window's size.

12.4.2 PCI Outbound Base Address Registers (POBAR*n*)

The PCI outbound base address register (POBAR*n*) defines the location of the outbound translation window in the local (source) memory space. Figure 12-3 shows the POBAR*n* register fields.

Offset 0x08, 0x20, 0x38,
0x50, 0x68, 0x80

Access: Read/Write

	0–11	12–31
R W	—	BA
Reset	All zeros	

Figure 12-3. PCI Outbound Base Address Registers (POBAR*n*)

Table 12-3 describes POBAR*n* fields.

Table 12-3. POBAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	BA	Base address. This field contains the starting address of the outbound translated window. This field corresponds to the most-significant 20 bits of a 32-bit address.

12.4.3 PCI Outbound Comparison Mask Registers (POCMR*n*)

The PCI outbound comparison mask register (POCMR*n*) defines the size and destination of the outbound translation window. It also defines some properties of the window in the PCI address space. See Section 12.5.1, "Transaction Forwarding," for more information. Figure 12-4 shows the POCMR*n* register fields.

Offset 0x10, 0x28, 0x40
0x58, 0x70, 0x88

Access: Read/Write

	0	1	2–11	12–31
R/W	EN	IO	—	CM
Reset	All zeros			

Figure 12-4. PCI Outbound Comparison Mask Registers (POCMR*n*)

Table 12-4 describes the bit settings of the POCMR*n* register.

Table 12-4. POCMR*n* Field Descriptions

Bits	Name	Description
0	EN	Enable. Enables the address translation window. 0 Address translation is disabled for this window. 1 Address translation is enabled for this window. Local addresses that match the definition of the window will be recognized by the device and translated to the PCI memory space.
1	IO	I/O space. Determines whether the window is mapped to the PCI memory space or PCI I/O space. 0 Memory space 1 I/O space
2–11	—	Reserved, should be cleared.

Table 12-4. POCMR*n* Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th colspan="4">Description</th></tr>
<tr><td rowspan="14">12–31</td><td rowspan="14">CM</td><td colspan="4">Comparison mask. Contains the size of the translation window. The bits that are 1 in this field indicate bits of the transaction address that should be matched to the value in the PCI outbound base address register and translated in case of a match. This field corresponds to the most-significant 20 bits of a 32-bit address.</td></tr>
<tr><td>Value</td><td>Meaning</td><td>Value</td><td>Meaning</td></tr>
<tr><td>0000_0000_0000_0000_0000</td><td>4 GB</td><td>1111_1111_1110_0000_0000</td><td>2 MB</td></tr>
<tr><td>1000_0000_0000_0000_0000</td><td>2 GB</td><td>1111_1111_1111_0000_0000</td><td>1 MB</td></tr>
<tr><td>1100_0000_0000_0000_0000</td><td>1 GB</td><td>1111_1111_1111_1000_0000</td><td>512 KB</td></tr>
<tr><td>1110_0000_0000_0000_0000</td><td>512 MB</td><td>1111_1111_1111_1100_0000</td><td>256 KB</td></tr>
<tr><td>1111_0000_0000_0000_0000</td><td>256 MB</td><td>1111_1111_1111_1110_0000</td><td>128 KB</td></tr>
<tr><td>1111_1000_0000_0000_0000</td><td>128 MB</td><td>1111_1111_1111_1111_0000</td><td>64 KB</td></tr>
<tr><td>1111_1100_0000_0000_0000</td><td>64 MB</td><td>1111_1111_1111_1111_1000</td><td>32 KB</td></tr>
<tr><td>1111_1110_0000_0000_0000</td><td>32 MB</td><td>1111_1111_1111_1111_1100</td><td>16 KB</td></tr>
<tr><td>1111_1111_0000_0000_0000</td><td>16 MB</td><td>1111_1111_1111_1111_1110</td><td>8 KB</td></tr>
<tr><td>1111_1111_1000_0000_0000</td><td>8 MB</td><td>1111_1111_1111_1111_1111</td><td>4 KB</td></tr>
<tr><td>1111_1111_1100_0000_0000</td><td>4 MB</td><td></td><td></td></tr>
<tr><td colspan="4">All others values are Reserved</td></tr>
</table>

12.4.4 Power Management Control Register (PMCR)

PMCR provides control of system-level low-power modes. Figure 12-5 shows the PMCR register fields.

Offset 0xF0 — Access: Read only

	0–30	31
R	—	IDLE
W		
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1

Figure 12-5. Power Management Control Register (PMCR)

Table 12-5 describes PMCR fields.

Table 12-5. PMCR Field Descriptions

Bits	Name	Description
0–30	—	Reserved
31	IDLE	This read-only bit indicates that the logic controlled by the power management function is idle. This bit could be used as an indication that it is okay to disable the clocks to this complex. 0 Logic is active. 1 Logic is idle. There are no outstanding transactions in the IOS, the DMA, or the PCI port.

12.4.5 Discard Timer Control Register (DTCR)

DTCR configures the discard timer, which is used to place a time limit on PCI delayed read transactions from non-prefetchable memory. Although prefetched reads may be discarded whenever the IOS is full and needs to allocate another buffer, other delayed reads must not be discarded until the originator actually receives the data. The DTCR is used to release stuck buffers in case of malfunctioning or disconnected masters that never come back to read the data they requested

Figure 12-6 shows the DTCR register fields.

Offset 0xF8 Access: Read/Write

	0	1–7	8–31
R/W	EN	—	PTV
Reset	All zeros		

Figure 12-6. Discard Timer Control Register (DTCR)

Table 12-6 describes DTCR fields.

Table 12-6. DTCR Field Descriptions

Bits	Name	Description
0	EN	Enable. This bit enables the discard timer. 0 Disabled 1 Enabled
1–7	—	Reserved
8–31	PTV	Preload timer value (PTV). This field contains the preload value for the discard timer. PCI delayed reads from non-prefetchable address space are discarded after (2^{24} – PTV) internal clock cycles if the master has not repeated the transaction. 0xFFFFFF is not valid for PTV. For example, to discard a delayed completion if the PCI master has not repeated the transaction in 2^{15} PCI clocks, • Assuming the internal frequency is twice the PCI frequency • The PTV should equal $2^{24} – 2^{16}$ (0xFF0000).

12.5 Functional Description

The IOS is a four-port switch with buffering. Each port has master and slave interfaces. When a port masters a transaction, the transaction attributes are stored in a buffer and the IOS generates a transaction to the slave interface of the destination port. The data is also buffered between the ports. The IOS contains 8 cache line (32-byte) transaction buffers, some of which are reserved for specific types of transactions.

The address and data phases of the transactions are independent. The data phases of the transactions are not required to be in order.

12.5.1 Transaction Forwarding

Although the ports use a similar interface, the I/O sequencer is not actually symmetrical. The transaction forwarding from each source is explained in the following sections.

12.5.1.1 Transactions from the Coherency System Bus (CSB) Port

Transactions from the CSB port are forwarded as follows:

- If the address matches the 12-byte PCI controller software configuration memory space of the PCI controller, the transaction is forwarded to the PCI port. These address values are configuration options of the I/O sequencer. See Table 14-3 for more information on PCI controller software configuration memory space.
- If the address matches the DMA register memory space, the transaction is forwarded to the DMA port.
- If the address hits any of the outbound translation windows, the transaction is forwarded to the PCI port, with the address translated. See Section 12.5.2, "PCI Outbound Address Translation," for more information.

12.5.1.2 Transactions from the PCI Port

Transactions from the PCI port are forwarded as follows:

- If the address matches the DMA register memory space, the transaction is forwarded to the DMA port.
- All other transactions are forwarded to the CSB port.

12.5.1.3 Transactions from the DMA Port

Transactions from the DMA port are forwarded as follows:

- If the address hits any of the outbound translation windows, the transaction is forwarded to the PCI port, with the address translated. See Section 12.5.2, "PCI Outbound Address Translation," for more information.
- All other transactions are forwarded to the CSB port.

12.5.2 PCI Outbound Address Translation

Outbound address translation is provided to allow the outbound transactions to access any address over the PCI memory or I/O space. Translation window base addresses are defined in the PCI outbound base address registers. See Section 12.4.2, "PCI Outbound Base Address Registers (POBARn)," for more information. Transactions to these address ranges are issued on the PCI bus with a translated address. The translation addresses are defined in the associated PCI outbound translation address registers (POTARs).

Figure 12-7 shows an example translation window for outbound memory accesses.

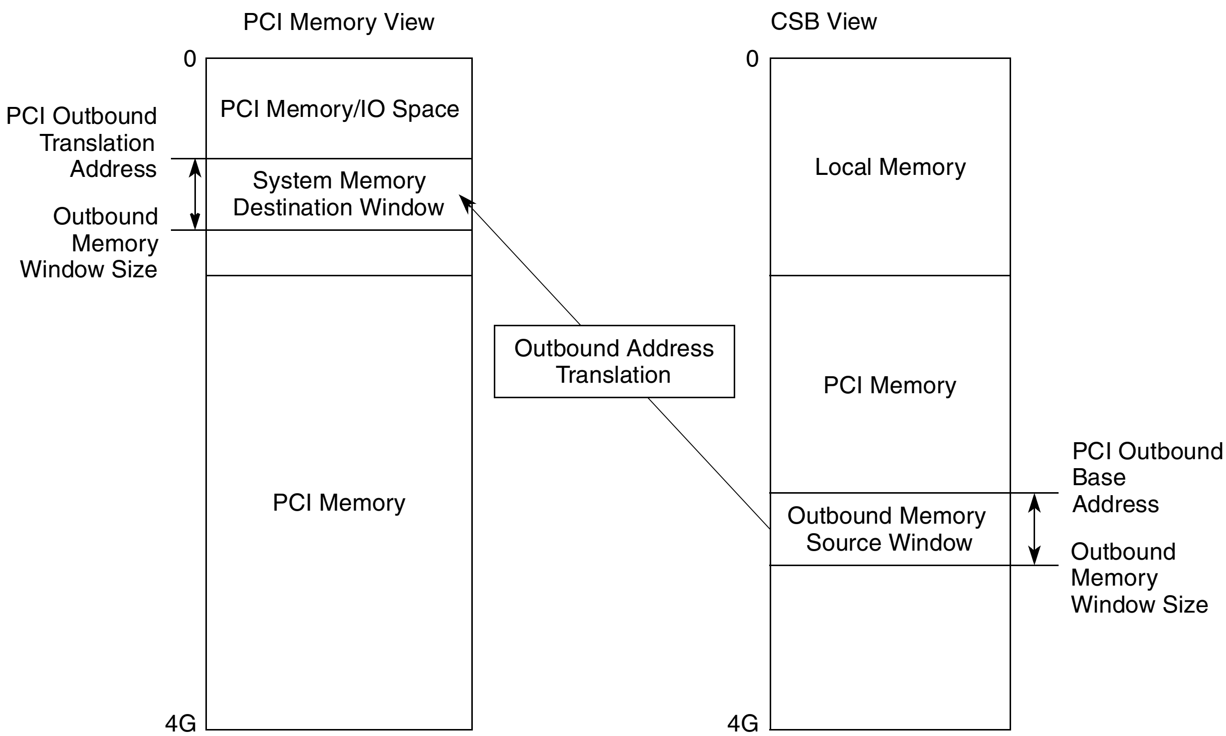


Figure 12-7. Outbound PCI Memory Address Translation

The six sets of outbound translation registers allow six simultaneous translation windows to the PCI port. Software can move and adjust the memory window translations and sizes during run-time. This allows software to access different PCI memory/IO spaces on-the-fly, but the PCI outbound translation source windows must not overlap. However, outbound translation destination windows can be overlapped.

12.5.3 Transaction Ordering

The following rules are applied to maintain proper ordering of transactions:

- The transactions arriving from each port are dispatched to the destination port in the order of arrival. The dispatch order of transactions arriving on different ports is not necessarily maintained.
- A read transaction that originates at the CSB port and reads from the PCI port pulls out of the IOS any posted writes that originated on the PCI port and were posted before the read data arrives from the PCI.
- The IOS can always accept a write from the PCI port without forcing the PCI port to first accept a read.

Chapter 13
DMA/Messaging Unit

The DMA/messaging unit supports communication between two processors on different buses, for example, a local processor and a processor on a PCI bus. This unit operates with generic messages and doorbell registers. Figure 13-1 is a block diagram of the DMA/messaging unit.

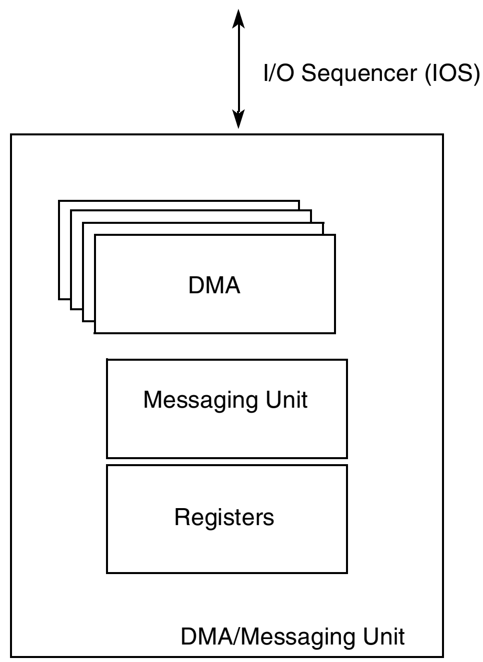


Figure 13-1. DMA/Messaging Unit Block Diagram

This block also provides a DMA controller, which transfers blocks of data independent of the local processor or PCI hosts. The DMA module has four high-speed DMA channels, which share buffer space in the I/O sequencer (IOS) to facilitate the gathering and sending of data.

13.1 Features

The DMA/messaging unit includes the following features:

- Message and doorbell registers for inter-processor communication
- DMA controller
  - Four DMA channels
  - Concurrent execution across multiple channels with programmable bandwidth control
  - Misaligned transfer capability

— Data chaining and direct mode
— Interrupt on completed segment, chain, and error
— Optional external control signals (REQ/ACK/DONE) per channel

13.2 External Signal Description

This section describes the DMA signals.

13.2.1 Detailed Signal Descriptions

Table 13-1 contains the detailed descriptions of the DMA interface signals.

Table 13-1. DMA Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
$\overline{\text{DREQ}}$[0:3]	I	DMA request signals, one per channel. The DMA request signal indicates the start or continuation of a DMA transfer. The fallingp edge of $\overline{\text{DREQ}n}$ causes DMAMR*n*[CS] to be set, thereby activating the DMA channel.	
		State Meaning	Asserted—Assertion of $\overline{\text{DREQ}n}$ starts or resumes a DMA transfer if DMAMR*n*[EMSEN] is 1. Negated—Negation of $\overline{\text{DREQ}n}$ has no effect.
		Timing	Assertion—Can be asserted asynchronously. Negation—Should remain asserted until $\overline{\text{DACK}n}$ is asserted or the requested transaction to the peripheral occurs.
$\overline{\text{DACK}}$[0:3]	O	DMA acknowledge signals, one per channel. The DMA acknowledge signal reflects the value of DMAMR*n*[CS].	
		State Meaning	Asserted—A DMA transfer is active. Negated—The DMA transfer is halted or complete.
		Timing	Assertion—Asserted asynchronously when a DMA transfer is started or resumed in the internal control logic. Negation—Negated asynchronously when a DMA transfer is halted or completed in the internal control logic. Note that there may still be outstanding write transactions in the bus pipeline after the negation of $\overline{\text{DACK}n}$.
$\overline{\text{DDONE}}$[0:3]	O	DMA done signals, one per channel. The DMA done signal indicates that the DMA transfer has completed.	
		State Meaning	Asserted—A DMA transfer is complete. Negated—A DMA transfer is active or halted.
		Timing	Assertion—Asserted asynchronously when a DMA transfer is completed in the internal control logic. Note that there may still be outstanding write transactions in the bus pipeline after the assertion of $\overline{\text{DDONE}n}$. Negation—Negated asynchronously when a DMA transfer begins in the internal control logic.

13.3 Memory Map/Register Definition

Table 13-2 lists the address and access of the memory map module.

Table 13-2. Module Memory Map

Offset	Register	Access	Reset	Section/Page
0x0_8030	OMISR—Outbound message interrupt status register	Mixed	0x0000_0000	13.4.1/13-4
0x0_8034	OMIMR—Outbound message interrupt mask register	R/W	0x0000_0000	13.4.2/13-5
0x0_8050	IMR0—Inbound message register 0	R/W	0x0000_0000	13.4.3/13-6
0x0_8054	IMR1—Inbound message register 1	R/W	0x0000_0000	13.4.3/13-6
0x0_8058	OMR0—Outbound message register 0	R/W	0x0000_0000	13.4.4/13-6
0x0_805C	OMR1—Outbound message register 1	R/W	0x0000_0000	13.4.4/13-6
0x0_8060	ODR—Outbound doorbell register	R/W	0x0000_0000	13.4.5/13-7
0x0_8068	IDR—Inbound doorbell register	R/W	0x0000_0000	13.4.5/13-7
0x0_8080	IMISR—Inbound message interrupt status register	Mixed	0x0000_0000	13.4.6/13-8
0x0_8084	IMIMR—Inbound message interrupt mask register	R/W	0x0000_0000	13.4.7/13-9
0x0_8100	DMAMR0—DMA 0 mode register	R/W	0x0000_0000	13.4.8.1/13-10
0x0_8104	DMASR0—DMA 0 status register	R/W	0x0000_0000	13.4.8.2/13-12
0x0_8108	DMACDAR0—DMA 0 current descriptor address register	R/W	0x0000_0000	13.4.8.3/13-13
0x0_8110	DMASAR0—DMA 0 source address register	R/W	0x0000_0000	13.4.8.4/13-14
0x0_8118	DMADAR0—DMA 0 destination address register	R/W	0x0000_0000	13.4.8.5/13-14
0x0_8120	DMABCR0—DMA 0 byte count register	R/W	0x0000_0000	13.4.8.6/13-15
0x0_8124	DMANDAR0—DMA 0 next descriptor address register	R/W	0x0000_0000	13.4.8.7/13-15
0x0_8180	DMAMR1—DMA 1 mode register	R/W	0x0000_0000	13.4.8.1/13-10
0x0_8184	DMASR1—DMA 1 status register	R/W	0x0000_0000	13.4.8.2/13-12
0x0_8188	DMACDAR1—DMA 1 current descriptor address register	R/W	0x0000_0000	13.4.8.3/13-13
0x0_8190	DMASAR1—DMA 1 source address register	R/W	0x0000_0000	13.4.8.4/13-14
0x0_8198	DMADAR1—DMA 1 destination address register	R/W	0x0000_0000	13.4.8.5/13-14
0x0_81A0	DMABCR1—DMA 1 byte count register	R/W	0x0000_0000	13.4.8.6/13-15
0x0_81A4	DMANDAR1—DMA 1 next descriptor address register	R/W	0x0000_0000	13.4.8.7/13-15
0x0_8200	DMAMR2—DMA 2 mode register	R/W	0x0000_0000	13.4.8.1/13-10
0x0_8204	DMASR2—DMA 2 status register	R/W	0x0000_0000	13.4.8.2/13-12
0x0_8208	DMACDAR2—DMA 2 current descriptor address register	R/W	0x0000_0000	13.4.8.3/13-13
0x0_8210	DMASAR2—DMA 2 source address register	R/W	0x0000_0000	13.4.8.4/13-14
0x0_8218	DMADAR2—DMA 2 destination address register	R/W	0x0000_0000	13.4.8.5/13-14
0x0_8220	DMABCR2—DMA 2 byte count register	R/W	0x0000_0000	13.4.8.6/13-15

Table 13-2. Module Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0_8224	DMANDAR2—DMA 2 next descriptor address register	R/W	0x0000_0000	13.4.8.7/13-15
0x0_8280	DMAMR3—DMA 3 mode register	R/W	0x0000_0000	13.4.8.1/13-10
0x0_8284	DMASR3—DMA 3 status register	R/W	0x0000_0000	13.4.8.2/13-12
0x0_8288	DMACDAR3—DMA 3 current descriptor address register	R/W	0x0000_0000	13.4.8.3/13-13
0x0_8290	DMASAR3—DMA 3 source address register	R/W	0x0000_0000	13.4.8.4/13-14
0x0_8298	DMADAR3—DMA 3 destination address register	R/W	0x0000_0000	13.4.8.5/13-14
0x0_82A0	DMABCR3—DMA 3 byte count register	R/W	0x0000_0000	13.4.8.6/13-15
0x0_82A4	DMANDAR3—DMA 3 next descriptor address register	R/W	0x0000_0000	13.4.8.7/13-15
0x0_82A8	DMAGSR—DMA general status register	R	0x0000_0000	13.4.8.8/13-16
0x0_82B0– 0x0_82FF	Reserved	—	—	—

13.4 Register Descriptions

The following sections describe the DMA/messaging unit configuration, control, and status registers.

NOTE

The registers described in this section use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI bus.

13.4.1 Outbound Message Interrupt Status Register (OMISR)

OMISR contains the interrupt status of the doorbell and outbound message registers. A PCI device acknowledges the outbound message interrupt by writing a 1 to the appropriate status bit: OMISR[OM1I] or OMISR[OM0I]. Setting one of these bits clears both the interrupt and the corresponding status bit. The local processor provokes an outbound message interrupt by writing to either of the two outbound message registers: OMR0 or OMR1. OMISR can be accessed from the CSB or the PCI bus, but it is normally accessed only from the PCI bus. Figure 13-2 shows the OMISR fields.

Offset 0x030 Access: Mixed

	31–4	3	2	1	0
R	—	ODI	—	OM1I	OM0I
W				w1c	w1c
Reset	All zeros				

Figure 13-2. Outbound Message Interrupt Status Register (OMISR)

Table 13-3 describes the OMISR register.

Table 13-3. OMISR Field Descriptions

Bits	Name	Description
31–4	—	Reserved
3	ODI	Outbound doorbell interrupt. This read-only bit indicates the status of the ODR bits. It is masked by OMIMR[ODIM]. 0 No outbound doorbell interrupt. 1 There is an outbound doorbell interrupt.
2	—	Reserved
1	OM1I	Outbound message 1 interrupt. When set, indicates that there is an outbound message 1 interrupt. Write 1 to this position to clear this bit. 0 No outbound message 1 interrupt. 1 There is an outbound message 1 interrupt.
0	OM0I	Outbound message 0 interrupt. When set, indicates that there is an outbound message 0 interrupt. Write 1 to this position to clear this bit. 0 No outbound message 0 interrupt. 1 There is an outbound message 0 interrupt.

13.4.2 Outbound Message Interrupt Mask Register (OMIMR)

OMIMR contains the interrupt mask of the doorbell and message register events generated by the local processor. OMIMR can be read from the CSB or the PCI bus, but it can be written only from the PCI bus. Figure 13-3 shows the OMIMR.

Offset 0x034 Access: User Read/Write

	31–4	3	2	1	0
R W	—	ODIM	—	OM1IM	OM0IM
Reset	All zeros				

Figure 13-3. Outbound Message Interrupt Mask Register (OMIMR)

Table 13-4 describes the OMIMR register.

Table 13-4. OMIMR Field Descriptions

Bits	Name	Description
31–4	—	Reserved
3	ODIM	Outbound doorbell interrupt mask. 0 Outbound doorbell interrupt is allowed 1 Outbound doorbell interrupt is masked
2	—	Reserved

Table 13-4. OMIMR Field Descriptions (continued)

Bits	Name	Description
1	OM1IM	Outbound message 1 interrupt mask. 0 Outbound message 1 interrupt is allowed 1 Outbound message 1 interrupt is masked
0	OM0IM	Outbound message 0 interrupt mask. 0 Outbound message 0 interrupt is allowed 1 Outbound message 0 interrupt is masked

13.4.3 Inbound Message Registers (IMR0–IMR1)

The inbound message registers can be read from the PCI bus and the CSB in both host and agent modes. They can be written only from the PCI bus. Figure 13-4 shows the IMR0 and IMR1 fields.

Offset 0x050, 0x054 — Access: User Read/Write

	31–0
R/W	IMSG*n*
Reset	All zeros

Figure 13-4. Inbound Message Registers (IMR0, IMR1)

Table 13-5 describes the IMR*n* register.

Table 13-5. IMR0 and IMR1 Field Descriptions

Bits	Name	Description
31–0	IMSG*n*	Inbound message *n*. Contains generic data to be passed between the local processor and external hosts.

13.4.4 Outbound Message Registers (OMR0–OMR1)

The outbound message registers can be read from the PCI bus and the CSB in both host and agent modes. They can be written only from the CSB.

Offset 0x058, 0x05C — Access: User Read/Write

	31–0
R/W	OMSG*n*
Reset	All zeros

Figure 13-5. Outbound Message Registers (OMR0–OMR1)

Table 13-6 describes the OMR*n* registers.

Table 13-6. OMR0 and OMR1 Field Descriptions

Bits	Name	Description
31–0	OMSG*n*	Outbound message *n*. Contains generic data to be passed between the local processor and external hosts.

13.4.5 Doorbell Registers

The following sections describe the outbound and inbound doorbell registers.

13.4.5.1 Outbound Doorbell Register (ODR)

ODR is accessible from the PCI bus and the CSB in both host and agent modes. Figure 13-6 shows the ODR*n* fields.

Offset: 0x060 Access: User Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R/W	—			ODR 28	ODR 27	ODR 26	ODR 25	ODR 24	ODR 23	ODR 22	ODR 21	ODR 20	ODR 19	ODR 18	ODR 17	ODR 16
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	ODR 15	ODR 14	ODR 13	ODR 12	ODR 11	ODR 10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
Reset	All zeros															

Figure 13-6. Outbound Doorbell Register (ODR)

Table 13-7 describes the ODR registers.

Table 13-7. ODR Field Descriptions

Bits	Name	Description
31–29	—	Reserved
28–0	ODR*n*	Outbound doorbell *n*. Write 1 from the CSB to set. Write 1 from the PCI bus to clear. Writing 0 has no effect. (Writing a bit in this register from the CSB causes an interrupt ($\overline{\text{PCI\_INTA}}$) to be generated.)

13.4.5.2 Inbound Doorbell Register (IDR)

IDR is accessible from the PCI bus and the CSB in both host and agent modes. Figure 13-7 shows the IDR fields.

Offset: 0x068 Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	IMC	IDR30	IDR29	IDR28	IDR27	IDR26	IDR25	IDR24	IDR23	IDR22	IDR21	IDR20	IDR19	IDR18	IDR17	IDR16
Reset	All zeros															

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
Reset	All zeros															

Figure 13-7. Inbound Doorbell Register (IDR)

Table 13-8 describes the IDR registers.

Table 13-8. IDR Field Descriptions

Bits	Name	Descriptions
31	IMC	Inbound machine check. Write 1 from the PCI bus to set. Write 1 from the CSB to clear. Writing 0 has no effect. Writing this bit from the PCI bus causes a machine check interrupt to be generated to the local processor.
30–0	IDR*n*	Inbound doorbell *n*. Write 1 from the PCI bus to set. Write 1 from the CSB to clear. Writing 0 has no effect. Writing a bit in this register from the PCI bus causes an interrupt to be generated to the local processor.

13.4.6 Inbound Message Interrupt Status Register (IMISR)

The IMISR contains the interrupt status of the doorbell and message register events. Writing a 1 to IM1I clears the bit. The events are generated by the PCI masters.

Figure 13-8 shows the IMISR fields.

Offset 0x080 Access: User Mixed

	31–4	3	2	1	0
R	—	MCI	IDI	—	IM1I
W					w1c
Reset	All zeros				

Figure 13-8. Inbound Message Interrupt Status Register (IMISR)

Table 13-9 describes the IMISR register.

Table 13-9. IMISR Field Descriptions

Bits	Name	Descriptions
31–5	—	Reserved
4	MCI	Machine check interrupt. Indicates whether a machine check interrupt condition was generated by setting the IDR[31]. The interrupt is cleared by writing a 1 to IDR[IMC] from the CSB. 0 No machine check interrupt 1 There is a machine check interrupt
3	IDI	Inbound doorbell interrupt. Indicates whether an inbound doorbell interrupt occurred. 0 No inbound doorbell interrupt 1 There is an inbound doorbell interrupt
2	—	Reserved
1	IM1I	Inbound message 1 interrupt. Indicates whether an inbound message 1 interrupt occurred. Write 1 to this position to clear this bit. 0 No inbound message 1 interrupt. 1 There is an inbound message 1 interrupt.
0	IM0I	Inbound message 0 interrupt. Indicates whether an inbound message 0 interrupt occurred. Write 1 to this position to clear this bit. 0 No inbound message 0 interrupt. 1 There is an inbound message 0 interrupt.

13.4.7 Inbound Message Interrupt Mask Register (IMIMR)

This register contains the interrupt mask of the doorbell and message register events generated by the PCI master. Figure 13-9 shows the IMIMR fields.

Offset 0x084 Access: User Read/Write

	31–4	3	2	1	0
R W	—	MCIM	IDIM	—	IM1IM
Reset	All zeros				

Figure 13-9. Inbound Message Interrupt Mask Register (IMIMR)

Table 13-10 describes the IMISR register.

Table 13-10. IMIMR Field Descriptions

Bits	Name	Description
31–5	—	Reserved
4	MCIM	Machine check interrupt mask. 0 Machine check interrupt from the IDR is allowed 1 Machine check interrupt is masked. IMISR[MC1] is cleared
3	IDIM	Inbound doorbell interrupt mask. 0 Inbound doorbell interrupt is allowed 1 Inbound doorbell interrupt is masked. IMISR[IDI] is cleared.
2	—	Reserved

Table 13-10. IMIMR Field Descriptions (continued)

Bits	Name	Description
1	IM1IM	Inbound message 1 interrupt mask. 0 Inbound message 1 interrupt is allowed 1 Inbound message 1 interrupt is masked. IMISR[IM1I] is cleared
0	IM0IM	Inbound message 0 interrupt mask. 0 Inbound message 0 interrupt is allowed 1 Inbound message 0 interrupt is masked. IMISR[IM0I] is cleared

13.4.8 DMA Registers

Each DMA channel has a set of seven 32-bit registers (mode, status, current descriptor address, next descriptor address, source address, destination address, and byte count) to support transactions. The following sections describe the format of the DMA support registers.

13.4.8.1 DMA Mode Register (DMAMR*n*)

This section describes the DMA mode register. The mode register allows software to start the DMA transfer and to control various DMA transfer characteristics. Figure 13-10 shows the DMAMR*n* fields.

Offset: 0x100, 0x180, 0x200, 0x280 — Access: User read/write

	31–28	27–24	23–21	20	19	18	17–16
R/W	—	DRCNT	BWC	DMSEN	IRQS	EMSEN	DAHTS
Reset	All zeros						

	15–14	13	12	11–10	9–8	7	6–4	3	2	1	0
R/W	SAHTS	DAHE	SAHE	PRC	—	EOTIE	—	TEM	CTM	CC	CS
Reset	All zeros										

Figure 13-10. DMA Mode Register (DMAMR*n*)

Table 13-11 describes the DMAMR*n* register.

Table 13-11. DMAMR*n* Field Descriptions

Bits	Name	Description
31–28	—	Reserved
27–24	DRCNT	DMA request count. This field specifies the number of cache lines transferred per DMA request assertion if EMSEN is 1. This field is not used if EMSEN is 0. 0101 1 cache line 0110 2 cache lines 0111 4 cache lines 1000 8 cache lines 1001 16 cache lines 1010 32 cache lines Others Reserved

Table 13-11. DMAMR*n* Field Descriptions (continued)

Bits	Name	Description
23–21	BWC	Bandwidth control. Only applies when multiple channels are executing transfers concurrently. The field determines how many cache lines a given channel is allowed to transfer after it is granted access to the IOS interface and before it releases the interface to the next channel. This allows the user to prioritize the DMA channels. The BWC values are listed as follows: 000 1 cache line 001 2 cache lines 010 4 cache lines 011 8 cache lines 100 16 cache lines Others Reserved
20	DMSEN	Direct mode snoop enable. This bit controls snooping of direct mode DMA transactions. 0 Snooping is disabled 1 Snooping is enabled
19	IRQS	Interrupt steer. This bit determines the destination of the DMA interrupts. 0 All DMA interrupts are routed to the on-chip interrupt controller 1 All DMA interrupts are routed to the PCI bus through $\overline{\text{PCI\_INTA}}$
18	EMSEN	External master start enable. This bit is cleared when the DMA transfer has completed, so it must be set again for each transfer. 0 The channel is started by software setting the CS bit 1 The channel is started by hardware asserting the $\overline{\text{DREQ}}$ pin
17–16	DAHTS	Destination address hold transfer size. This field indicates the transfer size used for each transaction when DAHE is 1. The byte count register must be in multiples of the size, and the destination address register must be aligned based on the size. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
15–14	SAHTS	Source address hold transfer size. This field indicates the transfer size used for each transaction when SAHE is 1. The byte count register must be in multiples of the size, and the source address register must be aligned based on the size. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
13	DAHE	Destination address hold enable. This bit allows the DMA controller to hold the destination address constant for every transfer. The size used for transfer is indicated by DAHTS. Note that hardware supports only aligned transfers for this feature. 0 Do not hold the destination address constant 1 Hold the destination address constant **Note:** The DMA does not support address hold when the external trigger mode is selected (EMSEN = 1). **Note:** The DMA does not support address hold for both the source and the destination at the same transfer.
12	SAHE	Source address hold enable. This bit allows the DMA controller to hold the source address constant for every transfer. The size used for transfer is indicated by SAHTS. Note that hardware supports only aligned transfers for this feature. 0 Do not hold the source address constant 1 Hold the source address constant **Note:** The DMA does not support address hold when the external trigger mode is selected (EMSEN = 1). **Note:** The DMA does not support address hold for both the source and the destination at the same transfer.

Table 13-11. DMAMR*n* Field Descriptions (continued)

Bits	Name	Description
11–10	PRC	PCI read command. This field indicates the type of PCI read command to use. 00 Reserved 01 PCI read line 10 PCI read multiple 11 Reserved
9–8	—	Reserved
7	EOTIE	End-of-transfer interrupt enable. This bit determines whether an interrupt is generated at the completion of a DMA transfer. End-of-transfer is defined as the end of a direct mode transfer or in chaining mode, as the end of the transfer of the last segment of a chain. 0 No EOT interrupt is generated 1 EOT interrupt is generated
6–4	—	Reserved
3	TEM	Transfer error mask. This bit determines the DMA response in the event of a transfer error. 0 The DMA will halt when a transfer error occurs. 1 The DMA will complete the transfer regardless of whether a transfer error occurs. **Note:** Regardless of the setting of TEM, if an error condition was detected during the DMA transfer, it will cause DMASR*n*[TE] to be set.
2	CTM	Channel transfer mode. 0 Chaining mode 1 Direct mode
1	CC	Channel continue. This bit applies only to chaining mode. Setting this bit indicates that the current descriptor segment should be repeated. CC is cleared by the DMA once the repeat takes effect, so it only causes a single repeat. 0 Normal chaining 1 DMACDAR is not loaded from DMANDAR, causing a repeat of the current descriptor segment
0	CS	Channel start. A 0-to-1 transition occurring on this bit when the channel is not busy (SR[CB] bit is 0) will start the DMA process. If the channel is busy and a 0-to-1 transition occurs, the DMA channel will restart from a previous halt condition. A 1-to-0 transition when the channel is busy (CB bit is 1) will halt the DMA process. Nothing happens if the channel is not busy and a 1-to-0 transition occurs. This bit is cleared by the DMA at the end of a transfer.

13.4.8.2 DMA Status Register (DMASR*n*)

This section describes the DMA status register. The status register reports various DMA conditions during and after the DMA transfer. Writing a 1 to a specific set bit clears the bit. Figure 13-11 shows the DMASR*n* fields.

Offset 0x104, 0x184, 0x204, 0x284 — Access: User Mixed

	31–8	7	6–3	2	1	0
R	—	TE	Ò	CB	EOSI	EOCDI
W						
Reset	All zeros					

Figure 13-11. DMA Status Register (DMASR*n*)

Table 13-12 describes the DMASR*n* register.

Table 13-12. DMASR*n* Field Descriptions

Bits	Name	Description
31–8	—	Reserved
7	TE	Transfer error. Set when there is an error condition during the DMA transfer.
6–3	—	Reserved
2	CB	Channel busy. This bit indicates whether the channel is busy. It is cleared as a result of any of the following conditions: an error or completion of the DMA transfer. 0 No DMA transfer is currently in progress 1 A DMA transfer is currently in progress
1	EOSI	End-of-segment interrupt. After transferring a segment of data, if the DMACDAR*n*[EOSIE] bit in the current descriptor address register is set, this bit is set and an interrupt is generated.
0	EOCDI	End-of-chain/direct interrupt. When the last DMA transfer is finished, either in chaining or direct mode, if DMAMR[EOTIE] is set, this bit is set and an interrupt is generated.

13.4.8.3 DMA Current Descriptor Address Register (DMACDAR*n*)

DMACDAR*n* contains the address of the current segment descriptor being transferred. In chaining mode, software must initialize this register to point to the first descriptor in the chain. After processing the first descriptor, the DMA controller moves the contents of the next descriptor address register into DMACDAR, loads the following descriptor into DMANDAR, and executes the current transfer. Figure 13-12 shows the DMACDAR*n* fields.

Offset 0x108, 0x188, 0x208, 0x288 Access: User Read/Write

	31–5	4	3	2–0
R W	CDA	SNEN	EOSIE	—
Reset	All zeros			

Figure 13-12. DMA Current Descriptor Address Register (DMACDAR*n*)

Table 13-13 describes the DMACDAR*n* register.

Table 13-13. DMACDAR*n* Field Descriptions

Bits	Name	Description
31–5	CDA	Current descriptor address. This field contains the current descriptor address of the segment descriptor in memory. It must be aligned on an 8-word boundary.
4	SNEN	Snoop enable. 0 Snooping is disabled on DMA transactions of the current segment. 1 Snooping is enabled on DMA transactions of the current segment.
3	EOSIE	End-of-segment interrupt enable 0 No end-of-segment interrupt is generated. 1 An interrupt is generated when the current DMA transfer for the current descriptor is finished.
2–0	—	Reserved

13.4.8.4 DMA Source Address Register (DMASAR*n*)

DMASAR*n* indicates the address from which the DMA controller will be reading data. The software must ensure that this is a valid memory address. Figure 13-13 shows the DMASAR*n*.

Offset 0x110, 0x190, 0x210, 0x290 Access: User Read/Write

	31 — 0
R W	SA
Reset	All zeros

Figure 13-13. DMA Source Address Register (DMASAR*n*)

Table 13-14 describes the DMASAR*n* register.

Table 13-14. DMASAR*n* Field Descriptions

Bits	Name	Description
31–0	SA	Source address of DMA transfer. The content of this field is updated after each DMA read operation.

13.4.8.5 DMA Destination Address Register (DMADAR*n*)

DMADAR*n* indicates the address to which the DMA controller will be writing data. The software must ensure that this is a valid memory address. Figure 13-14 shows the DMADAR*n* fields.

Offset 0x118, 0x198, 0x218, 0x298 Access: User Read/Write

	31 — 0
R W	DA
Reset	All zeros

Figure 13-14. DMA Destination Address Register (DMADAR*n*)

Table 13-15 describes the DMADAR*n* register.

Table 13-15. DMASAR*n* Field Descriptions

Bits	Name	Description
31–0	DA	Destination address of DMA transfer.Updated after each DMA write operation.

13.4.8.6 DMA Byte Count Register (DMABCR*n*)

DMABCR*n* contains the number of bytes per transfer (maximum transfer size is 64 Mbytes). Figure 13-15 shows the DMABCR*n*.

Offset 0x120, 0x1A0, 0x220, 0x2A0 — Access: User Read/Write

	31–26	25–0
R/W	—	BC
Reset	All zeros	

Figure 13-15. DMA Byte Count Register (DMABCR*n*)

Table 13-16 describes the DMABCR*n* register.

Table 13-16. DMABCR*n* Field Descriptions

Bits	Name	Description
31–26	—	Reserved
25–0	BC	Byte count. This field contains the number of bytes to transfer. The value in this register is decremented after each DMA read operation. Maximum transfer size is 64 Mbytes.

13.4.8.7 DMA Next Descriptor Address Register (DMANDAR*n*)

DMANDAR*n* contains the address for the next segment descriptor in the chain. In chaining mode, this register is loaded from the 'next descriptor' field of the descriptor to which the current descriptor address register is pointing. Figure 13-16 shows the DMANDAR*n*.

Offset 0x124, 0x1A4, 0x224, 0x2A4 — Access: User Read/Write

	31–5	4	3	2–1	0
R/W	NDA	NSNEN	NESIE	—	EOTD
Reset	All zeros				

Figure 13-16. DMA Next Descriptor Address Register (DMANDAR*n*)

Table 13-17 describes the DMANDAR*n* register.

Table 13-17. DMANDAR*n* Field Descriptions

Bits	Name	Descriptions
31–5	NDA	Next descriptor address. This field contains the next descriptor address of the next segment descriptor in memory. It must be aligned on an 8-word boundary.
4	NSNEN	Next snoop enable. 0 Snooping is disabled on DMA transactions. 1 Snooping is enabled on DMA transactions.
3	NEOSIE	Next end-of-segment interrupt enable. 0 No end-of-segment interrupt is generated. 1 An interrupt is generated when the DMA transfer for the next descriptor is finished.

Table 13-17. DMANDAR*n* Field Descriptions (continued)

Bits	Name	Descriptions
2–1	—	Reserved
0	EOTD	End-of-transfer descriptor. 0 This descriptor contains a link to another descriptor. 1 This descriptor is the last to be executed.

13.4.8.8 DMA General Status Register (DMAGSR)

DMAGSR provides faster access to the status bits by combining the status bits of all of the DMA channels into one register. Each byte of this register provides the value of bits 7–0 of a channel's DMA status register. These bits are cleared by writing to the individual DMA status registers. Figure 13-17 shows the DMAGSR fields.

Offset 0x2A8 — Access: User Read/Write

	31–24	23–16	15–8	7–0
R/W	Channel 0 Status	Channel 1 Status	Channel 2 Status	Channel 3 Status
Reset	All zeros			

Figure 13-17. DMA General Status Register (DMAGSR)

13.5 Functional Description

13.5.1 Message Unit

An embedded processor is often part of a larger system containing many processors and distributed memory. These processors tend to work on tasks independent of the host and other peripheral processors in the system. Because of the independent nature of the tasks, it is necessary to provide a communication mechanism between the peripheral processors and the rest of the system. One such method is the use of messages. This block provides a messaging unit to further facilitate communications between host and peripheral. The message unit uses generic messages and doorbell registers.

13.5.1.1 Messaging Registers (IMR0–IMR1, OMR0–OMR1)

There are two 32-bit inbound message registers (IMR0–IMR1) and two 32-bit outbound message registers (OMR0–OMR1). IMR0 and IMR1 allow a remote host or PCI master to write a 32-bit value that, in turn, causes an interrupt request to the on-chip interrupt controller that drives an interrupt line to the local processor. OMR0 and OMR1 allow the local processor to write an outbound message which, in turn, causes the outbound interrupt signal $\overline{\text{PCI\_INTA}}$ to assert.

The interrupt to the local processor is cleared by writing 1 to the appropriate IMISR bit. The interrupt to PCI ($\overline{\text{PCI\_INTA}}$) is cleared by writing 1 to the appropriate OMISR bit.

13.5.1.2 Doorbell Registers (IDR and ODR)

This block contains the inbound doorbell register (IDR) and the outbound doorbell register (ODR). The inbound doorbell allows a remote processor to set a bit in the register from the PCI bus. This, in turn, generates an interrupt request to the on-chip interrupt controller that drives an interrupt line to the local processor. The local processor can write to the ODR, which causes the outbound interrupt signal PCI_INTA to assert, thus interrupting the remote processor on the PCI bus.

The interrupt to the local processor is cleared by writing 1 to the appropriate IDR bit. The interrupt to PCI (PCI_INTA) is cleared by writing 1 to the appropriate ODR bit.

13.5.2 DMA Controller

The DMA controller transfers blocks of data independent of the local processor or PCI hosts. Data movement occurs on the PCI bus and/or CSB. The DMA module has four high-speed DMA channels, which share buffer space in the IOS to facilitate the gathering and sending of data. Both the local processor and PCI masters can initiate a DMA transfer.

Features of the DMA controller include the following:

- Four channels
- Concurrent execution across multiple channels with programmable bandwidth control
- All channels are accessible by local processor and remote PCI masters
- Unaligned transfer capability
- Data chaining and direct mode
- Interrupt on completed segment, chain, and error

Figure 13-18 shows a diagram of the DMA controller in the integrated device.

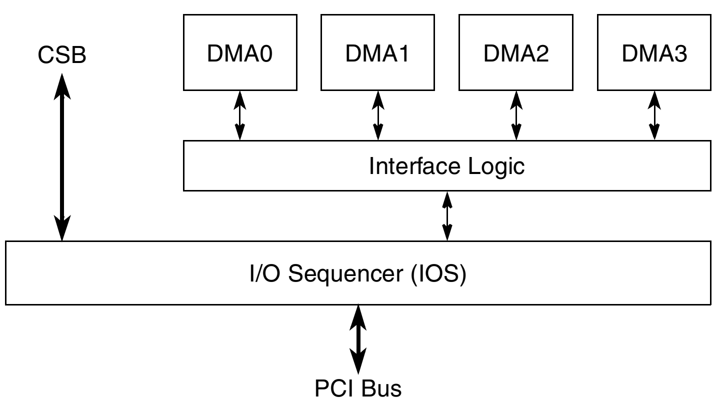


Figure 13-18. DMA Controller Block Diagram

13.5.3 DMA Operation

The DMA controller operates in the following two modes:

- Direct mode, in direct mode, the DMA controller does not read a chain of descriptors from memory but instead uses the current parameters in the DMA registers to start a DMA transfer. The DMA transfer finishes after all the bytes specified in the byte count register have been transferred. See Section 13.6.1, "Initialization Steps in Direct Mode," for more details on initialization steps.

- Chaining mode, in chaining mode, the DMA controller loads descriptors from memory prior to a DMA transfer. The DMA controller begins the transfer according to the descriptor information loaded for each segment. Once the current segment is finished, the DMA controller reads the next descriptor from memory and begins another DMA transfer. The process is finished if the current descriptor is the last one in the chain. See Section 13.6.2, "Initialization Steps in Chaining Mode," for more details on initialization steps.

In both modes, setting the start bit in the DMA mode register begins the DMA transfer.

The DMA controller supports unaligned transfers for both the source and destination addresses. It gathers data beginning at the source address and aligns the data accordingly before sending it to the destination address. The DMA controller assumes that the source and destination addresses are valid PCI or CSB memory addresses.

Accesses to CSB memory depend on the alignment of the source and destination addresses and the size of the transfer. The DMA controller transfers a full cache line whenever possible. Misaligned destination addresses result in sub-transfers of less than a cache line on the initial and final beats of the transfer; intermediate beats transfer full cache lines. Configuring a DMA channel for address hold mode DMAMR*n* precludes cache line transfers.

PCI memory read operations depend on the PRC (PCI read command) field in the mode register, the alignment of the source address, and the size of the transfer. The DMA controller attempts to read a full cache line whenever possible. Writing to PCI memory depends on the alignment of the destination address and the size of the transfer.

13.5.3.1 External Control

The DMA transfer of any channel, in either direct mode or chaining mode, can be controlled by the DMA request input signals. External control is enabled by setting the external master start enable (EMSEN) bit instead of the channel start (CS) bit in the DMA mode register (DMAMR*n*).

When using external control, the following restrictions apply:

- Both the source and destination addresses must be aligned to 32-byte boundaries.
- In chaining mode all byte count values except the last must be multiples of 32 bytes.

A falling edge on $\overline{\text{DREQ}n}$ sets DMAMR*n*[CS] to start the transfer and asserts the corresponding $\overline{\text{DACK}n}$ output signal. The number of cache lines specified by the DMA request count (DRCNT) bit in the DMA mode register (DMAMR*n*)—or the remaining byte count, if smaller— is transferred, and the CS bit and the $\overline{\text{DACK}n}$ output signal are then cleared and negated to halt the transfer until the next $\overline{\text{DREQ}n}$ assertion.

When using the $\overline{\text{DACK}n}$ handshake signal, $\overline{\text{DREQ}n}$ should remain asserted until $\overline{\text{DACK}n}$ is asserted, at which point $\overline{\text{DREQ}n}$ may be negated. $\overline{\text{DREQ}n}$ may be asserted again to resume the transfer or start a new transfer once $\overline{\text{DACK}n}$ has been negated.

If the $\overline{\text{DACK}n}$ handshake signal is not used, $\overline{\text{DREQ}n}$ should remain asserted until the first transaction of the DMA transfer appears on the external interface, at which point $\overline{\text{DREQ}n}$ may be negated. $\overline{\text{DREQ}n}$ may be asserted again to resume the transfer or start a new transfer as early as one clock cycle after its negation, even if the current transfer is still active. The DMA controller is able to record a new assertion of DREQ*n* while a transfer is in progress, with the effect of setting DMAMR*n*[CS] again once the transfer has been halted.

Once a transfer has been completed, DMAMR*n*[EMSEN] is cleared by the DMA controller, and DMAMR*n*[CS] will not be set again until DMAMR*n*[EMSEN] has been set by the user. The assertion of DREQ*n* and the setting of DMAMR*n*[EMSEN] may occur in either order; whichever occurs later will trigger the DMA transfer.

The DDONE*n* output signal is asserted when the DMA transfer has completed, that is, all bytes specified in the byte count register or the descriptors have been transferred. This signal could be used as an indication that the number of cache lines transferred might be smaller than that specified by the DRCNT field.

NOTE

The DACK*n* and DDONE*n* output signals are intended as handshake signals for DREQ*n*. They are asserted and negated according to the DMA controller internal logic. These signals are not synchronized to the transactions appearing on the external pins of the device. Specifically, the negation of DACK*n* or the assertion of DDONE*n* does not mean that all transactions of the DMA transfer have been completed as seen on the external pins.

See Section 13.6.3, "Initialization Steps in Direct Mode with External Control," and Section 13.6.4, "Initialization Steps in Chaining Mode with External Control," for more details on initialization steps.

13.5.3.2 DMA Coherency

The four DMA channels use up to four cache lines (128 bytes) of buffer space in the IOS in addition to 16 bytes of local buffer space. Because no address snooping occurs in these internal queues, data posted in these queues is not visible to the rest of the system while a DMA transfer is in progress. It is the responsibility of application software to ensure the coherency of the region being transferred during the DMA process.

Snooping of the CPU or processor data cache is selectable during DMA transactions. A snoop bit is provided in the DMA current descriptor address register (DMACDAR*n*) and the DMA next descriptor address register (DMANDAR*n*) that allows software to control when the cache is snooped on a per segment basis.

13.5.3.3 Halt and Error Conditions

DMA transfers are halted either by clearing the CS (channel start) bit in the DMA mode register (DMAMR*n*) or when encountering an error condition. In either case, the application software can do one of the following:

- Continue the DMA transfer
- Reconfigure the DMA for a new transfer
- Leave the channel in the halted state

When a DMA channel is halted, its programming model is completely accessible. If the DMA is halted due to an error condition, the TE (transfer error) bit in the DMA status register (DMASR*n*) must be cleared before the transfer can be resumed or a new transfer initiated. Note that the TE bit is not cleared automatically by hardware.

13.5.4 DMA Segment Descriptors

DMA segment descriptors contain the source and destination addresses of the data segment, the segment byte count, and a link to the next descriptor. Segment descriptors are built on cache-line (32-byte) boundaries in either CSB or PCI memory and are linked together into chains using the next-descriptor-address field.

Table 13-18. DMA Segment Descriptor Fields

Descriptor Field	Description
Source address	Contains the source address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field will be loaded into the DMA source address register (DMASAR*n*).
Destination address	Contains the destination address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field will be loaded into the DMA destination address register (DMADAR*n*).
Next descriptor address	Points to the next descriptor in memory. After the DMA controller reads the descriptor from memory, this field will be loaded into the DMA next descriptor address register (DMANDAR*n*).
Byte count	Contains the number of bytes to transfer. After the DMA controller reads the descriptor from memory, this field will be loaded into the DMA byte count register (DMABCR*n*).

Application software initializes the current DMA current descriptor address register (DMACDAR*n*) to point to the first descriptor in the chain. For each descriptor in the chain, the DMA controller starts a new DMA transfer with the control parameters specified by the descriptor. The DMA controller traverses the descriptor chain until reaching the last descriptor (with its EOTD bit set).

Figure 13-19 shows the DMA chain of segment descriptors.

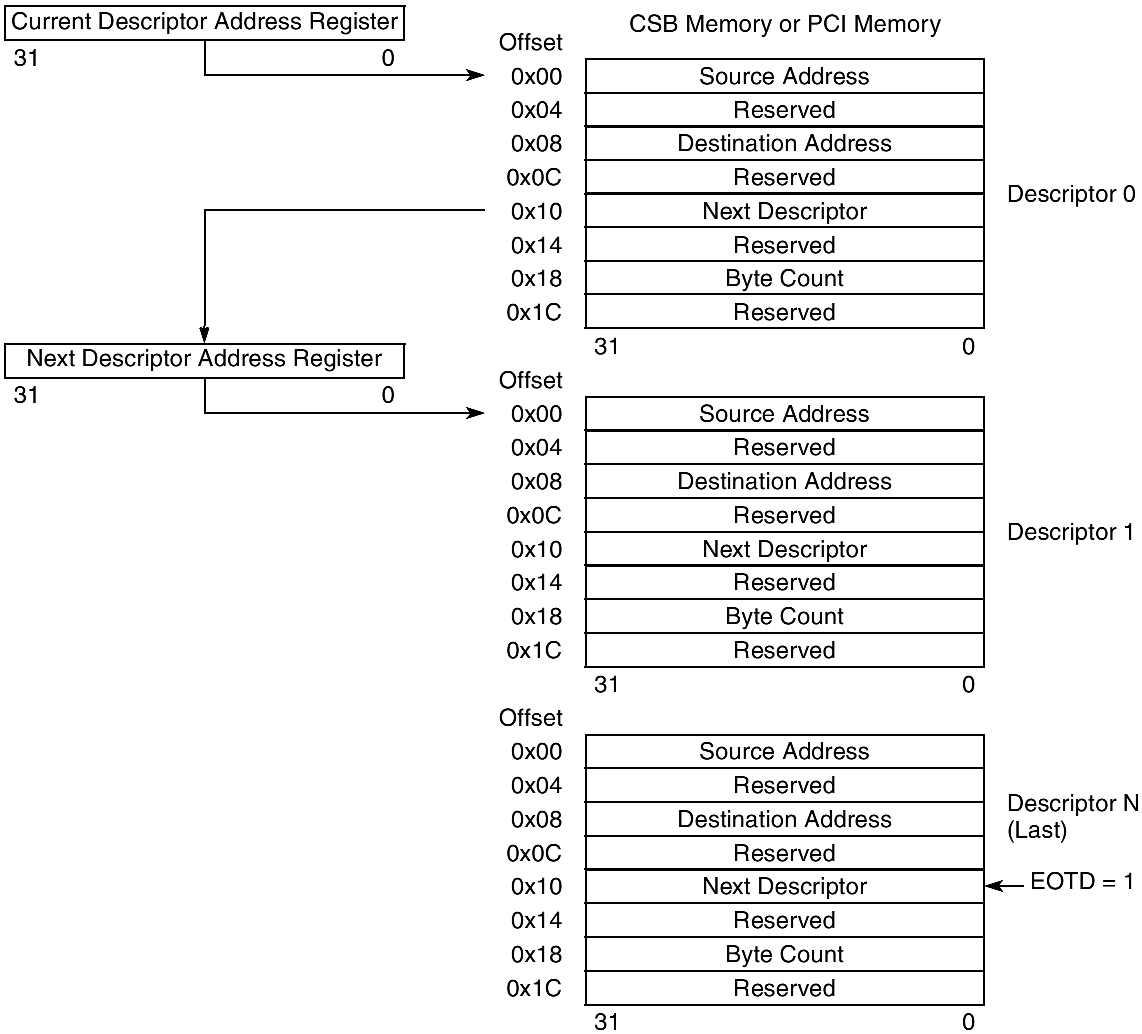


Figure 13-19. DMA Chain of Segment Descriptors

13.5.4.1 Descriptor in Big-Endian Mode

In big-endian mode, the descriptor in CSB memory should be programmed such that data appears in ascending significant-byte order. If segment descriptors are written to memory located in the CSB, they should be treated like they are translated from big-endian to little-endian mode.

Example 13-1 shows the big-endian mode descriptor's data structure. Note that the descriptor structure must be aligned on an 8-word boundary.

Example 13-1. Big-Endian Mode Descriptor's Data Structure.

```
struct {
          double a;           /* 0x1122334455667788 double word*/
          double b;           /* 0x55667788aabbccdd double word*/
          double c;           /* 0x8765432101234567 double word */
          double d;           /* 0x0123456789abcdef double word */
} Descriptor;
```

```
Results:  Source Address = 0x44332211 <MSB..LSB>
          Destination Address = 0x88776655 <MSB..LSB>
          Next Descriptor Address = 0x21436587 <MSB..LSB>
          Byte Count = 0x67452301 <MSB..LSB>
```

13.5.4.2 Descriptor in Little-Endian Mode

In little-endian mode, each segment descriptor should be programmed in descending significant-byte order.

Example 13-2 shows the little-endian mode descriptor's data structure. Note that the descriptor structure must be aligned on an 8-word boundary.

Example 13-2. Little-Endian Mode Descriptor's Data Structure.

```
struct {
          double a;           /* 0x8877665544332211 double word*/
          double b;           /* 0x1122334488776655 double word*/
          double c;           /* 0x7654321012345678 double word */
          double d;           /* 0x0123456776543210 double word */
} Descriptor;
Results:  Source Address = 0x44332211 <MSB..LSB>
          Destination Address = 0x88776655 <MSB..LSB>
          Next Descriptor Address = 0x12345678 <MSB..LSB>
          Byte Count = 0x76543210 <MSB..LSB>
```

13.6 Initialization/Application Information

13.6.1 Initialization Steps in Direct Mode

The initialization steps of a DMA transfer in direct mode are described as follows:

1. Poll the CB (channel busy) bit in the DMA status register (DMASR*n*) to make sure the DMA channel is idle.
2. Initialize the DMASAR*n*, DMADAR*n*, and the DMABCR*n*.
3. Initialize DMAMR*n*[CTM]) to indicate direct mode. Other control parameters in the mode register can also be initialized here if necessary.
4. First clear then set the DMAMR*n*[CS] to start the DMA transfer.

13.6.2 Initialization Steps in Chaining Mode

The initialization steps of a DMA transfer in chaining mode are described as follows:

1. Build a chain of descriptor segments in memory. Refer to Section 13.5.4, "DMA Segment Descriptors."
2. Poll the DMASR*n*[CB] to make sure the DMA channel is idle.
3. Initialize the DMACDAR*n* to point to the first descriptor in the chain.
4. Initialize the DMAMR*n*[CTM] to indicate chaining mode. Other control parameters in the mode register can also be initialized here if necessary.

5. First clear then set the DMAMR*n*[CS] to start the DMA transfer.

13.6.3 Initialization Steps in Direct Mode with External Control

The initialization steps of a DMA transfer in direct mode with external control are described as follows:

1. Poll the CB (channel busy) bit in the DMA status register (DMASR*n*) to make sure the DMA channel is idle.
2. Initialize the DMA source address register (DMASAR*n*), the DMA destination address register (DMADAR*n*), and the DMA byte count register (DMABCR*n*).
3. Set DMAMR*n*[CTM]) to indicate direct mode, program the DRCNT field, and set the EMSEN bit. Other control parameters in the mode register can also be initialized here if necessary.

13.6.4 Initialization Steps in Chaining Mode with External Control

The initialization steps of a DMA transfer in chaining mode with external control are described as follows:

1. Build a chain of descriptor segments in memory. Refer to Section 13.5.4, "DMA Segment Descriptors," for more information.
2. Poll the CB (channel busy) bit in the DMA status register (DMASR*n*) to make sure the DMA channel is idle.
3. Initialize the DMA current descriptor address register (DMACDAR*n*) to point to the first descriptor in the chain.
4. Clear the DMAMR*n*[CTM] to indicate chaining mode, program the DRCNT field, and set the EMSEN bit. Other control parameters in the mode register can also be initialized here if necessary.

Chapter 14
PCI Bus Interface

The PCI interface is compatible with the *PCI Local Bus Specification*, Rev. 2.3. It is beyond the scope of this manual to document the intricacies of PCI. This chapter describes the PCI controller and provides a basic description of the PCI bus operations. The specific emphasis is directed at how this device implements the PCI specification. Designers of systems incorporating PCI devices should refer to the respective specifications for a thorough description of the PCI buses.

NOTE

Much of the available PCI literature refers to a 16-bit quantity as a WORD and a 32-bit quantity as a DWORD. Because this is inconsistent with the terminology in this manual, the terms ‘word’ and ‘double word’ are not used in this chapter. Instead, the number of bits or bytes indicates the exact quantity.

14.1 Introduction

The PCI controller acts as a bridge between the PCI interface and the CSB. The I/O sequencer buffers the data. Figure 13-1 is a high-level block diagram of the PCI controller.

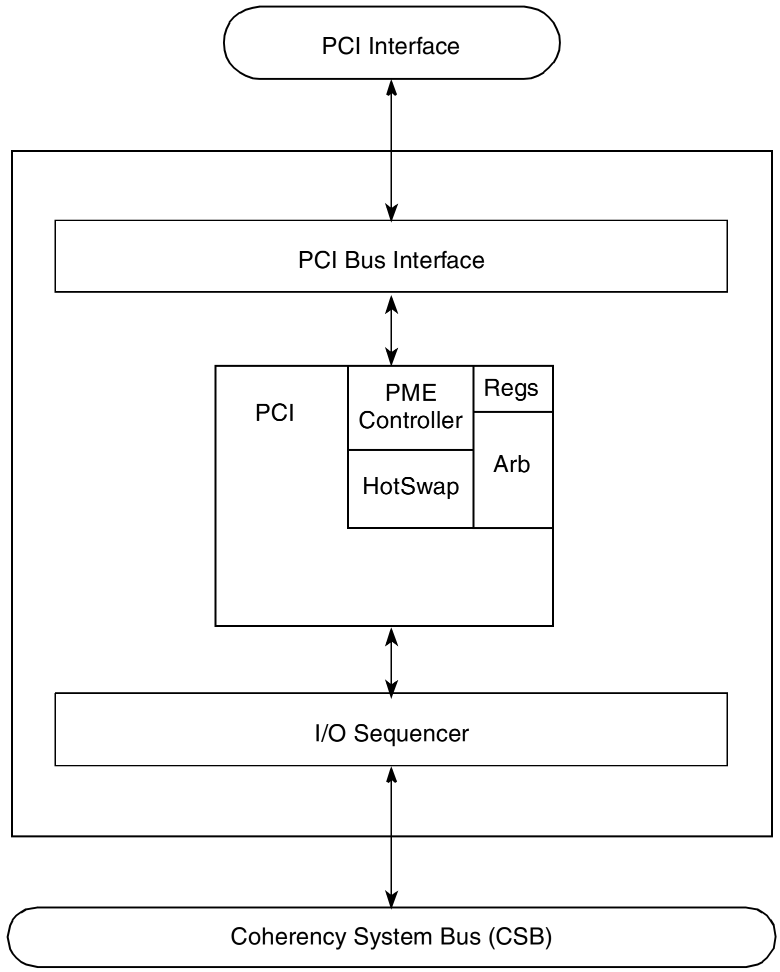


Figure 14-1. PCI Controller Block Diagram

The PCI controller connects the processor and memory system to the I/O components through the PCI system bus. This interface acts as both initiator (master) and target (slave) device. The PCI controller uses a 32-bit multiplexed, address/data bus that can run at frequencies up to 66-MHz. The interface provides address and data parity with error checking and reporting. The interface provides for three physical address spaces—64-bit address memory, 32-bit address I/O, and PCI configuration space.

The PCI interface can function as either a PCI host bridge referred to as host mode or a peripheral device on the PCI bus referred to as agent mode. See Section 14.4.4.4, “Host Mode Configuration Access,” for more information. Note that the PCI controller can be configured from the PCI bus while in agent mode. An address translation mechanism is provided to map PCI memory windows between the PCI bus and the internal bus.

The PCI interface does not flush pending outbound writes as a result of an inbound read command. Systems must not rely on inbound reads to ensure all pending outbound writes have completed. For example, consider the case where a core writes data to a PCI device and then updates a flag in the local DDR memory indicating the write to PCI has completed. An external PCI master may misread the flag ahead of the actual write transaction's completion on the PCI bus.

14.1.1 Features

The PCI controller includes the following features:

- PCI specification revision 2.3 compliant
- 32-bit PCI interface support
- Host and agent mode support
- PCI bus power management unit
- Supports accesses to all PCI address spaces
- 64-bit dual-address cycle (DAC) support (as a target only)
- Internal configuration registers accessible from PCI
- On-chip arbitration supporting five masters on PCI
- Arbiter supports two-level priority request/grant signal pairs
- Supports PCI-to-memory and memory-to-PCI streaming
- Memory prefetching of PCI read accesses and support for delayed read transactions
- Supports posting of processor-to-PCI and PCI-to-memory writes
- Supports selectable snooping for inbound transactions
- Address translation units for address mapping between host and peripheral
- Supports parity
- PCI 3.3-V compatible

14.1.2 Modes of Operation

PCI controller modes of operation are determined at reset by the reset configuration word high (RCWH) as described in Section 4.3.2, "Reset Configuration Words." Table 14-1 summarizes these modes.

Table 14-1. PCI Controller Modes

Parameter	Description	Section/Page
Host/agent configuration	Selects between host and agent mode for the PCI interface.	4.3.2.2.1/4-18
PCI arbiter enable	Enables the on-chip PCI bus arbiter	4.3.2.2/4-16

14.1.2.1 Host/Agent Mode Configuration

The PCI controller can function as either a PCI host bridge (referred to as host mode) or a peripheral device on the PCI bus (referred to as agent mode). Note that host/agent mode selection is determined at power-up as summarized in Section 4.3.2.2.1, "PCI Host/Agent Configuration."

When the device powers up in host m ode, all inbound configuration accesses are ignored (and thus master aborted). When the device powers up in agent mode, it acknowledges inbound configuration accesses. Note that in PCI agent mode, the PCI controller ignores all PCI memory accesses except those to the memory-mapped registers until inbound address translation is enabled. In agent mode, configuration cycles are acknowledged if CFG_LOCK is 0 (see Section 14.3.3.24, “PCI Function Configuration Register”), either from reset configuration or after being cleared by software.

14.1.2.2 PCI Arbiter Configuration

The interface can be configured to use an on-chip or off-chip PCI arbiter. Arbitration for PCI is determined by the value in RCWH[PCIARB]. See Section 4.3.2.2, “Reset Configuration Word High Register (RCWHR),” for more information.

14.2 External Signal Description

Table 13-2 shows the properties of the PCI signals.

Table 14-2. Signal Properties

Name	Function	Reset State	Pull Up
CPCI_HS_ENUM	CompactPCI hot swap enumerator	High impedance	Required
CPCI_HS_ES	CompactPCI hot swap ejector switch	—	—
CPCI_HS_LED	CompactPCI hot swap LED	Asserted	—
M66EN	66-MHz enable	—	—
PCI_AD[31:0]	PCI*n* address / data	High impedance	—
PCI_C/BE[3:0]	PCI bus command / byte enable	High impedance	—
PCI_DEVSEL	PCI device select	High impedance	Required
PCI_FRAME	PCI cycle frame	High impedance	Required
PCI_REQ[0:4]	PCI arbiter requests	Configuration-dependent	Required on inputs
PCI_GNT[0:4]	PCI arbiter grants	Configuration-dependent	—
PCI_IDSEL	PCI initialization device select	—	—
PCI_INTA	PCI interrupt A	High impedance	Required
PCI_IRDY	PCI initiator ready	High impedance	Required
PCI_PAR	PCI parity	High impedance	—
PCI_PERR	PCI parity error	High impedance	Required
PCI_RESET_OUT	PCI reset output	Asserted	
PCI_SERR	PCI system error	High impedance	Required
PCI_STOP	PCI stop	High impedance	Required
PCI_TRDY	PCI target ready	High impedance	Required
PCI_PME	PCI PME assertion request	High impedance	Required

Figure 14-2 shows the external PCI signals.

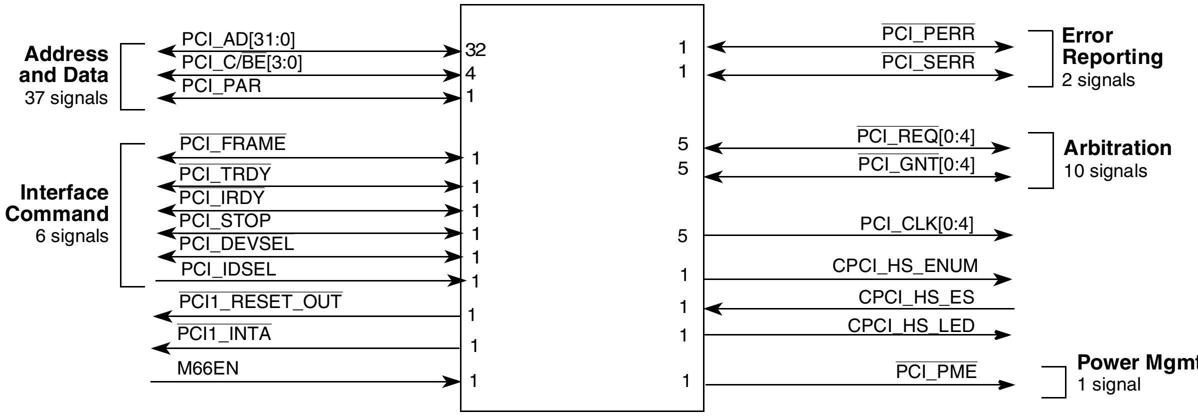


Figure 14-2. PCI Interface External Signals

Table 14-3 contains detailed descriptions of the external PCI interface signals.

Table 14-3. PCI Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
CPCI_HS_ENUM	O	CompactPCI hot swap enumerator. Used for the hot swap interface to connect to the host as the enumeration request in a compact PCI system. This signal is used for agent mode only.	
		State Meaning	Asserted—This card was inserted and needs to be configured, or this card is about to be extracted and needs to be removed from the system resources list. Negated—No action is needed.
		Timing	Assertion/Negation—No timing is specified
CPCI_HS_ES	I	CompactPCI hot swap ejector switch. Used for agent mode only. In a compact PCI system this input signal is used for the hot swap interface to connect to the ejector switch logic.	
		State Meaning	Asserted—The switch is open. Negated—The switch is closed.
		Timing	Assertion/Negation—No timing is specified
CPCI_HS_LED	O	CompactPCI hot swap LED. Used for the hot swap interface to connect to the hot swap LED in a CompactPCI system. This signal is used for agent mode only.	
		State Meaning	Asserted—Output is driving logic 1 to illuminate the hot swap LED. Negated—Output is driving logic 0 to turn off the hot swap LED.
		Timing	Assertion/Negation—No timing is specified
M66EN	I	66-MHz enable. Determines the AC timing of the PCI interface.	
		State Meaning	Asserted—The PCI interface signals use the 66-MHz PCI AC timing parameters. Negated—The PCI interface signals use the 33-MHz PCI AC timing parameters.
		Timing	Assertion/Negation—Constant

Table 14-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O		Description
PCI_AD[31:0]	I/O		PCI address/data bus. During an address phase, these signals contain a physical address. During a data phase, these signals contain the data bytes.
	O		Outputs for the bi-directional PCI address/data bus.
		State Meaning	Asserted/Negated—Represents the physical address during the address phase of a PCI transaction. During the data phase(s) of a PCI transaction, the PCI address/data bus contain the data being written. PCI_AD[7:0] define the LSB and, PCI_AD[31:24] define the MSB.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I		Inputs for the bi-directional PCI address/data bus.
		State Meaning	Asserted/Negated—Represents the address to be decoded as a check for device select during the address phase of a PCI transaction or the data being received during the data phase(s) of a PCI transaction. PCI_AD[7:0] define the LSB and, PCI_AD[31:24] define the MSB.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_C/$\overline{\text{BE}}$[3:0]	I/O		PCI bus command/byte enable.
	O		Outputs for the bi-directional command/byte enable.
		State Meaning	Asserted/Negated—During the address phase, PCI_CBE[3:0],define the bus command. Byte enables determine which byte lanes carry meaningful data for PCI bus data phases. The PCI_CBE[0] signal applies to the LSB.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I		Inputs for the bi-directional command/byte enable.
		State Meaning	Asserted/Negated—During the address phase, PCI_CBE[3:0], indicate the command that another master is sending. During the PCI bus data phase, PCI_CBE[3:0], indicate which byte lanes are valid.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
$\overline{\text{PCI\_DEVSEL}}$	I/O		PCI device select.
	O		Outputs for the bi-directional device select.
		State Meaning	Asserted—The PCI controller has decoded the address and is the target of the current access. Negated—The PCI controller has decoded the address and is not the target of the current access.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I		Inputs for the bi-directional device select.
		State Meaning	Asserted—Some PCI agents (other than this PCI controller) have decoded its address as the target of the current access. Negated—No PCI agent has been selected.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 14-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_FRAME	I/O	PCI cycle frame. Used by the current PCI master to indicate the beginning and duration of an access.	
	O	Outputs for the bi-directional frame.	
		State Meaning	Asserted—The PCI controller acting as a PCI master which is initiating a bus transaction. While PCI_FRAME is asserted, data transfers may continue. Negated—If PCI_IRDY is asserted, indicates that the PCI transaction is in the final data phase; if PCI_IRDY is negated, indicates that the PCI bus is idle.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional frame.	
		State Meaning	Asserted—Another PCI master is initiating a bus transaction. Negated—The transaction is in the final data phase or that the bus is idle.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_GNT0	I/O	PCI arbiter grants. Output signal on this PCI controller when the arbiter is enabled. Input signal when the arbiter is disabled. **Note:** PCI_GNT[0] is a point-to-point signal. Every master has its own bus grant signal.	
	O	Outputs for the bi-directional arbiter grants.	
		State Meaning	Asserted—The PCI controller granted control of the PCI bus to agent 0. Negated—The PCI controller did not grant control of the PCI bus to agent 0.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional arbiter grants.	
		State Meaning	Asserted—The PCI controller has been granted control of the PCI bus by an external arbiter. Negated—The PCI controller has not been granted control of the PCI bus by an external arbiter.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_GNT[1:4]	O	PCI arbiter grants. Output signals on this PCI controller when the arbiter is enabled. Note that PCI_GNT*n* is a point-to-point signal. Every master has its own bus grant signal.	
		State Meaning	Asserted—The PCI controller granted control of the PCI bus to agent *n*. Negated—The PCI controller did not grant control of the PCI bus to agent *n*.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_IDSEL	I	PCI initialization device select. Used as a chip select during a PCI configuration cycle in agent mode. This signal should be tied low in host mode.	
		State Meaning	Asserted—The PCI controller is being selected as a target of a configuration read or write transactions. Negated—The PCI controller is not being selected as a target of configuration read or write transactions.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 14-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_INTA	O	PCI interrupt A.	
		State Meaning	Asserted—The PCI controller signals an interrupt to the PCI host. Negated—The PCI controller is not currently signalling an interrupt.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_IRDY	I/O	PCI initiator ready. This signal is driven by the PCI controller when it is the initiator of a PCI transfer.	
	O	Outputs for the bi-directional initiator ready.	
		State Meaning	Asserted—The PCI controller, acting as a PCI master, can complete the current data phase of a PCI transaction. During a write, this PCI controller asserts PCI_IRDY to indicate that valid data is present on PCI_AD[31:0]. During a read, this PCI controller asserts PCI_IRDY to indicate that it is prepared to accept data. Negated—The PCI target needs to wait before this PCI controller, acting as a PCI master, can complete the current data phase. During a write, this PCI controller negates PCI_IRDY to insert a wait cycle when it cannot provide valid data to the target. During a read, this PCI controller negates PCI_IRDY to insert a wait cycle when it cannot accept data from the target.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional initiator ready.	
		State Meaning	Asserted—Another PCI master can complete the current data phase of a transaction. Negated—If PCI_FRAME is asserted, indicates a wait cycle from another master. If PCI_FRAME is negated, indicates that the PCI bus is idle.
PCI_PAR	I/O	PCI parity.	
	O	Outputs for the bi-directional parity.	
		State Meaning	Asserted—Odd parity across PCI_AD[31:0] and PCI_CBE[3:0] during address and data phases. Negated—Even parity across PCI_AD[31:0] and PCI_AD[31:0] during address and data phases.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional parity.	
		State Meaning	Asserted—Odd parity driven by another PCI master or the PCI target during address and data phases. Negated—Even parity driven by another PCI master or the PCI target during address and data phases.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 14-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_PERR	I/O	PCI parity error	
	O	Outputs for the bi-directional parity error.	
		State Meaning	Asserted—The PCI controller, acting as a PCI agent, detected a data parity error. (driven by the PCI initiator on reads; driven by the PCI target on writes.) Negated—No error.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional parity error.	
		State Meaning	Asserted—Another PCI agent detects a data parity error while this PCI controller is sourcing data (this PCI controller was acting as the PCI initiator during a write, or is acting as the PCI target during a read). Negated—No error.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_PME	O	PCI PME signal	
	O	Outputs for the PCI_PME signal. This is an open-drain signal.	
		State Meaning	Asserted—Indicates that a power management event has occurred Negated—Indicates that no power management event has occurred
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_REQ0	I/O	PCI bus request. Input signal on this PCI controller when the arbiter is enabled. Output signal when the arbiter is disabled. Note that PCI_REQ*n* is a point-to-point signal. Every master has its own bus request signal.	
	O	Outputs for the bi-directional bus request.	
		State Meaning	Asserted—The PCI controller is requesting control of the PCI bus to perform a transaction. Negated—The PCI controller does not require use of the PCI bus.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Input for the bi-directional bus request.	
		State Meaning	Asserted—Agent 0 is requesting control of the PCI bus to perform a transaction. Negated—Agent 0 does not require use of the PCI bus.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_REQ[1:4]	I	PCI bus request. Input signals on this PCI controller when the arbiter is enabled. Note that PCI_REQ[n] is a point-to-point signal. Every master has its own bus request signal. Following is the state meaning for the PCI_REQ[n] input.	
		State Meaning	Asserted—An agent *n* is requesting control of the PCI bus to perform a transaction. Negated—An agent *n* does not require use of the PCI bus.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_RESET_OUT	O	PCI reset. This signal is used only in host mode. It should be left unconnected in agent mode.	
		State Meaning	Asserted—Devices on the PCI bus are in reset. Negated—Devices on the PCI bus operate normally.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

Table 14-3. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_SERR	I/O	PCI system error	
	O	Outputs for the bi-directional system error.	
		State Meaning	Asserted—An address parity error, a target-abort (when this PCI controller is acting as the initiator), or some other system error (where the result is a catastrophic error) was detected. Negated—No error.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional system error.	
		State Meaning	Asserted—A device (other than this PCI controller) has detected a catastrophic error. Negated—No error.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_STOP	I/O	PCI stop.	
	O	Outputs for the bi-directional stop.	
		State Meaning	Asserted—The PCI controller, acting as a PCI target, is requesting that the initiator stop the current transaction. Negated—The current transaction can continue.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional stop.	
		State Meaning	Asserted—A target is requesting that this PCI controller, as the initiator, stop the current transaction. Negated—The current transaction can continue.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
PCI_TRDY	I/O	PCI target ready.	
	O	Outputs for the bi-directional target ready.	
		State Meaning	Asserted—The PCI controller, acting as a PCI target, can complete the current data phase of a PCI transaction. During a read, this PCI controller asserts PCI_TRDY to indicate that valid data is present on PCI_AD[31:0]. During a write, this PCI controller asserts PCI_TRDY to indicate that it is prepared to accept data. Negated—The PCI initiator needs to wait before this PCI controller, acting as a PCI target, can complete the current data phase. During a read, this PCI controller negates PCI_TRDY to insert a wait cycle when it cannot provide valid data to the initiator. During a write, this PCI controller negates PCI_TRDY to insert a wait cycle when it cannot accept data from the initiator.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3
	I	Inputs for the bi-directional target ready.	
		State Meaning	Asserted—Another PCI target is able to complete the current data phase of a transaction. Negated—A wait cycle from another target.
		Timing	Assertion/Negation—As specified by *PCI Local Bus Specification Rev* 2.3

14.3 Memory Map/Register Definitions

The PCI controller has the following types of registers:

- The PCI configuration access registers. Used for generating PCI configuration accesses from the CSB. These registers, listed in Table 14-4, are memory-mapped on the CSB and accessed through the IMMR window.
- The PCI memory-mapped registers. Used to manage error functions, general control and status, and address translation control for the inbound path. These registers are shown in Table 14-5. They can be accessed by PCI masters via the PCI controller to the CSB through the PIMMR inbound window. Note that Table 14-5 does not list outbound address translation registers; these are contained in the I/O sequencer (IOS) memory-mapped registers. See Chapter 13, "DMA/Messaging Unit," for more information.
- The PCI configuration space registers. Defined by the PCI specification. These registers are accessed by PCI masters using configuration accesses and are described in Section 14.3.3, "PCI Configuration Space Registers."

Table 14-4. PCI Configuration Access Registers

Offset	Register	Access	Reset	Section/Page
PCI Configuration Access Registers—Block Base Address 0x0_8300				
0x00	PCI_CONFIG_ADDRESS	W	0x0000_0000	14.3.1.1/14-13
0x04	PCI_CONFIG_DATA	R/W	0x0000_0000	14.3.1.2/14-14
0x08	PCI_INT_ACK	R	N/A	14.3.1.3/14-15
0x80	PCIPMR0—PCI power management register 0	R	0x7E4B_0001	14.3.3.27/14-42
0x84	PCIPMR1—PCI power management register 1	R/W	0x00*n*0_0000	14.3.3.28/14-43
0x88–0xFF	Reserved	—	—	—

Table 14-5. PCI Memory-Mapped Registers

Offset	Register	Access	Reset	Section/Page
PCI Controller—Block Base Address 0x0_8500				
PCI Error Management Registers				
0x00	PCI error status register (PCI_ESR)	w1c	0x0000_0000	14.3.2.1/14-15
0x04	PCI error capture disable register (PCI_ECDR)	R/W	0x0000_0000	14.3.2.2/14-16
0x08	PCI error enable register (PCI_EER)	R/W	0x0000_0000	14.3.2.3/14-17
0x0C	PCI error attributes capture register (PCI_EATCR)	R/W	0x0000_0000	14.3.2.4/14-18
0x10	PCI error address capture register (PCI_EACR)	R	0x0000_0000	14.3.2.5/14-19
0x14	PCI error extended address capture register (PCI_EEACR)	R	0x0000_0000	14.3.2.6/14-20
0x18	PCI error data capture register (PCI_EDCR)	R/W	0x0000_0000	14.3.2.7/14-20

Table 14-5. PCI Memory-Mapped Registers (continued)

Offset	Register	Access	Reset	Section/Page
PCI Control and Status Registers				
0x20	PCI general control register (PCI_GCR)	R/W	0x0000_0000	14.3.2.8/14-21
0x24	PCI error control register (PCI_ECR)	R/W	0x0000_0000	14.3.2.9/14-21
0x28	PCI general status register (PCI_GSR)	R	0x0000_0000	14.3.2.10/14-22
PCI Inbound ATU Registers				
0x38	PCI inbound translation address register 2 (PITAR2)	R/W	0x0000_0000	14.3.2.11/14-23
0x3C	Reserved	—	—	—
0x40	PCI inbound base address register 2 (PIBAR2)	R/W	0x0000_0000	14.3.2.12/14-24
0x44	PCI inbound extended base address register 2 (PIEBAR2)	R/W	0x0000_0000	14.3.2.13/14-24
0x48	PCI inbound window attributes register 2 (PIWAR2)	R/W	0x0000_0000	14.3.2.14/14-25
0x50	PCI inbound translation address register 1 (PITAR1)	R/W	0x0000_0000	14.3.2.11/14-23
0x54	Reserved	—	—	—
0x58	PCI inbound base address register 1 (PIBAR1)	R/W	0x0000_0000	14.3.2.12/14-24
0x5C	PCI inbound extended base address register 1 (PIEBAR1)	R/W	0x0000_0000	14.3.2.13/14-24
0x60	PCI inbound window attributes register 1 (PIWAR1)	R/W	0x0000_0000	14.3.2.14/14-25
0x68	PCI inbound translation address register 0 (PITAR0)	R/W	0x0000_0000	14.3.2.11/14-23
0x6C	Reserved	—	—	—
0x70	PCI inbound base address register 0 (PIBAR0)	R/W	0x0000_0000	14.3.2.12/14-24
0x78	PCI inbound window attributes register 0 (PIWAR0)	R/W	0x0000_0000	14.3.2.13/14-24
0x7C–0xFF	Reserved	—	—	—

14.3.1 PCI Configuration Access Registers

This section describes the registers used to allow a local bus master to access the PCI configuration space, and generate special cycle or interrupt acknowledge transactions on the PCI bus. A special case provides access to the PCI controller’s internal PCI configuration registers. The PCI registers, PCI_CONFIG_ADDRESS, PCI_CONFIG_DATA, and PCI_INT_ACK, are little-endian registers.

14.3.1.1 PCI_CONFIG_ADDRESS

Figure 14-3 shows the PCI_CONFIG_ADDRESS register fields.

Offset 0x0 — Access: Write only

	31	30–24	23–16	15–11	10–8	7–2	1–0
R		—					—
W	EN		BN	DN	FN	RN	
Reset	All zeros						

Figure 14-3. PCI_CONFIG_ADDRESS Register

The PCI_CONFIG_ADDRESS register holds the address for an access to the PCI configuration space from the local bus. This register must be programmed before accessing PCI_CONFIG_DATA to perform the transaction. Only 32-bit accesses are permitted.

If EN = 1, BN = 0, and DN = 0, the access is to the internal PCI configuration registers, so no transaction is generated on the PCI bus.

If EN = 1, BN = 0, DN = 31, FN = 7, and RN = 0, writing to PCI_CONFIG_DATA generates a special cycle transaction and reading from PCI_CONFIG_DATA generates an interrupt acknowledge transaction.

Table 14-6 shows the bit settings of the PCI_CONFIG_ADDRESS register.

Table 14-6. PCI_CONFIG_ADDRESS Field Descriptions

Bits	Name	Description
31	EN	Enable configuration transaction. Determines the type of transaction to be generated. 0 No configuration transaction will be generated by accessing the CONFIG_DATA register. Such an access will be passed through to the PCI bus as an I/O transaction. Since this is generally not desirable, the user should not access CONFIG_DATA when the EN bit is 0. 1 A configuration transaction will be generated by accessing the CONFIG_DATA register if BN and DN are not both zero.
30–24	—	Reserved
23–16	BN	Bus number. Specifies the bus segment to which a configuration transaction is directed. If this field is 0, a Type 0 configuration transaction is generated. Otherwise, a Type 1 configuration transaction is generated.

Table 14-6. PCI_CONFIG_ADDRESS Field Descriptions (continued)

Bits	Name	Description
15–11	DN	Device number. Specifies the device to which a configuration transaction is directed. For a Type 0 configuration transaction, this field is decoded to individual PCI1_IDSEL signals for the address phase according to the following values. For a Type 1 configuration transaction, this field is used directly for the address phase. **Value** — **AD Signal that is Driving High** 01010 — 31 01011 — 11 01100 — 12 01101 — 13 01110 — 14 01111 — 15 10000 — 16 10001 — 17 10010 — 18 10011 — 19 10100 — 20 10101 — 21 10110 — 22 10111 — 23 11000 — 24 11001 — 25 11010 — 26 11011 — 27 11100 — 28 11101 — 29 11110 — 30 11111 — Special cycle / interrupt acknowledge 00000 — Internal access Others — Reserved
10–8	FN	Function number. Specifies the function to which the configuration transaction is directed on a multi-function device. It is used directly in the address phase of the configuration transaction.
7–2	RN	Register number. Specifies the register being accessed in the PCI configuration space.
1–0	—	Reserved

14.3.1.2 PCI_CONFIG_DATA

An access to PCI_CONFIG_DATA usually generates a PCI configuration transaction if PCI_CONFIG_ADDRESS[EN] is set. There are some exceptions contained in the description of PCI_CONFIG_ADDRESS[EN].

This register may be accessed with an 8-, 16-, or 32-bit access, depending on the width of the register targeted by the configuration transaction.

Figure 14-4 shows the PCI_CONFIG_DATA register fields.

Offset 0x4 — Access: Read/Write

	31–0
R	CFG_DATA
W	
Reset	All zeros

Figure 14-4. PCI_CONFIG_DATA

Table 14-7 shows the bit settings of the PCI_CONFIG_DATA register.

Table 14-7. PCI_CONFIG_DATA Field Descriptions

Bits	Name	Description
31–0	CFG_DATA	Configuration data. This field contains the data transferred on a PCI configuration transaction.

14.3.1.3 PCI Interrupt Acknowledge Register (PCI_INT_ACK)

Reading this register generates an interrupt acknowledge transaction on the PCI bus. The value that is read is undefined.

14.3.2 PCI Memory-Mapped Control and Status Registers

This section describes the control and status registers.

14.3.2.1 PCI Error Status Register (PCI_ESR)

The PCI error status register (PCI_ESR) contains status bits for various types of error conditions captured by the PCI controller. Each status bit is set when the corresponding error condition is captured. PCI_ESR is a write-1-to-clear type register. A bit is cleared whenever the register is written and the data in the corresponding bit location is a 1. Figure 14-5 shows the PCI_ESR fields.

Offset 0x00 — Access: w1c

	0	1–20	21	22	23	24	25	26	27–31
R	MERR	—	APAR	PCISERR	MPERR	TPERR	NORSP	TABT	—
W	w1c		w1c	w1c	w1c	w1c	w1c	w1c	
Reset	All zeros								

Figure 14-5. PCI Error Status Register (PCI_ESR)

Table 14-8 describes the bit settings of the PCI_ESR register.

Table 14-8. PCI_ESR Field Descriptions

Bits	Name	Description
0	MERR	Multiple errors. Set if any other bit of this register is 1 and the same error type occurs again.
1–20	—	Reserved
21	APAR	Address parity error. Set when there is an address parity error on a PCI access initiated by a device other than this PCI controller.
22	PCISERR	PCI system error.Set when the $\overline{\text{PCI\_SERR}}$ input signal is asserted. See Table 14-3 for more information on $\overline{\text{PCI\_SERR}}$.
23	MPERR	Master parity error. Set when the $\overline{\text{PCI\_PERR}}$ input signal is asserted on a write access initiated by this PCI controller or when a data parity error is detected by this PCI controller on a read access that it initiated.
24	TPERR	Target parity error. Set when this PCI controller is the target of a transaction and the $\overline{\text{PCI\_PERR}}$ input signal is asserted on a read access or a data parity error is detected by this PCI controller on a write access.
25	NORSP	No response. Set when there is no response to a transaction initiated by this PCI controller on the PCI bus (no $\overline{\text{PCI\_DEVSEL}}$ assertion).
26	TABT	Target abort. Set when a PCI target abort occurs on a transaction initiated by this PCI controller.
27–31	—	Reserved

14.3.2.2 PCI Error Capture Disable Register (PCI_ECDR)

PCI_ECDR contains fields for controlling the capture of the transaction that caused an error. Each bit corresponds to the error condition reported in the PCI error status register (PCI_ESR). Note that only the first error is captured, so disabling the capture of some error types may allow greater visibility of the significant errors.

- 1 = Do not capture the transaction that caused this error.
- 0 = Capture the transaction that caused this error.

Figure 14-6 shows the PCI_ECDR fields.

Offset 0x04 — Access: Read/Write

	0–20	21	22	23	24	25	26	27–31
R/W	—	APAR	PCISERR	MPERR	TPERR	NORSP	TABT	—
Reset	All zeros							

Figure 14-6. PCI Error Capture Disable Register (PCI_ECDR)

Table 14-9 describes the bit settings of the PCI_ECDR register.

Table 14-9. PCI_ECDR Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21	APAR	Address parity error. Disable capture for address parity errors
22	PCISERR	PCI system error. Disable capture for received $\overline{\text{PCI\_SERR}}$ errors
23	MPERR	Master parity error. Disable capture for master $\overline{\text{PCI\_PERR}}$ errors
24	TPERR	Target parity error. Disable capture for target $\overline{\text{PCI\_PERR}}$ errors
25	NORSP	No response. Disable capture for master-abort errors
26	TABT	Target abort. Disable capture for target abort errors
27–31	—	Reserved

14.3.2.3 PCI Error Enable Register (PCI_EER)

PCI_EER contains fields for enabling the assertion of an interrupt for the error conditions reported in the PCI error status register (PCI_ESR).

- 1 = The interrupt is enabled.
- 0 = The interrupt is disabled.

Figure 14-7 shows the PCI_EER fields.

Offset 0x08 — Access: Read/Write

	0–20	21	22	23	24	25	26	27–31
R/W	—	APAR	PCISERR	MPERR	TPERR	NORSP	TABT	—
Reset	All zeros							

Figure 14-7. PCI Error Enable Register (PCI_EER)

Table 14-10 describes the bit settings of the PCI_EER register.

Table 14-10. PCI_EER Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21	APAR	Address parity error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
22	PCISERR	PCI system error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
23	MPERR	Master parity error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
24	TPERR	Target parity error. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
25	NORSP	No response. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.

Table 14-10. PCI_EER Field Descriptions (continued)

Bits	Name	Description
26	TABT	Target abort. Generate an interrupt when the corresponding bit of the PCI_ESR is 1.
27–31	—	Reserved

14.3.2.4 PCI Error Attributes Capture Register (PCI_EATCR)

PCI_EATCR contains fields for storing information associated with the first PCI error captured. Figure 14-8 shows the PCI_EATCR fields.

Offset 0x0C — Access: Read/Write

	0	1–3	4–7	8–9	10–11	12–15
R/W	—	ERRTYPE	BN	—	TS	ES
Reset	All zeros					

	16–19	20–23	24–27	28–29	30	31
R/W	CMD	—	BE	—	PB	VI
Reset	All zeros					

Figure 14-8. PCI Error Attributes Capture Register (PCI_EATCR)

Table 14-11 describes the bit settings of the PCI_EATCR register.

Table 14-11. PCI_EATCR Field Descriptions

Bits	Name	Description
0	—	Reserved
1–3	ERRTYPE	First error type. This field is encoded to indicate the type of the first PCI error captured. 000 Address parity error 001 Write data parity error 010 Read data parity error 011 Master abort 100 Target abort 101 System error indication received 110 Parity error indication received on a read 111 Parity error indication received on a write
4–7	BN	Beat number. This field provides the data beat number on which the error occurred for data parity errors. The value of this field is undefined for other error types. The beat values are described as follows: 0000 1st beat 0001 2nd beat 0010 3rd beat 0011 4th beat 0100 5th beat 0101 6th beat 0110 7th beat 0111 8th beat 1000 9th beat or beyond (transaction larger than one cache line) Others Reserved

Table 14-11. PCI_EATCR Field Descriptions (continued)

Bits	Name	Description
8–9	—	Reserved
10–11	TS	Transaction size. Indicates the size of the transaction in units of doublewords (8 bytes). If the transaction crossed a cache line (32-byte) boundary, this field indicates the number of actual double words in the cache line on which the error occurred. This field is valid only if the PCI controller was the master of the transaction. 00 4 double words 01 1 double word 10 2 double words 11 3 double words
12–15	ES	Error source. This field indicates the source of the PCI transaction. 0000 External master 0101 DMA Others reserved
16–19	CMD	PCI command. Contains the PCI command PCI_CBE[3:0] of the transaction.
20–23	—	Reserved
24–27	BE	PCI byte enables. Contains the PCI byte enables PCI_CBE[3:0] for the data word.
28–29	—	Reserved
30	PB	Parity bit. Contains the PCI parity bit for the captured data word.
31	VI	Error information valid. This bit indicates that the error information captured in this register, PCI_EACR, PCI_EEACR, and PCI_EDCR is valid. 0 No valid error information 1 Error information is valid

14.3.2.5 PCI Error Address Capture Register (PCI_EACR)

PCI_EACR contains fields for storing the low portion of the address associated with the first PCI error captured. Figure 14-9 shows the PCI_EACR fields.

Offset 0x10 — Access: Read only

	0–31
R	PCI_EA
W	
Reset	All zeros

Figure 14-9. PCI Error Address Capture Register (PCI_EACR)

Table 14-12 describes the bit settings of the PCI_EACR register.

Table 14-12. PCI_EACR Field Description

Bits	Name	Description
0–31	PCI_EA	PCI error address. Contains the low portion of the address associated with the first detected error. Read only.

14.3.2.6 PCI Error Extended Address Capture Register (PCI_EEACR)

PCI_EEACR contains fields for storing the high portion of the address associated with the first PCI error captured. Figure 14-10 shows the PCI_EEACR fields.

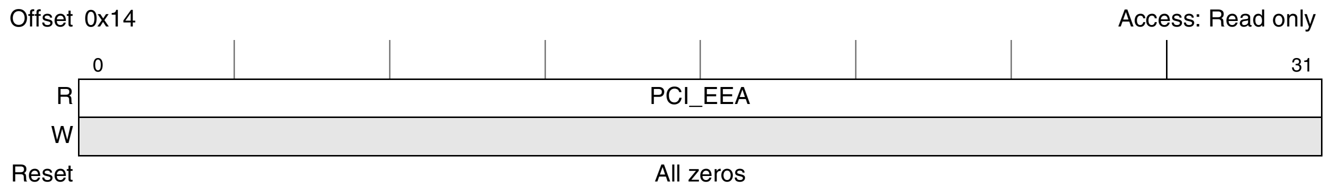


Figure 14-10. PCI Error Extended Address Capture Register (PCI_EEACR)

Table 14-13 describes the bit settings of the PCI_EEACR register.

Table 14-13. PCI_EEACR Field Description

Bits	Name	Description
0–31	PCI_EEA	PCI error extended address. Contains the high portion of the address associated with the first detected error.

14.3.2.7 PCI Error Data Low Capture Register (PCI_EDLCR)

PCI_EDLCR contains fields for storing the data associated with the first PCI error captured. Figure 14-11 shows the PCI_EDLCR fields.

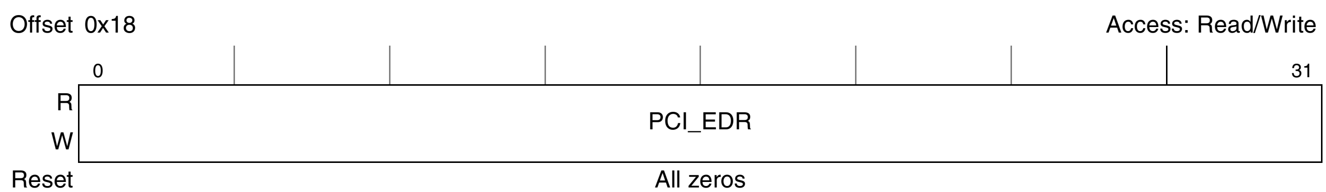


Figure 14-11. PCI Error Data Low Capture Register (PCI_EDLCR)

Table 14-14 describes the bit settings of the PCI_EDLCR register.

Table 14-14. PCI_EDLCR Field Description

Bits	Name	Description
0–31	PCI_EDR	PCI error data. Contains the data associated with the first detected error.

14.3.2.8 PCI General Control Register (PCI_GCR)

PCI_GCR contains fields for controlling the behavior of the internal arbiter, the state of the bus signals, and the PCI reset signal for host mode. Figure 14-12 shows the PCI_GCR fields.

Offset 0x20 — Access: Read/Write

	0–28	29	30	31
R/W	—	BBR	PPL	SPRST
Reset	All zeros			

Figure 14-12. PCI General Control Register (PCI_GCR)

Table 14-15 shows the bit settings of PCI_GCR. The bits that are not reserved have read and write capability.

Table 14-15. PCI_GCR Field Descriptions

Bits	Name	Description
0–28	—	Reserved
29	BBR	Block bus requests. This bit could be used to prepare for entering a low-power mode by preventing transactions on the PCI bus. 0 External bus requests are treated normally. 1 Block external bus requests. When this bit is set, all bus requests from external devices to the PCI controller's internal arbiter are blocked, and the bus is continuously granted to the PCI controller.
30	PPL	PCI pins low. This bit could be used to put the bus signals in a safe electrical state when the devices on the bus are powered down. This bit should never be set during normal operation of the PCI bus. 0 PCI pins function normally 1 PCI pins in the low state. Setting this bit forces all the output and bidirectional pins of the PCI bus to be driven low.
31	SPRST	Soft PCI reset. This bit provides software control of the $\overline{\text{PCI\_RESET\_OUT}}$ output signal. It is only valid in host mode. 0 $\overline{\text{PCI\_RESET\_OUT}}$ is driven low. 1 $\overline{\text{PCI\_RESET\_OUT}}$ is driven high.

14.3.2.9 PCI Error Control Register (PCI_ECR)

PCI_ECR contains fields for determining whether an interrupt or machine check is generated for the error conditions reported in the PCI error status register (PCI_ESR). Note that if the corresponding bit in the PCI error enable register (PCI_EER) is clear, the bit in the PCI error control register (PCI_ECR) has no effect.

1 = A machine check is generated.
0 = An interrupt is generated.

Figure 14-13 shows the PCI_ECR fields.

Offset 0x24 Access: Read/Write

	0–20	21	22	23	24	25	26	27–31
R W	—	APAR	PCISERR	MPERR	TPERR	NORSP	TABT	—
Reset	All zeros							

Figure 14-13. PCI Error Control Register (PCI_ECR)

Table 14-16 describes the bit settings of the PCI_ECR register.

Table 14-16. PCI_ECR Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21	APAR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
22	PCISERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
23	MPERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
24	TPERR	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
25	NORSP	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
26	TABT	0 An interrupt is generated if the corresponding bit of the PCI_ESR is 1. 1 A machine check is generated if the corresponding bit of the PCI_ESR is 1.
27–31	—	Reserved

14.3.2.10 PCI General Status Register (PCI_GSR)

PCI_GSR contains fields for providing status information, shown in Figure 14-14.

Offset 0x28 Access: Read-only

	0–30	31
R	—	IDLE
W		
Reset	All zeros	

Figure 14-14. PCI General Status Register (PCI_GSR)

Table 14-17 shows the bit settings of the PCI_GSR register. All bits are read-only.

Table 14-17. PCI_GSR Field Descriptions

Bits	Name	Description
0–30	—	Reserved
31	IDLE	PCI controller is idle. Indicates when the PCI bus is totally idle before setting PCI_GCR[PPL]. 0 The PCI controller is active. 1 The PCI controller is idle.

14.3.2.11 PCI Inbound Translation Address Registers (PITAR*n*)

PITAR*n* contains fields for defining the starting point of the inbound translation windows in the local memory space (see Section 14.4.6, "PCI Inbound Address Translation"; for more information on outbound address translation registers, see Chapter 12, "Sequencer"). Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.

Offset 0x38 (n = 2)
0x50 (n = 1)
0x68 (n = 0)

Access: Read/Write

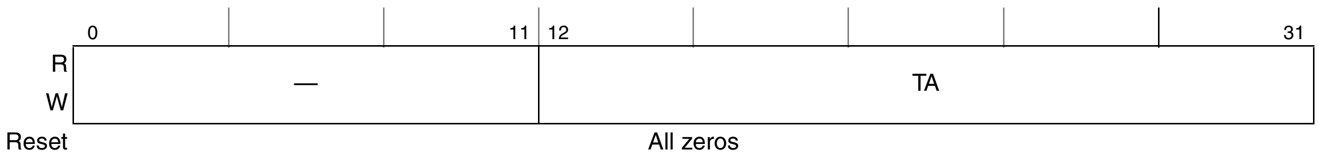


Figure 14-15. PCI Inbound Translation Address Registers (PITAR*n*)

Table 14-18 shows the bit settings of PITAR*n*.

Table 14-18. PITAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	TA	Translation address. Contains the starting address of the inbound translated address. TA corresponds to the 20 highest-order bits of a 32-bit local address. The specified address must be aligned to the window size, as defined by PIWAR*n*[IWS].

14.3.2.12 PCI Inbound Base Address Registers (PIBAR*n*)

PIBAR*n* contains fields for defining the starting point of the inbound windows in the PCI memory space. A write to a PIBAR*n* register also causes a change in the base address bits in the corresponding GPL base address register in the PCI configuration space. Figure 14-16 shows the PIBARx fields.

Offset 0x40 (n = 2)
0x58 (n = 1)
0x70 (n = 0)

Access: Read/Write

	0–31
R/W	BA
Reset	All zeros

Figure 14-16. PCI Inbound Base Address Registers (PIBAR*n*)

Table 14-19 shows the bit settings of PIBAR*n*.

Table 14-19. PIBAR*n* Field Descriptions

Bits	Name	Description
0–31	BA	Base address. Contains the starting address in the PCI memory space of the inbound window. This field corresponds to bits 43–12 of a 64-bit address. In PIBAR0, the upper 12 bits are reserved because only a 32-bit address is supported. The specified address must be aligned to the window size, as defined by PIWAR*n*[IWS].

14.3.2.13 PCI Inbound Extended Base Address Registers (PIEBAR*n*)

PIEBAR*n* contains fields for defining the high portion of the starting point of the inbound windows in the PCI memory space. Figure 14-17 shows the PIEBAR*n* fields.

Offset 0x44 (n = 2)
0x5C (n = 1)

Access: Read/Write

	0–11	12–31
R/W	—	EBA
Reset	All zeros	

Figure 14-17. PCI Inbound Extended Base Address Registers (PIEBAR*n*)

Table 14-20 shows the bit settings of PIEBAR*n*.

Table 14-20. PIEBAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	EBA	Extended base address. Contains the high portion of the starting address in the PCI memory space of the inbound base address. This 20-bit field corresponds to bits 63–44 of a 64-bit address.

14.3.2.14 PCI Inbound Window Attribute Registers (PIWAR*n*)

PIWAR*n* contains fields for defining the size of an inbound translation window. It also defines some properties of the window. Figure 14-18 shows the PIWAR*n* fields. See Section 4.3.1.1, "Reset Configuration Word Source," for more information on reset configuration.

Offset 0x48 (n = 2)
0x60 (n = 1)
0x78 (n = 0)

Access: Read/Write

	0	1	2	3–11	12–15	16–19	20–25	26–31
R/W	EN	—	PF	—	RTT	WTT	—	IWS
Reset	All zeros							

Figure 14-18. PCI Inbound Window Attribute Registers (PIWAR*n*)

Table 14-21 shows the bit settings of PIWAR*n*.

Table 14-21. PIWAR*n* Field Descriptions

Bits	Name	Description
0	EN	Enable. Used to enable the address translation window. 0 Address translation is disabled for this window. 1 Address translation is enabled for this window. PCI addresses that match the definition of the window will be recognized by the PCI controller and translated to the local memory space.
1	—	Reserved
2	PF	Prefetchable. Defines whether the transactions that are translated through this window are prefetchable on the local bus. Streaming the transactions requires the memory space to be prefetchable. 0 Not prefetchable 1 Prefetchable
3–11	—	Reserved
12–15	RTT	Read transaction type. Determines the type of transaction performed on the local bus when the PCI transaction is a read. The RTT values are described as follows: 0100 Read without snoop on system bus 0101 Read with snoop on system bus Others reserved
16–19	WTT	Write transaction type. Determines the type of transaction performed on the local bus when the PCI transaction is a write. The WTT values are described as follows: 0100 Write without snoop of local processor 0101 Write with snoop of local processor Others reserved

Table 14-21. PIWAR*n* Field Descriptions (continued)

Bits	Name	Description
20–25	—	Reserved
26–31	IWS	Inbound window size. Indicates the size of the inbound translation window. Inbound translation window size N which is the encoded 2^(N+1) bytes window size. The smallest window is 4 Kbytes (N = 11) 000000–001010 Reserved 001011 4-Kbyte window size 001100 8-Kbyte window size ... 011110 2-Gbyte window size 011111–111111 Reserved

14.3.3 PCI Configuration Space Registers

This section describes the PCI configuration space registers. These registers are shown with descending bit numbering to correspond to the PCI standard.

NOTE

The registers described in this section use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI bus.

Table 14-22 shows the PCI configuration registers that are mapped in PCI configuration space. Some fields are common to registers in both spaces to ensure consistency. These fields are discussed in the register definitions.

Table 14-22. PCI Configuration Space Registers

Address	Use	Access
00	Vendor ID configuration register	R
02	Device ID configuration register	R
04	PCI command configuration register	R/W
06	PCI status configuration register	Read/bit-reset
08	Revision ID configuration register	R
09	Standard programming interface	R
0A	Subclass code configuration register	R
0B	Base class code configuration register	R
0C	Cache line size configuration register	R/W
0D	Latency timer configuration register	R/W
0E	Header type configuration register	R
0F	BIST control configuration register	R

Table 14-22. PCI Configuration Space Registers (continued)

Address	Use	Access
10	PIMMR base address register	R/W
14	GPL base address register 0	R/W
18	GPL base address register 1	R/W
1C	GPL extended base address register 1	R/W
20	GPL base address register 2	R/W
24	GPL extended base address register 2	R/W
2C	Subsystem vendor ID configuration register	R
2E	Subsystem device ID configuration register	R
34	Capabilities pointer configuration register	R
3C	Interrupt line configuration register	R/W
3D	Interrupt pin configuration register	R
3E	Minimum grant configuration register	R
3F	Maximum latency configuration register	R
44	PCI function configuration register	R/W
46	PCI arbiter control register (PCIACR)	R/W
48	Hot swap register block	R/W
80	PCI power management register 0	R
84	PCI power management register 1	R/W

14.3.3.1 Vendor ID Configuration Register

Figure 14-19 shows the vendor ID fields. This is a read-only register.

Offset 0x00 Access: Read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	VID															
W																
Reset	0	0	0	1	1	0	0	1	0	1	0	1	0	1	1	1

Figure 14-19. Vendor ID Configuration Register

Table 14-23 shows the bit settings of the vendor ID register.

Table 14-23. Vendor ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	VID	Vendor ID. The read-only value 0x1957 specifies Freescale Semiconductor as the manufacturer of the device.

14.3.3.2 Device ID Configuration Register

Figure 14-20 shows the device ID fields. This is a read-only register.

Offset 02 | Access: Read-only

	15–0
R	DID
W	
Reset	Device-dependent

Figure 14-20. Device ID Configuration Register

Table 14-24 shows the bit settings of the device ID register.

Table 14-24. Device ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	DID	Device ID. This field identifies the device. 00C2 MPC8379E 00C3 MPC8379 00C4 MPC8378E 00C5 MPC8378 00C6 MPC8377E 00C7 MPC8377

14.3.3.3 PCI Command Configuration Register

Figure 14-21 shows the PCI command fields.

Offset 04 | Access: Mixed

	15–11	10	9	8	7	6	5	4	3	2	1	0
R	—	INTD	FB-B	SERREN	—	PERRR	—	MWI	SC	BMST	MEM	I/O
W												
Reset	All zeros											

Figure 14-21. PCI Command Configuration Register

Table 14-25 shows the bit settings of the PCI command register.

Table 14-25. PCI Command Configuration Register Field Descriptions

Bits	Name	Description
15–11	—	Reserved
10	INTD	Interrupt Disable. Setting this bit masks the $\overline{\text{PCI\_INTA}}$ output. 0 $\overline{\text{PCI\_INTA}}$ provides the device interrupt status. 1 $\overline{\text{PCI\_INTA}}$ is always negated.
9	FB-B	Fast back-to-back. Hard-wired to 0.

Table 14-25. PCI Command Configuration Register Field Descriptions (continued)

Bits	Name	Description
8	SERREN	SERR enable. This bit is an enable bit for the SERR driver. Address parity errors are reported only if this bit and bit 6 are 1. 0 $\overline{\text{PCI\_SERR}}$ is never asserted. 1 $\overline{\text{PCI\_SERR}}$ may be asserted to indicate error conditions.
7	—	Reserved
6	PERRR	Parity error response. Controls the PCI controller’s response to a parity error. 0 Parity errors are ignored and normal operation continues. 1 Standard parity error treatment.
5	—	Reserved
4	MWI	Memory-write-and-invalidate. Hard-wired to 0.
3	SC	Special cycles. Hard-wired to 0.
2	BMST	Bus master. Controls the PCI controller’s ability to be a master on the PCI bus. At reset, this bit is cleared in Agent Mode and set in Host Mode. 0 The PCI controller does not generate PCI accesses. 1 The PCI controller behaves as a bus master.
1	MEM	Memory space. Controls the response to memory space accesses. 0 The PCI controller does not respond to Memory Space accesses. 1 The PCI controller as a target responds to Memory Space accesses.
0	I/O	I/O space. Hard-wired to 0.

14.3.3.4 PCI Status Configuration Register

This register is used to record status information for PCI bus-related events. Some of the bits are hard-wired to indicate the capabilities of the PCI controller. Other bits can be cleared by writing 1 to the bit location. Figure 14-22 shows the PCI status fields.

Offset 06 Access: Mixed

	15	14	13	12	11	10	9	8	7	6	5	4	3	2		0
R	DPERR	SSERR	RMA	RTA	STA	DEVSEL_T		DPD	FB-BC	—	66M	CL	INTS	—		
W	w1c	w1c	w1c	w1c	w1c			w1c					w1c			
Reset	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0

Figure 14-22. PCI Status Configuration Register

Table 14-26 shows the bit settings of the PCI status register.

Table 14-26. PCI Status Configuration Register Field Descriptions

Bits	Name	Description
15	DPERR	Detected parity error. Set whenever the PCI controller detects a parity error on the PCI bus, even if parity error handling is disabled (as controlled by bit 6 in the PCI Command register).
14	SSERR	Signaled system error. Set whenever $\overline{\text{PCI\_SERR}}$ is asserted.

Table 14-26. PCI Status Configuration Register Field Descriptions (continued)

Bits	Name	Description
13	RMA	Received master abort. Set whenever the PCI controller, acting as the PCI master on the PCI bus, terminates a transaction (except for a special-cycle) using master-abort.
12	RTA	Received target abort. Set whenever a transaction initiated by this PCI controller on the PCI bus is terminated by a target-abort.
11	STA	Signaled target abort. Set whenever the PCI controller, acting as the PCI target on the PCI bus, issues a target-abort to a PCI master.
10–9	DEVSEL_T	DEVSEL timing. Hard-wired to 00.
8	DPD	Master data parity error. Set when a data parity error is detected on the PCI bus, if the PCI controller is the master that initiated the transaction and bit 6 in the PCI command register is set.
7	FB-BC	Fast back-to-back capable. Hard-wired to 1.
6	—	Reserved
5	66M	66-MHz capable. Hard-wired to 1.
4	CL	Capabilities list. Hard-wired to 1.
3	INTS	Interrupt status. Contains the status of the device interrupt. The value of this bit is not affected by the INTD bit of the PCI command configuration register.
2–0	—	Reserved

14.3.3.5 Revision ID Configuration Register

Figure 14-23 shows the revision ID fields.

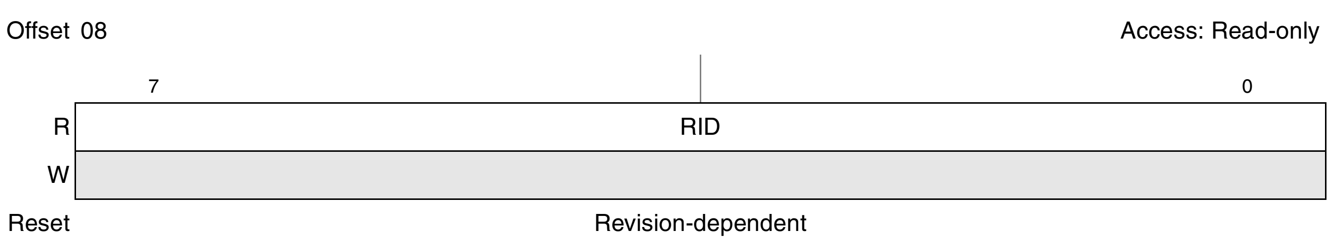


Figure 14-23. Revision ID Configuration Register

Table 14-27 shows the bit settings of the revision ID register.

Table 14-27. Revision ID Configuration Register Field Descriptions

Bits	Name	Description
7–0	RID	Revision ID. Specifies a revision code of the PCI controller.

14.3.3.6 Standard Programming Interface Configuration Register

Figure 14-24 shows the standard programming interface fields. This is the lower byte of the class code.

Offset 09 — Access: Read-only

	7–0
R	PI
W	
Reset	All zeros

Figure 14-24. Standard Programming Interface Configuration Register

Table 14-28 shows the bit settings of the standard programming interface register.

Table 14-28. Standard Programming Interface Configuration Register Field Descriptions

Bits	Name	Description
7–0	PI	Programming interface. This field is hard-wired to 0x00.

14.3.3.7 Subclass Code Configuration Register

Figure 14-25 shows the subclass code fields. This is the middle byte of the class code.

Offset 0x0A — Access: Read-only

	7							0
R	SC							
W								
Reset	0	0	1	0	0	0	0	0

Figure 14-25. Subclass Code Configuration Register

Table 14-29 shows the bit settings of the subclass code register.

Table 14-29. Subclass Code Configuration Register Field Descriptions

Bits	Name	Description
7–0	SC	Sub-class code. This field is hard-wired to 0x20, indicating a Power PC processor.

14.3.3.8 Base Class Code Configuration Register

Figure 14-26 shows the base class code fields. This is the upper byte of the class code.

Offset 0x0B — Access: Read-only

	7							0
R	BCC							
W								
Reset	0	0	0	0	1	0	1	1

Figure 14-26. Base Class Code Configuration Register

Table 14-30 shows the bit settings of the class code register.

Table 14-30. Class Code Configuration Register Field Descriptions

Bits	Name	Description
7–0	BCC	Base class code. This field is hard-wired to 0x0B, indicating a processor.

14.3.3.9 Cache Line Size Configuration Register

Figure 14-27 shows the cache line size fields.

Offset 0C — Access: Read/Write

	7 … 0
R / W	CLS
Reset	All zeros

Figure 14-27. Cache Line Size Configuration Register

Table 14-31 shows the bit settings of the cache line size register.

Table 14-31. Cache Line Size Configuration Register Field Descriptions

Bits	Name	Description
7–0	CLS	Cache line size. Cache-line in terms of 32-bit words. Although the register is writable, only the value 0x08 is legal.

14.3.3.10 Latency Timer Configuration Register

Figure 14-28 shows the latency timer fields.

Offset 0D — Access: Read/Write

	7–3	2–0
R/W	LT	—
Reset	All zeros	

Figure 14-28. Latency Timer Configuration Register

Table 14-32 shows the bit settings of the latency timer register.

Table 14-32. Latency Timer Configuration Register Field Descriptions

Bits	Name	Description
7–3	LT	Latency timer. Specifies a granularity of 8 PCI clocks, the length of time that the PCI controller, when mastering a transaction, may hold the bus as the result of a bus grant. Refer to the PCI 2.3 specification for the rules by which the PCI controller completes transactions when the timer has expired.
2–0	—	Reserved

14.3.3.11 Header Type Configuration Register

Figure 14-29 shows the read-only header type register, which is hard-wired to 0x00.

Offset 0E — Access: Read-only

	7–0
R	Header Type
W	
Reset	

Figure 14-29. Header Type Configuration Register

14.3.3.12 BIST Control Configuration Register

Figure 14-30 shows the read-only BIST control register, which is hard-wired to 0x00.

Offset 0x0F — Access: Read-only

	7–0
R	BIST
W	
Reset	All zeros

Figure 14-30. BIST Control Configuration Register

14.3.3.13 PIMMR Base Address Configuration Register

Figure 14-31 shows the PIMMR base address register fields.

Offset 10 | Access: Read/Write

	31–20	19–4	3	2–1	0
R	BA	—	PRE	T	MSI
W					
Reset	All zeros				

Figure 14-31. PIMMR Base Address Configuration Register

Table 14-33 shows the bit settings of the PIMMR base address register.

Table 14-33. PIMMR Base Address Configuration Register Field Descriptions

Bits	Name	Description
31–20	BA	Base address. Defines the base address for the internal (on-chip) memory-mapped register space. The size of this space is 1 MB.
19–4	—	Reserved
3	PRE	Prefetchable. Hard-wired to 0.
2–1	T	Type. Hard-wired to 00.
0	MSI	Memory space indicator. Hard-wired to 0

14.3.3.14 GPL Base Address Register 0

The GPL base address register 0 is provided to allow access to local memory space. This register is closely tied to PIBAR0 and PIWAR0 in the CSR memory space. A write to GPL base address register 0 also causes a change in the base address bits that are not masked according to the IWS field of PIWAR0 in PIBAR0. Note that this write operation will not change the bits that are masked by the IWS field. For read operation these masked bits will always return zeros.

Figure 14-32 shows the GPL base address register 0 fields.

Offset 0x14, 0x18 | Access: Read/Write

	31–12	11–4	3	2–1	0
R	BA	—	PRE	T	MSI
W					
Reset	All zeros				

Figure 14-32. GPL Base Address Register 0

Table 14-34 shows the bit settings of the GPL base address register 0.

Table 14-34. GPL Base Address Register 0 Field Descriptions

Bits	Name	Description
31–12	BA	Base address. Defines the base address for the inbound window. Bits 11–4 are hard-wired to 0 since the minimum window size is 4 Kbytes.
3	PRE	Prefetchable. This bit is read-only and contains the value of the PF bit in PIWAR0.
2–1	T	Type. Hard-wired to 00.
0	MSI	Memory space indicator. Hard-wired to 0

14.3.3.15 GPL Base Address Registers 1–2

The general purpose local access base address registers are provided to allow access to local memory space. These registers are closely tied to PIBAR*n* and PIWAR*n* in the CSR memory space. A write to a GPL base address register also causes a change in the base address bits that are not masked according to the IWS field of PIWAR*n* in the corresponding PIBAR*n*. Note that this write operation will not change the bits that are masked by the IWS field. For read operations, these masked bits always return zeros. Figure 14-33 shows the GPL base address register 1–2 fields.

Offset 0x18, 0x20 Access: Read/Write

	31–12	11–4	3	2–1	0
R/W	BA	—	PRE	T	MSI
Reset	0000 0000 0000 0000 0000	0000 0000	0	1 0	0

Figure 14-33. GPL Base Address Registers 1–2

Table 14-35 shows the bit settings of the GPL base address register 1–2.

Table 14-35. GPL Base Address Register 1,2 Field Descriptions

Bits	Name	Description
31–12	BA	Base address. Defines the two portion of the base address for the inbound window. Bits 11–4 are hard-wired to 0 since the minimum window size is 4 Kbytes.
11–4	—	Reserved
3	PRE	Prefetchable. This bit is read-only and contains the value of the PF bit in PIWAR*n*.
2–1	T	Type. Hard-wired to 10.
0	MSI	Memory space indicator. Hard-wired to 0.

14.3.3.16 GPL Extended Base Address Registers 1–2

Two general-purpose local access base address registers are provided to allow access to local memory space. These registers are closely tied to PIBAR*n*, PIEBAR*n*, and PIWAR*n* in the CSR memory space. A write to a GPL extended base address register also causes a change in the base address bits that are not masked according to the IWS field of PIWARx in the corresponding PIBAR*n*/PIEBAR*n*. Note that this write operation does not change bits that are masked by the IWS field. For read operations these masked bits will always return zeros.

Figure 14-34 shows the GPL extended base address registers 1–2 fields.

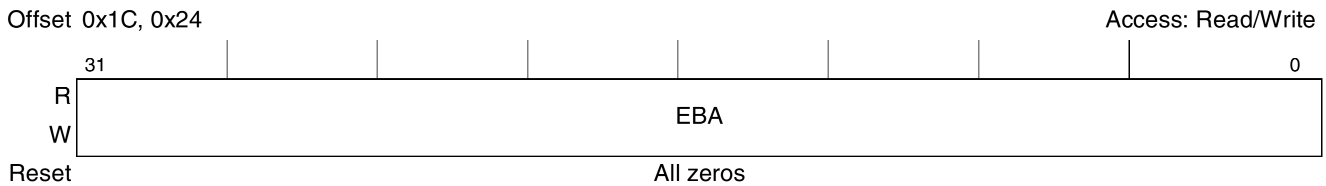


Figure 14-34. GPL Extended Base Address Registers 1–2

Table 14-36 shows the bit settings of the GPL extended base address register 1–2.

Table 14-36. GPL Extended Base Address Registers 1–2 Field Descriptions

Bits	Name	Description
31–0	EBA	Extended base address. Defines the high portion of the base address for the inbound window.

14.3.3.17 Subsystem Vendor ID Configuration Register

Figure 14-35 shows the subsystem vendor ID fields. The subsystem vendor ID configuration register is read-only from the PCI bus, but it can be programmed from the CSB.

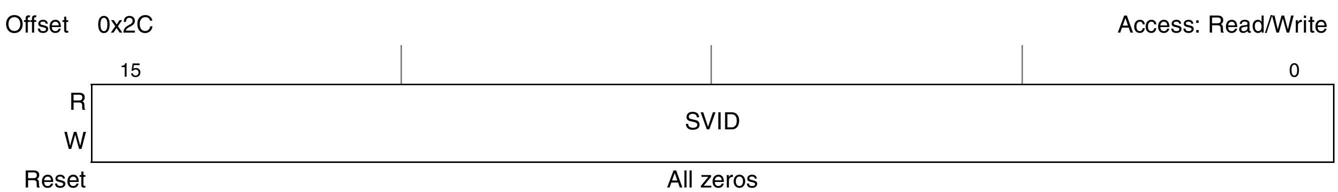


Figure 14-35. Subsystem Vendor ID Configuration Register

Table 14-37 shows the bit settings of the subsystem vendor ID configuration register.

Table 14-37. Subsystem Vendor ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	SVID	Subsystem vendor ID. Identifies the manufacturer of the board or subsystem that contains this device.

14.3.3.18 Subsystem Device ID Configuration Register

Figure 14-36 shows the subsystem device configuration register ID fields. The subsystem device ID configuration register is read-only from the PCI bus, but it can be programmed from the CSB.

Offset 0x2E | Access: Read/Write

	15 ... 0
R W	SDID
Reset	All zeros

Figure 14-36. Subsystem Device ID Configuration Register

Table 14-38 shows the bit settings of the subsystem device ID configuration register.

Table 14-38. Subsystem Device ID Configuration Register Field Descriptions

Bits	Name	Description
15–0	SDID	Subsystem device ID. This field identifies the board or subsystem that contains this device.

14.3.3.19 Capabilities Pointer Configuration Register

The capabilities pointer register specifies the byte offset in the PCI configuration space that contains the first item in the capabilities list. Figure 14-37 shows the capabilities pointer configuration register fields.

Offset 0x34 | Access: Read-only

	7							0
R	Capabilities Pointer							
W								
Reset	0	1	0	0	1	0	1	0

Figure 14-37. Capabilities Pointer Configuration Register

14.3.3.20 Interrupt Line Configuration Register

Figure 14-38 shows the interrupt line configuration register fields.

Offset 0x3C | Access: Read/Write

	7 ... 0
R W	IL
Reset	All zeros

Figure 14-38. Interrupt Line Configuration Register

Table 14-39 shows the bit settings of the interrupt line configuration register.

Table 14-39. Interrupt Line Configuration Register Field Descriptions

Bits	Name	Description
7–0	IL	Interrupt line. Used to communicate interrupt line routing information. The value has no effect on the operation of the PCI controller.

14.3.3.21 Interrupt Pin Configuration Register

The interrupt pin configuration register tells which interrupt pin is used (0x01 means PCI_INTA). Figure 14-39 shows the interrupt pin configuration register fields.

Offset 0x3D

	7							0
R	Interrupt Pin							
W								
Reset	0	0	0	0	0	0	0	1

Figure 14-39. Interrupt Pin Register

14.3.3.22 Minimum Grant Configuration Register

Figure 14-40 shows the minimum grant configuration register fields.

Offset 0x3E

	7 – 0
R	MIN_GNT
W	
Reset	All zeros

Figure 14-40. Minimum Grant Configuration Register

14.3.3.23 Maximum Latency Configuration Register

Figure 14-41 shows the maximum latency configuration register fields.

Offset 0x3F

	7	6	5	4	3	2	1	0
R	MAX_LAT							
W								
Reset	All zeros							

Figure 14-41. Maximum Latency Configuration Register

14.3.3.24 PCI Function Configuration Register

Figure 14-42 shows the PCI function configuration register fields.

Offset 0x44 Access: Read/Write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	—										CFG_ LOCK	—		TLTD	MLTD	HA
Reset	0	0	0	0	0	0	0	0	0	0	1	0	cfg	0	0	cfg

1

Figure 14-42. PCI Function Configuration Register

Table 14-40 shows the bit settings of the PCI function configuration register.

Table 14-40. PCI Function Configuration Register Field Descriptions

Bits	Name	Description
15–6	—	Reserved
5	CFG_LOCK	Configuration lock. Controls access to the PCI configuration space from the PCI port. In host mode the PCI configuration space is always inaccessible, so this bit is not used. Normally, this bit will be cleared in agent mode once the configuration of the PCI controller is complete to allow an external host to access the PCI configuration space. 0 Access to the configuration spaces is permitted. 1 Any inbound PCI access to the PCI configuration space is retried. SeeSection 4.3.1.1, "Reset Configuration Word Source," for more information on reset configuration.
3–4	—	Reserved
2	TLTD	Target latency timeout disable. Determines whether the PCI controller, while acting as a PCI target, times out when the first data phase of a transaction has not completed in 16 PCI cycles. 0 Target latency timeout enabled. 1 Target latency timeout disabled.
1	MLTD	Master latency timer disable. Determines whether the PCI controller, while acting as a PCI master, terminates a transaction upon the expiration of the master latency timer. 0 Master latency timer enabled. 1 Master latency timer disabled.
0	HA	Host/Agent. Indicates whether the PCI controller is in host mode or agent mode. It provides the value of the $\overline{\text{PCI\_HOST}}$—PCI host configuration bit is sampled at the end of the reset sequence. 0 Host mode 1 Agent mode

14.3.3.25 PCI Arbiter Control Register (PCIACR)

Figure 14-43 shows the PCI arbiter control register (PCIACR) fields.

Offset 0x46 Access: Read/Write

	15	14	13	12	11				7	6	5			2	1	0
R W	AD	PM	—	PMBD	—					PRI0	PRI1	PRI2	PRI3	PRI4	—	MPRI
Reset	cfg	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-43. PCI Arbiter Control Register (PCIACR)

Table 14-41 shows the bit settings of the PCIACR.

Table 14-41. PCI Arbiter Control Register (PCIACR) Field Descriptions

Bits	Name	Description
15	AD	Arbiter disable. Indicates whether the PCI controller functions as the arbiter for the PCI bus. It provides the value of the PCI arbiter enable configuration bit as sampled at the end of the reset sequence. SeeChapter 4, “Reset, Clocking, and Initialization,” for more information on reset configuration. 0 Arbiter enabled 1 Arbiter disabled
14	PM	Parking mode. Controls which device receives a bus grant when there are no outstanding bus requests and the bus is idle. 0 The bus is parked with the last device to use the bus. 1 The bus is parked with the PCI controller.
13	—	Reserved
12	PBMD	PCI broken master disable. Determines whether the PCI controller ignores the bus requests of an initiator that requests the bus for an excessive period without using it. 0 An initiator that requests the bus and receives the grant must begin using the bus within 16 PCI clock periods after the bus becomes idle or its request is subsequently ignored. 1 No requests are ignored.
11–7	—	Reserved
6–2	PRI*n*	Priority level for master *n*. When the PCI controller functions as the arbiter for the PCI bus, each PRIn bit determines the arbitration priority level for the PCI master connected to the REQ*n*/GNT*n* pair. 0 Low priority 1 High priority
1	—	Reserved
0	MPRI	My priority. When the PCI controller functions as the arbiter for the PCI bus, this bit determines the arbitration priority level for the PCI controller when it acts as a PCI master. 0 Low priority 1 High priority

14.3.3.26 Hot Swap Register Block

Figure 14-44 shows the hot swap register block fields.

Offset 0x48

	31–24	23	22	21–20	19	18	17	16	15–8	7–0
R	—	INS	EXT	—	LOO	—	EIM	—	NXT_PTR	CAP_ID
W		w1c	w1c							
Reset	0 0 0 0 0 0 0 0	0	0	0 0	0	0	0	0	0 0 0 0 0 0 0 0	0 0 0 0 0 1 1 0

Figure 14-44. Hot Swap Register Block

Table 14-42 shows the bit settings of the Hot Swap register block.

Table 14-42. Hot Swap Register Block Field Descriptions

Bits	Name	Description
31–24	—	Reserved
23	INS	Insertion status. Indicates that a card has been inserted. Write 1 to clear this bit.
22	EXT	Extraction status. Indicates that a card has been extracted. Write 1 to clear this bit.
21–20	—	Reserved
19	LOO	LED On/Off. Controls the LED when the hardware is in state H2 0 LED off 1 LED on
18	—	Reserved
17	EIM	ENUM mask. This bit masks the CPCI_HS_ENUM input. 0 Enabled 1 Masked
16	—	Reserved
15–8	NXT_PTR	Next pointer—hardwired to 0x80 to point to the address of the power management capability in the PCI controller.
7–0	CAP_ID	Capability ID for hot swap (hardwired to 0x06)

14.3.3.27 PCI Power Management Register 0 (PCIPMR0)

The PCI power management register 0 (PCIPMR0), shown in Figure 14-45, indicates the power management policies to implement in the system.

Table 14-43 describes the PCIPMR0 fields.

Offset 0x80 Access: Read only

Bits	31–27	26	25	24–22	21	20	19	18–16
R	PME_Support	D2_Support	D1_Support	Aux_Current	DSI	—	PME_Clock	Version
W								
Reset	0 1 1 1 1	1	1	0 0 0	0	0	1	0 1 1

Bits	15–8	7–0
R	NEXT_CAP_PTR	CAP_ID
W		
Reset	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1

Figure 14-45. PCI Power Management Register 0 (PCIPMR0)

Table 14-43. PCIPMR0 Field Descriptions

Bits	Name	Description
31–27	PME_Support	Indicates the power states in which the PCI Controller may assert PME#. PME_Support (bit 27): 0 PME# cannot be asserted from D0 1 PME# can be asserted from D0 PME_Support (bit 28): 0 PME# cannot be asserted from D1 1 PME# can be asserted from D1 PME_Support (bit 29): 0 PME# cannot be asserted from D2 1 PME# can be asserted from D2 PME_Support (bit 30): 0 PME# cannot be asserted from D3_hot 1 PME# can be asserted from D3_hot PME_Support (bit 31): 0 PME# cannot be asserted from D3_cold 1 PME# can be asserted from D3_cold
26	D2_Support	D2 power management state support 0 The PCI controller does not support D2 power management state. 1 The PCI controller supports the D2 power management state.
25	D1_Support	D1 power management state support 0 The PCI controller does not support D1 power management state. 1 The PCI controller supports the D1 power management state.
24–22	Aux_Current	Reports the 3.3 Vaux auxiliary current requirements
21	DSI	Device specific initialization. Indicates whether special initialization of this PCI controller is required.
20	—	Reserved

Table 14-43. PCIPMR0 Field Descriptions (continued)

Bits	Name	Description
19	PME_Clock	PME clock 0 Indicates that no PCI clock is required for the PCI controller to generate PME# 1 Indicates that PCI clock is required for the PCI controller to generate PME#
18–16	Version	PCI Power Management Interface Specification version. 011 Revision 1.2 of the PCI Power Management Interface Specification
15–8	NEXT_CAP_PTR	The next capability pointer points to the next item in the PCI controller's capability list. 0000_0000 The end of the capability list
7–0	CAP_ID	0000_0001 Indicates the power management support capability

14.3.3.28 PCI Power Management Register 1 (PCIPMR1)

The PCI power management register 1 (PCIPMR1), shown in Figure 14-46, contains the bit fields that software uses to manage the PCI controller's power management state, as well as to enable and monitor PMEs (power management events). This register can be accessed by the host in agent mode.

Offset 0x84 Access: Read / write

	31–24	23	22	21–16
R/W	Data	BPCC_En	B2_B3	—
Reset	0 0 0 0 0 0 0 0	0	0	0 0 0 0 0 0

	15	14–13	12–9	8	7–4	3	2	1–0
R/W	PME_Status	Data_Scale	Data_Select	PME_En	—	No_Soft_Reset	—	Power_State
Reset	0	0 0	0 0 0 0	0	0 0 0 0	1	0	0 0

Figure 14-46. PCI Power Management Register 1 (PCIPMR1)

Table 14-44 describes the PCIPMR1 fields.

Table 14-44. PCIPMR1 Field Descriptions

Bits	Name	Description
31–24	Data	Reports the state dependent data requested by the Data_Select field. The value of this field is scaled by the value reported by the Data_Scale field.
23	BPCC_En	Bus power/Clock control enable 0 Disable the bus power/clock control policies defined in section 4.7.1 of the PCI Bus Power Management Interface Specification Revision 1.2 1 Enable the bus power/clock control policies defined in section 4.7.1 of the PCI Bus Power Management Interface Specification Revision 1.2 Note: This bit field is not implemented, only required for all PCI-to-PCI Bridge

Table 14-44. PCIPMR1 Field Descriptions (continued)

Bits	Name	Description
22	B2_B3	The state of this bit determines what will happen as a direct result of programming the function to D3_hot. 0 Indicates that when the bridge function is programmed to D3_hot, its secondary bus will have its power removed (B3). 1 Indicates that when the bridge function is programed to D3_hot, its secondary bus's PCI clock will be stopped (B2). This bit only meaningful if bit 16 (BPCC_En) is set. Note: This bit field is not implemented, only required for all PCI-to-PCI Bridge
21–16	—	Reserved
15	PME_Status	This bit set when the PCI controller would normally assert PME# signal independent of the state of the PME_En bit. Writing a value of one to this bit will clear it and cause the PCI controller to stop asserting a PME# signal. 0 Default 1 If (Wake_Up & PME_En)
14–13	Data_Scale	The scale factor to be used when interpreting the value of the data register
12–9	Data_Select	Selects which data is to be reported through the data register and Data_Scale field
8	PME_En	Enables the function to assert PME# 0 Disables the function to assert PME# 1 Enables the function to assert PME#
7–4	—	Reserved
3	No_Soft_Reset	This bit field indicates whether an internal reset occurs during the transition from D3_hot to D0. 0 The Power_State command performs an internal reset. 1 The Power_State command does not perform an internal reset.
2	—	Reserved
1–0	Power_State	Determines the current power state of the PCI controller and sets the controller into a new power state. The power state definition is as follows: 00 D0 supports all PCI function. 01 D1 disables the inbound memory space, bus mastering and functional interrupt request. 10 D2 disables the inbound memory space, bus mastering and functional interrupt request. 11 D3_hot disables the inbound memory space, bus mastering and functional interrupt request.

14.4 Functional Description

The following sections discuss the operation of the PCI controller.

14.4.1 PCI Bus Arbitration

The PCI bus arbitration approach is access-based. Bus masters must arbitrate for each access performed on the bus. PCI uses a central arbitration scheme where each master has its own unique request ($\overline{\text{REQ}n}$) output and grant ($\overline{\text{GNT}n}$) input signal. A simple request-grant handshake is used to gain access to the bus. Arbitration for the bus occurs during the previous access so that no PCI bus cycles are consumed waiting for arbitration (except when the bus is idle).

The PCI internal arbiter supports five external masters (besides the PCI controller itself) by using the $\overline{REQ}$ signals and generating the $\overline{GNT}$ signals.

During reset, the PCI controller samples the reset configuration bit (and programs the PCI_ARB_DIS bit accordingly) to determine if the arbiter is enabled or disabled. SeeChapter 4, "Reset, Clocking, and Initialization," for more information. The arbiter can also be enabled or disabled by directly programming the PCI_ARB_DIS bit in the arbiter configuration register (see Section 14.3.3.25, "PCI Arbiter Control Register (PCIACR)," for more information). However, it is recommended to use the reset configuration bit to set the arbiter state because the arbiter state controls the direction of $\overline{REQ0}$ and $\overline{GNT0}$.

If the arbiter is disabled, the PCI controller uses $\overline{REQ0}$ to issue requests to an external arbiter, and uses $\overline{GNT0}$ to receive grants from the external arbiter.

14.4.1.1 Bus Parking

When no devices are requesting the bus, the bus is granted, or parked, for a specified device to prevent the AD, PCI_C/$\overline{BE}$ and PCI_PAR signals from floating. The PCI controller can be configured to either park on itself or park on the last master to use the bus (see Section 14.3.3.25, "PCI Arbiter Control Register (PCIACR)," for more information).

14.4.1.2 Arbitration Algorithm

The round-robin arbitration algorithm has two priority levels. Each of the external PCI bus masters, plus the PCI controller, are assigned either a high or a low priority level, as programmed in the arbiter configuration register (see Section 14.3.3.25, "PCI Arbiter Control Register (PCIACR).") Within each priority group (high or low), the bus grant is given to the next requesting device in numerical order, with the PCI controller itself positioned before device 0. $\overline{GNTn}$ is asserted for device *n* as soon as the previously granted device begins a transaction. Conceptually, the lowest priority device at any given time is the current bus master and the highest priority device is the next one to follow the current master. This is considered to be a fair algorithm because a given device cannot prevent other devices from having access to the bus—a given device automatically becomes the lowest priority device as soon as it begins to use the bus. If a master is not requesting the bus, the transaction slot is given to the next requesting device within the priority group.

The grant given to one device may be taken away and whenever a higher priority device asserts its request. If the bus is idle when a new device is to receive a grant, no device receives a grant for one clock; in the next clock, the new winner of the arbitration receives a grant. This operation allows for a turnaround clock when a device is using address stepping or when the bus is parked.

The low priority group collectively receives one bus transaction request slot in the high priority group. Therefore, if there are N high-priority devices, each high-priority device is guaranteed to get at least one of (N+1) bus transactions, and the M low priority devices are guaranteed to each get at least one of (N+1) x M bus transactions, with one of the low-priority devices receiving the grant in one of (N+1) bus transactions. If all devices are programmed to the same priority level or if there is only one device at the low priority, the algorithm provides each device an equal number of bus grants in a round-robin sequence.

An arbitration example with three masters in the high priority group and two in the low priority group is shown in Figure 14-47. Noting that one position in the high priority group is actually a place-holder for the low priority group, it can be seen that each high priority initiator is guaranteed at least 1 out of 3 transaction slots, and each low priority initiator is guaranteed at least 1 out of 6 slots. Assuming all devices are requesting the bus, the grant sequence (with device 1 being the current master) is as follows: 0, 2, the PCI controller, 0, 2, 1, 0, 2, the PCI controller, and so on. If, for example, device 2 is not requesting the bus, the grant sequence becomes 0, the PCI controller, 0, 1, 0, the PCI controller, and so on. If device 2 now requests the bus at a point in the sequence when device 0 is conducting a transaction and the PCI controller is the next grant, then the PCI controller's grant is removed, and the higher-priority device 2 is awarded the next grant.

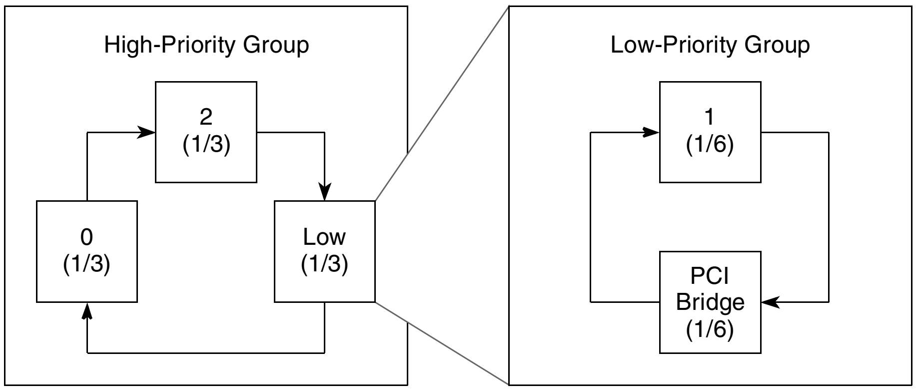


Figure 14-47. PCI Arbitration Example

14.4.1.3 Broken Master Lock-Out

The broken master feature allows the arbiter to lock out any masters that are broken or ill-behaved. This feature is controlled by programming the PCI arbiter control register. When the broken master feature is enabled, a granted device that does not assert $\overline{\text{PCI\_FRAME}}$ within 16 PCI clock cycles after the bus is idle, has its grant removed and subsequent requests are ignored until its $\overline{\text{REQ}}$ is negated for at least one clock cycle. This prevents ill-behaved masters from monopolizing the bus. When the broken master feature is disabled, a device that requests the bus and receives a grant never loses its grant until and unless it begins a transaction or negates its $\overline{\text{REQ}}$ signal. Note that disabling the broken master feature is not recommended.

14.4.1.4 Master Latency Timer

The PCI controller implements the master latency timer register (see Section 14.3.3.10, "Latency Timer Configuration Register") to prevent itself from monopolizing the bus. When the master latency timer expires, the PCI controller checks the state of its $\overline{\text{PCI\_GNT}}$ signals. If the $\overline{\text{PCI\_GNT}}$ signal is not asserted, the PCI controller completes one more data phase and relinquishes the bus. The master latency timer can be disabled if needed (see Section 14.3.3.24, "PCI Function Configuration Register," for more information).

14.4.2 Bus Commands

PCI bus commands indicate the type of transaction occurring on the bus. These commands are encoded on PCI_C/BE[3:0] during the address phase of the transaction. PCI bus commands are described in Table 14-45.

Table 14-45. PCI Command Definitions

PCI_C/ BE[3:0]	Command Type	Supported as:		Definition
		Initiator	Target	
0b0000	Interrupt acknowledge	Yes	No	A read implicitly addressed to the system interrupt controller. The size of the vector to be returned is indicated on the byte enables after the address phase.
0b0001	Special cycle	Yes	No	Provides a simple message broadcast mechanism. See Section 14.4.4.6, “Special Cycle Command,” for more information.
0b0010	I/O read	Yes	No	Accesses agents mapped in I/O address space.
0b0011	I/O write	Yes	No	Accesses agents mapped in I/O address space.
0b010x	—	—	—	Reserved. No response occurs.
0b0110	Memory read	Yes	Yes	Accesses agents mapped in memory address space. A read from prefetchable space, when seen as a target, fetches a cache line of data (32 bytes) from the starting address, even though all 32 bytes may not actually be sent to the initiator.
0b0111	Memory write	Yes	Yes	Accesses agents mapped in memory address space. Note that for inbound writes less than 4-bytes, the PCI controller splits the transaction into single byte writes to the target. Thus, the PCI interface cannot be used to perform single beat writes to 16-bit devices on the local peripheral interfaces.
0b100x	—	—	—	Reserved. No response occurs.
0b1010	Configuration read	Yes	Yes	Accesses the configuration space of each agent. An agent is selected when its IDSEL signal is asserted. See Section 14.4.4.4, “Host Mode Configuration Access,” for more information on configuration accesses. As a target, a configuration read is only accepted if the PCI controller is configured to be in agent mode.
0b1011	Configuration write	Yes	Yes	Accesses the configuration space of each agent. An agent is selected when its IDSEL signal is asserted. See Section 14.4.4.4, “Host Mode Configuration Access,” for more information. As a target, a configuration write is only accepted if the PCI controller is configured to be in agent mode.
0b1100	Memory read multiple	Yes	Yes	Causes a prefetch of the next cache line.
0b1101	Dual address cycle	No	Yes	Transfers an 8-byte address to devices.
0b1110	Memory read line	Yes	Yes	Indicates that the initiator intends to transfer an entire cache line of data.
0b1111	Memory write and invalidate	No	Yes	Indicates that the initiator will transfer an entire cache line of data, and if PCI has any cacheable memory, this line needs to be invalidated.

14.4.3 PCI Protocol Fundamentals

The bus transfer mechanism on the PCI bus is called a burst. A burst is comprised of an address phase and one or more data phases.

All signals are sampled on the rising edge of the PCI clock. Each signal has a setup and hold window with respect to the rising clock edge, in which transitions are not allowed. Outside this aperture, signal values or transitions have no significance.

14.4.3.1 Basic Transfer Control

PCI data transfers are controlled by the following signals:

- $\overline{\text{PCI\_FRAME}}$ is driven by an initiator to indicate the beginning and end of a transaction.
- $\overline{\text{PCI\_IRDY}}$ (initiator ready) is driven by an initiator, allowing it to force wait cycles.
- $\overline{\text{PCI\_TRDY}}$ (target ready) is driven by a target, allowing it to force wait cycles.

The bus is idle when both $\overline{\text{PCI\_FRAME}}$ and $\overline{\text{PCI\_IRDY}}$ are negated. The first clock cycle in which $\overline{\text{PCI\_FRAME}}$ is asserted indicates the beginning of the address phase. The address and the bus command code are transferred in that cycle. The next cycle ends the address phase and begins the data phase.

During the data phase, data is transferred in each cycle that both $\overline{\text{PCI\_IRDY}}$ and $\overline{\text{PCI\_TRDY}}$ are asserted. Once the PCI controller, as an initiator, has asserted $\overline{\text{PCI\_IRDY}}$, it does not change $\overline{\text{PCI\_IRDY}}$ or $\overline{\text{PCI\_FRAME}}$ until the current data phase completes, regardless of the state of $\overline{\text{PCI\_TRDY}}$. Once the PCI controller, as a target, has asserted $\overline{\text{PCI\_TRDY}}$ or $\overline{\text{PCI\_STOP}}$ it does not change $\overline{\text{PCI\_DEVSEL}}$, $\overline{\text{PCI\_TRDY}}$, or $\overline{\text{PCI\_STOP}}$ until the current data phase completes.

When the PCI controller (as a master) intends to complete only one more data transfer, $\overline{\text{PCI\_FRAME}}$ is negated and $\overline{\text{PCI\_IRDY}}$ is asserted (or kept asserted) indicating the initiator is ready. After the target indicates it is ready ($\overline{\text{PCI\_TRDY}}$ asserted) the bus returns to the idle state.

14.4.3.2 Addressing

The PCI specification defines three physical address spaces—memory, I/O, and configuration. The memory and I/O address spaces are standard for all systems. The configuration address space supports the PCI hardware configuration. Each PCI device decodes the address for each PCI transaction with each agent responsible for its own address decode.

The information contained in the two lower address bits (AD1 and AD0) depends on the address space. In the I/O address space, all 32 address/data lines provide the full byte address. AD[1:0] are used for the generation of $\overline{\text{PCI\_DEVSEL}}$ and indicate the least significant valid byte involved in the transfer. Once a target has claimed an I/O access, it first determines if it can complete the entire access as indicated by the byte enable signals. If all the selected bytes are not in the address range, the entire access should not be completed; that is, the target should not transfer any data and should terminate the transaction with a target-abort operation. See Section 14.4.3.6, “Bus Transactions,” for more information.

In the configuration address space, accesses are decoded to a 4-byte address using AD[7:2]. An agent determines if it is the target of the access when a configuration command is decoded, IDSEL is asserted, and AD[1:0] are 0b00; otherwise, the agent ignores the current transaction. The PCI controller determines a configuration access is for a device on the PCI bus by decoding a configuration command. When in agent mode, the PCI controller responds to host-generated PCI configuration cycles when its IDSEL is asserted during a configuration cycle.

For memory accesses, the address is decoded using AD[31:2]; thereafter, the address is incremented internally by 4 bytes until the end of the burst transfer. Another initiator in a memory access should drive 0b00 on AD[1:0] during the address phase to indicate a linear incrementing burst order. The PCI controller checks AD[1:0] during a memory command access and provides the linear incrementing burst order. On reads, if AD[1:0] is 0b10, which represents a cache line wrap, the PCI controller linearly increments the burst order starting at the critical 64-bit address, wraps at the end of the cache line, and disconnects after reading one cache line. If AD[1:0] is 0bx1 (a reserved encoding) and the PCI_C/BE[3:0] signals indicate a memory transaction, it executes a target disconnect after the first data phase is completed. Note that AD[1:0] are included in parity calculations.

14.4.3.3 Device Selection

As a target, the PCI controller drives PCI_DEVSEL one clock following the address phase as indicated in the configuration space status register; see Section 14.3.3.4, “PCI Status Configuration Register,” for more information. The PCI controller as a target qualifies the address/data lines with PCI_FRAME before asserting PCI_DEVSEL. The PCI_DEVSEL signal is asserted at or before the clock edge at which the PCI controller enables its PCI_TRDY, PCI_STOP, or data (for a read). The PCI_DEVSEL signal is not negated until PCI_FRAME is negated, with PCI_IRDY asserted and either PCI_STOP or PCI_TRDY asserted. The exception to this is a target-abort; see Section 14.4.3.8, “Transaction Termination,” for more information.

As an initiator, if the PCI controller does not see the assertion of PCI_DEVSEL within 4 clocks of PCI_FRAME, it terminates the transaction with a master-abort as described in Section 14.4.3.8, “Transaction Termination,” for more information.

14.4.3.4 Byte Enable Signals

The byte enable signals (BE[3:0]) indicate which byte lanes carry valid data. The byte enable signals may enable different bytes for each of the data phases. The byte enable signals are valid on the edge of the clock that starts each data phase and remain valid for the entire data phase.

If the PCI controller, as a target, sees no byte enable signals asserted, it completes the current data phase with no permanent change. This implies that on a read transaction, the PCI controller expects the data not to be changed, and on a write transaction, the data is not stored.

14.4.3.5 Bus Driving and Turnaround

The turnaround-cycle is one clock cycle and is required to avoid contention. This cycle occurs at different times for different signals. PCI_IRDY, PCI_TRDY, and PCI_DEVSEL use the address phase as their turnaround-cycle. PCI_FRAME, PCI_C/BE[3:0], and AD[31:0] use the idle cycle between transactions as their turnaround-cycle. (An idle cycle in PCI is when both PCI_FRAME and PCI_IRDY are negated).

Byte lanes not involved in the current data transfer are driven to a stable condition even though the data is not valid.

14.4.3.6 Bus Transactions

The timing diagrams in this section show the relationship of significant signals involved in bus transactions.

Note the following conventions:

- When a signal is drawn as a solid line, it is actively being driven by the current initiator or target.
- When a signal is drawn as a dashed line, no agent is actively driving it.
- Three-stated signals with slashes between the two rails have indeterminate values.
- The terms ‘edge’ and ‘clock edge’ refer to the rising edge of the clock.
- The terms ‘asserted’ and ‘negated’ refer to the globally visible state of the signal on the clock edge, and not to signal transitions.
- The symbol represents a turnaround-cycle.

14.4.3.7 Read and Write Transactions

Both read and write transactions begin with an address phase followed by a data phase. The address phase occurs when PCI_FRAME is asserted for the first time, and the AD[31:0] signals contain a byte address and the PCI_C/BE[3:0] signals contain a bus command. The data phase consists of the actual data transfer and possible wait cycles; the byte enable signals remain actively driven from the first clock of the data phase through the end of the data transfer.

A read transaction starts when PCI_FRAME is asserted for the first time and the PCI_C/BE[3:0] signals indicate a read command. Figure 14-48 shows an example of a single beat read transaction.

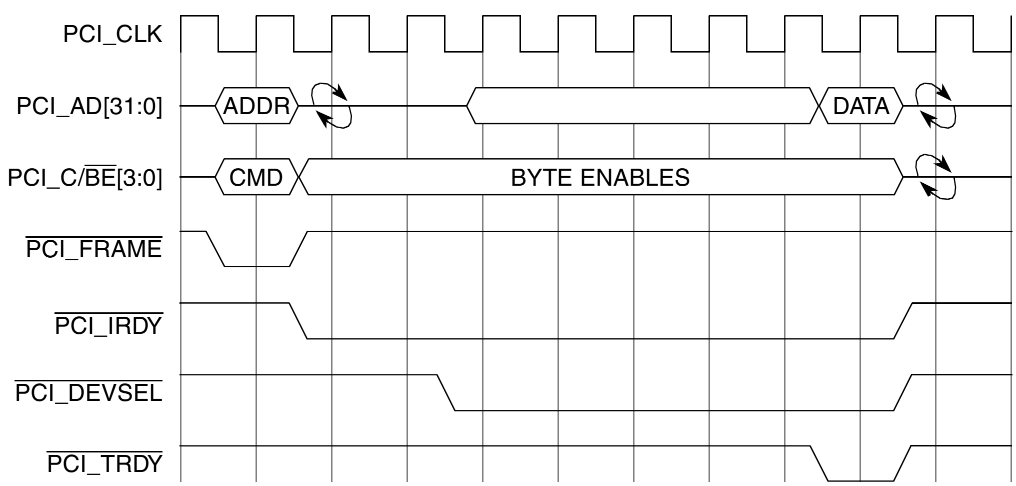


Figure 14-48. Single Beat Read Example

Figure 14-49 shows an example of a burst read transaction.

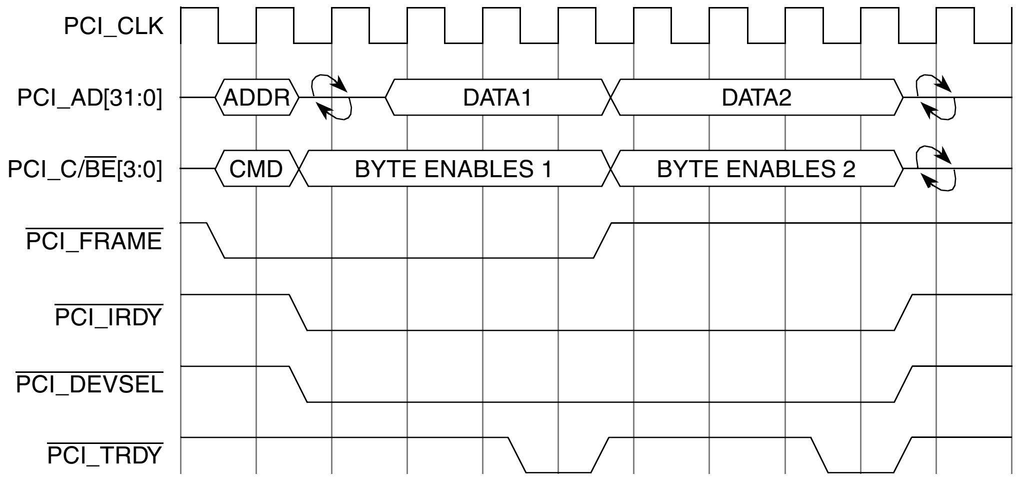


Figure 14-49. Burst Read Example

During the turnaround-cycle following the address phase, the PCI_C/BE[3:0] signals indicate which byte lanes are involved in the data phase. The turnaround-cycle must be enforced by the target with the PCI_TRDY signal if using fast PCI_DEVSEL assertion. The earliest the target can provide valid data is one cycle after the turnaround cycle. The target must drive the AD[31:0] signals when PCI_DEVSEL is asserted except during the turnaround cycle.

The data phase completes when data is transferred, which occurs when both PCI_IRDY and PCI_TRDY are asserted on the same clock edge. When either is negated, a wait cycle is inserted and no data is transferred. To indicate the last data phase PCI_IRDY must be asserted when PCI_FRAME is negated.

A write transaction starts when PCI_FRAME is asserted for the first time and the PCI_C/BE[3:0] signals indicate a write command. Figure 14-50 shows an example of a single-beat write transaction.

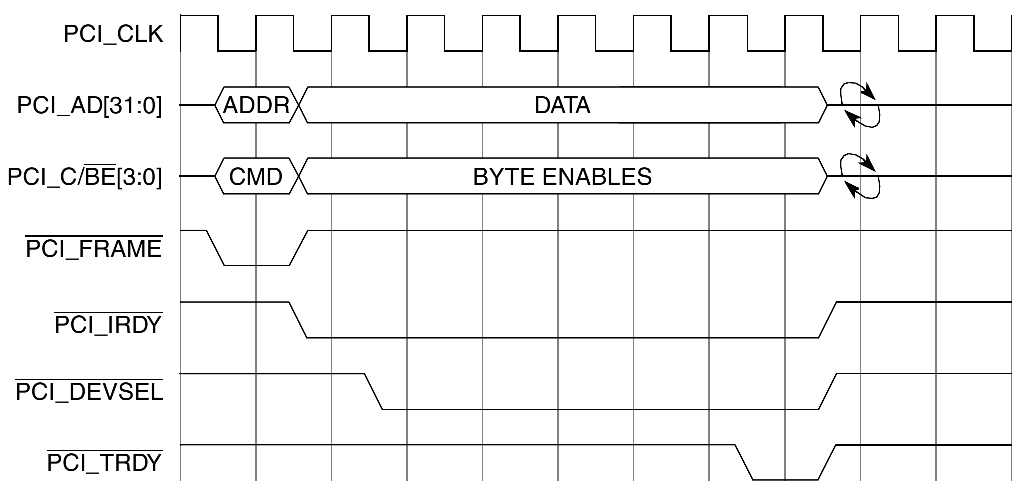


Figure 14-50. Single Beat Write Example

Figure 14-51 shows an example of a burst write transaction.

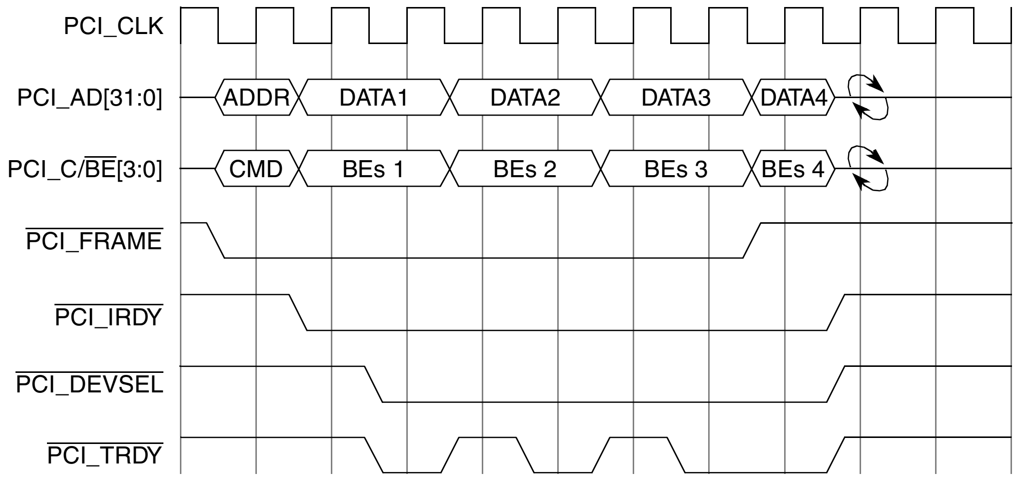


Figure 14-51. Burst Write Example

A write transaction is similar to a read transaction except no turnaround cycle is needed following the address phase because the initiator provides both address and data. Data phases are the same for both read and write transactions.

14.4.3.8 Transaction Termination

The termination of a PCI transaction is orderly and systematic, regardless of the cause of the termination. All transactions end when PCI_FRAME and PCI_IRDY are both negated, indicating the idle cycle.

The PCI controller as an initiator terminates a transaction when PCI_FRAME is negated and PCI_IRDY is asserted. This indicates that the final data phase is in progress. The final data transfer occurs when both PCI_TRDY and PCI_IRDY are asserted. A master-abort is an abnormal case of a master initiated termination. If the PCI controller detects that PCI_DEVSEL has remained negated for more than four clocks after the assertion of PCI_FRAME, it negates PCI_FRAME and then, on the next clock, negates PCI_IRDY. On aborted reads, the PCI controller returns 0xFFFF_FFFF. The data is lost on aborted writes.

When the PCI controller as a target needs to suspend a transaction, it asserts PCI_STOP. Once asserted, PCI_STOP remains asserted until PCI_FRAME is negated. Depending on the circumstances, data may or may not be transferred during the request for termination. If PCI_TRDY and PCI_IRDY are asserted during the assertion of PCI_STOP**,** data is transferred. This type of target-initiated termination is called a disconnect B, shown in Figure 14-52. If PCI_TRDY is asserted when PCI_STOP is asserted but PCI_IRDY is not, PCI_TRDY must remain asserted until PCI_IRDY is asserted and the data is transferred. This is called a disconnect A target-initiated termination, also shown in Figure 14-52. However, if PCI_TRDY is negated when PCI_STOP is asserted, no more data is transferred, and the initiator therefore does not have to wait for a final data transfer (see the retry diagram in Figure 14-50).

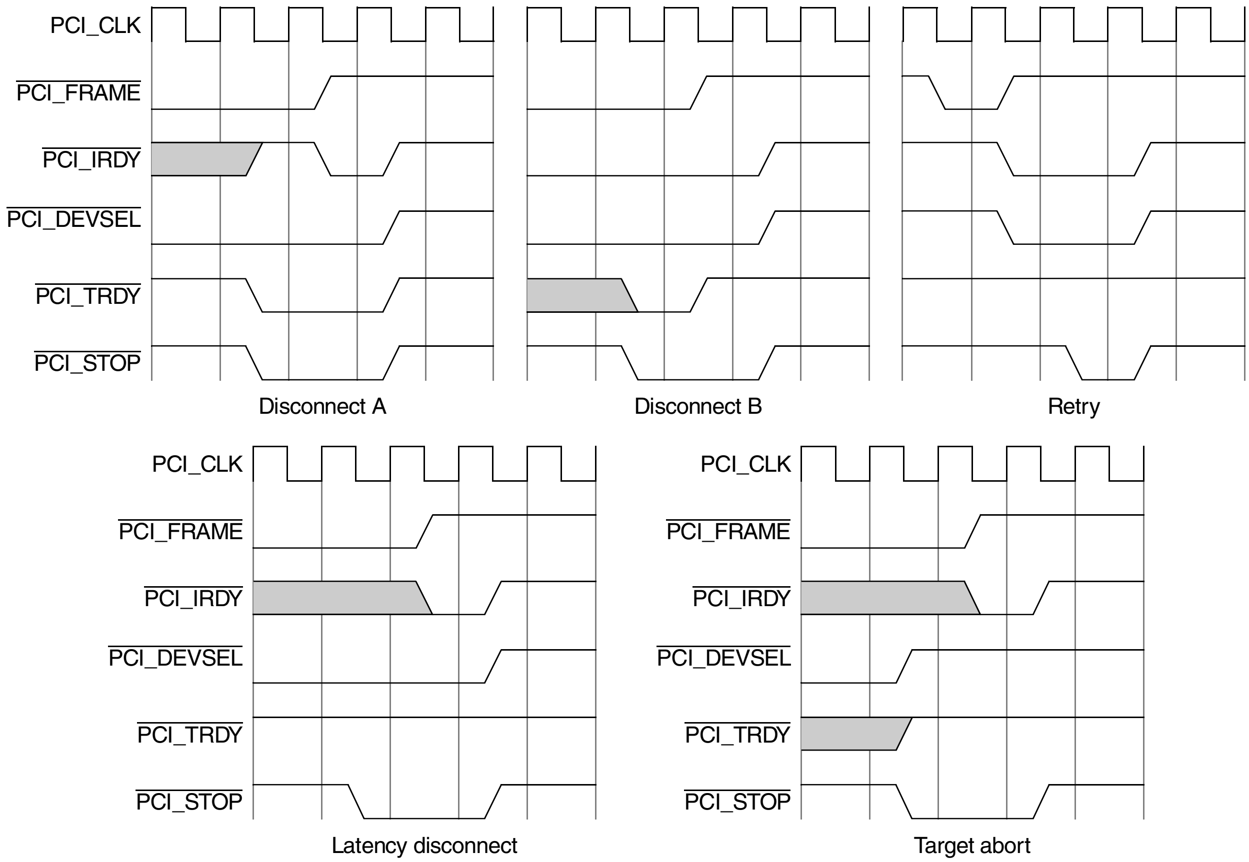


Figure 14-52. Target-Initiated Terminations

Note that when an initiator is terminated by PCI_STOP, it must negate its REQ*n* signal for a minimum of two PCI clocks (of which one clock is needed for the bus to return to the idle state). If the initiator intends to complete the transaction, it should reassert its REQ*n* immediately following the two clocks or potential starvation may occur. If the initiator does not intend to complete the transaction, it can assert REQ*n* whenever it needs to use the PCI bus again.

The PCI controller terminates a transaction in the following cases:

- Eight PCI clock cycles have elapsed between data phases. This is a ‘latency disconnect’ (see Figure 14-50).
- AD[1:0] is 0bx1 (a reserved burst ordering encoding) during the address phase and one data phase has completed.
- The PCI command is a configuration command and one data phase has completed.
- A streaming transaction crosses a 4-Kbyte page boundary.
- A streaming transaction runs out of I/O sequencer buffer entries.
- A cache line wrap transaction has completed a cache line transfer.

Another target-initiated termination is the retry termination. Retry refers to termination requested because the target is currently in a state where it is unable to process the transaction. This can occur because no buffer entries are available in the I/O sequencer, or the sixteen clock latency timer has expired without transfer of the first data. The target latency timer of the PCI controller can be optionally disabled. See Section 14.3.3.24, “PCI Function Configuration Register,” for more information.

When the PCI controller is in host mode it does not respond to any PCI configuration transactions. When the PCI controller is in agent mode and the CFG_LOCK lock bit is set (see Section 14.3.3.24, “PCI Function Configuration Register”) the PCI controller retries all transactions to the PCI configuration space or the internal (on-chip) memory-mapped register space. Note that all retried accesses need to be completed. An example of a retry is shown in Figure 14-50.

Note that because a target can determine whether or not data is transferred (when both $\overline{\text{PCI\_IRDY}}$ and $\overline{\text{PCI\_TRDY}}$ are asserted), if it wants to do only one more data transfer and then stop, it may assert $\overline{\text{PCI\_TRDY}}$ and $\overline{\text{PCI\_STOP}}$ at the same time.

Target-abort refers to the abnormal termination that is used when a fatal error has occurred, or when a target will never be able to respond. Target-abort is indicated when $\overline{\text{PCI\_STOP}}$ is asserted and $\overline{\text{PCI\_DEVSEL}}$ is negated. This indicates that the target requires the transaction to be terminated and does not want the transaction tried again. Note that any transferred data may have been corrupted.

The PCI controller terminates a transaction with target-abort in the case in which it is the intended target of a read transaction from system memory and the data from memory is corrupt. If the PCI controller is the intended target of a transaction and an address parity error occurs, or a data parity error occurs on a write transaction to system memory, it continues the transaction on the PCI bus but aborts internally. The PCI controller does not target-abort in this case.

If the PCI controller is mastering a transaction that terminates with a target-abort, undefined data is returned on a read and write data is lost. An example of a target-abort is shown in Figure 14-50.

An initiator may retry any target disconnect accesses, except target-abort, at a later time starting with the address of the next non-transferred data. Retry is actually a special case of disconnect where no data transfer occurs at all and the initiator must start the entire transaction over again.

14.4.4 Other Bus Operations

The following sections provide information on additional PCI bus operations.

14.4.4.1 Fast Back-to-Back Transactions

In the two types of fast back-to-back transactions, the first type places the burden of avoiding contention on the initiator while the second places the burden on all potential targets. The PCI controller as a target supports both types of fast back-to-back transactions but does not support them as an initiator. The PCI controller as a target has the fast back-to-back enable bit hardwired to one, that is, enabled.

For the first type (governed by the initiator), the initiator may only run a fast back-to-back transaction to the same target. For the second type, when the PCI controller detects a fast-back-to-back operation and did not drive $\overline{\text{PCI\_DEVSEL}}$ in the previous cycle, it delays the assertion of $\overline{\text{PCI\_DEVSEL}}$ and $\overline{\text{PCI\_TRDY}}$ for one cycle to allow the other target to get off the bus.

14.4.4.2 Dual Address Cycles

The PCI controller supports dual address cycle (DAC) commands (64-bit addressing on PCI bus) as a target only. DACs are different from single address cycles (SACs) in that the address phase takes two PCI beats instead of one PCI beat to transfer (64-bit vs. 32-bit addressing). Only PCI memory commands can use DAC cycles; I/O, configuration, interrupt acknowledge, and special cycle command cannot use DAC cycles. The PCI controller supports single-beat and burst DAC transactions.

14.4.4.3 Data Streaming

The PCI controller provides data streaming for PCI transactions to and from prefetchable memory. In other words, when the PCI controller is a target for a PCI initiated transaction, it supplies or accepts multiple cache lines of data without disconnecting. For PCI transactions to non-prefetchable space, the PCI controller disconnects after the first data phase so streaming cannot occur.

For PCI memory reads, streaming is achieved by performing speculative reads from memory in prefetchable space. A block of memory may be marked as prefetchable by setting the PCI configuration registers bit for the inbound address translation (see Section 14.3.2.14, "PCI Inbound Window Attribute Registers (PIWARn)," for more information) in the following cases:

- When reads do not alter the contents of memory (reads have no side effects)
- When reads return all bytes regardless of the byte enable signals
- When writes can be merged without causing errors

For a memory read command or a memory read line command, the PCI controller reads one cache line from memory. If the transaction crosses a cache line boundary, the PCI controller starts the read of a new cache line. For a memory read multiple command, the PCI controller reads two cache lines from memory. When the PCI transaction finishes the read for the first cache line, the PCI controller performs a speculative read of a third cache line. The PCI controller continues this prefetching until the end of the transaction.

For PCI writes to memory, streaming is achieved by buffering the transaction in the space available within the I/O sequencer. This allows PCI memory writes to execute with no wait states.

A disconnect occurs if the PCI controller runs out of buffer space on writes, or the PCI controller cannot supply consecutive data beats for reads within eight PCI bus clocks of each other. A disconnect also occurs if the transaction crosses a 4-Kbyte page boundary.

14.4.4.4 Host Mode Configuration Access

The PCI controller provides two types of configuration accesses to support hierarchical bridges. To access configuration space, a value is written to the CONFIG_ADDR register specifying which PCI bus, which device, and which configuration register to be accessed.

When the PCI controller sees an access that falls inside the 4 bytes beginning at the CONFIG_DATA address, it checks the enable bit, the device number and the bus number in the CONFIG_ADDR register. If the enable bit is set and the device number is not equal to all ones, a configuration cycle translation is performed. When the device number field is equal to all ones, it has a special meaning (see Section 14.4.4.6, "Special Cycle Command," for more information).

There are two types of translations supported:

- Type 0 translations—For when the device is on the PCI bus connected to the PCI controller.
- Type 1 translations—For when the device is on another bus somewhere behind the PCI controller.

For type 0 translations, the PCI controller decodes the device number field to assert the appropriate IDSEL line and perform a configuration cycle on the PCI bus with AD[1:0] as 0b00. All 21 IDSEL bits are decoded, starting with bit AD11. That is, if the device number field contains 0b01011, AD11 on the PCI bus is set. The IDSEL lines are bit-wise associated with increasing values for the device number such that AD12 corresponds to 0b01100, and so on up to bit 30 as shown in Table 13-41. AD31 is selected with 0b01010. A device number of 0b11111 indicates a special cycle. Device number 0b00000 is used for configuring the PCI controller itself. Bits 10 through 8 are copied to the PCI bus as an encoded value for components which contain multiple functions. Bits 7 through 2 are also copied onto the PCI bus. The PCI controller implements address stepping on configuration cycles so that the target's PCI_IDSEL, which is connected directly to one of the AD lines, reaches a stable value. This means that a valid address and command are driven on the AD and PCI_C/$\overline{\text{BE}}$ lines one cycle before the assertion of $\overline{\text{PCI\_FRAME}}$.

For type 1 translations, the PCI controller copies the contents of the CONFIG_ADDR register directly onto the PCI address/data lines during the address phase of a configuration cycle, with the exception that AD[1-0] contains 0b01 (not 0b00 as in Type 0 translations).

When the PCI controller is configured as a host device, a local master sometimes needs to perform configuration reads from unpopulated PCI slots (as part of the system configuration). To avoid getting a machine check interrupt, the following steps should be taken:

1. Mask the NORSP bit in the error mask register. See Section 14.3.2.9, "PCI Error Control Register (PCI_ECR)."
2. Perform the PCI configuration reads.
3. Clear the NORSP bit in the error status register.
4. Unmask (write 1) the NORSP bit in the error mask register. See Section 14.3.2.3, "PCI Error Enable Register (PCI_EER)."

14.4.4.5 Agent Mode Configuration Access

When the PCI controller is configured as an agent device, it responds to remote host generated PCI configuration accesses to the PCI interface. This is indicated by decoding the configuration command along with the PCI controller's IDSEL being asserted. A remote host can access the 256-byte PCI configuration area and the memory-mapped configuration registers within the PCI controller.

14.4.4.6 Special Cycle Command

A special cycle command contains no explicit destination address but is broadcast to all PCI agents. Each receiving agent must determine whether the message is applicable to itself. No assertion of $\overline{\text{PCI\_DEVSEL}}$ in response to a special cycle command is necessary.

A special cycle command is like any other bus command in that it has an address phase and a data phase. The address phase starts like all other commands with the assertion of $\overline{\text{PCI\_FRAME}}$ and completes when

PCI_FRAME and PCI_IRDY are negated. Special cycles terminate with a master-abort. (In the special cycle case, the received-master-abort bit in the configuration status register is not set.)

The address phase contains no valid information other than the command field. Even though there is no explicit address, the address/data lines are driven to a stable state and parity is generated. During the data phase, the address/data lines contain the message type and an optional data field. The message is encoded on the sixteen least-significant bits (AD[15:0]). The data field is encoded on AD[31:16]. When running a special cycle, the message and data are valid on the first clock PCI_IRDY is asserted.

When the PCI_CONFIG_ADDRESS register is written with a value so that the bus number matches the bridge bus, the device number is all ones, the function number is all ones, and the register number is zero. The next time the PCI_CONFIG_DATA register is accessed, the PCI controller executes either a special cycle or an interrupt acknowledge command. When the PCI_CONFIG_DATA register is written, the PCI controller generates a special cycle encoding on the command/byte enable lines during the address phase and drives the data from the PCI_CONFIG_DATA register onto the address/data lines during the first data phase.

If the bus number field of the PCI_CONFIG_ADDRESS does not match one of the PCI controller bus numbers, the PCI controller passes the write to PCI_CONFIG_DATA through to the PCI bus as a type 1 configuration cycle as it does any other time the bus number field does not match.

Table 14-46. Special Cycle Commands

Address (AD[15–0])	Message Type	Description
0x0000	SHUTDOWN (SLEEP)	Indicates the processor is entering its most power saving mode
0x0001	HALT (DOZE)	Indicates the processor is entering a power save mode where address decoding is still available
0x0002–0xFFFF	—	Reserved for future commands

14.4.4.7 Interrupt Acknowledge

When the PCI_CONFIG_ADDRESS register is written with a value such that the bus number is 0x00, the device number is all ones, the function number is all ones, and the register number is zero, the next time the PCI_CONFIG_DATA register is accessed the PCI controller does either a special cycle command or an interrupt acknowledge command. When the PCI_CONFIG_DATA register is read, the PCI controller generates an interrupt acknowledge command encoding on the command/byte enable lines during the address phase. During the address phase, AD[31:0] do not contain a valid address but are driven with stable data and valid parity (PCI_PAR). During the data phase, the byte enable signals determine which bytes are involved in the transaction. The interrupt vector must be returned when PCI_TRDY is asserted.

An interrupt acknowledge transaction can also be issued on the PCI bus by reading from the PCI_INT_ACK register.

14.4.5 Error Functions

This section describes PCI bus errors.

14.4.5.1 Parity

During valid 32-bit address and data transfers, parity covers all 32 address/data lines and the 4 command/byte enable lines regardless of whether or not all lines carry meaningful information. Byte lanes not actually transferring data are driven with stable (albeit meaningless) data and are included in the parity calculation. During configuration, special cycle or interrupt acknowledge commands, some address lines are not defined but are still driven to stable values and included in the parity calculation.

Even parity is calculated for all PCI operations: the value of PCI_PAR is generated such that the number of ones on PCI_AD[31:0], PCI_C/BE[3:0] and PCI_PAR equals an even number. The PCI_PAR signal is driven when the address/data lines are driven and follow the corresponding address or data by one clock.

The PCI controller checks the parity after all valid address phases (the assertion of PCI_FRAME) and for valid data transfers (PCI_IRDY and PCI_TRDY asserted) involving the PCI controller. When an address or data parity error is detected, the detected-parity-error bit in the configuration space status register is set (see Section 14.3.3.4, “PCI Status Configuration Register.”)

14.4.5.2 Error Reporting

Except for setting the detected-parity-error bit, all parity error reporting and response is controlled by the parity-error-response bit (see Section 14.3.3.3, “PCI Command Configuration Register,” for more information). If the parity-error-response bit is cleared, the PCI controller completes all transactions regardless of parity errors (address or data). If the bit is set, the PCI controller asserts PCI_PERR two clocks after the actual data transfer in which a data parity error is detected, and keeps PCI_PERR asserted for one clock. When acting as an initiator during a read transaction or as a target involved in a write to system memory the PCI controller asserts PCI_PERR.

Figure 14-53 shows the possible assertion points for PCI_PERR if the PCI controller detects a data parity error.

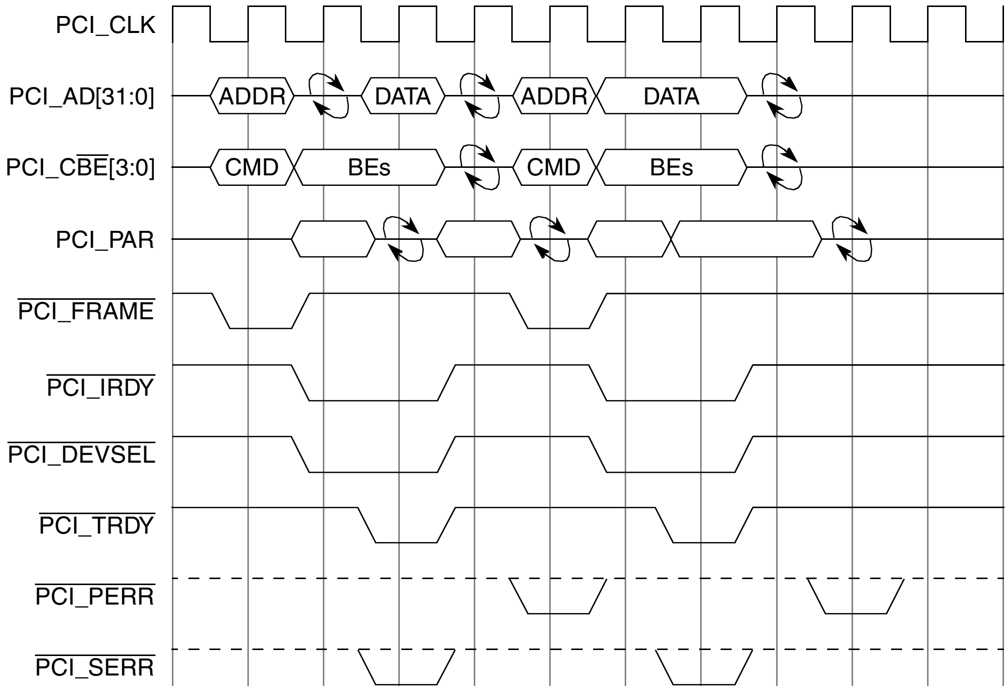


Figure 14-53. PCI Parity Operation

As an initiator, the PCI controller attempts to complete the transaction on the PCI bus if a data parity error is detected and sets the data-parity-reported bit in the configuration space status register. If a data parity error occurs on a read transaction, the PCI controller aborts the transaction internally. As a target, the PCI controller completes the transaction on the PCI bus even if a data parity error occurs. If parity error occurs during a write to system memory, the transaction completes on the PCI bus, but is aborted internally, insuring that potentially corrupt data does not go to memory.

When the PCI controller asserts PCI_SERR, it sets the signaled-system-error bit in the configuration space status register. Additionally, if the error is an address parity error, the parity-error-detected bit is set; reporting an address parity error on PCI_SERR is conditioned on the parity-error-response bit being enabled in the command register. PCI_SERR is asserted when the PCI controller detects an address parity error while acting as a target. The system error is passed to the PCI controller's interrupt processing logic to assert MCP. Figure 14-53 shows where the PCI controller could detect an address parity error and assert PCI_SERR or where the PCI controller, acting as an initiator, checks for the assertion of PCI_SERR signaled by the target detecting an address parity error.

As a target that asserts PCI_SERR on an address parity, the PCI controller completes the transaction on the PCI bus, aborting internally if the transaction is a write to system memory. If PCI_PERR is asserted during a PCI controller write to PCI, the PCI controller attempts to continue the transfer, allowing the target to abort/disconnect if desired. If the PCI controller detects a parity error on a read from PCI, the PCI controller aborts the transaction internally and continues the transfer on the PCI bus, allowing the target to abort/disconnect if desired.

In all cases of parity errors on the PCI bus, regardless of the parity-error-response bit, information about the transaction is logged in the PCI error control capture register, the PCI error address capture register and the PCI error data capture register; MCP is also asserted to the core as an option.

14.4.6 PCI Inbound Address Translation

For inbound transactions (transactions generated by an external master on the PCI bus where the PCI controller responds as a slave device), the PCI controller only responds to PCI addresses within the windows mapped by the PCI inbound base address registers (PIBARs). If there is an address hit in one of the PIBARs, the PCI address is translated from PCI space to local memory space through the associated PCI inbound translation address registers (PITARs). This allows an external master to access local memory. Each PIBAR register is associated with a PITAR and PIWAR which are located in the PCI controller's PCI CSR space. Figure 14-54 shows an example translation window for inbound memory accesses.

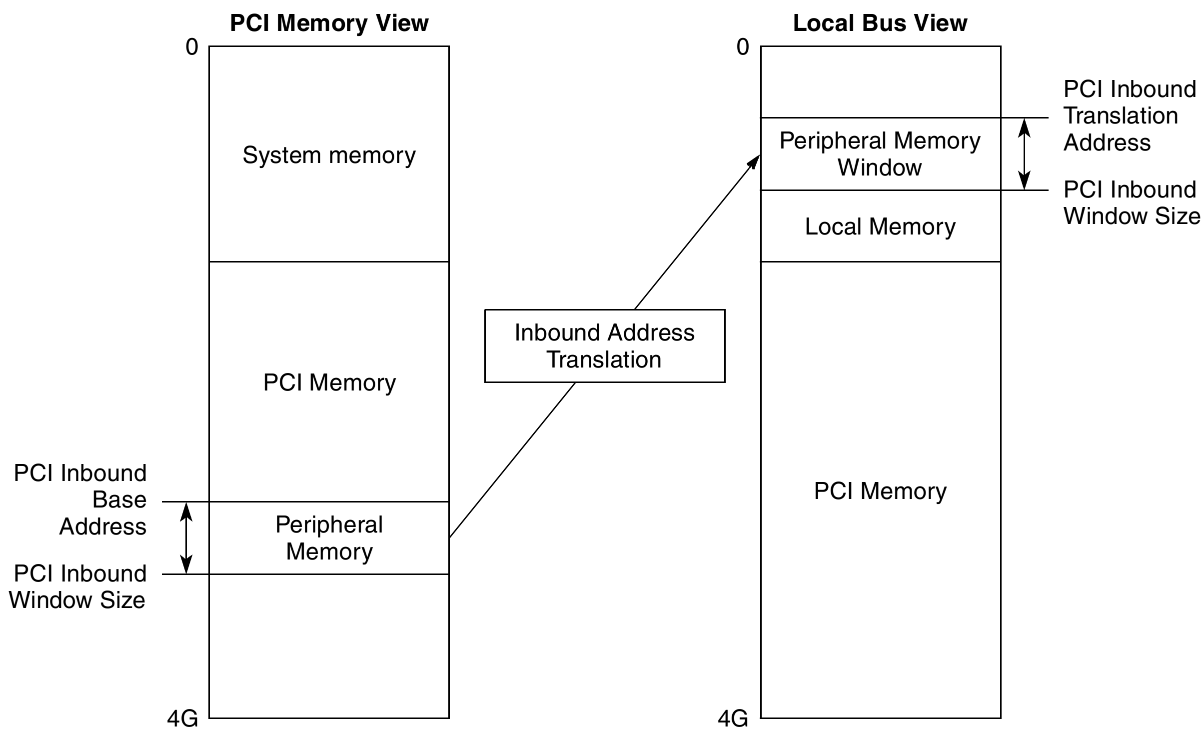


Figure 14-54. Inbound PCI Memory Address Translation

There are three full sets of inbound translation registers, in addition to the PIMMR base address register, allowing four simultaneous translation windows, one to a fixed destination and three programmable. Only two of the programmable windows can be mapped anywhere in the 64-bit PCI address space. Window 0 can only be mapped within the lowest 4-Gbyte space. Software can move the programmable translation base addresses during run-time to access different portions of local memory, but the PCI inbound translation windows may not overlap.

The translation windows are disabled after reset, that is, after reset, the PCI controller does not acknowledge externally mastered transactions on the PCI bus by asserting PCI_DEVSEL until the inbound translation windows are enabled.

14.4.7 CompactPCI Hot Swap Specification Support

CompactPCI is an open specification supported by the PCI Industrial Computer Manufacturers Group (PICMG) and is intended for embedded applications using PCI. CompactPCI Hot Swap is an extension of the CompactPCI specification and allows the insertion and extraction (or “hot swapping”) of boards without adversely affecting system operation. The hot swap specification defines the following levels of support:

- Hot swap capable
- Hot swap friendly
- Hot swap ready

The PCI controller is hot swap friendly, meaning that it supports the hardware and software connection processes as defined in the hot swap specification. This level of support allows the board and system designers to build full Hot Swap and high availability systems based on the PCI controller as a PCI target device. For details on the hot swap process, refer to the *Hot Swap Specification PICMG 2.1*, R1.0, August 3, 1998.

14.5 Initialization/Application Information

The following sections describe initialization sequences for host and agent modes.

14.5.1 Initialization Sequence for Host Mode

The sequence below must be followed in host mode:

1. Enable PCI output clocks and select desired frequency ratios. See Section 4.4.1, “Clocking in PCI Host Mode.”
2. Wait for at least 1 ms to enable stable clocks into agent devices.
3. Deactivate PCI_RESET_OUT signal for PCI. See Table 13-3 for more information on PCI_RESET_OUT signal.
4. Wait for at least 1 ms to enable devices to complete the powerup sequence.
5. Configure PCI internal registers and PCI agents to desired modes of operation.

14.5.2 Initialization Sequence for Agent Mode

The sequence belown must be followed in agent mode to optionally initialize subsystem vendor ID/device ID:

1. Initialize PCI inbound window size in PIWAR[1–3] desired window size.
2. Unlock configuration lock in PCI function configuration register.

Chapter 15
PCI Express Interface Controller

The MPC8378E and MPC8377E PCI Express interface complies with the *PCI Express™ Base Specification*, Revision 1.0a (available from http://www.pcisig.org). It is beyond the scope of this manual to document the intricacies of the PCI Express protocol. This chapter describes the PCI Express controller of this device and provides a basic description of the PCI Express protocol. The specific emphasis is directed at how the device implements the PCI Express specification. Designers of systems incorporating PCI Express devices should refer to the specification for a thorough description of PCI Express.

NOTE

Much of the available PCI Express literature refers to a 16-bit quantity as a WORD and a 32-bit quantity as a DWORD. This is inconsistent with the terminology in the rest of this manual where the terms 'word' and 'double word' refer to a 32-bit and 64-bit quantity, respectively. Where necessary to avoid confusion, the precise number of bits or bytes is specified.

NOTE

The PCI Express engine does not support misaligned byte transfers. It must be DWORD aligned to the CSB bus.

15.1 Introduction

The PCI Express controller is a mechanism for communicating with PCI Express devices. The controller contains three major parts:

- PCI Express core—Handles the transaction, data link and MAC layers and contains the configuration header and control registers.
- CSB bridge—Controls the transfer of the transactions between the PCI Express transaction layer and the CSB, and include Write and Read DMA engines, a message manager and a set of configuration registers.
- SerDes—Controls the transfer between the PCI Express MAC layer and the physical link, and includes another set of configuration registers (described in Chapter 19, "SerDes PHY.").

Figure 15-1 is a high-level block diagram of the PCI Express controller.

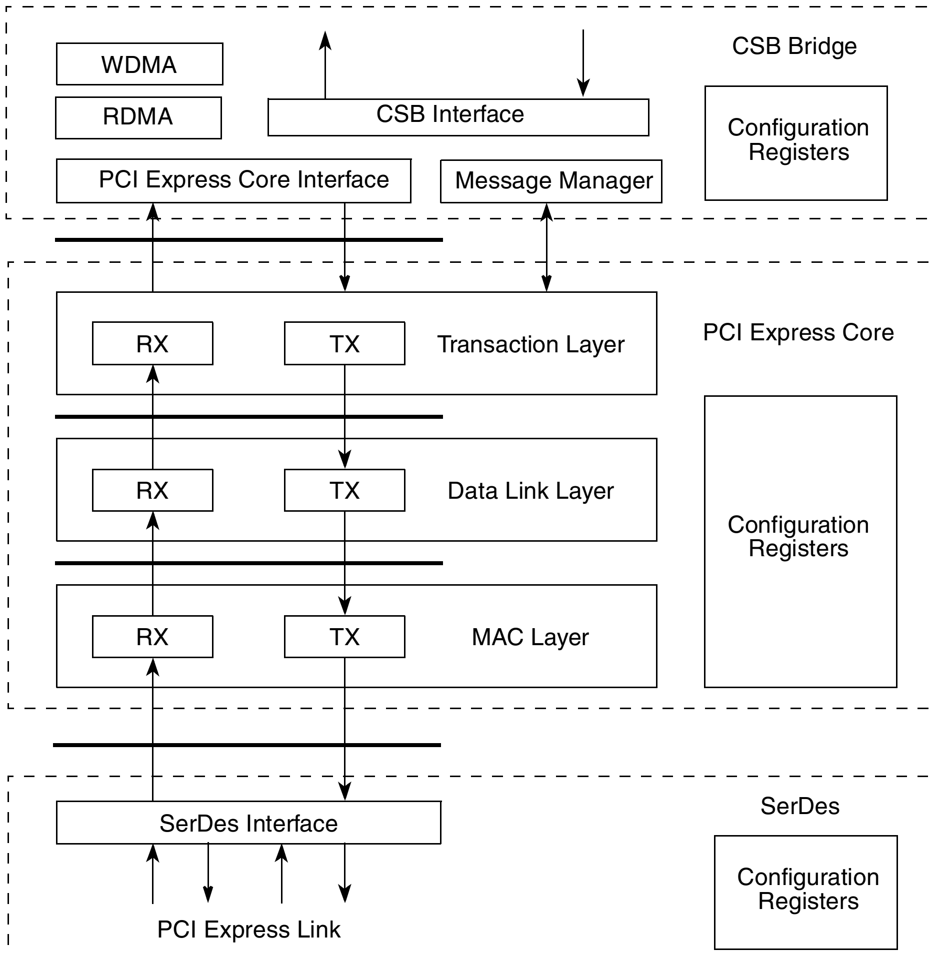


Figure 15-1. PCI Express Controller Block Diagram

The PCI Express controller connects the coherent system bus (CSB) to the PCI Express bus, which is a 2.5-GHz serial interface that supports up to a x2 lane. As both a master (initiator) and a target device, the PCI Express interface is capable of high bandwidth data transfer and is designed to support the next generation of I/O devices. When it comes out of reset, the PCI Express interface performs link width negotiation and exchanges flow control credits with its link partner. When link auto-negotiation finishes, the controller is ready for operation.

Internally, the design contains queues to keep track of inbound and outbound transactions. Control logic handles buffer management, bus protocol, transaction spawning, and tag generation. In addition, memory blocks store inbound and outbound data.

The device can be configured to operate in either root complex (RC) or endpoint (EP) mode. An RC device connects the CPU/memory subsystem to the I/O devices, while an EP device typically denotes a peripheral or I/O device. In RC mode, a type 1 configuration header is used. In EP mode, a type 0 configuration header is used.

As an initiator, the device supports memory read and write operations with a maximum payload of 128 bytes and I/O transactions. In addition, outbound configuration are supported if the device is in RC mode. As a target interface, the device accepts read and write operations to local memory space. Furthermore, as an EP device, the device accepts configuration transactions to the internal PCI Express configuration registers. Inbound I/O transactions are not supported.

15.1.1 MPC8378E/MPC8377E as a PCI Express Initiator

Outbound CSB transactions to PCI Express are first mapped to a translation window to determine which PCI Express transactions are to be issued. A transaction from the CSB can become a memory, I/O, or configuration transaction on the PCI Express bus depending on the window attributes. A transaction can be broken up into smaller transactions depending on the original request size, transaction type, PCI Express Device Control register’s Max_Payload_Size field (for writes) and PCI Express Device Control Register’s Max_Read_Request_Size field (for reads). The device performs PCI Express ordering rule checks to determine the next transaction to be sent on the PCI Express bus. In general, transactions are serviced in the order they are received from the CSB. The device allows reordering of higher-priority transactions to bypass lower-priority transactions only when there is a stall condition. For posted write transactions, after all data is received on the CSB, the data is forwarded to the PCI Express bus and the transaction is considered to be complete. For non-posted write transactions, the device waits for a completion to return from the link partner before considering the transaction to be complete. For non-posted read transactions, the device waits for all completion packets to return from the link partner and then forwards all data back to the CSB before terminating the transaction.

There are two methods of generating PCI Express outbound transactions:

- One of the CSB masters, such as the e300 host, directly initiates a transaction. This is referred to as “PIO.”
- The write or read DMA engines, which are part of the PCI Express controller CSB bridge, is used.

The DMA method is more efficient for transferring large chunks of data. The outbound widows are used and shared by both methods.

15.1.2 MPC8378E/MPC8377E as a PCI Express Target

Inbound PCI Express transactions to the CSB are first mapped to the CSB address space through a translation window. A transaction can be broken up into smaller transactions when it is sent to the CSB depending on the original size, byte enables and starting/ending addresses. The device performs PCI Express ordering rule checks to determine the next transaction to be sent to the CSB. In general, transactions are serviced in the order they are received from the PCI Express bus. The device allows reordering of higher-priority transactions to bypass lower-priority transactions only when there is a stall condition. For posted write transactions, after all data is received on the PCI Express bus, the data is forwarded to the CSB and the transaction is considered to be complete. For non-posted read transactions, the device waits for enough completion packets (dependent on the packet length) to return and then forwards data back to the PCI Express bus. This process continues until there are no more completion packets left to be sent.

15.1.3 Features

The following is a list of PCI Express controller features:

- Designed to comply with the *PCI Express Base Specification, Version 1.0a*
- Root complex (RC) and endpoint (EP) configurations
- 32- and 64-bit address support
- Two PCI Express links of x1 lane each or one x2 link
- Access to all PCI Express memory
- Access to I/O address spaces as requestor only in RC mode
- Posting of processor-to-PCI Express and PCI Express-to-memory writes
- Strong and relaxed transaction ordering rules
- PCI Express configuration registers
- Baseline and advanced error reporting
- One virtual channel (VC0) per controller
- 128-byte maximum payload size (Max_Payload_Size) for memory read and write operations
- Four inbound general-purpose translation windows per controller
- Four outbound translation windows per controller
- Up to four outstanding PCI Express transactions from each controller (posted or non-posted)
- Credit-based flow control management
- PCI Express messages and interrupts
- Maximum 32-byte payload transactions from the CSB
- Interrupt generation from messages or upon detection of errors
- Read and Write DMA engines per controller

15.1.4 Modes of Operation

This section describes how some parameters that affect the PCI Express controller operating modes are determined by dedicated memory mapped registers.

15.1.4.1 Root Complex/Endpoint Modes

The PCI Express controller can function as either a root complex (RC) or an endpoint (EP) device. The PCI Express control registers 1 and 2 determine the RC/EP mode; see Section 5.3.2.11, "PCI Express Control Registers (PECR1 and PECR2)."

15.1.4.2 Link Width

The link width of each PCI Express controller is determined by SRDSCR4. See Section 19.3.5, "SerDesn Control Register 4 (SRDSnCR4)." The device can be configured to either two separate PCI express controllers with an x1 lane each, or one PCI Express controller with x2 lanes link width.

15.1.4.3 Reference Clock

The reference clock for the PCI Express PHY is determined by SRDSCR4. See Section 19.3.5, "SerDesn Control Register 4 (SRDSnCR4)."

15.2 External Signal Descriptions

PCI Express defines the connection between two devices as a link, which can be composed of a single lane or multiple lanes. Each lane consists of a differential pair for transmitting (TX*n* and $\overline{\text{TX}}n$) and a differential pair for receiving (RX*n* and $\overline{\text{RX}}n$) with an embedded data clock.

Table 15-1 describes the external PCI Express interface signals.

Table 15-1. PCI Express Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description
L2_SD_RX[0] $\overline{\text{L2\_SD\_RX[0]}}$	I	Receive data lane 0. Receive data differential signal pair carry PCI Express packet information. Used by the PCI Express core 1.
L2_SD_RX[1] $\overline{\text{L2\_SD\_RX[1]}}$	I	Receive data lane 1. Receive data differential signal pair carry PCI Express packet information. Used by the PCI Express core 1 in two lanes mode or by the PCI Express core 2 in one lane mode.
L2_SD_TX[0] $\overline{\text{L2\_SD\_TX[0]}}$	O	Transmit data lane 0. The transmit data differential signal pair carry PCI Express packet information. Used by the PCI Express core 1.
L2_SD_TX[1] $\overline{\text{L2\_SD\_TX[1]}}$	O	Transmit data lane 1. The transmit data differential signal pair carry PCI Express packet information. Used by the PCI Express core 1 in two lanes mode or by the PCI Express core 2 in one lane mode.

15.3 Memory Map/Register Definitions

The PCI Express interface supports the following register types:

- Memory-mapped registers—Control PCI Express address translation, PCI error management, and PCI Express configuration register access on the device. These registers are described in Section 15.3.1, "PCI Express Memory Map."
- PCI Express configuration registers within the PCI Express configuration header—Specified by the PCI Express specification for every PCI Express device. They are described in Section 15.4.1, "Common PCI-Compatible Configuration Header Registers."

From the PCI Express side, the configuration header registers can be accessed through configuration access, and the memory-mapped registers can be accessed through memory transactions after the inbound translation window is programmed. From the CSB side, all these registers are memory-mapped.

15.3.1 PCI Express Memory Map

The PCI Express memory-mapped registers, listed in Table 15-3, are accessed by reading and writing to an address composed of the base address (specified in the IMMRBAR on the CSB side, or the ATMU windows on the PCI Express side) plus the offset of the specific register to be accessed. In this table and in the register figures and fields description, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.

- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

Also note that although the table explicitly lists only the registers for the PCI Express Controller 1, the register map for PCI Express Controller 2 is the same except the for the block base address. Memory-mapped registers for PCI Express Controller 1 begin at block base address 0x0_9000 and the memory-mapped registers for PCI Express Controller 2 begin at block base address 0x0_A000.

Table 15-2 below lists the address ranges for each type of register. Undefined address spaces are reserved.

Table 15-2. PCI Express Controller Register Groups

Register	Offset Range	Section/Page
PCI Express Core Registers		
Common PCI-Compatible Configuration Header Registers	0x000–0x3FF	15.4.1/15-15
PCI Express Core Control and Status Registers (CSRs)	0x400–0x4CF	15.4.6/15-62
PCI Express BAR Configuration Registers (EP Mode)	0x4D8–0x4FF	15.4.7/15-73
PCI Express Extended Status and Control Registers	0x590–0x7FF	15.4.8/15-75
PCI Express CSB Bridge Registers		
Global Registers	0x800–0x83F	15.5.2/15-78
PCI Express Outbound PIO Registers	0x840–0x8DF	15.5.3/15-81
PCI Express Inbound PIO Registers	0x8E0–0x87F	15.5.4/15-83
PCI Express DMA Registers	0x990–0xADF	15.5.5/15-84
Mailbox Registers	0xB20–0xB9F	15.5.6/15-88
PCI Express Host Interrupt Registers	0xBA0–0XBDF	15.5.7/15-90
CSB System Interrupt Registers	0xBE0–0xC1F	15.5.8/15-95
PCI Express Outbound Address Mapping Registers	0xCA0–0xDDF	15.5.10/15-104
PCI Express EP Inbound Address Translation Registers	0xDE0–0xE5F	15.5.11/15-107
PCI Express RC Inbound Address Mapping Registers	0xE60–0xFFF	15.5.12/15-108

Table 15-3. PCI Express Memory Map

Offset	Register	Access	Reset	Section/Page
PCI Express 1—Block Base Address 0x0_9000 **PCI Express 2—Block Base Address 0x0_A000**				
PCI Express Controller 1 Registers				
PCI Express1 Core Configuration Header Registers				
0x000	PCI Express Vendor ID Register	R	0x1957	15.4.1.1/15-15
0x002	PCI Express Device ID Register	R	Device-specific	15.4.1.2/15-16
0x004	PCI Express Command Register	Mixed	0x0000	15.4.1.3/15-16
0x006	PCI Express Status Register	Mixed	0x0010	15.4.1.4/15-17
0x008	PCI Express Revision ID Register	R	Revision-specific	15.4.1.5/15-18
0x009	PCI Express Class Code Register	Mixed	0x0B20	15.4.1.7/15-20
0x00C	PCI Express Cache Line Size Register	R/W	0x00	15.4.1.7/15-20
0x00D	PCI Express Latency Timer Register	R	0x00	15.4.1.8/15-20
0x00E	PCI Express Header Type Register	R	0x00 (EP mode) 0x01 (RC mode)	15.4.1.10/15-22
0x00F	PCI Express BIST Register	R	0x00	15.4.1.10/15-22
0x010–0x014	Base Address Registers 0 and 1 (BAR0/BAR1) (EP mode only)	Mixed	0x0008	15.4.2.1.1/15-23
0x018–0x020	Base Address Registers 2 and 4 (BAR2/BAR4) (EP mode only)	Mixed	0x0000_000C	15.4.2.1.2/15-23
0x01C–0x024	Base Address Registers 3 and 5 (BAR3/BAR5) (EP mode only)	R/W	0x0000_0000	15.4.2.1.3/15-24
0x02C	PCI Express Subsystem Vendor ID Register (EP mode only)	Special	0x0000	15.4.2.2/15-24
0x02E	PCI Express Subsystem ID Register (EP mode only)	Special	0x0000	15.4.2.3/15-25
0x034	PCI Express Capabilities Pointer Register	R	0x0044	15.4.2.4/15-25
0x03C	PCI Express Interrupt Line Register (EP mode only)	R/W	0x0000	15.4.2.5/15-26
0x03D	PCI Express Interrupt Pin Register	R	0x0001	15.4.2.6/15-26
0x03E	PCI Express Minimum Grant Register (EP mode only)	R	0x0000	15.4.2.7/15-27
0x03F	PCI Express Maximum Latency Register (EP mode only)	R	0x0000	15.4.2.8/15-27
0x018	PCI Express Primary Bus Number Register (RC mode only)	R/W	0x0000	15.4.3.1/15-28
0x019	PCI Express Secondary Bus Number Register (RC mode only)	R/W	0x0000	15.4.3.2/15-29
0x01A	PCI Express Subordinate Bus Number Register (RC mode only)	R/W	0x0000	15.4.3.3/15-29
0x01B	Secondary Latency Timer Register 2 (RC mode only)		0x0000	
0x01C	PCI Express I/O Base Register (RC mode only)	R	0x0000	15.4.3.5/15-30
0x01D	PCI Express I/O Limit Register (RC mode only)	R	0x0000	15.4.3.6/15-30
0x01E	PCI Express Secondary Status Register (RC mode only)	Mixed	0x0000	15.4.3.7/15-31
0x020	PCI Express Memory Base Register (RC mode only)	R/W	0x0000	15.4.3.8/15-31

Table 15-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
PCI Express 1—Block Base Address 0x0_9000 **PCI Express 2—Block Base Address 0x0_A000**				
0x022	PCI Express Memory Limit Register (RC mode only)	R/W	0x0000	15.4.3.9/15-32
0x024	PCI Express Prefetchable Memory Base Register (RC mode only)	R/W	0x0000	15.4.3.10/15-32
0x026	PCI Express Prefetchable Memory Limit Register (RC mode only)	R/W	0x0000	15.4.3.11/15-33
0x028	PCI Express Prefetchable Base Upper 32-Bit Register (RC mode only)	R/W	0x0000	15.4.3.12/15-33
0x02C	PCI Express Prefetchable Limit Upper 32-Bit Register (RC mode only)	R/W	0x0000	15.4.3.13/15-34
0x030	PCI Express I/O Base Upper 16-Bit Register (RC mode only)	R	0x0000	15.4.3.14/15-34
0x032	PCI Express I/O Limit Upper 16-Bit Register (RC mode only)	R‘	0x0000	15.4.3.15/15-35
0x034	PCI Express Capabilities Pointer Register	R	0x044	15.4.3.16/15-35
0x03C	PCI Express Interrupt Line Register	R/W	0x0000	15.4.3.17/15-36
0x03D	PCI Express Interrupt Pin Register	R	0x0001	15.4.3.18/15-36
0x03E	PCI Express Bridge Control Register (RC mode only)	R/W	0x0000	15.4.3.19/15-36
0x044	PCI Express Power Management Capability ID Register	R	0x01	15.4.4.1/15-39
0x045	PCI Express Power Management Next Capabilities Pointer Register	R	0x4C	15.4.4.2/15-39
0x046	PCI Express Power Management Capabilities Register	R	0x7E02	15.4.4.3/15-39
0x048	PCI Express Power Management Status and Control Register	Mixed	0x0000	15.4.4.4/15-40
0x04B	PCI Express Power Management Data Register	R	0x0000	15.4.4.5/15-41
0x04C	PCI Express Capability ID Register	R	0x10	15.4.4.6/15-41
0x04D	PCI Express Next Capabilities Pointer Register	R	0x70	15.4.4.7/15-41
0x04E	PCI Express Capabilities Register	R	0x00*n*1	15.4.4.8/15-42
0x050	PCI Express Device Capabilities Register	R	0x0000_0000	15.4.4.9/15-42
0x054	PCI Express Device Control Register	R/W	0x2810	15.4.4.10/15-43
0x056	PCI Express Device Status Register	Mixed	0x0000	15.4.4.11/15-44
0x058	PCI Express Link Capabilities Register	R	0x0003_D421	15.4.4.12/15-45
0x05C	PCI Express Link Control Register	R/W	0x0000	15.4.4.13/15-45
0x05E	PCI Express Link Status Register	R	0x0011	15.4.4.14/15-46
0x060	PCI Express Slot Capabilities Register	R	0x000007c0	15.4.4.15/15-46
0x064	PCI Express Slot Control Register	R/W	0x0000	15.4.4.16/15-47
0x066	PCI Express Slot Status Register	Mixed	0x0040	15.4.4.17/15-48
0x068	PCI Express Root Control Register (RC mode only)	R/W	0x0000	15.4.4.18/15-49

Table 15-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
0x06C	PCI Express Root Status Register (RC mode only)	Mixed	0x0000_0000	15.4.4.19/15-49
0x070	PCI Express MSI Message Capability ID Register (EP mode only)	R	0x05	15.4.4.20/15-50
0x072	PCI Express MSI Message Control Register (EP mode only)	Mixed	0x0088	15.4.4.21/15-50
0x074	PCI Express MSI Message Address Register (EP mode only)	R/W	0x0000_0000	15.4.4.22/15-51
0x078	PCI Express MSI Message Upper Address Register (EP mode only)	R/W	0x0000_0000	15.4.4.23/15-51
0x07C	PCI Express MSI Message Data Register (EP mode only)	R/W	0x0000	15.4.4.24/15-51
0x100	PCI Express Advanced Error Reporting Capability ID Register	R	0x1381_0001	15.4.5.1/15-53
0x104	PCI Express Uncorrectable Error Status Register	R/W	0x0000_0000	15.4.5.2/15-53
0x108	PCI Express Uncorrectable Error Mask Register	R/W	0x0000_0000	15.4.5.3/15-54
0x10C	PCI Express Uncorrectable Error Severity Register	R/W	0x0006_2010	15.4.5.4/15-55
0x110	PCI Express Correctable Error Status Register	w1c	0x0000_0000	15.4.5.5/15-56
0x114	PCI Express Correctable Error Mask Register	R/W	0x0000_0000	15.4.5.6/15-57
0x118	PCI Express Advanced Error Capabilities and Control Register	R/W	0x0000_00A0	15.4.5.7/15-57
0x11C	PCI Express Header Log Register	R	0x0000_0000	15.4.5.8/15-59
0x120	PCI Express Header Log Register	R	0x0000_0000	
0x124	PCI Express Header Log Register	R	0x0000_0000	
0x128	PCI Express Header Log Register	R	0x0000_0000	
0x12C	PCI Express Root Error Command Register	R/W	0x0000_0000	15.4.5.9/15-60
0x130	PCI Express Root Error Status Register	Mixed	0x0000_0000	15.4.5.10/15-60
0x134	PCI Express Error Source Identification Register	R	0x0000_0000	15.4.5.11/15-61
PCI Express Core Control and Status Registers (CSRs)				
0x404	PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)	R	0x0000_0000	15.4.6.1/15-62
0x41C	PCI Express N_FTS Control Register (PEX_NFTS_CTRL)	R/W	0x0000_4040	15.4.6.2/15-63
0x438	PCI Express ACK Replay Timeout Register (PEX_ACKRPLY_TO)	R/W	0x00C2_415C	15.4.6.3/15-64
0x440	PCI Express Core Clock Ratio Register (PEX_GCLK_RATIO)	Mixed	0x0000_0010	15.4.6.4/15-65
0x450	PCI Express Power Management Timer Register (PEX_PM_TIMER)	Mixed	0x0019_0960	15.4.6.5/15-66
0x454	PCI Express PME Time-Out Register (PEX_PME_TIMEOUT)	Mixed	0x0262_5A00	15.4.6.6/15-67
0x45C	PCI Express ASPM Request Timer Register (PEX_ASPM_REQTMR) (RC mode only)	R/W	0x0000_0ED8	15.4.6.7/15-68
0x478	PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE)	R/W	0x0000_0000	15.4.6.8/15-68

Table 15-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
0x47C	PCI Express Device Capabilities Update Register (PEX_DEVCAP_UPDATE)	R/W	0x0000_0000	15.4.6.9/15-69
0x480	PCI Express Link Capabilities Update Register (PEX_LINKCAP_UPDATE)	R/W	0x0000_3D42	/15-69
0x490	PCI Express Slot Capabilities Update Register (PEX_SLCAP_UPDATE)	R/W	0x000007c0	15.4.6.11/15-71
0x4B0	PCI Express Configuration Ready Register (PEX_CFG_READY)	Mixed	0x0000_0000	15.4.6.12/15-72
PCI Express BAR Configuration Registers (EP Mode)				
0x4D8	PCI Express BAR Size Low Configuration Register (PEX_BAR_SIZEL)	R/W	0xFC00_0000	15.4.7.1/15-73
0x4DC	Reserved	R/W	0xFFFF_FFFF	15.4.7.2/15-74
0x4E0	PCI Express BAR Select Configuration Register (PEX_BAR_SEL)	R/W	0x0000_0400	15.4.7.2/15-74
0x504	PCI Express BAR Prefetch Configuration Register (PEX_BAR_PF)	R/W	0x0000_0400	15.4.7.3/15-74
PCI Express Extended Status and Control Register				
0x590	PCI Express PME_To_Ack Timeout Register (PEX_PME_TO_ACK_TOR)	Mixed	0x0262_5A00	15.4.8.1/15-75
0x594	PCI Express PME_To_Ack Status Register (PEX_PME_TO_ACK_SR)	w1c	0x0000_0000	15.4.8.2/15-76
0x5A0	PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK)	Mixed	0x0000_003F	15.4.8.3/15-77
PCI Express CSB Bridge Registers				
Global Registers				
0x800	Reserved	RO	0x0110_1010	—
0x804	Reserved	RO	0x0003_249F	—
0x808	PCI Express CSB Bridge Control register (PEX_CSB_CTRL)	R/W	0x0000_0130	15.5.2.1/15-78
0x80C	Reserved	RO	0x0000_0000	—
0x814	PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR)	R/W	0x0000_0000	15.5.2.2/15-79
0x818	Reserved	RO	0x0000_0000	—
0x81C	PCI Express CSB Bridge Status register (PEX_CSB_STAT)	RO	0x0000_0000	15.5.2.3/15-80
0x820	Reserved	RO	0x0000_0000	—
PCI Express Outbound PIO Registers				
0x840	PCI Express Outbound PIO Control Register (PEX_CSB_OBCTRL)	R/W	0x0000_0000	15.5.3.1/15-81

Table 15-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
PCI Express 1—Block Base Address 0x0_9000 **PCI Express 2—Block Base Address 0x0_A000**				
0x844	PCI Express Outbound PIO Status Register (PEX_CSB_OBSTAT)	w1c	0x0000_0000	15.5.3.2/15-82
0x848	Reserved	RO	0x0000_0000	—
PCI Express Inbound PIO Registers				
0x8E0	PCI Express Inbound PIO Control Register (PEX_CSB_IBCTRL)	R/W	0x0000_0000	15.5.4.1/15-83
0x8E4	PCI Express Inbound PIO Status Register (PEX_CSB_IBSTAT)	w1c	0x0000_0000	15.5.4.2/15-83
0x8E8	Reserved	RO	0x0000_0000	—
PCI Express DMA Registers				
0x990	Reserved	RO	0x0000_0000	—
0x9A0	PCI Express Write DMA Control Register (PEX_WDMA_CTRL)	R/W	0x0000_0000	15.5.5.1/15-84
0x9A4	PCI Express Write DMA first Address Register (PEX_WDMA_ADDR)	R/W	0x0000_0000	15.5.5.2/15-85
0x9A8	PCI Express Write DMA Status Register (PEX_WDMA_STAT)	w1c	0x0000_0000	15.5.5.3/15-85
0x9AC	Reserved	RO	0x0000_0000	—
0xA40	PCI Express Read DMA Control Register (PEX_RDMA_CTRL)	R/W	0x0000_0000	15.5.5.4/15-86
0xA44	PCI Express Read DMA first Address Register (PEX_RDMA_ADDR)	R/W	0x0000_0000	15.5.5.5/15-87
0xA48	PCI Express Read DMA Status Register (PEX_RDMA_STAT)	w1c	0x0000_0000	15.5.5.6/15-87
Mailbox Registers				
0xB20	PCI Express Outbound Mailbox Control Register (PEX_OMBCR)	R/W	0x0000_0000	15.5.6.1/15-88
0xB24	PCI Express Outbound Mailbox Data Register (PEX_OMBDR)	R/W	0x0000_0000	15.5.6.2/15-89
0xB60	PCI Express Inbound Mailbox Control Register (PEX_IMBCR)	R/W	0x0000_0000	15.5.6.3/15-89
0xB64	PCI Express Inbound Mailbox Data Register (PEX_IMBDR)	R/W	0x0000_0000	15.5.6.4/15-90
PCI Express Host Interrupts Registers				
0xBA0	PCI Express Host Interrupt Enable Register (PEX_HIER)	R/W	0x0000_0000	15.5.7.1/15-90
0xBA4	PCI Express Host Interrupt Status Register (PEX_HISR)	w1c	0x0000_0000	15.5.7.2/15-91
0xBA8	PCI Express Host Outbound PIO Interrupt Vector Register (PEX_HOPIVR)	R/W	0x0000_0000	15.5.7.3/15-92
0xBC0	PCI Express Host Inbound PIO Interrupt Vector Register (PEX_HIPIVR)	R/W	0x0000_0000	15.5.7.4/15-93
0xBC8	PCI Express Host Write DMA Interrupt Vector Register (PEX_HWDIVR)	R/W	0x0000_0000	15.5.7.5/15-93
0xBD0	PCI Express Host Read DMA Interrupt Vector Register (PEX_HRDIVR)	R/W	0x0000_0000	15.5.7.6/15-94

Table 15-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
0xBD8	PCI Express Host Miscellaneous Interrupt Vector Register (PEX_HMIVR)	R/W	0x0000_0000	15.5.7.7/15-94
CSB System Interrupts Registers				
0xBE0	CSB System PIO Interrupt Enable Register (PEX_CSPIER)	R/W	0x0000_0000	15.5.8.1/15-95
0xBE4	CSB System Write DMA Interrupt Enable Register (PEX_CSWDIER)	R/W	0x0000_0000	15.5.8.2/15-96
0xBE8	CSB System Read DMA Interrupt Enable Register (PEX_CSRDIER)	R/W	0x0000_0000	15.5.8.3/15-97
0xBEC	CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER)	R/W	0x0000_0002	15.5.8.4/15-98
0xBF0	CSB System PIO Interrupt Status Register (PEX_CSPISR)	w1c	0x0000_0000	15.5.8.5/15-99
0xBF4	CSB System Write DMA Interrupt Status Register (PEX_CSWDISR)	w1c	0x0000_0000	15.5.8.6/15-100
0xBF8	CSB System Read DMA Interrupt Status Register (PEX_CSRDISR)	w1c	0x0000_0000	15.5.8.7/15-101
0xBFC	CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR)	w1c	0x0000_0000	15.5.8.8/15-101
Power Management Registers				
0xC80	PCI Express PM Control Register (PEX_PM_CTRL)	R/W	0x0000_0000	15.5.9.1/15-103
PCI Express Outbound Address Mapping Registers				
0xCA0	PCI Express Outbound Window Attributes Register 0 (PEX_OWAR0)	R/W	0x0000_0000	15.5.10.1/15-104
0xCA4	PCI Express Outbound Window Base Address Register 0 (PEX_OWBAR0)	R/W	0x0000_0000	15.5.10.2/15-105
0xCA8	PCI Express Outbound Window Translation Address Register Low 0 (PEX_OWTARL0)	R/W	0x0000_0000	15.5.10.3/15-106
0xCAC	PCI Express Outbound Window Translation Address Register High 0 (PEX_OWTARH0)	R/W	0x0000_0000	15.5.10.4/15-106
0xCB0	PCI Express Outbound Window Attributes Register 1 (PEX_OWAR1)	R/W	0x0000_0000	15.5.10.1/15-104
0xCB4	PCI Express Outbound Window Base Address Register 1 (PEX_OWBAR1)	R/W	0x0000_0000	15.5.10.2/15-105
0xCB8	PCI Express Outbound Window Translation Address Register Low 1 (PEX_OWTARL1)	R/W	0x0000_0000	15.5.10.3/15-106
0xCBC	PCI Express Outbound Window Translation Address Register High 1 (PEX_OWTARH1)	R/W	0x0000_0000	15.5.10.4/15-106
0xCC0	PCI Express Outbound Window Attributes Register 2 (PEX_OWAR2)	R/W	0x0000_0000	15.5.10.1/15-104

Table 15-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
PCI Express 1—Block Base Address 0x0_9000 **PCI Express 2—Block Base Address 0x0_A000**				
0xCC4	PCI Express Outbound Window Base Address Register 2 (PEX_OWBAR2)	R/W	0x0000_0000	15.5.10.2/15-105
0xCC8	PCI Express Outbound Window Translation Address Register Low 2 (PEX_OWTARL2)	R/W	0x0000_0000	15.5.10.3/15-106
0xCCC	PCI Express Outbound Window Translation Address Register High 2 (PEX_OWTARH2)	R/W	0x0000_0000	15.5.10.4/15-106
0xCD0	PCI Express Outbound Window Attributes Register 3 (PEX_OWAR3)	R/W	0x0000_0000	15.5.10.1/15-104
0xCD4	PCI Express Outbound Window Base Address Register 3 (PEX_OWBAR3)	R/W	0x0000_0000	15.5.10.2/15-105
0xCD8	PCI Express Outbound Window Translation Address Register Low 3 (PEX_OWTARL3)	R/W	0x0000_0000	15.5.10.3/15-106
0xCDC	PCI Express Outbound Window Translation Address Register High 3 (PEX_OWTARH3)	R/W	0x0000_0000	15.5.10.4/15-106
PCI Express EP Inbound Address Translation Registers				
0xDE0	PCI Express EP Inbound Window Translation Address Register 0 (PEX_EPIWTAR0)	R/W	0x0000_0000	15.5.11.1/15-107
0xDE4	PCI Express EP Inbound Window Translation Address Register 1 (PEX_EPIWTAR1)	R/W	0x0000_0000	15.5.11.1/15-107
0xDE8	PCI Express EP Inbound Window Translation Address Register 2 (PEX_EPIWTAR2)	R/W	0x0000_0000	15.5.11.1/15-107
0xDEC	PCI Express EP Inbound Window Translation Address Register 3 (PEX_EPIWTAR3)	R/W	0x0000_0000	15.5.11.1/15-107
PCI Express RC Inbound Address Mapping Registers				
0xE60	PCI Express RC Inbound Window Attributes Register 0 (PEX_RCIWAR0)	R/W	0x0000_0000	15.5.12.1/15-108
0xE64	PCI Express RC Inbound Window Translation Address Register 0 (PEX_RCIWTAR0)	R/W	0x0000_0000	15.5.12.2/15-109
0xE68	PCI Express RC Inbound Window Base Address Register Low 0 (PEX_RCIWBARL0)	R/W	0x0000_0000	15.5.12.3/15-110
0xE6C	PCI Express RC Inbound Window Base Address Register High 0 (PEX_RCIWBARH0)	R/W	0x0000_0000	15.5.12.4/15-110
0xE70	PCI Express RC Inbound Window Attributes Register 1 (PEX_RCIWAR1)	R/W	0x0000_0000	15.5.12.1/15-108
0xE74	PCI Express RC Inbound Window Translation Address Register 1 (PEX_RCIWTAR1)	R/W	0x0000_0000	15.5.12.2/15-109
0xE78	PCI Express RC Inbound Window Base Address Register Low 1 (PEX_RCIWBARL1)	R/W	0x0000_0000	15.5.12.3/15-110
0xE7C	PCI Express RC Inbound Window Base Address Register High 1 (PEX_RCIWBARH1)	R/W	0x0000_0000	15.5.12.4/15-110

Table 15-3. PCI Express Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
PCI Express 1—Block Base Address 0x0_9000 **PCI Express 2—Block Base Address 0x0_A000**				
0xE80	PCI Express RC Inbound Window Attributes Register 2 (PEX_RCIWAR2)	R/W	0x0000_0000	15.5.12.1/15-108
0xE84	PCI Express RC Inbound Window Translation Address Register 2 (PEX_RCIWTAR2)	R/W	0x0000_0000	15.5.12.2/15-109
0xE88	PCI Express RC Inbound Window Base Address Register Low 2 (PEX_RCIWBARL2)	R/W	0x0000_0000	15.5.12.3/15-110
0xE8C	PCI Express RC Inbound Window Base Address Register High 2 (PEX_RCIWBARH2)	R/W	0x0000_0000	15.5.12.4/15-110
0xE90	PCI Express RC Inbound Window Attributes Register 3 (PEX_RCIWAR3)	R/W	0x0000_0000	15.5.12.1/15-108
0xE94	PCI Express RC Inbound Window Translation Address Register 3 (PEX_RCIWTAR3)	R/W	0x0000_0000	15.5.12.2/15-109
0xE98	PCI Express RC Inbound Window Base Address Register Low 3 (PEX_RCIWBARL3)	R/W	0x0000_0000	15.5.12.3/15-110
0xE9C	PCI Express RC Inbound Window Base Address Register High 3 (PEX_RCIWBARH3)	R/W	0x0000_0000	15.5.12.4/15-110
PCI Express Controller 2 Memory-Mapped Registers				
0x000–0xFFC	PCI Express Controller 2 registers **Note:** All registers defined for PCI Express controller 1 are also defined for PCI Express controller 2; the offsets of the PCI Express controller 2 registers are the same except they have a different block base address of 0x0_A000.			

15.4 PCI Express Core Configuration Header Registers

The PCI Express core implements a standard type 0/type 1 configuration space, which consists of a 64-byte type 0 configuration space header and capability structures listed in PCI/PCI Express specification.

The various capabilities supported are as follows:

- Power management (PM)
- PCI Express (PCI_EX)
- Message signaled interrupt (MSI) (not present for RC)
- Vital product data (VPD) (not present for RC)
- Subsystem ID and subsystem vendor ID (SSID/SSVID) (optional capability only for type-1 header devices)

The supported PCI Express extended capabilities are as follows:

- Advanced error reporting
- Device serial number
- Power budgeting VC capability (including VC arbitration table for WRR-32/port arbitration table)
- Vendor specific capability (VSEC)

15.4.1 Common PCI-Compatible Configuration Header Registers

The first 64 bytes of the 256-byte PCI-compatible configuration space consists of a predefined header that every PCI device must support. The first 16 bytes of the predefined header are defined the same way for all PCI Express devices. These common registers are shown in Figure 15-2. They are common to both type 0 and type 1 configuration headers.

Reserved

				Address Offset (Hex)
Device ID		Vendor ID		00
Status		Command		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	Cache Line Size	0C

Figure 15-2. PCI Express PCI-Compatible Configuration Header Common Registers

The remaining 48 bytes of the header may have differing layouts depending on the function of the device. Two header types apply to PCI Express. Type 0 headers, described in Section 15.4.2, “Type 0 PCI-Compatible Configuration Header Registers,” are typically used by endpoints; Type 1 headers described in Section 15.4.3, “Type 1 PCI-Compatible Configuration Header Registers,” are used by root complexes and switches/bridges.

NOTE

The registers described in this section use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI Express bus.

15.4.1.1 PCI Express Vendor ID Register

The vendor ID register, shown in Figure 15-3, identifies the manufacturer of the device.

Offset 0x000 Access: Read-only

	15															0
R	Vendor ID															
W																
Reset	0	0	0	1	1	0	0	1	0	1	0	1	0	1	1	1

Figure 15-3. PCI Express Vendor ID Register

Table 15-4 describes the vendor ID register fields.

Table 15-4. PCI Express Vendor ID Register Field Description

Bits	Name	Description
15–0	Vendor ID	0x1957 (Freescale)

15.4.1.2 PCI Express Device ID Register

The device ID register, shown in Figure 15-4, identifies the device.

Offset 0x002 — Access: Read-only

	15–0
R	Device ID
W	
Reset	Device-specific; see field description

Figure 15-4. PCI Express Device ID Register

Table 15-5 describes the device ID register fields.

Table 15-5. PCI Express Device ID Register Field Description

Bits	Name	Description
15–0	Device ID	Device ID. This field identifies the device. 00C4 MPC8378E 00C5 MPC8378 00C6 MPC8377E 00C7 MPC8377

15.4.1.3 PCI Express Command Register

The PCI Express command register, shown in Figure 15-5, controls the ability to generate and respond to PCI Express cycles. The error control and status bits in the command and status registers control PCI-compatible error reporting. Note that PCI Express advanced error reporting is controlled by the PCI Express device control register described in Section 15.4.4.10, "PCI Express Device Control Register," and the advance error reporting capability structure described in Section 15.4.5.1, "PCI Express Advanced Error Reporting Capability ID Register," through Section 15.4.5.11, "PCI Express Error Source Identification Register."

Offset 0x004 — Access: Mixed

	15–11	10	9	8	7	6	5–3	2	1	0
R	—	Interrupt Disable	—	SERR	—	Parity error response	—	Bus master	Memory space	I/O space
W										
Reset	All zeros									

Figure 15-5. PCI Express Command Register

Table 15-6 describes the bits of the command register.

Table 15-6. PCI Express Command Register Fields Description

Bits	Name	Description
15–11	—	Reserved
10	Interrupt Disable	Controls the ability to generate INTx interrupt messages. 0 Enables messages 1 Disables messages Any INTx emulation interrupts already asserted by this device must be negated when this bit is set.

Table 15-6. PCI Express Command Register Fields Description (continued)

Bits	Name	Description
9	—	Reserved
8	SERR	Controls the reporting of fatal and non-fatal errors detected by the device to the root complex. 0 Disables reporting 1 Enables reporting
7	—	Reserved
6	Parity error response	Controls whether this PCI Express controller responds to parity errors. 0 Parity errors are ignored and normal operation continues. 1 Parity errors cause the appropriate bit in the PCI Express status register to be set. However, note that errors are reported based on the values set in the PCI Express error enable and detection registers.
5–3	—	Reserved
2	Bus master	Enables/disables this PCI Express device to behave as a PCI Express bus master. 0 Disables the ability to generate PCI Express accesses. 1 Enables this PCI Express controller to behave as a bus master. Clearing this bit prevent the device from issuing any memory or I/O transactions. Because MSI interrupts are effectively memory writes, clearing this bit also disables the ability of the device to issue MSI interrupts.
1	Memory space	Controls whether this PCI Express device (as a target) responds to memory accesses. 0 Device does not respond to PCI Express memory space accesses. 1 Device responds to PCI Express memory space accesses. Clearing this bit prevents the device from accepting any memory transaction. It does not affect outbound memory transactions.
0	I/O space	I/O space. This bit is hard-wired to 0.

15.4.1.4 PCI Express Status Register

The status register, shown in Figure 15-6, records status information for PCI Express events.

Offset 0x006 Access: Mixed

	15	14	13	12	11	10	9	8
R	Detected parity error	Signaled system error	Received master-abort	Received target-abort	Signaled target-abort	—		Master data parity error
W	w1c	w1c	w1c	w1c	w1c			w1c
Reset	All zeros							

	7	6	5	4	3	2		0
R	—	—	—	Capabilities list	Interrupt Status	—		
W								
Reset	0	0	0	1	0	0	0	0

Figure 15-6. PCI Express Status Register

Table 15-7 describes the PCI Express status register bits.

Table 15-7. PCI Express Status Register Fields Description

Bits	Name	Description
15	Detected parity error[1]	Set when a device receives a poisoned TLP regardless of the state of bit 6 in the command register.
14	Signaled system error[1]	Set when a device sends a ERR_FATAL or ERR_NONFATAL message and the SERR enable bit in the command register is set.
13	Received master-abort[1]	Set when a requestor receives a completion with unsupported request completion status.
12	Received target-abort[1]	Set when a device receives a completion with completer abort completion status.
11	Signaled target-abort[1]	Set when a device completes a request using completer abort completion status.
10–9	—	Reserved
8	Master data parity error detected[1]	Set by the requestor (primary side for Type1 headers) when either the requestor receives a completion marked poisoned or the requestor poisons a write request. Note that the parity error enable bit (bit 6) in the command register must be set before this bit can be set.
7–5	—	Reserved.
4	Capabilities List	All PCI Express devices are required to implement the PCI Express capability structure.
3	Interrupt Status	Set when an INTx interrupt message is pending internally to the device. Note that this bit is associated with INTx messages and not Message Signaled Interrupts.
2–0	—	Reserved.

[1] The error control and status bits in the command and status registers control PCI-compatible error reporting. PCI Express advanced error reporting is controlled by the PCI Express device control register described in Section 15.4.4.10, "PCI Express Device Control Register," and the advance error reporting capability structure described in Section 15.4.5.1, "PCI Express Advanced Error Reporting Capability ID Register," through Section 15.4.5.11, "PCI Express Error Source Identification Register."

15.4.1.5 PCI Express Revision ID Register

The revision ID register, shown in Figure 15-7, identifies the revision of the device.

Offset 0x008 Access: Read-only

	7 … 0
R	Revision ID
W	
Reset	Revision specific

Figure 15-7. PCI Express Revision ID Register

Table 15-8 describes the revision ID register fields.

Table 15-8. PCI Express Revision ID Register Fields Description

Bits	Name	Description
7–0	Revision ID	Revision specific. The value is 0x10.

15.4.1.6 PCI Express Class Code Register

The PCI Express class code register, shown in Figure 15-8, is composed of three single-byte fields—base class (offset 0x00B), sub-class (offset 0x00A), and programming interface (offset 0x009)—that indicate the basic functionality.

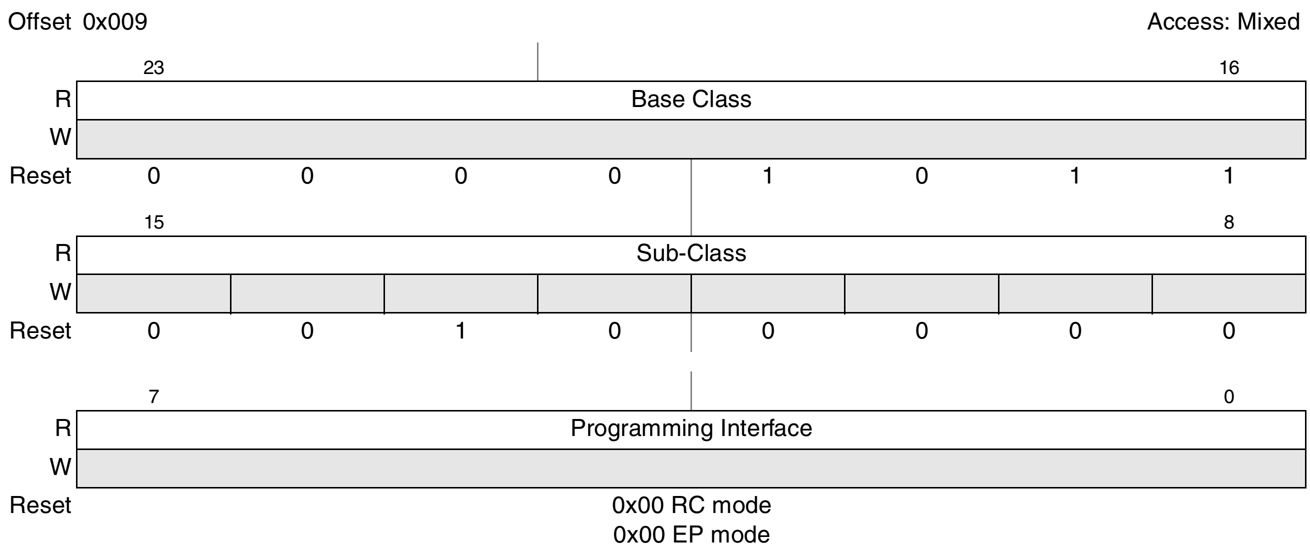


Figure 15-8. PCI Express Class Code Register

Table 15-9 describes the class code register fields.

Table 15-9. PCI Express Class Code Register Fields Description

Bits	Name	Description
23–16	Base Class	0x0B—Processor
15–8	Sub-Class	0x20—PowerPC
7–0	Programming Interface	0x00—RC mode 0x00—EP mode

15.4.1.7 PCI Express Cache Line Size Register

The cache line size register, shown in Figure 15-9, is provided for legacy compatibility (PCI 2.3); it is not used for PCI Express device functionality.

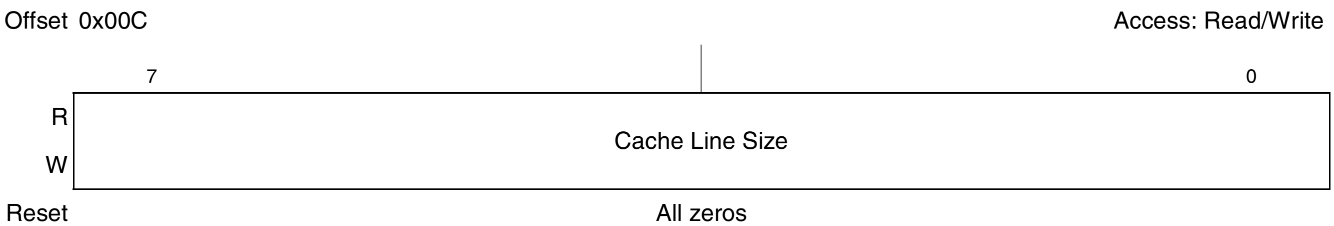


Figure 15-9. PCI Express Bus Cache Line Size Register

Table 15-10 describes the cache line size register.

Table 15-10. PCI Express Bus Cache Line Size Register Fields Description

Bits	Name	Description
7–0	Cache Line Size	Represents the cache line size of the processor in terms of 32-bit words (eight 32-bit words = 32 bytes). Note that for PCI Express operation this register is ignored.

15.4.1.8 PCI Express Latency Timer Register

The latency timer register, shown in Figure 15-10, is provided for legacy compatibility (PCI 2.3); it is not used for PCI Express device functionality.

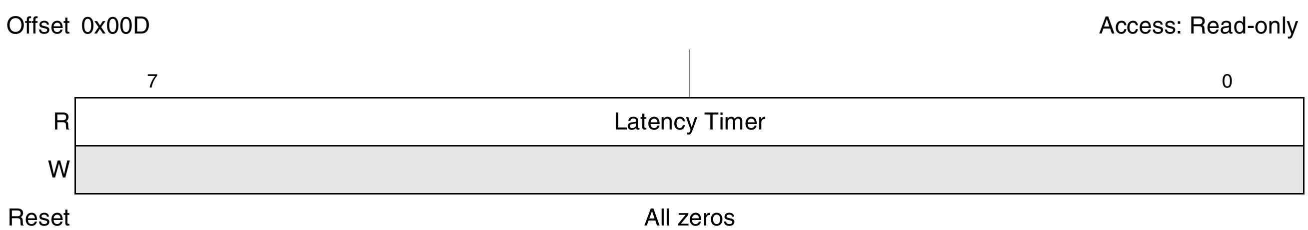


Figure 15-10. PCI Express Latency Timer Register

Table 15-11 describes the PCI Express latency timer register (PLTR).

Table 15-11. PCI Express Latency Timer Register Fields Description

Bits	Name	Description
7–0	Latency Timer	Note that for PCI Express operation this register is ignored.

15.4.1.9 PCI Express Header Type Register

The PCI Express header type register, shown in Figure 15-9, identifies the layout of the PCI-compatible header.

Offset 0x00E — Access: Read-only

	7	6–0
R	Multifunction	Header Layout
W		
Reset	0x00 (EP mode) 0x01 (RC mode)	

Figure 15-11. PCI Express Header Type Register

Table 15-11 describes the PCI Express header type register.

Table 15-12. PCI Express Header Type Register Fields Description

Bits	Name	Description
7	Multifunction	Identifies whether a device supports multiple functions 0 Single-function device 1 Multiple-function device
6–0	Header Layout	0x00 Endpoint. See Figure 15-12 for type 0 layout. 0x01 Root Complex. See Figure 15-23 for type 1 layout. All other encodings are reserved.

15.4.1.10 PCI Express BIST Register

The BIST register is optional and reserved on the PCI Express controller.

15.4.2 Type 0 PCI-Compatible Configuration Header Registers

The type 0 header is shown in Figure 15-12.

Reserved (shaded)

				Address Offset (Hex)
Device ID		Vendor ID		00
Status		Command		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	Cache Line Size	0C
Base Address Registers				10
				14
				18
				1C
				20
				24
(Reserved)				28
Subsystem ID		Subsystem Vendor ID		2C
(Reserved)				30
(Reserved)			Capabilities Pointer	34
Expansion ROM Base Address				38
MAX_LAT	MIN_GNT	Interrupt Pin	Interrupt Line	3C

Figure 15-12. PCI Express PCI-Compatible Configuration Header—Type 0

Section 15.4.1, “Common PCI-Compatible Configuration Header Registers,” describes the registers in the first 16 bytes of the header. This section describes the registers that are unique to the type 0 header beginning at offset 0x010.

15.4.2.1 PCI Express Base Address Registers (EP Mode Only)

The PCI Express base address registers (BARs) point to the beginning of distinct address ranges which the device should claim. The device supports two 32-bit memory space BARs and two 64-bit memory space BARs. These registers in the header configuration space are used for inbound PCI express transactions, in EP mode only. Note that in RC mode, the device only supports BARs defined by the inbound ATMUs.

For a standard enumeration sequence, the base address registers (BARs) registers are accessed by the Root Complex device to determine the EP attributes. The EP local host should initialize these attributes (if different than the default values) before accepting configuration accesses. This BARs size and prefetch attributes are programmed by an indirect registers access, using the PCI Express BAR Configuration Registers. For further details see Section 15.4.7, “PCI Express BAR Configuration Registers (EP Mode).”

NOTE

To access the device internal memory-mapped configuration registers space from the PCI Express side, the IMMRBAR address should be programmed to one of the PCI Express EP Inbound Window Translation Address Registers (PEX_EPIWTAR*n*) corresponding to one of the base address registers described in this section.

15.4.2.1.1 Base Address Registers 0 and 1 (BAR0/BAR1)

BAR0 and BAR1, shown in Figure 15-13, defines the inbound memory windows in the 32-bit memory space.

Offset 0x010 (EP mode only)
0x014 (EP mode only)

Access: Mixed

	31–12	11–4	3	2–1	0
R	ADDRESS	—	PREF	TYPE	MemSp
W					
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1	0 0	0

Figure 15-13. 32-Bit Base Address Registers (BAR0/BAR1)

Table 15-13 describes the BAR0/BAR1 fields.

Table 15-13. BAR0 and BAR1 Register Fields Description

Bits	Name	Description
31–12	ADDRESS	Indicates the base address where the inbound memory window begins. The number of upper bits that the device allows to be writable is selected through the PCI Express BAR configuration registers (EP mode).
11–4	—	Reserved. The device allows a 4 Kbyte window minimum.
3	PREF	Prefetchable. This bit is determined by PCI Express BAR prefetch configuration register (PEX_BAR_PF)
2–1	TYPE	Type. 00 Locate anywhere in 32-bit address space.
0	MemSp	Memory space indicator.

15.4.2.1.2 Base Address Registers 2 and 4 (BAR2/BAR4)

BAR2 and BAR4, shown in Figure 15-14, define the lower portion of the 64-bit inbound memory windows.

Offset 0x018 (EP mode only)
0x020 (EP mode only)

Access: Mixed

	31–12	11–4	3	2–1	0
R	ADDRESS	—	PREF	TYPE	MemSp
W					
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1	1 0	0

Figure 15-14. 64-Bit Low Memory Base Address Register (BAR2)

Table 15-14 describes the PCI Express 64-bit low memory BAR2 and BAR4 fields.

Table 15-14. BAR2 and BAR4 Register Fields Description

Bits	Name	Description
31–12	ADDRESS	Indicates the lower portion of the base address where the inbound memory window begins. The number of upper bits that the device allows to be writable is selected through the PCI Express BAR configuration registers (EP mode).
11–4	—	Reserved. The device allows a 4 Kbyte window minimum.
3	PREF	Prefetchable. This bit is determined by PCI Express BAR prefetch configuration register (PEX_BAR_PF)
2–1	TYPE	Type. 0b10 Locate anywhere in 64-bit address space.
0	MemSp	Memory space indicator

15.4.2.1.3 Base Address Registers 3 and 5 (BAR3/BAR5)

BAR3/BAR5, shown in Figure 15-15, define the upper portion of the 64-bit inbound memory windows.

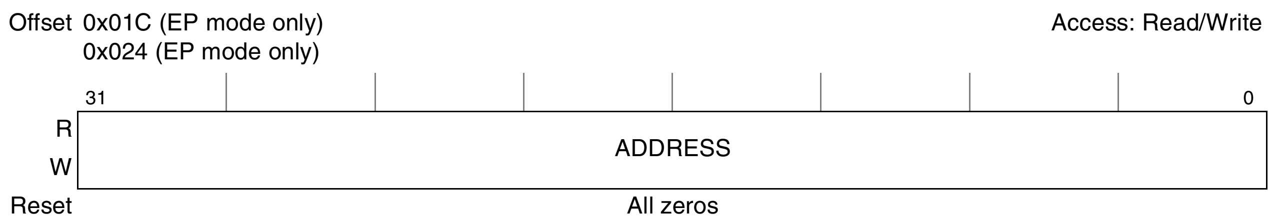


Figure 15-15. 64-Bit High Memory Base Address Registers 3 and 5 (BAR3/BAR5)

Table 15-15 describes the BAR3 and BAR5 fields.

Table 15-15. BAR3 and BAR5 Register Fields Description

Bits	Name	Description
31–0	ADDRESS	Indicates the upper portion of the base address where the inbound memory window begins. Since the local (CSB) address space of the device is only 32 bits (4 Gbytes), this register is all masked (all ones) when accessed during the eumeration sequence.

15.4.2.2 PCI Express Subsystem Vendor ID Register (EP Mode Only)

The PCI Express subsystem vendor ID register identifies the subsystem.

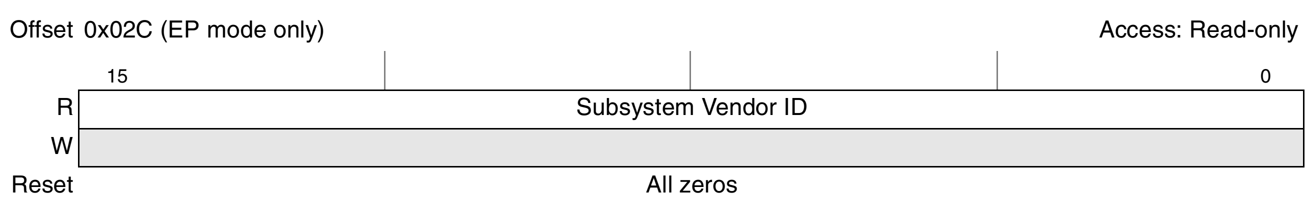


Figure 15-16. PCI Express Subsystem Vendor ID Register

Table 15-16. PCI Express Subsystem Vendor ID Register Fields Description

Bits	Name	Description
15–0	Subsystem Vendor ID	Subsystem Vendor ID. The value of this register can be set by programming the SSVID field of the PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE). See Section 15.4.6.8, "PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE). This value has to be programmed before setting the config-ready bit in the PCI Express Configuration Ready Register so that the host reads the correct information during enumeration.

15.4.2.3 PCI Express Subsystem ID Register (EP Mode Only)

The PCI Express subsystem ID register identifies the subsystem.

Offset 0x02E (EP mode only) — Access: Read-only

	15 … 0
R	Subsystem ID
W	
Reset	All zeros

Figure 15-17. PCI Express Subsystem ID Register

Table 15-17. PCI Express Subsystem ID Register Fields Description

Bits	Name	Description
15–0	Subsystem ID	Subsystem ID. The value of this register can be set by programming the SSID field of the PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE). See Section 15.4.6.8, "PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE). This value has to be programmed before setting the config-ready bit in the PCI Express Configuration Ready Register so that the host reads the correct information during enumeration.

15.4.2.4 PCI Express Capabilities Pointer Register

The PCI Express capabilities pointer identifies additional functionality supported by the device.

Offset 0x034 — Access: Read-only

	7							0
R	Capabilities Pointer							
W								
Reset	0	1	0	0	0	1	0	0

Figure 15-18. PCI Express Capabilities Pointer Register

Table 15-18. PCI Express Capabilities Pointer Register Fields Description

Bits	Name	Description
7–0	Capabilities Pointer	The capabilities pointer provides the offset (0x44) for additional PCI-compatible registers above the common 64-byte header. Refer to Section 15.4.4, "PCI Compatible Device-Specific Configuration Space Registers."

15.4.2.5 PCI Express Interrupt Line Register (EP-Mode Only)

The PCI Express interrupt line register is used by device drivers and OS software to communicate interrupt line routing information. Values in this register are programmed by system software and are system-specific.

Offset 0x03C (EP mode only) Access: Read/Write

	7 … 0
R / W	Interrupt Line
Reset	All zeros

Figure 15-19. PCI Express Interrupt Line Register

Table 15-19. PCI Express Interrupt Line Register Fields Description

Bits	Name	Description
7–0	Interrupt Line	Communicates interrupt line routing information.

15.4.2.6 PCI Express Interrupt Pin Register

The interrupt pin register identifies the legacy interrupt (INTx) messages the device uses.

Offset 0x03D Access: Read-only

	7							0
R	Interrupt Pin							
W								
Reset	0	0	0	0	0	0	0	1

Figure 15-20. PCI Express Interrupt Pin Register

Table 15-20. PCI Express Interrupt Pin Register Fields Description

Bits	Name	Description
7–0	Interrupt pin	Legacy INTx message used by this device. 0x01 INTA only is supported by this device.

15.4.2.7 PCI Express Minimum Grant Register (EP Mode Only)

This register does not apply to PCI Express. It is present for legacy purposes.

Offset 0x03E (EP mode only) | Access: Read-only

	7 … 0
R	MIN_GNT
W	
Reset	All zeros

Figure 15-21. PCI Express Minimum Grant Register (MAX_GNT)

Table 15-21. PCI Express MInimum Grant Register Fields Description

Bits	Name	Description
7–0	MIN_GNT	Does not apply for PCI Express.

15.4.2.8 PCI Express Maximum Latency Register (EP Mode Only)

This register does not apply to PCI Express. It is present for legacy purposes.

Offset 0x03F (EP mode only) | Access: Read-only

	7 … 0
R	MAX_LAT
W	
Reset	All zeros

Figure 15-22. PCI Express Maximum Latency Register (MAX_LAT)

Table 15-22. PCI Express Maximum Latency Register Fields Description

Bits	Name	Description
7–0	MAX_LAT	Does not apply for PCI Express.

15.4.3 Type 1 PCI-Compatible Configuration Header Registers

The type 1 header is shown in Figure 15-23.

Reserved (shaded)

				Address Offset (Hex)
Device ID		Vendor ID		00
Status		Command		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	Cache Line Size	0C
Reserved				10
Reserved				14
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	18
Secondary Status		I/O Limit	I/O Base	1C
Memory Limit		Memory Base		20
Prefetchable Memory Limit		Prefetchable Memory Base		24
Prefetchable Base Upper 32 Bits				28
Prefetchable Limit Upper 32 Bits				2C
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		30
Reserved			Capabilities Pointer	34
Expansion ROM Base Address				38
Bridge Control		Interrupt Pin	Interrupt Line	3C

Figure 15-23. PCI Express PCI-Compatible Configuration Header—Type 1

Section 15.4.1, "Common PCI-Compatible Configuration Header Registers," describes the registers in the first 16 bytes of the header. This section describes the registers that are unique to the type 1 header beginning at offset 0x010.

15.4.3.1 PCI Express Primary Bus Number Register (RC Mode Only)

The primary bus number register is shown in Figure 15-24.

Offset 0x018 — Access: Read/Write

	7 — 0
R / W	Primary Bus Number
Reset	All zeros

Figure 15-24. PCI Express Primary Bus Number Register

Table 15-23 describes the primary bus number register fields.

Table 15-23. PCI Express Primary Bus Number Register Fields Description

Bits	Name	Description
7–0	Primary Bus Number	Bus that is connected to the upstream interface. Note that this register is programmed during system enumeration; in RC mode this register should remain 0x00.

15.4.3.2 PCI Express Secondary Bus Number Register (RC Mode Only)

The secondary bus number register is shown in Figure 15-25.

Offset 0x019 Access: Read/Write

	7 … 0
R W	Secondary Bus Number
Reset	All zeros

Figure 15-25. PCI Express Secondary Bus Number Register

Table 15-24 describes the secondary bus number register fields.

Table 15-24. PCI Express Secondary Bus Number Register Fields Description

Bits	Name	Description
7–0	Secondary Bus Number	Bus that is directly connected to the downstream interface. Note that this register is programmed during system enumeration; in RC mode, this register is typically programmed to 0x01.

15.4.3.3 PCI Express Subordinate Bus Number Register (RC Mode Only)

The subordinate bus number register is shown in Figure 15-26.

Offset 0x01A Access: Read/Write

	7 … 0
R W	Subordinate Bus Number
Reset	All zeros

Figure 15-26. PCI Express Subordinate Bus Number Register

Table 15-25 describes the subordinate bus number register fields.

Table 15-25. PCI Express Subordinate Bus Number Register Fields Description

Bits	Name	Description
7–0	Subordinate Bus Number	Highest bus number that is on the downstream interface.

15.4.3.4 PCI Express Secondary Latency Timer Register (RC Mode Only)

The secondary latency timer register does not apply to PCI Express. It must be read-only and return all zeros when read.

15.4.3.5 PCI Express I/O Base Register (RC Mode Only)

Note that this device does not support inbound I/O transactions. The I/O base register is shown in Figure 15-26.

Offset 0x01C — Access: Read-only

	7–4	3–0
R	I/O Start Address	Address Decode Type
W		
Reset	All zeros	

Figure 15-27. PCI Express I/O Base Register

Table 15-25 describes the I/O base register fields.

Table 15-26. PCI Express I/O Base Register Fields Description

Bits	Name	Description
7–4	I/O Start Address	Specifies bits 15:12 of the I/O space start address
3–0	Address Decode Type	Specifies the number of I/O address bits. 0x00 16-bit I/O address decode 0x01 32-bit I/O address decode All other settings reserved.

15.4.3.6 PCI Express I/O Limit Register (RC Mode Only)

Note that this device does not support inbound I/O transactions. The I/O limit register is shown in Figure 15-26.

Offset 0x01D — Access: Read-only

	7–4	3–0
R	I/O Limit Address	Address Decode Type
W		
Reset	All zeros	

Figure 15-28. PCI Express I/O Limit Register

Table 15-25 describes the I/O limit register fields.

Table 15-27. PCI Express I/O Limit Register Fields Description

Bits	Name	Description
7–4	I/O Limit Address	Specifies bits 15:12 of the I/O space ending address
3–0	Address Decode Type	Specifies the number of I/O address bits. 0x00 16-bit I/O address decode 0x01 32-bit I/O address decode All other settings reserved.

15.4.3.7 PCI Express Secondary Status Register (RC Mode Only)

The PCI Express secondary status register is shown in Figure 15-29. Note that the errors in this register can be masked by corresponding bits in the secondary status interrupt mask register (PEX_SS_INTR_MASK) and that by default all the errors are masked. See Section 15.4.8.3, “Secondary Status Interrupt Mask Register (PEX_SS_INTR_MASK) (RC Mode Only),” for more information.

Offset 0x01E Access: Mixed

	15	14	13	12	11	10	9	8	7–0
R	DPE	SSE	RMA	RTA	STA	—		MDPE	—
W	w1c	w1c	w1c	w1c	w1c			w1c	
Reset	All zeros								

Figure 15-29. PCI Express Secondary Status Register

Table 15-28 describes the PCI Express secondary status register fields.

Table 15-28. PCI Express Secondary Status Register Fields Description

Bits	Name	Description
15	DPE	Detected parity error. This bit is set when the secondary side receives a poisoned TLP regardless of the state of the parity error response bit.
14	SSE	Signaled system error. This bit is set when a device sends a ERR_FATAL or ERR_NONFATAL message if the SERR enable bit in the command register is set to enable reporting.
13	RMA	Received master abort. This bit is set when the secondary side receives an unsupported request (UR) completion.
12	RTA	Received target abort. This bit is set when the secondary side receives a completer abort (CA) completion.
11	STA	Signaled target abort. This bit is set when the secondary side issues a CA completion.
10–9	—	Reserved.
8	MDPE	Master data parity error. This bit is set when the parity error response bit is set and the secondary side requestor receives a poisoned completion or poisons a write request. If the parity error response bit is cleared, this bit is never set.
7–0	—	Reserved

15.4.3.8 PCI Express Memory Base Register (RC Mode Only)

The memory base register is shown in Figure 15-30.

Offset 0x020 Access: Read/Write

	15–4	3–0
R	Memory Base	—
W		
Reset	All zeros	

Figure 15-30. PCI Express Memory Base Register

Table 15-29 describes the memory base register fields.

Table 15-29. PCI Express Memory Base Register Fields Description

Bits	Name	Description
15–4	Memory Base	Specifies bits 31:20 of the non-prefetchable memory space start address. Typically used for specifying memory-mapped I/O space. **Note:** Inbound posted transactions hitting into the mem base/limit range are ignored; inbound non-posted transactions hitting into the mem base/limit range results in an unsupported request response.
3–0	—	Reserved

15.4.3.9 PCI Express Memory Limit Register (RC Mode Only)

The memory limit register is shown in Figure 15-31.

Offset 0x022 Access: Read/Write

	15–4	3–0
R	Memory Limit	—
W		
Reset	All zeros	

Figure 15-31. PCI Express Memory Limit Register

Table 15-30 describes the memory base register fields.

Table 15-30. PCI Express Memory Limit Register Fields Description

Bits	Name	Description
15–4	Memory Limit	Specifies bits 31:20 of the non-prefetchable memory space ending address. Typically used for specifying memory-mapped I/O space. **Note:** Inbound posted transactions hitting into the mem base/limit range are ignored; inbound non-posted transactions hitting into the mem base/limit range result in an unsupported request response.
3–0	—	Reserved

15.4.3.10 PCI Express Prefetchable Memory Base Register (RC Mode Only)

The prefetchable memory base register is shown in Figure 15-32.

Offset 0x024 Access: Read/Write

	15–4	3–0
R	PF Memory Base	Address Decode Type
W		
Reset	All zeros	

Figure 15-32. PCI Express Prefetchable Memory Base Register

Table 15-31 describes the prefetchable memory base register fields.

Table 15-31. PCI Express Prefetchable Memory Base Register Fields Description

Bits	Name	Description
15–4	PF Memory Base	Specifies bits 31:20 of the prefetchable memory space start address.
3–0	Address Decode Type	Number of prefetchable memory address bits. 0x00 32-bit memory address decode 0x01 64-bit memory address decode All other settings reserved.

15.4.3.11 PCI Express Prefetchable Memory Limit Register (RC Mode Only)

The PCI Express prefetchable memory limit register is shown in Figure 15-33.

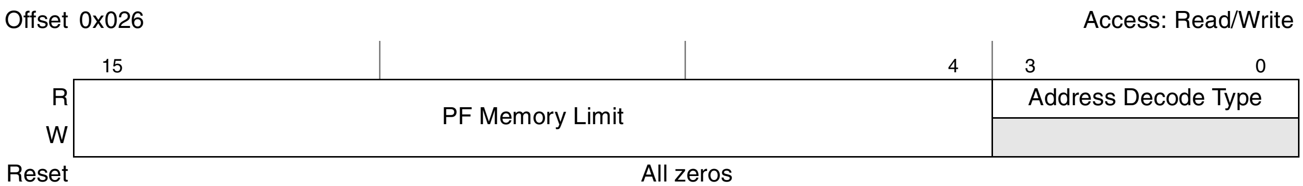


Figure 15-33. PCI Express Prefetchable Memory Limit Register

Table 15-32 describes the prefetchable memory limit register fields.

Table 15-32. PCI Express Prefetchable Memory Limit Register Fields Description

Bits	Name	Description
15–4	PF Memory Limit	Specifies bits 31:20 of the prefetchable memory space ending address.
3–0	Address Decode Type	Specifies the number of prefetchable memory address bits. 0x00 32-bit memory address decode 0x01 64-bit memory address decode All other settings reserved.

15.4.3.12 PCI Express Prefetchable Base Upper 32-Bit Register (RC Mode Only)

The PCI Express prefetchable memory base upper 32-bit register is shown in Figure 15-34.

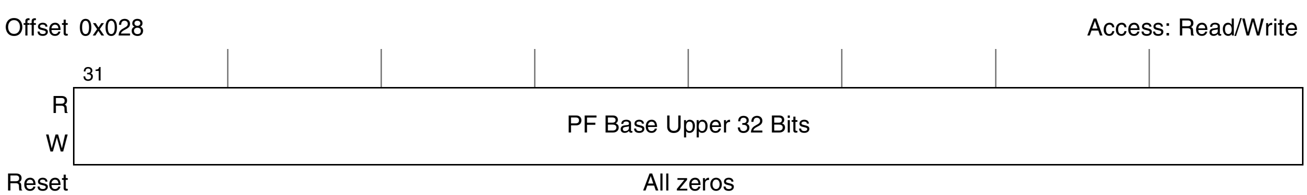


Figure 15-34. PCI Express Prefetchable Base Upper 32-Bit Register

Table 15-33 describes the PCI Express prefetchable memory base upper 32-bit register fields.

Table 15-33. PCI Express Prefetchable Base Upper 32-Bit Register Fields Description

Bits	Name	Description
31–0	PF Base Upper 32 Bits	Specifies bits 64:32 of the prefetchable memory space start address when the address decode type field in the prefetchable memory base register is 0x01.

15.4.3.13 PCI Express Prefetchable Limit Upper 32-Bit Register (RC Mode Only)

The PCI Express prefetchable memory base upper 32-bit register is shown in Figure 15-35.

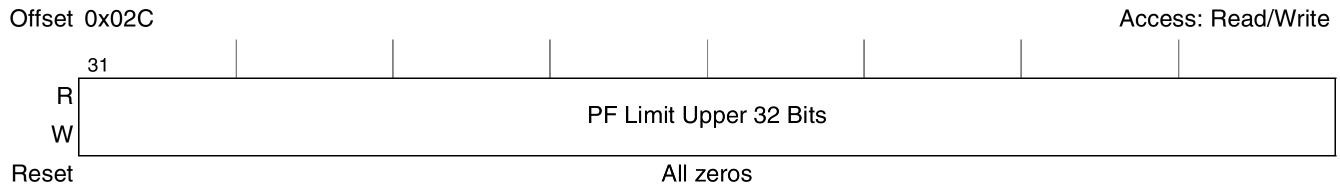


Figure 15-35. PCI Express Prefetchable Limit Upper 32-Bit Register

Table 15-34 describes the PCI Express prefetchable memory limit upper 32-bit register fields.

Table 15-34. PCI Express Prefetchable Limit Upper 32-Bit Register Fields Description

Bits	Name	Description
31–0	PF Limit Upper 32 Bits	Specifies bits 64–32 of the prefetchable memory space ending address when the address decode type field in the prefetchable memory limit register is 0x01.

15.4.3.14 PCI Express I/O Base Upper 16-Bit Register (RC Mode Only)

Note that this device does not support inbound I/O transactions. The I/O base upper 16-bit register is shown in Figure 15-36.

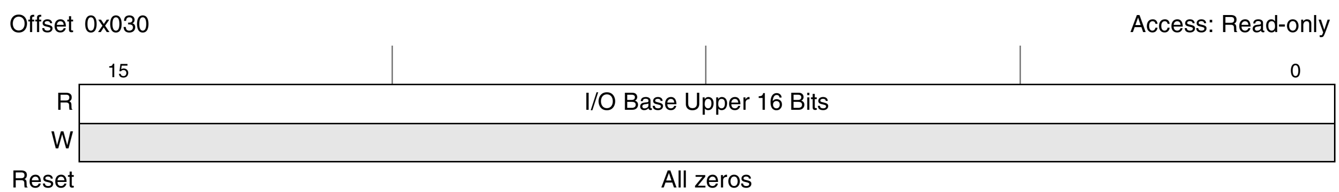


Figure 15-36. PCI Express I/O Base Upper 16-Bit Register

Table 15-35 describes the I/O base upper 16 bits register fields.

Table 15-35. PCI Express I/O Base Upper 16-Bit Register Fields Description

Bits	Name	Description
15–0	I/O Base Upper 16 Bits	Specifies bits 31–16 of the I/O space start address when the address decode type field in the I/O base register is 0x01.

15.4.3.15 PCI Express I/O Limit Upper 16-Bit Register (RC Mode Only)

Note that this device does not support inbound I/O transactions. The I/O limit upper 16-bit register is shown in Figure 15-37.

Offset 0x032 — Access: Read-only

	15 … 0
R	I/O Limit Upper 16 Bits
W	
Reset	All zeros

Figure 15-37. PCI Express I/O Limit Upper 16-Bit Register

Table 15-36 describes the I/O limit upper 16-bit register fields.

Table 15-36. PCI Express I/O Limit Upper 16-Bit Register Fields Description

Bits	Name	Description
15–0	I/O Limit Upper 16 Bits	Specifies bits 31–16 of the I/O space ending address when the address decode type field in the I/O limit register is 0x01.

15.4.3.16 PCI Express Capabilities Pointer Register

The PCI Express capabilities pointer identifies additional functionality supported by the device.

Offset 0x034 — Access: Read-only

	7							0
R	Capabilities Pointer							
W								
Reset	0	1	0	0	0	1	0	0

Figure 15-38. PCI Express Capabilities Pointer Register

Table 15-37. PCI Express Capabilities Pointer Register Fields Description

Bits	Name	Description
7–0	Capabilities Pointer	Provides the offset (0x44) for additional PCI-compatible registers above the common 64-byte header. Refer to Section 15.4.4, “PCI Compatible Device-Specific Configuration Space Registers.”

15.4.3.17 PCI Express Interrupt Line Register

The PCI Express interrupt line register is used by device drivers and OS software to communicate interrupt line routing information. Values in this register are programmed by system software and are system-specific.

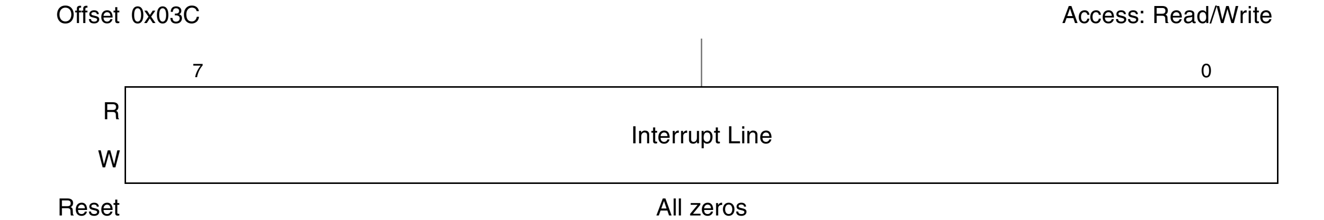


Figure 15-39. PCI Express Interrupt Line Register

Table 15-38. PCI Express Interrupt Line Register Fields Description

Bits	Name	Description
7–0	Interrupt Line	Communicates interrupt line routing information.

15.4.3.18 PCI Express Interrupt Pin Register

The interrupt pin register identifies the legacy interrupt (INTx) messages of the device (or function).

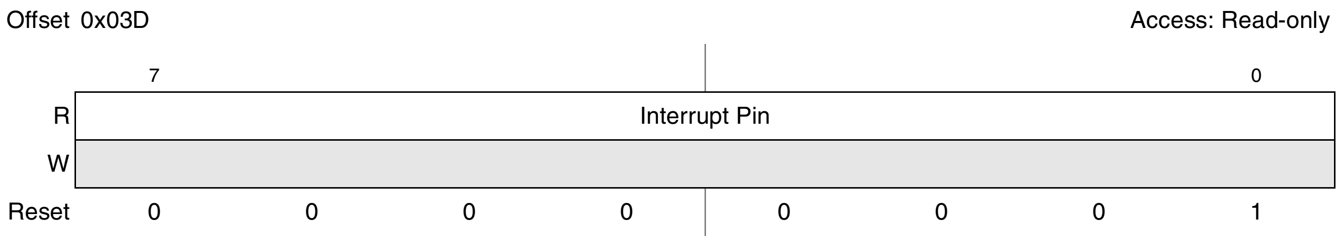


Figure 15-40. PCI Express Interrupt Pin Register

Table 15-39. PCI Express Interrupt Pin Register Fields Description

Bits	Name	Description
7–0	Interrupt Pin	Legacy INTx message used by this device. 0x01 INTA only is supported by this device.

15.4.3.19 PCI Express Bridge Control Register (RC Mode Only)

The PCI Express bridge control register is shown in Figure 15-41.

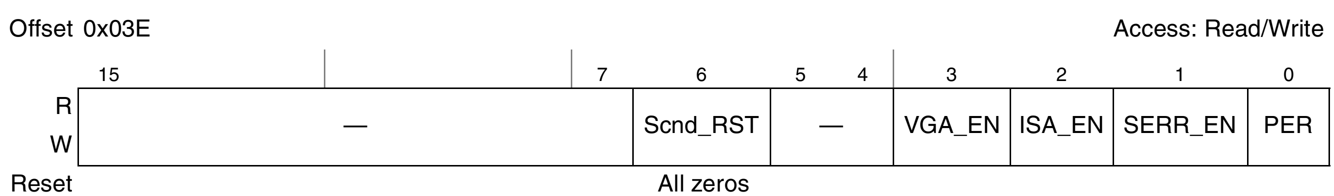


Figure 15-41. PCI Express Bridge Control Register

Table 15-40 describes the PCI Express bridge control register fields.

Table 15-40. PCI Express Bridge Control Register Fields Description

Bits	Name	Description
15–7	—	Reserved
6	Scnd_RST	Secondary bus reset
5–4	—	Reserved
3	VGA_EN	VGA enable
2	ISA_EN	ISA enable
1	SERR_EN	SERR enable. Controls the propagation of ERR_COR, ERR_NONFATAL, and ERR_FATAL responses received on the secondary side.
0	PER	Parity error response.

15.4.4 PCI Compatible Device-Specific Configuration Space Registers

The PCI-compatible device-specific configuration space is a PCI-compatible configuration space from 0x040 to 0x0FF (just above the 64-byte PCI-compatible configuration header).

Reserved (shaded)

				Address Offset (Hex)
PCI-Compatible Configuration Header (See Section 15.4.1, "Common PCI-Compatible Configuration Header Registers," for more information.)				000–03F
Reserved				040
Power Mgmt Capabilities		Next Pointer (0x4C)	Power Mgmt Capability ID	044
Data	Reserved	Power Management Status & Control		048
PCI Express Capabilities		Next Pointer (0x70 — EP mode) (NULL — RC mode)	PCI Express Capability ID	04C
Device Capabilities				050
Device Status		Device Control		054
Link Capabilities				058
Link Status		Link Control		05C
Slot Capabilities				060
Slot Status		Slot Control		064
Reserved		Root Control (RC mode only)		068
Root Status				06C
MSI Message Control		Next Pointer (NULL)	MSI Message Capability ID	070
MSI Message Address				074
MSI Upper Message Address				078
Reserved		MSI Message Data		07C
Reserved				080–0FF

Figure 15-42. PCI-Compatible Device-Specific Configuration Space

15.4.4.1 PCI Express Power Management Capability ID Register

The PCI Express power management capability ID register is shown in Figure 15-43.

Offset 0x044 Access: Read-only

	7							0
R	Power Mgmt Capability ID							
W								
Reset	0	0	0	0	0	0	0	1

Figure 15-43. PCI Express Power Management Capability ID Register

Table 15-41. PCI Express Power Management Capability ID Register Fields Description

Bits	Name	Description
7–0	Power Management Capability ID	Power Management = 0x01

15.4.4.2 PCI Express Power Management Next Capabilities Pointer Register

The PCI Express power management next capabilities pointer register is shown in Figure 15-44.

Offset 0x045 Access: Read-only

	7							0
R								
W								
Reset	0	1	0	0	1	1	0	0

Figure 15-44. PCI Express Power Management Next Capabilities Pointer

Table 15-42. PCI Express Power Management Next Capabilities Pointer Fields Description

Bits	Name	Description
7–0	—	Points to the PCI Express Capability Registers

15.4.4.3 PCI Express Power Management Capabilities Register

The PCI Express power management capabilities register is shown in Figure 15-45.

Offset 0x046 Access: Read-only

	15				11	10	9	8		6	5	4	3	2		0
R	PME Support					D2	D1	AUX Curr			DSI	—	PME CLK	Version		
W																
Reset	0	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0

Figure 15-45. PCI Express Power Management Capabilities Register

Table 15-43. PCI Express Power Management Capabilities Register Fields Description

Bits	Name	Description
15–11	PME Support	For a device, this 5-bit field indicates the power states in which the device may generate a PME. PME can be issues from D0, D1, D2 and D3hot.
10	D2	D2 power state is supported.
9	D1	D1 power state is supported.
8–6	AUX Curr	AUX Current. Vaux and D3cold is not supported by this device.
5	DSI	A Device Specific Initialization is not required.
4	—	Reserved
3	PME CLK	Does not apply to PCI Express
2–0	Version	0x02 indicates compatibility to the PCI EXPRESS BASE SPECIFICATION, REV. 1.0a

15.4.4.4 PCI Express Power Management Status and Control Register

The PCI Express power management status and control register is shown in Figure 15-46.

Offset 0x048 Access: Mixed

	15	14–13	12–9	8	7–2	1–0
R	PME_STAT	Data Scale	Data Select	PME_EN	—	Power State
W	w1c		Data Select	Sticky		Power State
Reset	All zeros					

Figure 15-46. PCI Express Power Management Status and Control Register

Table 15-44. PCI Express Power Management Status and Control Register Fields Description

Bits	Name	Description
15	PME_STAT	PME Status. This bit is set when PME is generated. Writing a "1" to this bit will clear it. Writing a "0" has no effect.
14–13	Data Scale	—
12–9	Data Select	—
8	PME_EN	PME Enable
7–2	—	Reserved
1–0	Power State	Power state. Indicates the current power state of the function. 00 D0 01 D1 02 D2 03 D3

15.4.4.5 PCI Express Power Management Data Register

The PCI Express power management data register is shown in Figure 15-47.

Offset 0x04B — Access: Read-only

	7–0
R	Data
W	
Reset	All zeros

Figure 15-47. PCI Express Power Management Data Register

Table 15-45. PCI Express Power Management Data Register Fields Description

Bits	Name	Description
7–0	Data	—

15.4.4.6 PCI Express Capability ID Register

The PCI Express capability ID register is shown in Figure 15-48.

Offset 0x04C — Access: Read-only

	7							0
R	PCI Express Capability ID							
W								
Reset	0	0	0	1	0	0	0	0

Figure 15-48. PCI Express Capability ID Register

Table 15-46. PCI Express Capability ID Register Fields Description

Bits	Name	Description
7–0	PCI Express Capability ID	PCI Express = 0x10

15.4.4.7 PCI Express Next Capabilities Pointer Register

The PCI Express next capabilities pointer register is shown in Figure 15-49.

Offset 0x04D — Access: Read-only

	7							0
R								
W								
Reset	0	1[1]	1[1]	1[1]	0	0	0	0

1 The reset value of 0b0111_0000 is only true in EP mode. In RC mode, the reset values should be 0b0000_0000.

Figure 15-49. PCI Express Next Capabilities Pointer

Table 15-47. PCI Express Next Capabilities Pointer Fields Description

Bits	Name	Description
7–0		Points to the PCI Express MSI capability registers. The reset value is 0x70 in EP mode and 0x00 in RC mode.

15.4.4.8 PCI Express Capabilities Register

The PCI Express capabilities register is shown in Figure 15-50.

Offset 0x04E | Access: Read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	—		Interrupt Message Number					Slot	Device/Port Type				Version			
W																
Reset	0	0	0	0	0	0	0	0	0	*n*	0	0	0	0	0	1

Figure 15-50. PCI Express Capabilities Register

Table 15-48. PCI Express Capabilities Register Fields Description

Bits	Name	Description
15–14	—	Reserved
13–9	Interrupt Message Number	This device supports only a single MSI number.
8	Slot	Slot Implemented (RC mode only)
7–4	Device/Port Type	0100 (RC mode) 0000 (EP mode)
3–0	Version	Indicates PCI-SIG defined PCI Express capability structure version number. 0x1identifies version 1.0a.

15.4.4.9 PCI Express Device Capabilities Register

The PCI Express device capabilities register is shown in Figure 15-51. Note that for End Point mode some of these fields can be set indirectly, using the PCI Express Device Capabilities Update Register. See

Section 15.4.6.9, “PCI Express Device Capabilities Update Register (PEX_DEVCAP_UPDATE) for additional details.

Offset 0x050 | Access: Read-only

	31			28	27	26	25							18	17	16
R	—				CSPLS		CSPLV								—	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11		9	8		6	5	4	3	2		0
R	—	PIP	AIP	ABP	EP_L1_LAT			EP_L0s_LAT			ET	PHAN_FCT		MAX_PL_SIZE_SUP		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-51. PCI Express Device Capabilities Register

Table 15-49. PCI Express Device Capabilities Register Fields Description

Bits	Name	Description
31–28	—	Reserved
27–26	CSPLS	Captured slot power limit scale
25–18	CSPLV	Captured slot power limit value
17–15	—	Reserved
14	PIP	Power indicator present
13	AIP	Attention indicator present
12	ABP	Attention button present
11–9	EP_L1_LAT	Endpoint L1 acceptable latency
8–6	EP_L0s_LAT	Endpoint L0s acceptable latency
5	ET	Extended tag field supported
4–3	PHAN_FCT	Phantom functions supported
2–0	MAX_PL_SIZE_SUP	Maximum payload size supported. 000 = 128 bytes

15.4.4.10 PCI Express Device Control Register

The PCI Express device control register is shown in Figure 15-52.

Offset 0x054 | Access: Read/write

	15	14		12	11	10	9	8	7		5	4	3	2	1	0
R/W	—	MAX_READ_SIZE			NSE	APE	PFE	ETE	MAX_PAYLOAD_SIZE			RO	URR	FER	NFER	CER
Reset	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0

Figure 15-52. PCI Express Device Control Register

Table 15-50. PCI Express Device Control Register Fields Description

Bits	Name	Description
15	—	Reserved
14–12	MAX_READ_SIZE	Maximum read request size
11	NSE	No snoop enable
10	APE	AUX power PM enable
9	PFE	Phantom functions enable
8	ETE	Extended tag field enable
7–5	MAX_PAYLOAD_SIZE	Maximum payload size
4	RO	Relaxed ordering
3	URR	Unsupported request reporting
2	FER	Fatal error reporting
1	NFER	Non-fatal error reporting
0	CER	Correctable error reporting

15.4.4.11 PCI Express Device Status Register

The PCI Express device status register is shown in Figure 15-53.

Offset 0x056 Access: Mixed

	15–6	5	4	3	2	1	0
R	—	TP	APD	URD	FED	NFED	CED
W				w1c	w1c	w1c	w1c
Reset	All zeros						

Figure 15-53. PCI Express Device Status Register

Table 15-51. PCI Express Device Status Register Fields Description

Bits	Name	Description
15–6	—	Reserved
5	TP	Transactions pending
4	APD	AUX power detected
3	URD	Unsupported request detected
2	FED	Fatal error detected
1	NFED	Non-fatal error detected
0	CED	Correctable error detected

15.4.4.12 PCI Express Link Capabilities Register

The PCI Express link capabilities register is shown in Figure 15-54. Note that for End Point mode, some of these fields can indirectly be set using the PCI Express Link Capabilities Update Register (PEX_LINKCAP_UPDATE). See Section 15.4.6.10, "PCI Express Link Capabilities Update Register (PEX_LINKCAP_UPDATE) for more details.

Offset 0x058 Access: Read-only

	31–24	23–18	17–16
R	Port Number	—	L1_EX_LAT...
W			
Reset	0 0 0 0 0 0 0 0	0 0 0 0 0 0	1 1

	15	14–12	11–10	9–4	3–0
R	...	L0s_EX_LAT	ASPM	MAX_LINK_W	MAX_LINK_SP
W					
Reset	1	1 0 1	0 1	0 0 0 0 1 0	0 0 0 1

Figure 15-54. PCI Express Link Capabilities Register

Table 15-52. PCI Express Link Capabilities Register Fields Description

Bits	Name	Description
31–24	Port Number	
23–18	—	Reserved
17–15	L1_EX_LAT	L1 exit latency. 0b111 indicates more than 64 microseconds
14–12	L0s_EX_LAT	L0s exit latency. 0b011 indicates 256 ns to less than 512 ns
11–10	ASPM	Active state power management (ASPM) Support, L0s Entry Supported
9–4	MAX_LINK_W	Maximum link width 0b000001 x1 0b000010 x2
3–0	MAX_LINK_SP	Maximum link speed, 0b0001 indicates 2.5 Gb/s

15.4.4.13 PCI Express Link Control Register

The PCI Express link control register is shown in Figure 15-55.

Offset 0x05C Access: Read/Write

	15–8	7	6	5	4	3	2	1–0
R/W	—	EXT_SYNC	CCC	RL	LD	RCB	—	ASPM_CTL
Reset	All zeros							

Figure 15-55. PCI Express Link Control Register

Table 15-53. PCI Express Link Control Register Fields Description

Bits	Name	Description
15–8	—	Reserved
7	EXT_SYNC	Extended synch
6	CCC	Common clock configuration
5	RL	Retrain link
4	LD	Link disable
3	RCB	Read completion boundary
2	—	Reserved
1–0	ASPM_CTL	Active state power management (ASPM) control

15.4.4.14 PCI Express Link Status Register

The PCI Express link status register is shown in Figure 15-56.

Offset 0x05E — Access: Read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	—			SCC	LT	—	NEG_LINK_W						LINK_SP			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Figure 15-56. PCI Express Link Status Register

Table 15-54. PCI Express Link Status Register Fields Description

Bits	Name	Description
15–13	—	Reserved
12	SCC	Slot clock configuration
11	LT	Link training
10	—	Reserved.
9–4	NEG_LINK_W	Negotiated link width
3–0	LINK_SP	Link speed

15.4.4.15 PCI Express Slot Capabilities Register

The PCI Express slot capabilities register is shown in Figure 15-57. For End Point applications implementing a slot, the content of this register can be modified by using the PCI Express Slot Capabilities

Update Register as described in Section 15.4.6.11, “PCI Express Slot Capabilities Update Register (PEX_SLCAP_UPDATE)”

Offset 0x060 Access: Read-only

	31–19	18–17	16
R	Physical Slot Number	—	SPLS
W			
Reset	0x0000		

	15	14–7	6	5	4	3	2	1	0
R	—	SPLV	HPC	HPS	PIP	AIP	MRLSP	PCP	ABP
W									
Reset	0x07C0								

Figure 15-57. PCI Express Slot Capabilities Register

Table 15-55. PCI Express Slot Capabilities Register Fields Description

Bits	Name	Description
31–19	Physical Slot Number	This field indicates the physical slot number attached to this Port.
18–17	—	Reserved
16–15	SPLS	Slot power limit scale.
14–17	SPLV	Slot power limit value
6	HPD	Hot plug capable
5	HPS	Hot plug surprise
4	PIP	Power indicator present
3	AIP	Attention indicator present
2	MRLSP	MRL sensor present
1	PCP	Power controller present
0	ABP	Attention button present

15.4.4.16 PCI Express Slot Control Register

The PCI Express slot control register is shown in Figure 15-58.

Offset 0x064 Access: Read/Write

	15–11	10	9–8	7–6	5	4	3	2	1	0
R/W	—	PCC	PIC	AIC	HPIE	CCIE	PDCE	MRLSCE	PFDE	ABPE
Reset	All zeros									

Figure 15-58. PCI Express Slot Control Register

Table 15-56. PCI Express Slot Control Register Fields Description

Bits	Name	Description
15–11	—	Reserved
10	PCC	Power controller control
9–8	PIC	Power indicator control
7–6	AIC	Attention indicator control
5	HPIE	Hot plug interrupt enable
4	CCIE	Command completed interrupt enable
3	PDCE	Presence detect changed enable
2	MRLSCE	MRL sensor changed enable
1	PFDE	Power fault detected enable
0	ABPE	Attention button pressed enable

15.4.4.17 PCI Express Slot Status Register

The PCI Express slot status register is shown in Figure 15-59.

Offset 0x066 Access: Mixed

	15–7	6	5	4	3	2	1	0
R	—	PDS	MRLSS	CC	PDC	MRLSC	PFD	ABP
W				w1c	w1c	w1c	w1c	w1c
Reset	0 0 0 0 0 0 0 0 0	1	0	0	0	0	0	0

Figure 15-59. PCI Express Slot Status Register

Table 15-57. PCI Express Slot Status Register Fields Description

Bits	Name	Description
15–7	—	Reserved
6	PDS	Presence detect state. Indicates whether a card is present in the slot. 0 Slot empty 1 Card is present
5	MRLSS	MRL sensor state 0 MRL closed 1 MRL open
4	CC	Command completed
3	PDC	Presence detect changed
2	MRLSC	MRL sensor changed
1	PFD	Power fault detected
0	ABP	Attention button pressed

15.4.4.18 PCI Express Root Control Register (RC Mode Only)

The PCI Express root control register is shown in Figure 15-60.

Offset 0x068 Access: Read/Write

	15–4	3	2	1	0
R/W	—	PMEIE	SEFEE	SENFEE	SECEE
Reset	All zeros				

Figure 15-60. PCI Express Root Control Register

Table 15-58. PCI Express Root Control Register Fields Description

Bits	Name	Description
15–4	—	Reserved
3	PMEIE	PME interrupt enable.
2	SEFEE	System error on fatal error enable.
1	SENFEE	System error on non-fatal error enable.
0	SECEE	System error on correctable error enable.

15.4.4.19 PCI Express Root Status Register (RC Mode Only)

The PCI Express root status register is shown in Figure 15-61.

Offset 0x06C Access: Mixed

	31–18	17	16
R	—	PMEP	PMES
W			w1c
Reset	All zeros		

	15–0
R	PME Requestor ID
W	
Reset	All zeros

Figure 15-61. PCI Express Root Status Register

Table 15-59. PCI Express Root Status Register Fields Description

Bits	Name	Description
31–18	—	Reserved
17	PMEP	PME pending.
16	PMES	PME status.
15–0	PME Requestor ID	PME requestor ID.

15.4.4.20 PCI Express MSI Message Capability ID Register (EP Mode Only)

The PCI Express MSI message capability ID register is shown in Figure 15-62.

Offset 0x070 Access: Read-only

	7							0
R	MSI Message Capability ID							
W								
Reset	0	0	0	0	0	1	0	1

Figure 15-62. PCI Express Capability ID Register

Table 15-60. PCI Express Capability ID Register Fields Description

Bits	Name	Description
7–0	MSI Message Capability ID	MSI Message = 0x05

NOTE

The value of the Next Pointer register at offset 0x071 is 0x00 (NULL), as this is the last capability of the list.

15.4.4.21 PCI Express MSI Message Control Register (EP Mode Only)

The PCI Express MSI message control register is shown in Figure 15-63.

Offset 0x072 Access: Mixed

	15							8	7	6		4	3		1	0
R	—								64AC	MME			MMC			MSIE
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0

Figure 15-63. PCI Express MSI Message Control Register

Table 15-61. PCI Express MSI Message Control Register Fields Description

Bits	Name	Description
15–8	—	Reserved
7	64AC	64-bit address capable
6–4	MME	Multiple message enable
3–1	MMC	Multiple message capable
0	MSIE	MSI enable

15.4.4.22 PCI Express MSI Message Address Register (EP Mode Only)

The PCI Express MSI message address register is shown in Figure 15-64.

Offset 0x074 Access: Read/Write

	31–2	1–0
R	Message Address	00
W		
Reset	All zeros	

Figure 15-64. PCI Express MSI Message Address Register

Table 15-62. PCI Express MSI Message Address Register Fields Description

Bits	Name	Description
31–2	Message Address	System-specified message address
1–0	00	Always returns 00 on reads; write operations have no effect.

15.4.4.23 PCI Express MSI Message Upper Address Register (EP Mode Only)

The PCI Express MSI message upper address register is shown in Figure 15-65.

Offset 0x078 Access: Read/Write

	31–0
R	Message Upper Address
W	
Reset	All zeros

Figure 15-65. PCI Express MSI Message Upper Address Register

Table 15-63. PCI Express MSI Message Upper Address Register Fields Description

Bits	Name	Description
31–0	Message Upper Address	System-specified message upper address

15.4.4.24 PCI Express MSI Message Data Register (EP Mode Only)

The PCI Express MSI message data register is shown in Figure 15-66.

Offset 0x07C Access: Read/Write

	15–0
R	Message Data
W	
Reset	All zeros

Figure 15-66. PCI Express MSI Message Data Register

Table 15-64. PCI Express MSI Message Data Register Fields Description

Bits	Name	Description
15–0	Message Data	System-specified message

15.4.5 PCI Express Extended Configuration Space

Reserved

Register		Address Offset (Hex)
PCI Compatible Configuration Header (See Section 15.4.1, "Common PCI-Compatible Configuration Header Registers," for more information.)		000–03F
PCI-Compatible Device-Specific Configuration Space (See Section 15.4.4, "PCI Compatible Device-Specific Configuration Space Registers," for more information.)		040–0FF
Next Capability Offset (NULL[1])/Capability Version	Advanced Error Reporting Capability ID	100
Uncorrectable Error Status		104
Uncorrectable Error Mask		108
Uncorrectable Error Severity		10C
Correctable Error Status		110
Correctable Error Mask		114
Advanced Error Capabilities and Control		118
Header Log		11C, 120, 124, 128
Root Error Command		12C
Root Error Status		130
Error Source ID	Correctable Error Source ID	134
		138–3FF
PCI Express Controller Internal CSRs		400–5A3
		5A4–FFF

Figure 15-67. PCI Express Extended Configuration Space

1 Even though the default value of this field is not NULL, it should be considered so by the software.

15.4.5.1 PCI Express Advanced Error Reporting Capability ID Register

The PCI Express advanced error reporting capability ID register is shown in Figure 15-68.

Offset 0x100 | Access: Read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Next Capability Pointer												Capability Version			
W																
Reset	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Capability ID															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 15-68. PCI Express Advanced Error Reporting Capability ID Register

Table 15-65. PCI Express Advanced Error Reporting Capability ID Register Fields Description

Bits	Name	Description
13–20	Next Capability Pointer	**Note:** even though the default value of this field is not NULL, it should be considered so by the software.
19–16	Capability Version	—
15–0	Capability ID	Advanced error reporting capability.

15.4.5.2 PCI Express Uncorrectable Error Status Register

The PCI Express uncorrectable error status register is shown in Figure 15-69. When a particular bit of this status register is set, it indicates that the error has occurred.

Offset 0x104 | Access: w1c

	31–21	20	19	18	17	16
R	—	URE	ECRCE	MTLP	RXO	UC
W		w1c	w1c	w1c	w1c	w1c
Reset	All zeros					

	15	14	13	12	11–5	4	3–1	0
R	CA	CTO	FCPE	PTLP	—	DLPE	—	TE
W	w1c	w1c	w1c	w1c		w1c		w1c
Reset	All zeros							

Figure 15-69. PCI Express Uncorrectable Error Status Register

Table 15-66. PCI Express Uncorrectable Error Status Register Fields Description

Bits	Name	Description
31–21	—	Reserved
20	URE	Unsupported request error status
19	ECRCE	ECRC error status
18	MTLP	Malformed TLP status
17	RXO	Receiver overflow status
16	UC	Unexpected completion status
15	CA	Completer abort status
14	CTO	Completion timeout status
13	FCPE	Flow control protocol error status
12	PTLP	Poisoned TLP status
11–5	—	Reserved
4	DLPE	Data link protocol error status
3–1	—	Reserved
0	TE	Training error status

15.4.5.3 PCI Express Uncorrectable Error Mask Register

The PCI Express uncorrectable error mask register is shown in Figure 15-70. An error is masked if the corresponding mask bit in this register is set.

Offset 0x108 Access: Read/Write

	31–21	20	19	18	17	16
R/W	—	UREM	ECRCEM	MTLPM	RXOM	UCM
Reset	All zeros					

	15	14	13	12	11–5	4	3–1	0
R/W	CAM	CTOM	FCPEM	PTLPM	—	DLPEM	—	TEM
Reset	All zeros							

Figure 15-70. PCI Express Uncorrectable Error Mask Register

Table 15-67. PCI Express Uncorrectable Error Mask Register Fields Description

Bits	Name	Description
31–21	—	Reserved
20	UREM	Unsupported request error mask
19	ECRCEM	ECRC error mask

Table 15-67. PCI Express Uncorrectable Error Mask Register Fields Description (continued)

Bits	Name	Description
18	MTLPM	Malformed TLP mask
17	RXOM	Receiver overflow mask
16	UCM	Unexpected completion mask
15	CAM	Completer abort mask
14	CTOM	Completion timeout mask
13	FCPEM	Flow control protocol error mask
12	PTLPM	Poisoned TLP mask
11–5	—	Reserved
4	DLPEM	Data link protocol error mask
3–1	—	Reserved
0	TEM	Training error mask

15.4.5.4 PCI Express Uncorrectable Error Severity Register

The PCI Express uncorrectable error severity register is shown in Figure 15-71. An error is reported as fatal if the corresponding severity bit is set; otherwise it is reported as non-fatal.

Offset 0x10C | Access: Read/Write

	31										21	20	19	18	17	16
R W	—											URES	ECRCES	MTLPS	RXOS	UCS
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

	15	14	13	12	11						5	4	3		1	0
R W	CAS	CTOS	FCPES	PTLPS	—							DLPES	—			TES
Reset	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 15-71. PCI Express Uncorrectable Error Severity Register

Table 15-68. PCI Express Uncorrectable Error Severity Register Fields Description

Bits	Name	Description
31–21	—	Reserved
20	URES	Unsupported request error severity
19	ECRCES	ECRC error severity
18	MTLPS	Malformed TLP severity
17	RXOS	Receiver overflow severity
16	UCS	Unexpected completion severity

Table 15-68. PCI Express Uncorrectable Error Severity Register Fields Description (continued)

Bits	Name	Description
15	CAS	Completer abort severity
14	CTOS	Completion timeout severity
13	FCPES	Flow control protocol error severity
12	PTLPS	Poisoned TLP severity
11–5	—	Reserved
4	DLPES	Data link protocol error severity
3–1	—	Reserved
0	TES	Training error severity

15.4.5.5 PCI Express Correctable Error Status Register

The PCI Express correctable error status register is shown in Figure 15-72. When an individual error status bit of this read-only register is set, it indicates that this particular error has occurred.

Offset 0x110 Access: w1c

	31–16
R	—
W	
Reset	All zeros

	15–13	12	11–9	8	7	6	5–1	0
R	—	RTTO	—	RNR	BDLLP	BTLP	—	RxE
W		w1c		w1c	w1c	w1c		w1c
Reset	All zeros							

Figure 15-72. PCI Express Correctable Error Status Register

Table 15-69. PCI Express Correctable Error Status Register Fields Description

Bits	Name	Description
31–13	—	Reserved
12	RTTO	Replay timer time-out status
11–9	—	Reserved
8	RNR	REPLAY_NUM rollover status
7	BDLLP	Bad DLLP status
6	BTLP	Bad TLP status
5–1	—	Reserved
0	RXE	Receiver error status

15.4.5.6 PCI Express Correctable Error Mask Register

The PCI Express correctable error mask register is shown in Figure 15-73. An error is masked if the corresponding mask bit in this register is set.

Offset 0x114 — Access: Read/Write

	31–16
R/W	—
Reset	All zeros

	15–13	12	11–9	8	7	6	5–1	0
R/W	—	RTTOM	—	RNRM	BDLLPM	BTLPM	—	RXEM
Reset	All zeros							

Figure 15-73. PCI Express Correctable Error Mask Register

Table 15-70. PCI Express Correctable Error Mask Register Fields Description

Bits	Name	Description
31–13	—	Reserved
12	RTTOM	Replay timer timeout mask
11–9	—	Reserved
8	RNRM	REPLAY_NUM rollover mask
7	BDLLPM	Bad DLLP mask
6	BTLPM	Bad TLP mask
5–1	—	Reserved
0	RXEM	Receiver error mask

15.4.5.7 PCI Express Advanced Error Capabilities and Control Register

The PCI Express advanced error capabilities and control register is shown in Figure 15-74.

Offset 0x118 — Access: Mixed

	31–16
R/W	—
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

	15–9	8	7	6	5	4–0
R	—	ECRCCE	ECRCCC	ECRCGE	ECRCGC	First Error Pointer
W		ECRCCE		ECRCGE		
Reset	0 0 0 0 0 0 0	0	1	0	1	0 0 0 0 0

Figure 15-74. PCI Express Advanced Error Capabilities and Control Register

Table 15-71. PCI Express Advanced Error Capabilities and Control Register Fields Description

Bits	Name	Description
31–9	—	Reserved
8	ECRCCE	ECRC checking enable. Set this bit to enable ECRC checking.
7	ECRCCC	ECRC checking capable. Status bit indicates if this capability has been enabled. 0 ECRC checking capability is disabled. 1 ECRC checking capability is enabled.
6	ECRCGE	ECRC generation enable. Set this bit to enable ECRC generation.
5	ECRCGC	ECRC generation capable. Status bit indicates if this capability has been enabled. 0 ECRC generation capability is disabled. 1 ECRC generation capability is enabled.
4–0	First Error Pointer	First error pointer. Identifies the bit position of the first error reported in uncorrectable error status register, Section 15.4.5.2, “PCI Express Uncorrectable Error Status Register.”

15.4.5.8 PCI Express Header Log Register

The PCI Express header log register is shown in Figure 15-75.

Offset 0x11C — Access: Read-only

	31–24	23–16	15–8	7–0
R	Byte 0	Byte 1	Byte 2	Byte 3
W				
Reset	All zeros			

Offset 0x120 — Access: Read-only

	31–24	23–16	15–8	7–0
R	Byte 4	Byte 5	Byte 6	Byte 7
W				
Reset	All zeros			

Offset 0x124 — Access: Read-only

	31–24	23–16	15–8	7–0
R	Byte 8	Byte 9	Byte A	Byte B
W				
Reset	All zeros			

Offset 0x128 — Access: Read-only

	31–24	23–16	15–8	7–0
R	Byte C	Byte D	Byte E	Byte F
W				
Reset	All zeros			

Figure 15-75. PCI Express Header Log Register

Table 15-72. PCI Express Header Log Register Fields Description

Bits	Name	Description
127–0	Header Log	Header of TLP associated with error.

15.4.5.9 PCI Express Root Error Command Register

The PCI Express root error command register is shown in Figure 15-76.

Offset 0x12C — Access: Read/Write

Bits	31–16
R	—
W	
Reset	All zeros

Bits	15–3	2	1	0
R	—	FERE	NFERE	CERE
W				
Reset	All zeros			

Figure 15-76. PCI Express Root Error Command Register

Table 15-73. PCI Express Root Error Command Register Fields Description

Bits	Name	Description
31–3	—	Reserved
2	FERE	Fatal error reporting enable.
1	NFERE	Non-fatal error reporting enable
0	CERE	Correctable error reporting enable

15.4.5.10 PCI Express Root Error Status Register

The PCI Express root error status register is shown in Figure 15-77.

Offset 0x130 — Access: Mixed

Bits	31–27	26–16
R	AEIMN	—
W		
Reset	All zeros	

Bits	15–7	6	5	4	3	2	1	0
R	—	FEMR	NFEMR	FUF	MEFNFR	EFNFR	MECR	ECR
W		w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros							

Figure 15-77. PCI Express Root Error Command Register

Table 15-74. PCI Express Root Error Command Register Fields Description

Bits	Name	Description
31–27	AEIMN	Advanced error interrupt message number.
26–7	—	Reserved

Table 15-74. PCI Express Root Error Command Register Fields Description (continued)

Bits	Name	Description
6	FEMR	Fatal error messages received.
5	NFEMR	Non-fatal error messages received.
4	FUF	First uncorrectable fatal.
3	MEFNFR	Multiple ERR_FATAL/NONFATAL received.
2	EFNFR	ERR_FATAL/NONFATAL received.
1	MECR	Multiple ERR_COR received.
0	ECR	ERR_COR received.

15.4.5.11 PCI Express Error Source Identification Register

The Error Source Identification register shown in Figure 15-78 identifies the source (Requestor ID) of first correctable and uncorrectable (Non-fatal/Fatal) errors reported in the Root Error Status register. This register is relevant for RC only.

Offset 0x134 Access: Read-only

	31–16	15–0
R	ERR_FATAL/NONFATAL Source ID	ERR_COR Source ID
W		
Reset	All zeros	

Figure 15-78. PCI Express Error Source Identification Register

Table 15-75. PCI Express Error Source Identification Register Fields Description

Bits	Name	Description
31–16	ERR_FATAL/NONFATAL Source ID	ERR_FATAL/NONFATAL Source Identification. Loaded with the Requestor ID indicated in the received ERR_FATAL or ERR_NONFATAL Message when the ERR_FATAL/NONFATAL Received register is not already set.
15–0	ERR_COR Source ID	Correctable Error Source Identification. Loaded with the Requestor ID indicated in the received ERR_COR Message when the ERR_COR Received register is not already set.

15.4.6 PCI Express Controller Internal Control and Status Registers (CSRs)

15.4.6.1 PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)

PEX_LTSSM_STAT, shown in Figure 15-79, provides details on link training status. This register is useful for debugging link training failures.

Offset 0x404 Access: Read-only

	31–7	6–0
R	—	Status Code
W		
Reset	All zeros	

Figure 15-79. PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)

The fields of the PEX_LTSSM_STAT are described in Table 15-76.

Table 15-76. PEX_LTSSM_STAT Fields Description

Bits	Name	Description
31–7	—	Reserved
6–0	Status code	Status code. See Table 15-77 for encodings.

Table 15-77 provides the encodings for the status code field of the PEX_LTSSM_STAT register.

Table 15-77. PEX_LTSSM_STAT Status Codes

Status Code (Hex)	LTSSM State Description	Status Code (Hex)	LTSSM State Description
00	Detect quiet	27	TX L0s FTS; RX L0s FTS
01	Detect active (0)	28	L0 to L1 (0)
02	Detect active (1)	29	L0 to L1 (1)
03	Detect active (2)	2A	L1 entry
04	Polling active (0)	2B	L1 idle (0)
05	Polling active (1)	2C	L1 idle (1)
06	Polling config (0)	2D	L0 to L2 (0)
07	Polling config (1)	2E	L0 to L2 (1)
08	Polling compliance	2F	L2 entry
09	Configuration link width start (0)	30	L2 idle (0)
0A	Configuration link width start (1)	31	L2 idle (1)
0B	Configuration link width accept (0)	32	Recovery lock (0)
0C	Configuration link width accept (1)	33	Recovery lock (1)
0D	Configuration lane number wait (0)	34	Recovery lock (2)
0E	Configuration lane number wait (1)	35	Recovery cfg (0)
0F	Configuration lane number wait (2)	36	Recovery cfg (1)
10	Configuration lane number wait (3)	37	Recovery idle (0)

Table 15-77. PEX_LTSSM_STAT Status Codes (continued)

Status Code (Hex)	LTSSM State Description	Status Code (Hex)	LTSSM State Description
11	Configuration lane number accept	38	Recovery idle (1)
12	Configuration complete (0)	39	Recovery to configuration
13	Configuration complete (1)	3A	Recovery cfg to configuration
14	Configuration idle (0)	3F	L0 no training
15	Configuration idle (1)	7F	Detect quiet EI
16	L0	49	Configuration link width start—RC
17	TX L0; RX L0s entry	4A	Configuration link width accept—RC
18	TX L0; RX L0s idle	4B	Configuration lane number wait—RC
19	TX L0; RX L0s fast training sequence (FTS)	4C	Configuration lane number accept—RC
1A	TX L0s entry (0); RX L0	60	Loopback slave active (0)
1B	TX L0s entry (0); RX L0s idle	61	Loopback slave active (1)
1C	TX L0s entry (0); RX L0s FTS	62	Loopback slave exit
1D	TX L0s entry (1); RX L0	68	Hot reset (0)
1E	TX L0s entry (1); RX L0s idle	69	Hot reset (1)
1F	TX L0s entry (1); RX L0s FTS	6A	Hot reset (0)—RC
20	TX L0s idle; RX L0	6B	Hot reset (1)—RC
21	TX L0s idle; RX L0s entry	75	Disabled (0)
22	TX L0s idle; RX L0s idle	71	Disabled (1)
23	TX L0s idle; RX L0s FTS	72	Disabled (2)
24	TX L0s FTS; RX L0	73	Disabled (3)
25	TX L0s FTS; RX L0s entry	74	Disabled (4)
26	TX L0s FTS; RX L0s idle	78	L0 to L1/L2—RC

15.4.6.2 PCI Express N_FTS Control Register (PEX_NFTS_CTRL)

The PCI Express N_FTS Control Register, shown in Figure 15-80, is used to set the N_FTS value that is advertised by the PCI Express controller during link training. It is preferable to set it before the PCI Express core internal reset is negated. If this value is changed after the link is up, the new value will take effect during the next link training. The N_FTS value is should be set according to the L0s exit latency of the Rx link of PHY. At a given time, either N_FTS or N_FTS_COM value is used based on the setting of common clock configuration bit in the configuration space.

The fields of the PCI Express N_FTS Control Register are described in Table 15-78.

Offset 0x41C | Access: R/W

	31–16	15–8	7–0
R/W	–	N_FTS_COM	N_FTS
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0	0 1 0 0 0 0 0 0

Figure 15-80. PCI Express N_FTS Control Register

Table 15-78. PCI Express N_FTS Control Register Fields Description

Bits	Name	Description
31–16	–	Reserved
15–8	N_FTS_COM	Number of Fast Training Sequence ordered sets in common clock mode. This is the number of Fast Training Sequence (FTS) ordered sets that PHY requires to enable its Rx circuits to achieve bit and symbol lock and come out of ASPM L0s link power state when devices on either side of the link use common reference clock. This N_FTS value is advertised by the PCI Express controller to the remote device during link training if the common clock configuration bit in configuration space is set.
7–0	N_FTS	Number of Fast Training Sequence ordered sets. This is the number of Fast Training Sequence (FTS) ordered sets that PHY requires to enable its Rx circuits to achieve bit and symbol lock and come out of ASPM L0s link power state. This N_FTS value is advertised by the PCI Express controller to the remote device during link training. Its value has to be calculated based on the L0s_exit latency time required by the PHY layer circuits.

15.4.6.3 PCI Express ACK Replay Timeout Register (PEX_ACKRPLY_TO)

The PCI Express ACK Replay Timeout Register, shown in Figure 15-81, is used to program timeout values for ACK DLLP transmission and reception in the DLL. Ack receive timeout is termed as Replay timeout since TLPs are re-transmitted after this timeout. Both values should be in terms of system clock cycles number and have to be set based on Max-Payload-size and operating link width as specified by the protocol.

Ack time-out will also depend upon ASPM L0s enabling for the Tx link of the device. The PCI Express controller implements a look-up table for automatic updates of the ack and replay time-out values, based on max-payload size, link_width and ASPM L0s enabled information. There for the reset value of this register may be different that shown in the figure. The automatic updating of these values will be disabled upon the first write to this register assuming that the software will take care of the updates based on the above factors from then on.

The fields of the PCI Express ACK Replay Timeout Register are described in Table 15-79.

Offset 0x438 Access: R/W

	31–27	26–13	12–0
R W	–	ACKRTV	ACKLTV
Reset	0x00c2415c		

Figure 15-81. PCI Express ACK Replay Timeout Register

Table 15-79. PCI Express ACK Replay Timeout Register Fields Description

Bits	Name	Description
31–27	–	Reserved
26–13	ACKRTV	Ack Replay Timeout Value. Timeout value to wait for reception of ACK DLLP from the link side by the DLL before re-transmitting TLPs. The protocol specifies this value in symbol times for various combinations of max-payload size and negotiated link width. The value programmed into this field should be in terms of system clock cycles number, and can be calculated as: $\frac{(REPLAY\_TIMER\_TIEMOUT + Rx\_L0s\_Adjustment) \times SYSTEM\_CLOCK}{250}$ • REPLAY_TIMER_TIMEOUT – Tiemout value for the replay timer, specified in symbol times. • Rx_L0s_Adjustment – The time required by the component’s receive circuits to exit from L0s to L0 specified in symbol times. • SYSTEM_CLOCK – The PCI Express controller system clock, specified in MHz. **Note:** The “250” denominator represent the frequency of a symbol (250 MHz).
12–0	ACKLTV	Ack Latency Timeout Value. Timeout value to force transmission of ACK DLLP by the DLL after a TLP is received. The protocol specifies this value in symbol times for various combinations of max-payload size & negotiated link width. The value programmed into this field should be in terms of system clock cycles number, and can be calculated as: $\frac{(ACK\_LATENCY\_TIEMOUT + Tx\_L0s\_Adjustment) \times SYSTEM\_CLOCK}{250}$ • ACK_LATENCY_TIMEOUT – Tiemout value to force transmission of ACK DLLP, specified in symbol times. • Tx_L0s_Adjustment – The time required for the Transmitter to exit L0s, specified in symbol times. • SYSTEM_CLOCK – The PCI Express controller system clock, specified in MHz. **Note:** The “250” denominator represent the frequency of a symbol (250 MHz).

15.4.6.4 PCI Express Controller Core Clock Ratio Register (PEX_GCLK_RATIO)

The PCI Express Controller Core Clock Ratio Register, shown in Figure 15-82, is used to program the ratio of the actual PCI Express controller core clock (csb_clk divided according to SCCR[PCIEXP*n*CM]) frequency to the maximum possible controller core frequency (400 MHz). Changing the default value of this register is required only when a PCI Express controller core clock frequency is different than its maximum.This ratio will be used by the PCI Express controller only to calculate the actual timer values to be used for Ack Latency and Replay timeout values. These two timer values have to dynamically change based on the negotiated link-width and max-payload size. The default value by itself is not enough for these two timers. By programming the clock ratio in this register, the calculation is automatically adjusted by hardware. Note that other timer registers in the PCI Express controller may still have to be programmed to a new value based on the actual controller core clock used in the specific application.

NOTE

The default PCI Express controller core clock is csb_clk/3 (SCCR[PCIEXPnCM] = 0b11). The user should program this register according to the actual application CSB frequency and SCCR[PCIEXPnCM] value.

The fields of PEX_GCLK_RATIO are described in Table 15-80.

Offset 0x440 — Access: Mixed

	Bits 31–6	Bits 5–0
R/W	—	CLKRN
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 1 0 0 0 0

Figure 15-82. PCI Express Core Clock Ratio Register (PEX_GCLK_RATIO)

Table 15-80. PEX_GCLK_RATIO Fields Description

Bits	Name	Description
31–6	—	Reserved
5–0	CLKRN	Clock Ratio Numerator. The numerator of the ratio of the actual PCI Express controller core clock used to the maximal core clock of 400 MHz. These bits should be programmed only when the PCI Express controller is required to run at a clock frequency other than the maximum. The denominator of the ratio is fixed at 16. The default value of this register is 0x10 (16 decimal), which corresponds to a ratio of 1:1 (or 16/16). For example, if the PCI Express controller clock is 200 MHz, the ratio of the clocks is 1:2 (equivalent to 8/16). Therefore these bits should be programmed to a value of 8, or 0x0000_0008.

15.4.6.5 PCI Express Power Management Timer Register (PEX_PM_TIMER)

PEX_PM_TIMER, shown in Figure 15-83, is used to program the time-in values for entering L0s and L1 power management states.

Offset 0x450 — Access: Mixed

	Bits 31–24	Bits 23–12	Bits 11–0
R	—	L1_WAIT_PERIOD	L0s_TIME_IN
W			
Reset	0 0 0 0 0 0 0 0	0 0 0 1 1 0 0 1 0 0 0 0	1 0 0 1 0 1 1 0 0 0 0 0

Figure 15-83. PCI Express Power Management Timer Register (PEX_PM_TIMER)

The fields of the PEX_PM_TIMER are described in Table 15-81.

Table 15-81. PEX_PM_TIMER Fields Description

Bits	Name	Description
31–24	—	Reserved
23–12	L1_WAIT_PERIOD	Wait period (in PCI Express controller core clock cycles) before entering L1 power state after all functions are in a non-D0 power state. The value is calculated as: Time (in μsec) × PCI Express controller core clock frequency (in MHz) The time value must be less than 2 μsec; the default value (0x14D) is 1 μsec for the default clock frequency of 400 MHz.
11–0	L0s_TIME_IN	Time in value (in PCI Express controller core clock cycles) for entering L0s power state. The value is calculated as: Time (in μsec) × PCI Express controller core clock frequency (in MHz) The maximum time value is 7 μsec; the default value (0x7CE) is 6 μsec for the default clock frequency of 400 MHz.

15.4.6.6 PCI Express PME Time-Out Register (PEX_PME_TIMEOUT) (EP Mode Only)

PEX_PME_TIMEOUT, shown in Figure 15-84, is used to program the time-out value that the controller uses before re-sending a PME message to the host. If PME is requested by a function and the host does not clear the associated PME_STAT bit even after this time-out has expired, the PME message is sent again to the host by the PCI Express controller. This register is supported only for EP mode.

Offset 0x454 (EP mode only) Access: Mixed

	31–26	25–0
R/W	—	PME_TIMEOUT

Bits	31–28	27–24	23–20	19–16	15–12	11–8	7–4	3–0
Reset	0 0 0 0	0 0 1 0	0 1 1 0	0 0 1 0	0 1 0 1	1 0 1 0	0 0 0 0	0 0 0 0

Figure 15-84. PCI Express PME Time-Out Register (PEX_PME_TIMEOUT)

The fields of PEX_PME_TIMEOUT are described in Table 15-81.

Table 15-82. PEX_PME_TIMEOUT Fields Description

Bits	Name	Description
31–26	—	Reserved
25–0	PME_TIMEOUT	The interval before PME messages are resent by the controller if the PME_STAT bit in the PCI Express power management status and control register (offset 0x048) is not cleared by the host. The value for PME_TIMEOUT is specified in terms of PCI Express controller core clock cycles. The value is calculated as: Time (in μsec) × PCI Express controller core clock frequency (in MHz) The minimum time value is 100 msec; the default value (0x1FC1E20) is 100 msec for the default clock frequency of 400 MHz.

15.4.6.7 PCI Express ASPM Request Timer Register (PEX_ASPM_REQTMR)

The PCI Express ASPM request timer register (PEX_ASPM_REQTMR) shown in Figure 15-85 is used to program the time interval between two ASPM L1 entry requests from a downstream device before deciding that the second request is a new request. This timer value is required because if the upstream port rejects an ASPM L1 entry request by sending a NAK message, the L1 request DLLPs continue to be received for some more time due to the link and processing latency involved before downstream device processes the NAK msg and stops the request. Meanwhile, these request DLLPs should not be considered as a new L1 entry request and responded with Ack DLLP. For more details refer to the PCI Express Base Specification Rev 1.0a errata, C7. ASPM & PCI-PM L1.

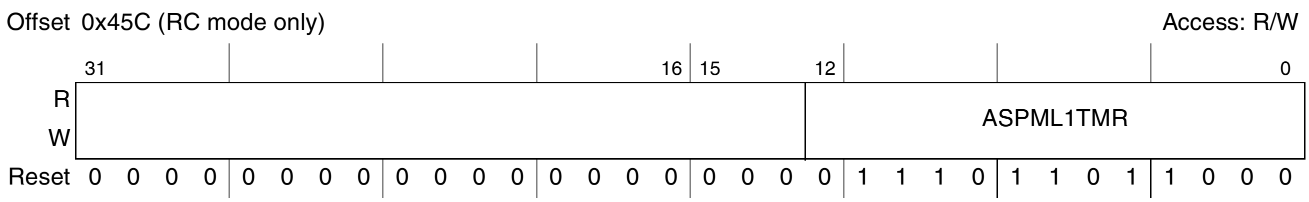


Figure 15-85. PCI Express ASPM Request Timer Register

The fields of the PCI Express ASPM request timer register are described in Table 15-83.

Table 15-83. PCI Express ASPM Request Timer Register Fields Description

Bits	Name	Description
31–13	–	Reserved
12–0	ASPML1TMR	ASPM L1 request timer value. This is the time-out interval after sending NAK message, before a new ASPM L1 request from a downstream device is treated as a new ASPM L1 entry request. E.g.: If the upstream device rejects an ASPM L1 entry request from a downstream device with ASPM NAK message, the next ASPM L1 entry request from downstream device will be entertained only after this timeout interval or only after the Rx link of the downstream port enters L0s state. This value is specified in terms of system clock cycles (CSB clock / PCIEXP*n*CM). The default value is equivalent to 9.5us in a 400MHz system clock. This value can be calculated as [Time in microseconds * SYSTEM_CLK in MHz]. For example 9.5[us]*400[MHz] = 3800 (0xED8). This register is used only in RC mode.

15.4.6.8 PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE)

The PCI Express subsystem vendor ID update register (PEX_SSVID_UPDATE) shown in Figure 15-86 is used to configure subsystem vendor ID and subsystem ID fields of configuration header (offset 0x2C) for End Point devices. This register has to be programmed before setting the config-ready bit in the PCI Express configuration ready register so that the host reads the correct information during enumeration.

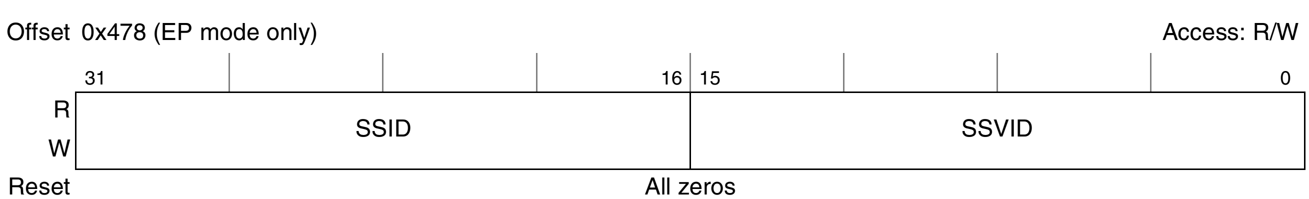


Figure 15-86. PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE)

Table 15-84. PEX_SSVID_UPDATE Fields Description

Bits	Name	Description
31–16	SSID	Subsystem ID
15–0	SSVID	Subsystem Vendor ID

15.4.6.9 PCI Express Device Capabilities Update Register (PEX_DEVCAP_UPDATE)

The PCI Express device capabilities update register shown in Figure 15-87 is used to set the values to the PCI Express device capabilities register in the PCI Express configuration header (offset 0x50). It can be used when the device is configured as an End Point to make the correct device information available to the upstream device. This register has to be programmed before setting the config-ready bit in the PCI Express configuration ready register so that the host reads the correct information during enumeration.

Offset 0x47C — Access: R/W

Bits	31–15	14	13	12	11–9	8–6	5	4–3	2–0
R/W	—	PIP	AIP	APB	L1AL	L0SAL			MPLS
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0	0 0 0	0	0 0	0 0 0

Figure 15-87. PCI Express Device Capabilities Update Register

The fields of the PCI Express device capabilities update register are described in Table 15-85.

Table 15-85. PCI Express Device Capabilities Update Register Fields Description

Bits	Name	Description
31–15	—	Reserved
14	PIP	Power Indicator Present
13	AIP	Attention Indicator Present
12	APB	Attention Button Present
11–9	L1AL	Endpoint L1 Acceptable Latency. This field indicates the acceptable latency that an Endpoint can withstand due to the transition from L1 state to the L0 state. Defined encodings are: 000b Less than 1us 001b 1 us to less than 2 us 010b 2 us to less than 4 us 011b 4 us to less than 8 us 100b 8 us to less than 16 us 101b 16 us to less than 32 us 110b 32 us-64 ìs 111b More than 64 us

Table 15-85. PCI Express Device Capabilities Update Register Fields Description (continued)

Bits	Name	Description
8–6	L0SAL	Endpoint L0s Acceptable Latency. This field indicates the acceptable total latency that an Endpoint can withstand due to the transition from L0s state to the L0 state. Defined encodings are: 000b Less than 64 ns 001b 64 ns to less than 128 ns 010b 128 ns to less than 256 ns 011b 256 ns to less than 512 ns 100b 512 ns to less than 1 us 101b 1 us to less than 2 us 110b 2 us-4 us 111b More than 4 us
5	—	Reserved (Extended Tag Field Supported). Must be set to 0b.
4–3	—	Reserved (Phantom Functions Supported). Must be set to 00b.
2–0	MPLS	Max Payload Size Supported. Must be set to 000b (128bytes)

15.4.6.10 PCI Express Link Capabilities Update Register (PEX_LINKCAP_UPDATE)

The PCI Express link capabilities update register shown in Figure 15-88 is used to set the values to the PCI Express link capabilities register in the PCI Express configuration header (offset 0x58). It can be used when the device is configured as an End Point to make the correct link information available to the upstream device. This register has to be programmed before setting the config-ready bit in the PCI Express configuration ready register so that the host reads the correct information during enumeration.

Offset 0x480 Access: R/W

Bits	31–14	13–11	10–8	7–6	5–0
R/W	—	L1EXL	L0SEXL	ASPM	MLW
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 1 1	1 0 1	0 1	0 0 0 0 1 0

Figure 15-88. PCI Express Link Capabilities Update Register

The fields of the PCI Express link capabilities update register are described in Table 15-86.

Table 15-86. PCI Express Link Capabilities Update Register Fields Description

Bits	Name	Description
31–14	–	Reserved
13–11	L1EXL	L1 Exit Latency for the given PCI Express Link. The value reported indicates the length of time this port requires to complete transition from L1 to L0. Defined encodings are: 000b Less than 1us 001b 1us to less than 2us 010b 2us to less than 4us 011b 4us to less than 8us 100b 8us to less than 16us 101b 16us to less than 32us 110b 32us to 64us 111b More than 64us **Note:** Exit latencies may be influenced by PCI-Express reference clock configuration depending upon whether a component uses a common or separate reference clock.
10–8	L0SEXL	L0s Exit Latency for the given PCI Express Link. The value reported indicates the length of time this port requires to complete transition from L0s to L0. Defined encodings are: 000b Less than 64ns 001b 64ns to less than 128ns 010b 128ns to less than 256ns 011b 256ns to less than 512ns 100b 512ns to less than 1us 101b 1us to less than 2us 110b 2us to 4us 111b More than 4us **Note:** Exit latencies may be influenced by PCI-Express reference clock configuration depending upon whether a component uses a common or separate reference clock. This field is automatically updated to a new value if the common clock configuration bit in config space is set by host.
7–6	ASPM	ASPM Support - Indicates the level of ASPM supported on the given PCI Express Link. Defined encodings are: 00b Reserved 01b L0s Entry Supported 10b Reserved 11b Reserved (L0s and L1 not supported by this device)
5–0	MLW	Maximum Link Width of the given PCI Express Link. Defined encodings are: 000001b x1 000010b x2 Other: Reserved

15.4.6.11 PCI Express Slot Capabilities Update Register (PEX_SLCAP_UPDATE)

The PCI Express slot capabilities update register shown in Figure 15-89 is used to set the values to the PCI Express slot capabilities register in the PCI Express configuration header (offset 0x60). It can be used when the device is configured as an End Point to make the correct slot information available to the upstream device. This register has to be programmed before setting the config-ready bit in the PCI Express configuration ready register so that the host reads the correct information during enumeration.

Offset 0x490 Access: R/W

Bits	31–30	29–17	16–15	14–7	6	5	4	3	2	1	0
R	—	PSN	SPLS	SPLV	HPC	HPS	PIP	AIP	MRL	PCP	ABP
W											
Reset	0 0	0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0 1 1 1 1 1	0	0	0	0	0	0	0

Figure 15-89. PCI Express Slot Capabilities Update Register

Table 15-87. PCI Express Slot Capabilities Update Register Fields Description

Bits	Name	Description
31–30	—	Reserved
29–17	PSN	Physical Slot Number. Physical Slot number attached to the port.
16–15	SPLS	Slot Power Limit Scale. Specifies the scale used for the Slot Power Limit Value. Range of Values: 00 1.0x 01 0.1x 10 0.01x 11 0.001x
14–7	SPLV	Slot power limit value. In combination with the Slot Power Limit Scale value, specifies the upper limit on power supplied by slot. Power limit (in Watts) calculated by multiplying the value in this field by the value in the Slot Power Limit Scale field.
6	HPC	Hot plug capable
5	HPS	Hot plug surprise
4	PIP	Power indicator present
3	AIP	Attention indicator present
2	MRL	MRL sensor present
1	PCP	Power controller present
0	ABP	Attention button present

15.4.6.12 PCI Express Configuration Ready Register

The PCI Express configuration ready register, shown in Figure 15-90, indicates configuration complete status to the transaction layer. The transaction layer handles configuration requests from external hosts only after the CFG_READY bit is set. All the configuration requests received from external hosts before the CFG_READY bit is set are completed with configuration request retry status (CRS). To ensure that the external host reads the correct capability advertisements during enumeration, the CFG_READY bit in this register should be set only after all relevant configuration registers are programmed.

Offset 0x4B0 Access: Mixed

Bits	31–1	0
R	—	CFG_READY
W		
Reset	All zeros	

Figure 15-90. PCI Express Configuration Ready Register (PEX_CFG_READY)

The fields of the PCI Express configuration ready register are described in Table 15-88.

Table 15-88. PEX_CFG_READY Fields Description

Bits	Name	Description
31–1	—	Reserved
0	CFG_READY	Configuration ready 1 The transaction layer accepts inbound configuration requests. 0 The transaction layer responds to all inbound configuration requests with retry (CRS)

15.4.7 PCI Express BAR Configuration Registers (EP Mode)

The registers described in this section configure the size and the prefetchable attributes of the base address registers (BAR) in the PCI Express configuration space. The local host should program these attributes before setting the config-ready bit in the PCI Express configuration space so that the RC host reads the correct information during enumeration. These registers are used only in EP mode.

15.4.7.1 PCI Express BAR Size Low Configuration Register (PEX_BAR_SIZEL)

PEX_BAR_SIZEL, shown in Figure 15-91, configures the size of the 32-bit address windows and low portion of the 64-bit address windows by setting the mask value for the base address registers (BAR) in the PCI Express configuration space. The specific PCI Express BAR is selected by the PCI Express BAR Select Configuration Register (PEX_BAR_SEL). Before programming this register, the PEX_BAR_SEL register should be programmed to select the correct BAR, whose size needs to be set. This register is used only in EP mode.

NOTE

Because the CSB address space of the device is limited to 4 Gigabytes, the high portion of the 64-bit address window mask value is always set to all ones (0xFFFF_FFFF) to comply with the standard enumeration sequence.

Offset 0x4D8 Access: Mixed

Bits	31–12	11–0
R/W	MASK	—
Reset	1111_1100_0000_0000_0000	0000_0000_0000

Reset: 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 0 0 0 0 0 0

Figure 15-91. PCI Express BAR Size Low Configuration Register (PEX_BAR_SIZEL)

The fields of the PEX_BAR_SIZEL register are described in Table 15-89.

Table 15-89. PEX_BAR_SIZEL Fields Description

Bits	Name	Description
31–12	MASK	Mask. Sets the mask for the BAR, and any bit with a value of zero is masked. When the RC does a configuration write to the BAR during the enumeration sequence, bits that are masked cannot be modified and remain zeros. All ones and zeros in this register must be consecutive. The actual size is according to the location of the least significant bit in the MASK[31:12] field, which is set. If MASK[31:m] is all ones, the size is 2^m bytes. If MASK[31:12] is all zeros, the window size is 4 Gigabytes. For example: 1111...1111 - 2^{12}, 4 Kilobytes window. 1111...1110 - 2^{13}, 8 Kilobytes window. ... 1100...0000 - 2^{30}, 1 Gigabytes window. 1000...0000 - 2^{31}, 2Gigabytes window. 0000...0000 - 4 Gigabytes window.
11–0	—	Reserved. Must be zeros

15.4.7.2 PCI Express BAR Select Configuration Register (PEX_BAR_SEL)

(PEX_BAR_SEL, shown in Figure 15-91, is used to select the specific BAR for which the size is being configured by the PEX_BAR_SIZEL register. This register should be programmed before the PEX_BAR_SIZEL register is accessed, and it is used only in EP mode.

Offset 0x4E0 Access: Mixed

	31–2	1–0
R/W	—	SEL
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	0 0

Figure 15-92. PCI Express BAR Select Configuration Register (PEX_BAR_SEL)

The fields of the PEX_BAR_SEL register are described in Table 15-90.

Table 15-90. PEX_BAR_SEL Fields Description

Bits	Name	Description
31–2	—	Reserved. Must be zeros.
1–0	SEL	Select. Selects the specific BAR size to be programmed by the PEX_BAR_SIZEL and PEX_BAR_SIZEH registers. 00 PEX_BAR_SIZEL points to BAR0 in address 0x010 (Window 0, 32 bit address). 01 PEX_BAR_SIZEL points to BAR1 in address 0x014 (Window 1, 32 bit address). 10 PEX_BAR_SIZEL points to BAR2 in address 0x018 (low portion of window 2, 64 bit address). 11 PEX_BAR_SIZEL points to BAR4 in address 0x020 (low portion of window 3, 64 bit address).

15.4.7.3 PCI Express BAR Prefetch Configuration Register (PEX_BAR_PF)

PEX_BAR_PF, shown in Figure 15-93, sets the Prefetchable field in the base address registers (BAR) of the PCI Express configuration space. The local host should program this register before setting the config-ready bit in the PCI Express c5onfiguration space so that the RC host reads the correct information during enumeration. It is used only in EP mode.

Offset 0x504 Access: Read/Write

	31–4	3	2	1	0
R/W	—	PF			
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0	0	0	0	0

Figure 15-93. PCI Express BAR Prefetch Configuration Register (PEX_BAR_PF)

The fields of the PEX_BAR_PF register are described in Table 15-91.

Table 15-91. PEX_BAR_PF Fields Description

Bits	Name	Description
31–4	—	Reserved. Must be zeros.
3–0	PF	Prefetchable. Sets the prefetchable attribute for all the BARs in the PCI Express configuration space. Each bit determines the prefetchable attribute of the corresponding BAR. PF[3] - Prefetchable attribute of BAR4 in address 0x020 (low portion of window 3, 64 bit address) PF[2] - Prefetchable attribute of BAR2 in address 0x018 (low portion of window 2, 64 bit address) PF[1] - Prefetchable attribute of BAR1 in address 0x014 (Window 1, 32 bit address) PF[0] - Prefetchable attribute of BAR0 in address 0x010 (Window 0, 32 bit address)

15.4.8 PCI Express Extended Status and Control Registers

15.4.8.1 PME_To_Ack Timeout Register (RC Mode Only)

PEX_PME_TO_ACK_TOR, shown in Figure 15-94, is used to program the timeout value for a PME_To_Ack message response in terms of PCI Express controller core clock cycles.

Offset 0x590 (RC mode only) Access: Mixed

	31–22	21–0
R	—	PME_TO_ACK_TIMEOUT
W		
Reset	0x02625a00	

Figure 15-94. PCI Express PME_To_Ack Timeout Register (PEX_PME_TO_ACK_TOR)

The fields of the PEX_PME_TO_ACK_TOR are described in Table 15-92.

Table 15-92. PEX_PME_TO_ACK_TOR Fields Description

Bits	Name	Description
31–22	—	Reserved
21–0	PME_TO_ACK_ TIMEOUT	After the RC broadcasts a PME_Turn_Off message, the power management module waits for the duration of the PME_To_Ack timeout interval to receive a PME_To_Ack message from the downstream device. If the Ack message is not received within this interval, the power manager indicates that it is safe to switch off power because a timeout has occurred. The value is calculated as: Time (in µsec) × PCI Express controller core clock frequency (in MHz) The recommended timeout duration is 1 msec to 10 msec to make sure that the downstream devices get enough time to prepare for power-off condition.

15.4.8.2 PME_To_Ack Status Register (RC Mode Only)

The PME_To_Ack Status Register (PEX_PME_TO_ACK_SR) shown in Figure 15-95, can be used by the Power Manager software to decide whether it is safe to switch off power to downstream devices, after PME_Turn_Off message has been broadcast by the Root Port.

Offset 0x594 Access: w1c

	31–3	2	1	0
R	—	PTACKTO	L2L3RDY	PTACKMR
W		w1c	w1c	w1c
Reset	All zeros			

Figure 15-95. PME_To_Ack Status Register (PEX_PME_TO_ACK_SR)

The fields of the PEX_PME_TO_ACK_SR are described in Table 15-93.

Table 15-93. PEX_PME_TO_ACK_SR Fields Description

Bits	Name	Description
31–3	—	Reserved
2	PTACKTO	PME_To_Ack timeout occurred. This bit is set by the hardware when PME_To_Ack message is not received by the Root Port from downstream device within the timeout duration indicated by PME_To_ACK_timeout register. When this bit is set, it is safe for the Power manager to switch off the power of the downstream device. Once set, this bit will remain set, till software clears it by writing 1’b1 to this bit.
1	L2L3RDY	Entered L2/L3 ready state. This bit is set by hardware when the current power management state is L2/L3 Ready. 100nsec after this bit is set, it is safe for the Power manager to switch off the power of the downstream device. Once set, this bit will remain set, till software clears it by writing 1’b1 to this bit.
0	PTACKMR	PME_To_Ack message received. This bit is set by hardware when PME_To_Ack message is received by the Root Port from the downstream device. When this bit is set, it is safe for the Power manager to switch off the power of the downstream device. Once set, this bit will remain set, till software clears it by writing 1’b1 to this bit.

15.4.8.3 Secondary Status Interrupt Mask Register (PEX_SS_INTR_MASK) (RC Mode Only)

PEX_SS_INTR_MASK, shown in Figure 15-96, can be used to disable sideband interrupt generation when error bits in the PCI Express secondary status register are set. See Section 15.4.3.7, "PCI Express Secondary Status Register (RC Mode Only)." By default, interrupt generation due to secondary status errors is disabled.

Offset 0x5A0 (RC mode only) Access: Mixed

	31–6	5	4	3	2	1	0
R	—	M_DPE	M_SSE	M_RMA	M_RTA	M_STA	M_MDPE
W							
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	1	1	1	1

Figure 15-96. PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK)

The fields of the PEX_SS_INTR_MASK are described in Table 15-94.

Table 15-94. PEX_SS_INTR_MASK Fields Description

Bits	Name	Description
31–6	—	Reserved
5	M_DPE	Mask detected parity error
4	M_SSE	Mask signaled system error
3	M_RMA	Mask received master abort
2	M_RTA	Mask received target abort
1	M_STA	Mask signaled target abort
0	M_MDPE	Mask master data parity error

15.5 PCI Express CSB Bridge

The CSB bridge serve as an interface between the PCI Express core and the CSB domain. It controls the transfer of the transactions between the PCI Express transaction layer and the CSB, and include Write and Read DMA engines, a message manager and a set of configuration registers. Note that programming errors may result in undefined behavior.

15.5.1 PCI Express CSB Bridge Configuration Space

The PCI Express CSB Bridge contains configuration registers for controlling and monitoring the PCI Express and CSB related operations.

The various features supported are as follows:

- ATMUs configuration
- PIO transactions control

- Write DMA engine control
- Read DMA engine control
- Mailbox
- Message signaled interrupts (MSI) generation
- Events monitoring

NOTE

The registers described in this section use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI bus.

15.5.2 Global Registers

15.5.2.1 PCI Express CSB Bridge Control Register (PEX_CSB_CTRL)

PEX_CSB_CTRL, shown in Figure 15-97, controls the operation of the PCI Express to CSB bridge.

Offset 0x808 Access: Read/Write

Bits	31–16
R/W	—
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Bits	15–10	9–8	7–4	3	2	1	0
R/W	—	DSAD	—	RDMAE	WDMAE	IBPIOE	OBPIOE
Reset	0 0 0 0 0 0	0 1	0 0 1 1	0	0	0	0

Figure 15-97. PCI Express CSB Bridge Control Register (PEX_CSB_CTRL)

Table 15-95 defines the bit fields of PEX_CSB_CTRL.

Table 15-95. PEX_CSB_CTRL Register Fields Description

Bits	Name	Description
31–10	—	Reserved
9–8	DSAD	Depth of descriptors array. Indicates the number of descriptors that should be placed at a contiguous addresses block. The PCI Express controller DMA uses this information to fetch each block of descriptors by a burst transaction. The implicit address of the next descriptor is the next memory location. The last descriptor in the contiguous block contains the explicit address pointer of the next set of descriptors. See Section 15.8.4, “Descriptor-Based DMA,” for detailed description. Note that for most usages this field should be programmed to 00. 00 1—Single fetch descriptor chain mode. Each descriptor explicitly contains the address of the next descriptor. 01 2—Burst fetch descriptor chain mode. The PCI Express controller will fetch two contiguous descriptors in a burst. 10 4—Burst fetch descriptor chain mode. The PCI Express controller will fetch four contiguous descriptors in a burst. 11 Reserved
7–4	—	Reserved
3	RDMAE	Read DMA enable. Must be set to enable the read DMA operation.
2	WDMAE	Write DMA enable. Must be set to enable the write DMA operation.
1	IBPIOE	Inbound PIO enable. Must be set to enable the PCI Express Inbound PIO operation.
0	OBPIOE	Outbound PIO enable.Must be set to enable the PCI Express outbound PIO operation.

15.5.2.2 PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR)

PEX_DMA_DSTMR, shown in Figure 15-98, contains the timer value the DMA engine should wait for before reading the next descriptor once it encounters a descriptor that is not ready. The timer should be programmed to allow sufficient number of clocks before the DMA tries to fetch the descriptors again.

Offset 0x814 Access: Read/Write

	31 … 0
R W	DSRT
Reset	All zeros

Figure 15-98. PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR)

Table 15-96 defines the bit field for PEX_DMA_DSTMR.

Table 15-96. PEX_DMA_DSTMR Fields Description

Bits	Name	Description
31–0	DSRT	Descriptor ready timer. Represents the number of CSB bridge clocks that the DMA engine should wait before checking whether the next descriptor is ready when it encounters invalid descriptor.

15.5.2.3 PCI Express CSB Bridge Status Register (PEX_CSB_STAT)

PEX_CSB_STAT, shown in Figure 15-99, maintains the activity status of the DMA and PIO transactions. When a transaction is initiated by the PCI Express DMA or PIO, the corresponding pending bit is set until a response is received.

Offset 0x81C — Access: Read only

	31–25	24	23–17	16
R	—	RDMARP	—	WDMARP
W				
Reset	All zeros			

	15–8	7	6–1	0
R	—	IBPIORP	—	OBPIORP
W				
Reset	All zeros			

Figure 15-99. PCI Express CSB Bridge Status Register (PEX_CSB_STAT)

Table 15-97 defines the bit field for PEX_CSB_STAT.

Table 15-97. PEX_CSB_STAT Register Fields Description

Bits	Name	Description
31–25	—	Reserved
24	RDMARP	Read DMA read transaction pending. Indicates whether a response is pending from the PCI Express bus to a transfer by the read DMA engine. 0 No response pending 1 Response is pending
23–17	—	Reserved
16	WDMARP	Write DMA read transaction pending.Indicates whether a response is pending from the CSB bus to a transfer from the write DMA engine. 0 No response pending 1 Response is pending
15–9	—	Reserved
8	IBPIORP	PCI Express inbound PIO read transaction pending. Indicates whether a response is pending from the CSB bus for an inbound transfer from the PCI Express bus 0 No response pending 1 Response is pending
7–1	—	Reserved
0	OBPIORP	PCI Express outbound PIO read transaction pending. Indicates whether a response is pending from the PCI Express bus for a transfer initiated on the CSB bus. 0 No response pending 1 Response is pending

15.5.3 PCI Express Outbound PIO Registers

The registers discussed in this section control PIO outbound transactions initiated by a CSB master.

15.5.3.1 PCI Express Outbound PIO Control Register (PEX_CSB_OBCTRL)

PEX_CSB_OBCTRL, shown in Figure 15-100, controls the PCI Express Outbound PIO operations.

Offset 0x840 Access: Read/Write

	31–16
R/W	—
Reset	All zeros

	15–10	9–7	6–4	3	2	1	0
R/W	—	TC	—	CFGWE	IOWE	MEMWE	PIOE
Reset	All zeros						

Figure 15-100. PCI Express Outbound PIO Control Register (PEX_CSB_OBCTRL)

Table 15-98 defines the bit fields for PEX_CSB_OBCTRL.

Table 15-98. PEX_CSB_OBCTRL Register Fields Description

Bits	Name	Description
31–4	—	Reserved
9–7	TC	Traffic class. Indicates TC value to be used for TLP generation corresponding to traffic received by the CSB slave.
3	CFGWE	Configuration window enable. Must be set to enable an outbound configuration transaction. Indicates that a CSB transactions directed to an outbound window can be mapped to Config write and read TLPs and transmitted to the PCI Express link.
2	IOWE	I/O window enable. Must be set to enable an outbound I/O transaction. Indicates that a CSB transactions directed to an outbound window can be mapped to I/O write and read TLPs and transmitted to the PCI Express link.
1	MEMWE	Memory window enable. Must be set to enable an outbound Memory transaction. Indicates that a CSB transactions directed to an outbound window can be mapped to Memory write and read TLPs and transmitted to the PCI Express link.
0	PIOE	PIO enable. Must be set to enable an outbound PIO transaction. This field controls the general enable of the PCI Express CSB bridge outbound PIO operation and should be set together with the other window enable fields in this register.

15.5.3.2 PCI Express Outbound PIO Status Register (PEX_CSB_OBSTAT)

PEX_CSB_OBSTAT, shown in Figure 15-101, maintains the status of the CSB PIO operations through the CSB slave controller. This register is replicated for each CSB PIO engine implemented.

Offset 0x844 Access: w1c

	31–16
R	—
W	
Reset	All zeros

	15–4	3	2	1	0
R	—	PEXER	CSBER	BMPER	BENER
W		w1c	w1c	w1c	w1c
Reset	All zeros				

Figure 15-101. PCI Express Outbound PIO Status Register (PEX_CSB_OBSTAT)

Table 15-99 defines the bit fields for PEX_CSB_OBSTAT.

Table 15-99. PEX_CSB_OBSTAT Register Fields Description

Bit	Name	Description
31–4	—	Reserved
3	PEXER	PCI Express error. Hardware sets this bit to indicate that an outbound PIO operation could not be completed successfully because there was an error in PCI Express completion received.
2	CSBER	CSB Bridge error. Hardware sets this bit to indicate that an error has occurred during a PIO outbound access to the CSB bridge by a CSB master.
1	BMPER	Bridge mapping error. Hardware sets this bit to indicate that an outbound PIO operation could not be completed successfully because a CSB bridge address mapping error has occurred.
0	BENER	Bridge enable error.Hardware sets this bit to indicate that the an outbound PIO operation could not be completed successfully because the CSB bridge outbound PIO operation was not enabled.

15.5.4 PCI Express Inbound PIO Registers

The registers discussed in this section control PIO inbound transactions initiated by a PCI Express device.

15.5.4.1 PCI Express Inbound PIO Control Register (PEX_CSB_IBCTRL)

PEX_CSB_IBCTRL, shown in Figure 15-102, controls the PCI Express inbound PIO operations.

Offset 0x8E0 — Access: Read/Write

	31–1	0
R	—	PIOE
W		
Reset	All zeros	

Figure 15-102. PCI Express Inbound PIO Control Register (PEX_CSB_IBCTRL)

Table 15-100 defines the bit fields for PEX_CSB_IBCTRL.

Table 15-100. PEX_CSB_IBCTRL Register Fields Description

Bit	Name	Description
31–1	—	Reserved
0	PIOE	PIO enable. Must be set to enable an inbound PIO transaction. This field controls the general enable of the PCI Express CSB bridge inbound PIO operation.

15.5.4.2 PCI Express Inbound PIO Status Register (PEX_CSB_IBSTAT)

PEX_CSB_IBSTAT, shown in Figure 15-103, maintains the status of PCI Express Inbound PIO operations through the PCI Express CSB bridge.

Offset 0x8E4 — Access: w1c

	31–16
R	—
W	
Reset	All zeros

	15–3	2	1	0
R	—	CSBER	BRGER	BENER
W		w1c	w1c	w1c
Reset	All zeros			

Figure 15-103. PCI Express Inbound PIO Status Register (PEX_CSB_IBSTAT)

Table 15-101 defines the bit fields for PEX_CSB_IBSTAT.

Table 15-101. PEX_CSB_IBSTAT Register Fields Description

Bit	Name	Description
31–3	—	Reserved
2	CSBER	CSB Bus Error. Hardware sets this bit to indicate that a CSB transaction bus error was encountered during an inbound PIO operation.
1	BRGER	CSB Bridge Error. Hardware sets this bit to indicate that an inbound PIO operation cannot complete successfully because of a CSB bridge error.
0	BENER	Bridge enable error. Hardware sets this bit to indicate that the an inbound PIO operation cannot complete successfully because the CSB bridge inbound PIO operation is not enabled.

15.5.5 DMA Registers

15.5.5.1 PCI Express Write DMA Control Register (PEX_WDMA_CTRL)

PEX_WDMA_CTRL, shown in Figure 15-104, controls the WDMA operations.

Offset 0x9A0 Access: Read/Write

	31–16
R / W	—
Reset	All zeros

	15–11	10	9	8–5	4–2	1	0
R / W	—	SNOOP	RLXO		TC	SUS	START
Reset	All zeros						

Figure 15-104. PCI Express Write DMA Control Register (PEX_WDMA_CTRL)

Table 15-102 defines the bit fields of PEX_WDMA_CTRL.

Table 15-102. PEX_WDMA_CTRL Register Fields Description

Bit	Name	Description
31–11	—	Reserved
10	SNOOP	Snoop for write and read transactions to the descriptor. Controls the snooping of the e300 core on the CSB bus to a transaction initiated by the WDMA. 0 Snoop disabled. 1 Snoop enabled.
9	RLXO	Relaxed ordering for PCI Express. Indicates the relaxed ordering bit to be used for all PCI Express transactions initiated by the DMA controller.
8	—	Reserved.
7–5	—	Reserved.

Table 15-102. PEX_WDMA_CTRL Register Fields Description (continued)

Bit	Name	Description
4–2	TC	Traffic Class. Indicates the traffic class value to be used for TLP generation corresponding to the traffic generated by the DMA.
1	SUS	DMA suspend. Software sets this bit to suspend the DMA controller.
0	START	DMA start. Software should set this bit to indicate that a descriptor is ready and the DMA controller can start transmission. Hardware resets this bit when the descriptor fetch cycle is initiated.

15.5.5.2 PCI Express Write DMA First Address Register (PEX_WDMA_ADDR)

PEX_WDMA_ADDR, shown in Figure 15-105, contains the CSB local memory address of the first descriptor. Note that the content is byte swapped from a CSB native address.

Offset 0x9A4 Access: Read/Write

	31–0
R W	FDSA
Reset	All zeros

Figure 15-105. PCI Express Write DMA First Address Register (PEX_WDMA_ADDR)

Table 15-103 defines the bit fields of PEX_WDMA_ADDR.

Table 15-103. PEX_WDMA_ADDR Register Fields Description

Bit	Name	Description
31–0	FDSA	First descriptor address. Indicates the address of the first descriptor on the CSB local memory (byte-swapped).

15.5.5.3 PCI Express Write DMA Status Register (PEX_WDMA_STAT)

PEX_WDMA_STAT, shown in Figure 15-106, maintains the status of write DMA operations.

Offset 0x9A8 Access: w1c

	31–16
R W	—
Reset	All zeros

	15–7	6	5	4	3	2	1	0
R	—	DAFER	BRER	DSUER	—	DSFER	DSCPL	CHCPL
W		w1c	w1c	w1c		w1c	w1c	w1c
Reset	All zeros							

Figure 15-106. PCI Express Write DMA Status Register (PEX_WDMA_STAT)

Table 15-104 defines the bit fields of PEX_WDMA_STAT.

Table 15-104. PEX_WDMA_STAT Register Fields Description

Bit	Name	Description
31–7	—	Reserved
6	DAFER	DMA data fetch error. Hardware sets this bit to indicate an error during the data fetch operation.
5	BRER	Bridge error. Hardware sets this bit to indicate that DMA operation cannot complete successfully because of a CSB bridge error.
4	DSUER	Descriptor update error. Hardware sets this bit to indicate an error during descriptor update operation.
3	—	Reserved
2	DSFER	Descriptor fetch error. This bit is set by hardware to indicate that a descriptor read from the CSB has terminated with an error.
1	DSCPL	Descriptor DMA transfer completed. Hardware sets this bit after completing the transaction for the descriptor.
0	CHCPL	DMA chain transfer completed. Hardware sets this bit after completing the transaction in all the descriptors that are currently programmed. This bit is set when DMA operation is complete and the DMA controller encounters a NULL descriptor. **Note:** When hardware sets this bit it is not guaranteed that the transferred data has fully reached its final destination. Software should guarantee this another way. For additional information see the PEX2 erratum in the errata document of the device.

15.5.5.4 PCI Express Read DMA Control Register (PEX_RDMA_CTRL)

PEX_RDMA_CTRL, shown in Figure 15-107, controls the RDMA operations.

Offset 0xA40 Access: Read/Write

	31–16
R W	—
Reset	All zeros

	15–11	10	9	8–2	1	0
R W	—	SNOOP	RLXO	—	SUS	START
Reset	All zeros					

Figure 15-107. PCI Express Read DMA Control Register (PEX_RDMA_CTRL)

Table 15-105 defines the bit fields of PEX_RDMA_CTRL.

Table 15-105. PEX_RDMA_CTRL Register Fields Description

Bits	Name	Description
31–11	—	Reserved
10	SNOOP	Snoop for write and read transactions to the descriptor. Controls the snooping of the e300 core on the CSB bus to a transaction initiated by the RDMA. 0 Snoop disabled. 1 Snoop enabled.

Table 15-105. PEX_RDMA_CTRL Register Fields Description (continued)

Bits	Name	Description
9	RLXO	Relaxed ordering for PCI Express. Indicates the relaxed ordering bit to be used for all PCI Express transactions initiated by the DMA controller.
8–2	—	Reserved
1	SUS	DMA suspend. Software sets this bit to suspend the DMA controller.
0	START	DMA start. Software can set this bit to indicate that a descriptor is ready and the DMA controller can start transmission. Hardware resets this bit when a descriptor fetch cycle is initiated.

15.5.5.5 PCI Express Read DMA First Address Register (PEX_RDMA_ADDR)

PEX_RDMA_ADDR, shown in Figure 15-108, contains the local memory address of the first descriptor. Note that the content is byte swapped from a CSB native address.

Offset 0xA44 Access: Read/Write

	31–0
R	FDSA
W	
Reset	All zeros

Figure 15-108. PCI Express Read DMA First Address Register (PEX_RDMA_ADDR)

Table 15-106 defines the bit fields of PEX_RDMA_ADDR.

Table 15-106. PEX_RDMA_ADDR Register Fields Description

Bits	Name	Description
31–0	FDSA	First descriptor address. Indicates the address of the first descriptor on the CSB local memory (byte swapped).

15.5.5.6 PCI Express Read DMA Status Register (PEX_RDMA_STAT)

PEX_RDMA_STAT, shown in Figure 15-109, maintains the status of read DMA operations.

Offset 0xA48 Access: w1c

	31–16
R	—
W	
Reset	All zeros

	15–8	7	6	5	4	3	2	1	0
R	—	CPLER	DAFER	BRER	DSUER	—	DSFER	DSCPL	CHCPL
W		w1c	w1c	w1c	w1c		w1c	w1c	w1c
Reset	All zeros								

Figure 15-109. PCI Express Read DMA Status Register (PEX_RDMA_STAT)

Table 15-107 defines the bit fields of PEX_RDMA_STAT.

Table 15-107. PEX_RDMA_STAT Register Fields Description

Bits	Name	Description
31–8	—	Reserved
7	CPLER	PCI Express completion error. Hardware sets this bit to indicate that DMA operation cannot complete successfully because of a PCI Express error.
6	DAFER	DMA data write error. The hardware sets this bit to indicate an error during data write operation.
5	BRER	Bridge error. Hardware sets this bit to indicate that DMA operation cannot complete successfully because of a CSB bridge error.
4	DSUER	Descriptor update error. Hardware sets this bit to indicate an error during descriptor update operation.
3	—	Reserved
2	DSFER	Descriptor fetch error. This bit is set by hardware to indicate that a descriptor read from the CSB has terminated with an error.
1	DSCPL	Descriptor DMA transfer completed. Hardware sets this bit after completing the transaction for the descriptor.
0	CHCPL	DMA chain transfer completed. Hardware sets this bit after completing the transaction in all the descriptors that are currently programmed. This bit is set when DMA operation is complete and the DMA controller encounters a NULL descriptor. **Note:** When hardware sets this bit it is not guaranteed that the transferred data has fully reached its final destination. Software should guarantee this another way.

15.5.6 Mailbox Registers

15.5.6.1 PCI Express Outbound Mailbox Control Register (PEX_OMBCR)

PEX_OMBCR, shown in Figure 15-110, controls the generation of an outbound interrupt from the CSB local host to the PCI Express and indicates that the local host has programmed the data mailbox register, and it is ready to be read. Setting the ready bit generates an interrupt to the PCI Express host if enabled. The PCI Express host should clear the ready bit after reading the data mailbox register.

Offset 0xB20 Access: Read/Write

	31–1	0
R W	—	READY
Reset	All zeros	

Figure 15-110. PCI Express Outbound Mailbox Control Register (PEX_OMBCR)

Table 15-108 defines the bit fields of PEX_OMBCR.

Table 15-108. PEX_OMBCR Register Fields Description

Bits	Name	Description

Table 15-108. PEX_OMBCR Register Fields Description (continued)

31–1	—	Reserved
0	READY	Outbound mailbox ready. If set, indicates that mailbox has valid data to be read by the PCI Express host and generates an interrupt if enabled.

15.5.6.2 PCI Express Outbound Mailbox Data Register (PEX_OMBDR)

PEX_OMBDR, shown in Figure 15-111, contains the data to be read by the PCI Express host when it receives an interrupt.

Offset 0xB24 — Access: Read/Write

	31–0
R/W	MBD
Reset	All zeros

Figure 15-111. MPCI Express Outbound Mailbox Data Register (PEX_OMBDR)

Table 15-109 defines the bit fields of PEX_OMBDR.

Table 15-109. PEX_OMBDR Register Fields Description

Bits	Name	Description
31–0	MBD	Mailbox Data. Contains the data to be read by the PCI Express host upon receiving an interrupt.

15.5.6.3 PCI Express Inbound Mailbox Control Register (PEX_IMBCR)

PEX_IMBCR, shown in Figure 15-112, controls the generation of an interrupt to the local host indicating that the PCI Express host has programmed the data mailbox register, and it is ready to be read. The CSB host clears the bit after reading out the mailbox register. Note that setting the ready bit generates an interrupt to the CSB host if enabled. The CSB host should clear the ready bit after reading the data mailbox register.

Offset 0xB60 — Access: Read/Write

	31–1	0
R/W	—	READY
Reset	All zeros	

Figure 15-112. PCI Express Inbound Mailbox Control Register (PEX_IMBCR)

Table 15-110 defines the bit fields of PEX_IMBCR.

Table 15-110. PEX_IMBCR Register Fields Description

Bits	Name	Description
31–1	—	Reserved
0	READY	Inbound mailbox ready. If set, indicates that mailbox has valid data to be read by the CSB local host and generates an interrupt if enabled.

15.5.6.4 PCI Express Inbound Mailbox Data Register (PEX_IMBDR)

PEX_IMBDR, shown in Figure 15-113, contains the data to be read by the local CSB host.

Offset 0xB64 | Access: Read/Write

	31–0
R/W	MBD
Reset	All zeros

Figure 15-113. PCI Express Inbound Mailbox Data Register (PEX_IMBDR)

Table 15-111 defines the bit fields of PEX_IMBDR.

Table 15-111. PEX_IMBDR Register Fields Description

Bits	Name	Description
31–0	MBD	Mailbox data. Contains the data to be read by the CSB local host upon receiving an interrupt.

15.5.7 PCI Express Host Interrupt Registers

This section describes the registers for generating interrupts to the PCI Express host. It consists of interrupt status registers and enable registers. Interrupts are generated only if the corresponding enable bit is set. The device supports generation of INTx and MSI interrupts. Using these registers to generate interrupts to the PCI Express host is applicable for only PCI Express EP applications.

15.5.7.1 PCI Express Host Interrupt Enable Register (PEX_HIER)

PEX_HIER, shown in Figure 15-114, enables the generation of interrupts to the PCI Express host at various events during the CBS bridge, PIO and DMA operation.

Offset 0xBA0 | Access: Read/Write

	31–16
R/W	—
Reset	All zeros

	15–12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	—	CBRIE	OMBIE	RDAIE	RDSIE	RDCIE	WDAIE	WDSIE	WDCIE	IPAIE	IPCIE	OPAIE	OPCIE
Reset	All zeros												

Figure 15-114. PCI Express Host Interrupt Enable Register (PEX_HIER)

Table 15-112 defines the bit fields of PEX_HIER.

Table 15-112. PEX_HIER Register Fields Description

Bits	Name	Description
31–12	—	Reserved
11	CBRIE	CSB bridge reset interrupt enable. If set, enables the generation of an interrupt upon a CSB bridge reset.
10	OMBIE	Outbound mailbox ready interrupt enable. If set, enables the generation of interrupt when the outbound mailbox control register ready bit (PEX_OMBCR[READY]) is set.
9	RDAIE	RDMA transfer aborted interrupt enable. If set, enables the generation of interrupt for every Read DMA transaction aborted.
8	RDSIE	RDMA descriptor transfer completed interrupt enable. If set, enables the generation of an interrupt when a read DMA transaction corresponding to a descriptor successfully completes.
7	RDCIE	RDMA chain descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when a read DMA transaction for an end-of-descriptor successfully completes.
6	WDAIE	WDMA transfer aborted interrupt enable. If set, enables the generation of interrupt for every DMA transaction aborted.
5	WDSIE	WDMA descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when DMA transactions corresponding to a descriptor successfully complete.
4	WDCIE	WDMA chain descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when DMA transactions for end-of-descriptor successfully complete.
3	IPAIE	Inbound PIO transaction aborted interrupt enable. If set, enables the generation of an interrupt for every inbound PCI Express PIO transaction aborted.
2	IPCIE	Inbound PIO transaction completed interrupt enable. If set, enables the generation of an interrupt for every inbound PCI Express PIO transaction that successfully completes.
1	OPAIE	Outbound PIO transaction abort interrupt enable. If set, enables the generation of an interrupt for every outbound PIO transaction aborted.
0	OPCIE	Outbound PIO transaction completed interrupt enable. If set, enables the generation of an interrupt for every outbound PIO transaction that successfully completes.

15.5.7.2 PCI Express Host Interrupt Status Register (PEX_HISR)

PEX_HISR, shown in Figure 15-115, maintains the status for interrupts issued to the PCI Express host.

Offset 0xBA4 Access: w1c

	31–16
R	—
W	
Reset	All zeros

	15–12	11	10	9	8	7	6	5	4	3	2	1	0
R	—	CBR	OMB	RDA	RDS	RDC	WDA	WDS	WDC	IPA	IPC	OPA	OPC
W		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros												

Figure 15-115. PCI Express Host Interrupt Status Register (PEX_HISR)

Table 15-113 defines the bit fields of PEX_HISR.

Table 15-113. PEX_HISR Register Fields Description

Bits	Name	Description
31–12	—	Reserved
11	CBR	CSB bridge reset. Hardware sets this bit upon a CSB bridge reset.
10	OMB	Outbound mailbox ready. Hardware sets this bit when the outbound mailbox control register ready bit (PEX_OMBCR[READY]) is set, indicating that the outbound mailbox is loaded with data to be read by the PCI-Express host.
9	RDA	RDMA transfer aborted. Hardware sets this bit when a read DMA transaction aborts.
8	RDS	RDMA descriptor transfer completed. Hardware sets this bit when a read DMA transaction corresponding to a descriptor successfully completes.
7	RDC	RDMA chain descriptor transfer completed. Hardware sets this bit when a read DMA transaction for the last descriptor in a chain descriptor successfully completes.
6	WDA	WDMA transfer aborted. Hardware sets this bit when a write DMA transaction aborts.
5	WDS	WDMA descriptor transfer completed. Hardware sets this bit when a write DMA transaction corresponding to a descriptor successfully completes.
4	WDC	WDMA chain descriptor transfer completed. Hardware sets this bit when a write DMA transaction for the last descriptor in a chain descriptor successfully completes.
3	IPA	Inbound PIO transaction aborted. Hardware sets this bit when an inbound PCI Express PIO transaction aborts.
2	IPC	Inbound PIO transaction completed. Hardware sets this bit when an inbound PCI Express PIO transaction successfully completes.
1	OPA	Outbound PIO transaction aborted. Hardware sets this bit when an outbound PIO transaction aborts.
0	OPC	Outbound PIO transaction completed. Hardware sets this bit when an outbound PIO transaction successfully completes.

15.5.7.3 PCI Express Host Outbound PIO Interrupt Vector Register (PEX_HOPIVR)

PEX_HOPIVR, shown in Figure 15-116, contains the interrupt vector for MSI interrupt generation upon an outbound PIO event. This vector is sent by the MSI interrupts to the PCI Express host if the interrupt is enabled.

Offset 0xBA8 | Access: Read/Write

	31–5	4–0
R/W	—	IVEC
Reset	All zeros	

Figure 15-116. PCI Express Host Outbound PIO Interrupt Vector Register (PEX_HOPIVR)

Table 15-114 defines the bit fields of PEX_HOPIVR.

Table 15-114. PEX_HOPIVR Register Fields Description

Bits	Name	Description
31–5	—	Reserved
4–0	IVEC	Interrupt Vector. Contains the vector value for MSI.

15.5.7.4 PCI Express Host Inbound PIO Interrupt Vector Register (PEX_HIPIVR)

PEX_HIPIVR, shown in Figure 15-117, contains the interrupt vector for MSI interrupt generation upon an inbound PIO event. This vectors will be sent by the MSI interrupts to the PCI Express host if the interrupt is enabled.

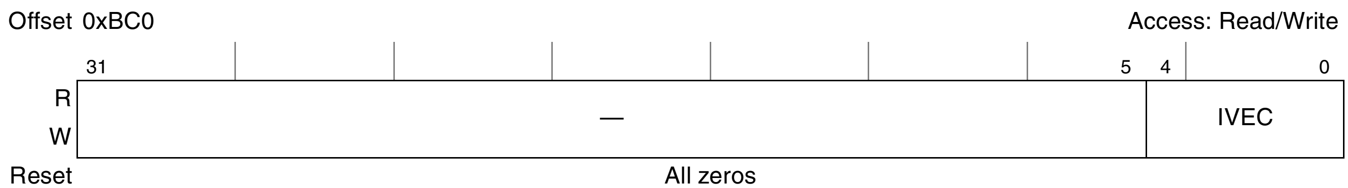


Figure 15-117. PCI Express Host Inbound PIO Interrupt Vector Register (PEX_HIPIVR)

Table 15-115 defines the bit fields of PEX_HIPIVR.

Table 15-115. PEX_HIPIVR Register Fields Description

Bits	Name	Description
31–5	—	Reserved
4–0	IVEC	Interrupt vector. Contains the vector value for MSI.

15.5.7.5 PCI Express Host Write DMA Interrupt Vector Register (PEX_HWDIVR)

PEX_HWDIVR, shown in Figure 15-118, contains the interrupt vector for an MSI interrupt issued to the PCI Express host upon WDMA events.

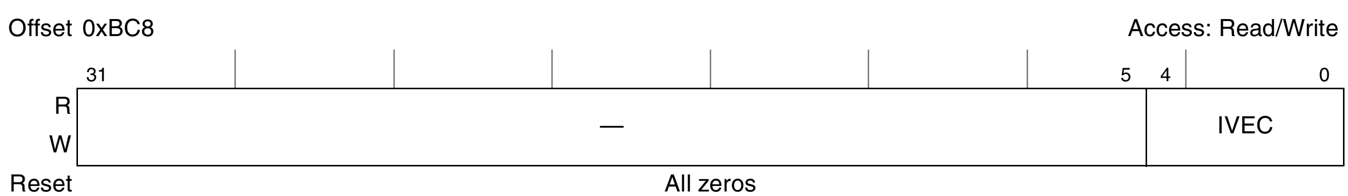


Figure 15-118. PCI Express Host Write DMA Interrupt Vector Register (PEX_HWDIVR)

Table 15-116 defines the bit fields of PEX_HWDIVR.

Table 15-116. PEX_HWDIVR Register Fields Description

Bits	Name	Description
31–5	—	Reserved
4–0	IVEC	Interrupt vector. Contains the vector value for MSI.

15.5.7.6 PCI Express Host Read DMA Interrupt Vector Register (PEX_HRDIVR)

PEX_HRDIVR, shown in Figure 15-119, contains the RDMA interrupt vector for MSI interrupt issued to the PCI Express host upon RDMA events.

Offset 0xBD0 Access: Read/Write

	31–5	4–0
R/W	—	IVEC
Reset	All zeros	

Figure 15-119. PCI Express Host Read DMA Interrupt Vector Register (PEX_HRDIVR)

Table 15-117 defines the bit fields of PEX_HRDIVR.

Table 15-117. PEX_HRDIVR Register Fields Description

Bit	Name	Description
31–5	—	Reserved
4–0	IVEC	Interrupt Vector. Contains the vector value for MSI.

15.5.7.7 PCI Express Host Miscellaneous Interrupt Vector Register (PEX_HMIVR)

PEX_HMIVR, shown in Figure 15-120, contains the interrupt vector for mailbox message for MSI interrupts issued to the PCI Express host. This includes CSB bridge reset and mailbox message.

Offset 0xBD8 Access: Read/Write

	31–5	4–0
R/W	—	IVEC
Reset	All zeros	

Figure 15-120. PCI Express Host Miscellaneous Interrupt Vector Register (PEX_HMIVR)

Table 15-118 defines the bit fields of PEX_HMIVR.

Table 15-118. PEX_HMIVR Register Fields Description

Bit	Name	Description
31–5	—	Reserved
4–0	IVEC	Interrupt Vector. Contains the vector value for MSI upon miscellaneous events.

15.5.8 CSB System Interrupt Registers

This section describes the registers for generating interrupts to the CSB system host (through the IPIC). It consists of interrupt status registers and enable registers. Interrupts are generated only if the corresponding enable bit is set, upon PIO, DMA and Miscellaneous events. The user has the flexibility of combining all or partial interrupt signals to generate interrupts to the CSB system host.

15.5.8.1 CSB System PIO Interrupt Enable Register (PEX_CSPIER)

PEX_CSPIER, shown in Figure 15-121, controls the generation of interrupt to the CSB processor at various events during an CSB or PCI Express PIO operation.

Offset 0xBE0 — Access: Read/Write

	31–25	24	23–17	16
R/W	—	IPAIE		IPCIE
Reset	All zeros			

	15–9	8	7–1	0
R/W		OPAIE		OPCIE
Reset	All zeros			

Figure 15-121. CSB System PIO Interrupt Enable Register (PEX_CSPIER)

Table 15-119 defines the bit fields of PEX_CSPIER.

Table 15-119. PEX_CSPIER Register Fields Description

Bit	Name	Description
31–25	—	Reserved
24	IPAIE	Inbound PIO transaction aborted interrupt enable. If set, enables the generation of an interrupt for every inbound PCI Express PIO transaction aborted.
23–17	—	Reserved
16	IPCIE	Inbound PIO transaction completed interrupt enable. If set, enables the generation of an interrupt for every inbound PCI Express PIO transaction that successfully completes.
15–9	—	Reserved
8	OPAIE	Outbound PIO transaction abort interrupt enable. If set, enables the generation of an interrupt for every outbound PIO transaction aborted.
7–1	—	Reserved
0	OPCIE	Outbound PIO transaction completed interrupt enable. If set, enables the generation of an interrupt for every outbound PIO transaction that successfully completes.

15.5.8.2 CSB System Write DMA Interrupt Enable Register (PEX_CSWDIER)

PEX_CSWDIER, shown in Figure 15-122, controls the generation of interrupt to the CSB processor at various events during a WDMA operation.

Offset 0xBE4 Access: Read/Write

	31–17	16
R/W	—	WDAIE
Reset	All zeros	

	15–9	8	7–1	0
R/W	—	WDSIE	—	WDCIE
Reset	All zeros			

Figure 15-122. CSB System Write DMA Interrupt Enable Register (PEX_CSWDIER)

Table 15-120 defines the bit fields of PEX_CSWDIER.

Table 15-120. PEX_CSWDIER Register Fields Description

Bit	Name	Description
31–17	—	Reserved
16	WDAIE	WDMA transfer aborted interrupt enable. If set, enables the generation of interrupt for every DMA transaction aborted.
15–9	—	Reserved
8	WDSIE	WDMA descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when DMA transactions corresponding to a descriptor complete successfully.
7–1	—	Reserved
0	WDCIE	WDMA chain descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when DMA transactions for end-of-descriptor complete successfully.

15.5.8.3 CSB System Read DMA Interrupt Enable Register (PEX_CSRDIER)

PEX_CSRDIER, shown in Figure 15-123, controls the generation of interrupt to the CSB system host at various events during a RDMA operation.

Offset 0xBE8 | Access: Read/Write

	31–17	16
R W	—	RDAIE
Reset	All zeros	

	15–9	8	7–1	0
R W	—	RDSIE	—	RDCIE
Reset	All zeros			

Figure 15-123. CSB System Read DMA Interrupt Enable Register (PEX_CSRDIER)

Table 15-121 defines the bit fields of PEX_CSRDIER.

Table 15-121. PEX_CSRDIER Register Fields Description

Bit	Name	Description
31–17	—	Reserved
16	RDAIE	RDMA transfer aborted interrupt enable. If set, enables the generation of interrupt for every Read DMA transaction aborted.
15–9	—	Reserved.
8	RDSIE	RDMA descriptor transfer completed interrupt enable. If set, enables the generation of an interrupt when a Read DMA transactions corresponding to a descriptor complete successfully.
7–1	—	Reserved.
0	RDCIE	RDMA chain descriptor transfer completed interrupt enable. If set, enables the generation of interrupt when a Read DMA transactions for end-of-descriptor complete successfully.

15.5.8.4 CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER)

PEX_CSMIER, shown in Figure 15-124, controls the generation of interrupt to the CSB processor at various events.

Offset 0xBEC — Access: Read/Write

	31	30	29	28		21	20	19	18	17	16
R/W	—		ERRD IE	—			PSCH IE	—		PMETO IE	—
Reset	All zeros										

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	ROF IE	ECRC IE	PTLP IE	MFTLP IE	CTO IE	CAIE	URIE	INTD IE	INTC IE	INTB IE	INTA IE	—			RSTIE	IMBIE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 15-124. CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER)

Table 15-122 defines the bit fields of PEX_CSMIER.

Table 15-122. PEX_CSMIER Register Fields Description

Bit	Name	Description
31–30	—	Reserved
29	ERRDIE	Error detected interrupt enable. If set, enables the generation of an interrupt when a PCI Express event occurs. The PCI Express event is reported by the Secondary status register (PCI Express Secondary Status Register) at address 0x901E. Note that a secondary mask exist by the PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK) at address 0x95A0. Valid only for RC.
28–21	—	Reserved
20	PSCHIE	Power state change interrupt enable. If set, enables the generation of an interrupt when a device power state change occurs.
19–18	—	Reserved
17	PMETOIE	PME to Ack timeout event interrupt enable. If set, enables the generation of an interrupt when a PME to Ack timeout occurs. This bit is valid only in RC mode.
16	—	Reserved
15	ROFIE	Receive overflow error interrupt enable. If set, enables the generation of an interrupt when PCI Express reports receive overflow error.
14	ECRCIE	ECRC error interrupt enable. If set, enables the generation of an interrupt when a TLP is received by PCI Express and it fails ECRC check.
13	PTLPIE	Poisoned TLP interrupt enable. If set, enables the generation of an interrupt when a poisoned TLP is received by PCI Express.
12	MFTLPIE	Malformed TLP interrupt enable. If set, enables the generation of an interrupt when a malformed TLP is received by PCI Express
11	CTOIE	Completion timeout interrupt enable. If set, enables the generation of an interrupt when PCI Express completion timeout occurs.
10	CAIE	Completer abort interrupt enable. If set, enables the generation of an interrupt when PCI Express completion Abort is received.

Table 15-122. PEX_CSMIER Register Fields Description (continued)

Bit	Name	Description
9	URIE	Unsupported request interrupt enable. If set, enables the generation of an interrupt when an unsupported request is received.
8	INTDIE	PCI Express INTD interrupt enable. If set, enables the generation of an interrupt when an INTD interrupt is received on the PCI Express link. Valid for RC applications only.
7	INTCIE	PCI Express INTC interrupt enable. If set, enables the generation of an interrupt when an INTC interrupt is received on the PCI Express link. Valid for RC applications only.
6	INTBIE	PCI Express INTB interrupt enable. If set, enables the generation of an interrupt when an INTB interrupt is received on the PCI Express link. Valid for RC applications only.
5	INTAIE	PCI Express INTA interrupt enable. If set, enables the generation of an interrupt when an INTA interrupt is received on the PCI Express link. Valid for RC applications only.
4–2	—	Reserved
1	RSTIE	PCI Express reset interrupt enable. If set, enables the generation of an interrupt when PCI Express is reset.
0	IMBIE	Inbound mailbox ready interrupt enable. If set, enables the generation of interrupt whenever the inbound mailbox control register ready bit (PEX_IMBCR[READY]) is set.

15.5.8.5 CSB System PIO Interrupt Status Register (PEX_CSPISR)

PEX_CSPISR, shown in Figure 15-125, maintains the status for interrupts issued to the CSB system host related to PIO operation.

Offset 0xBF0 Access: w1c

	31–25	24	23–17	16
R		IPA		IPC
W		w1c		w1c
Reset	All zeros			

	15–9	8	7–1	0
R		OPA		OPC
W		w1c		w1c
Reset	All zeros			

Figure 15-125. CSB System PIO Interrupt Status Register (PEX_CSPISR)

Table 15-123 defines the bit fields of PEX_CSPISR

Table 15-123. PEX_CSPISR Register Fields Description

Bit	Name	Description
31–25	—	Reserved
24	IPA	Inbound PIO transaction aborted. Indicates that an inbound PCI Express PIO transaction was aborted.
23–17	—	Reserved

Table 15-123. PEX_CSPISR Register Fields Description (continued)

Bit	Name	Description
16	IPC	Inbound PIO transaction completed. Indicates that an inbound PCI Express PIO transaction was successfully completes.
15–9	—	Reserved
8	OPA	Outbound PIO transaction aborted. Indicates that an outbound PIO transaction was aborted.
7–1	—	Reserved
0	OPC	Outbound PIO transaction completed. Indicates that an outbound PIO transaction was successfully completes.

15.5.8.6 CSB System Write DMA Interrupt Status Register (PEX_CSWDISR)

PEX_CSWDISR, shown in Figure 15-126, maintains the status for interrupts issued to the CSB host related to WDMA operation.

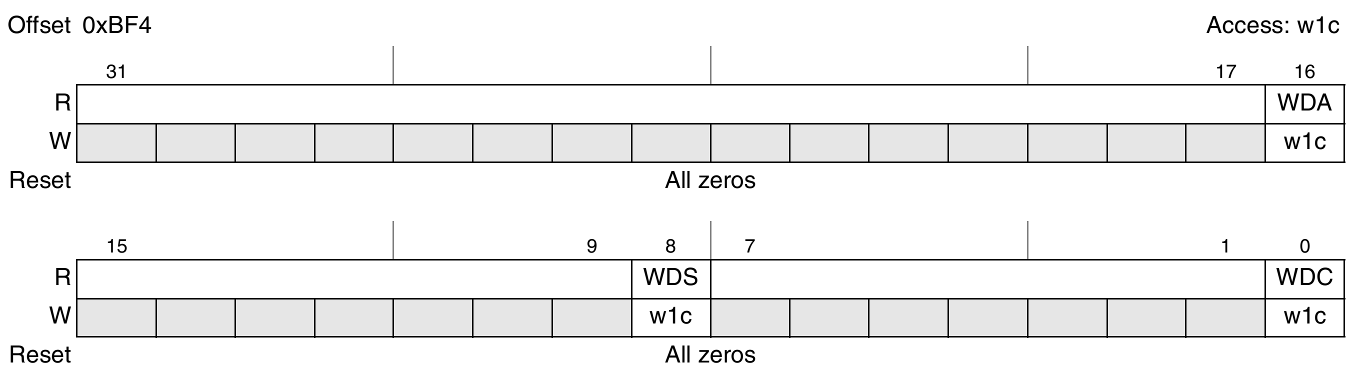


Figure 15-126. CSB System Write DMA Interrupt Status Register (PEX_CSWDISR)

Table 15-124 defines the bit fields of PEX_CSWDISR.

Table 15-124. PEX_CSWDISR Register Fields Description

Bits	Name	Description
31–17	—	Reserved
16	WDA	WDMA transfer aborted. Indicates that a write DMA transaction was aborted.
15–9	—	Reserved
8	WDS	WDMA descriptor transfer completed. Indicates that a write DMA transaction corresponding to a descriptor successfully completed.
7–1	—	Reserved
0	WDC	WDMA chain descriptor transfer completed.Indicates that a write DMA transaction corresponding to the last descriptor in a chain successfully completed.

15.5.8.7 CSB System Read DMA Interrupt Status Register (PEX_CSRDISR)

PEX_CSRDISR, shown in Figure 15-127, maintains the status for interrupts issued to the CSB host related to the RDMA operation.

Offset 0xBF8 — Access: w1c

	31–17	16
R	—	RDA
W		w1c
Reset	All zeros	

	15–9	8	7–1	0
R	—	RDS	—	RDC
W		w1c		w1c
Reset	All zeros			

Figure 15-127. CSB System Read DMA Interrupt Status Register (PEX_CSRDISR)

Table 15-125 defines the bit fields of PEX_CSRDISR.

Table 15-125. PEX_CSRDISR Register Fields Description

Bits	Name	Description
31–17	—	Reserved
16	RDA	RDMA transfer aborted. Indicates that a Read DMA transaction was aborted.
15–9	—	Reserved
8	RDS	RDMA descriptor transfer completed. Indicates that a read DMA transaction corresponding to a descriptor successfully completed.
7–1	—	Reserved
0	RDC	RDMA chain descriptor transfer completed. Indicates that a read DMA transaction corresponding to the last descriptor in a chain successfully completed.

15.5.8.8 CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR)

PEX_CSMISR, shown in Figure 15-128, maintains the status for interrupt issued to the CSB.

Offset 0xBFC — Access: w1c

	31–30	29	28–21	20	19–18	17	16
R	—	ERRD	—	PSCH	—	PMETO	—
W		w1c		w1c		w1c	
Reset	All zeros						

	15	14	13	12	11	10	9	8	7	6	5	4–2	1	0
R	ROF	ECRC	PTLP	MFTLP	CTO	CA	UR	INTD	INTC	INTB	INTA	—	RST	IMB
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c	w1c
Reset	All zeros													

Figure 15-128. CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR)

Table 15-126 defines the bit fields of PEX_CSMISR.

Table 15-126. PEX_CSMISR Register Fields Description

Bits	Name	Description
31–30	—	Reserved
29	ERRD	Error detected. Indicates that a PCI Express event occurred. The PCI Express event is reported by the secondary status register (PCI Express secondary status register) at address 0x901E. Note that there is a secondary mask, the PCI Express interrupt mask register (PEX_SS_INTR_MASK), at address 0x95A0. Valid only for RC. This bit must be cleared after the interrupt is serviced and after the associated status registers in the PCI Express controller causing the interrupt are cleared.
28–21	—	Reserved
20	PSCH	Power state change. Indicates that a device power state change occurred. Change in Device power state (D state) for function-0. D-state can transition between the supported values of D0, D1, D2 and D3-hot. PM software changes D-state with a configuration write to PMCSR register in PM capability of the PCI Express. The new D-state is available in the corresponding field of the PMCSR register. This bit must be cleared after the interrupt is serviced.
19	EXL2L3	Exited L2/L3 state. Indication that the L2/L3 ready state has been exited and the current link state is L0. Traffic can be re-started on the link. This bit is set when the link switches from the L2/L3 ready to L0 state in response to an Exit L2 command. After issuing this command, the user must wait until the exited EXL2L3 bit is set before initiating traffic. This bit is valid only in RC mode. Setting this bit causes an interrupt to be sent to CSB side. The bit must be cleared after the interrupt is serviced.
18	ENL2L3	Entered L2/L3 state. Indication to the Power manager that it is safe to switch off power to the downstream device 100nsec after this bit is set. It is set when the PCI Express controller enters L2/L3 ready state. This bit is valid only in RC mode. Setting this bit causes an interrupt to be sent to the CSB side. The bit must be cleared after the interrupt is serviced.
17	PMETMO	PME Turn Off Ack timeout event. Indication to power manager software that it is safe to switch off power to the downstream device. It is set when the PCI Express controller detects that the timeout interval for receiving a PME_To_Ack message from the downstream device has expired. This bit is valid only in RC mode. This bit must be cleared after the interrupt is serviced.
16	RPMETO	Received PME Turn off message. Notifies that main power to the device is to be removed. After this notification is received, Uplink must not try to transmit TLPs or initiate PME requests because the power may be switched off. After this message is received, the user should indicate the readiness to lose power by setting the Initiate L2/L3 entry bit. This bit is valid only in EP mode. Setting this bit causes an interrupt to be sent to the CSB side. The bit must be cleared after the interrupt is serviced.
15	ROF	Receive overflow error. Indicates that the PCI Express reported a receive overflow error.
14	ECRC	ECRC error. Indicates that a TLP received by the PCI Express failed the ECRC check.
13	PTLP	Poisoned TLP. Indicates that a poisoned TLP was received by the PCI Express.
12	MFTLP	Malformed TLP. Indicates that a malformed TLP was received by the PCI Express
11	CTO	Completion timeout. Indicates that a PCI Express completion timeout occurred.
10	CA	Completer abort. Indicates that a PCI Express completion Abort was received.
9	UR	Unsupported request. Indicates that an unsupported request was received.
8	INTD	PCI Express INTD. Indicates that an INTD interrupt was received on the PCI Express link. Valid for RC applications only.
7	INTC	PCI Express INTC. Indicates that an INTC interrupt was received on the PCI Express link. Valid for RC applications only.

Table 15-126. PEX_CSMISR Register Fields Description (continued)

Bits	Name	Description
6	INTB	PCI Express INTB. Indicates that an INTB interrupt was received on the PCI Express link. Valid for RC applications only.
5	INTA	PCI Express INTA. Indicates that an INTA interrupt was received on the PCI Express link. Valid for RC applications only.
4–2	—	Reserved
1	RST	PCI Express reset. Indicates that a PCI Express reset occurred.
0	IMB	Inbound mailbox ready. Indicates that the inbound mailbox control register ready bit (PEX_IMBCR[READY]) was set and the CSB host can read the mailbox data.

15.5.9 PCI Express Power Management Registers

15.5.9.1 PCI Express PM Control Register (PEX_PM_CTRL)

This PCI Express PM Control Register shown in Figure 15-129, is used to control the link power management by the PCI Express controller.

Offset 0xC80 | Access: Write Only

	31–16
R	
W	—
Reset	All zeros

	15–4	3	2	1	0
R					
W	—	PMESS	EXL2	PMETO	IL2L3
Reset	All zeros				

Figure 15-129. PCI Express PM Control Register (PEX_PM_CTRL)

Table 15-127 defines the bit fields of the PEX_PM_CTRL.

Table 15-127. PEX_PM_CTRL Register Fields Description

Bits	Name	Description
31–4	—	Reserved. Must be zeros.
3	PMESS	PME Status Set. Set the PME Status bit in the PCI Express Power Management Status and Control Register of the configuration space (Offset 0x048). In EP mode, this also causes the PCI Express controller to send PM_PME message. PME message transmission is not supported in RC mode, but the PME status bit can still be set. PM PME message can be used to request for a device power state change. The message will be transmitted only if PME is enabled and if PME Turn off message has not been received by the endpoint. This field is Write only. Read always returns zero.

Table 15-127. PEX_PM_CTRL Register Fields Description (continued)

Bits	Name	Description
2	EXL2	Exit L2. This bit is valid only in RC mode and instructs the PCI Express controller to exit from L2/L3 ready state and move to L0 active state so that traffic can be re-started on the PCI-Express link. This bit can be set under the following conditions: the downstream device was shut down by the power manager (through L2/L3 protocol) and later has its power restored, whereas the upstream device (CI Express controller as RC) was in L2/L3 ready state (with power and clocks available). This field is Write only. Read always returns zero.
1	PMETO	Send PME Turn off message. This bit is valid only in RC mode and instructs the PCI Express controller to send PME_Turn_Off message to downstream devices. After setting this bit, the user must not try to transmit TLPs or initiate PME requests as the power may be switched off. This field is Write only. Read always returns zero.
0	IL2L3	Initiate L2/L3 entry. This bit is valid only in EP mode and instructs the PCI Express controller to transition to L2/L3 ready state. This bit has to be asserted only after preparing for power removal. After setting this bit, the user must not try to transmit TLPs or initiate PME requests as the power may be switched off. This field is Write only. Read always returns zero.

15.5.10 PCI Express Outbound Address Mapping Registers

The registers discussed in this section control the outbound transactions attributes and address mapping from the CSB domain to the PCI Express domain. These registers are used in both RC and EP modes, and serve both PIO and DMA transactions.

15.5.10.1 PCI Express Outbound Window Attributes Register *n* (PEX_OWAR0–PEX_OWAR3)

PEX_OWAR0–PEX_OWAR3, shown in Figure 15-130, sets the attributes for the respective address window defined by the base and translation address registers for mapping of addresses related to CSB outbound transactions to PCI Express addresses.

Offset 0xCA0, 0xCB0, 0xCC0, 0xCD0 — Access: Read/Write

	31–16
R/W	SIZE
Reset	All zeros

	15–12	11–8	7–5	4	3	2–1	0
R/W	SIZE	—	TC	NSNP	RLXO	TYPE	EN
Reset	All zeros						

Figure 15-130. PCI Express Outbound Window Attributes Register *n* (PEX_OWAR0–PEX_OWAR3)

Table 15-128 defines the bit fields of the PEX_OWAR0–PEX_OWAR3.

Table 15-128. PEX_OWAR0–PEX_OWAR3 Register Fields Description

Bits	Name	Description
31–12	SIZE	CSB window size. Indicates the size of window in bytes. The actual size is a concatenation of the SIZE field as most significant bits and 12 zeroes as least significant bits {SIZE[31:12], 0x000}.
11–8	—	Reserved. Must be zeros.
7–5	TC	Traffic class. Indicates the traffic class of the packet. Applicable only if user wants to send traffic using multiple TC but single VC.
4	NSNP	No snoop enable. When this bit and the PCI Express device control register [Enable No Snoop] bit are set, the No Snoop bit for the packet is enabled. This attribute is not applicable and must be cleared for configuration requests, I/O requests, and memory requests that are Message Signaled Interrupts. 0 PCI Express TLP snoop enabled 1 PCI Express TLP snoop disabled
3	RLXO	Relax ordering enable. When this bit and the PCI Express device control register [Enable Relaxed] bit are set, this bit enables the relaxed ordering bit for the packet. This applies only to memory transactions.
2–1	TYPE	Window type. Indicates the type to which CSB transactions to the window address are mapped. 00 CFG 01 I/O 10 Memory 11 Reserved
0	EN	Enable. Must be set to enable this window.

15.5.10.2 PCI Express Outbound Window Base Address Register *n* (PEX_OWBAR0–PEX_OWBAR3)

PEX_OWBAR0–PEX_OWBAR3, shown in Figure 15-131, contains the base address of the CSB window for mapping to a PCI Express address. Note that the CSB base address must be aligned to 1 Kbyte. Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.

Offset 0xCA4, 0xCB4, 0xCC4, 0xCD4 — Access: Read/Write

	31–12	11–0
R / W	BA	—
Reset	All zeros	

Figure 15-131. PCI Express Outbound Window Base Address Register *n* (PEX_OWBAR0–PEX_OWBAR3)

Table 15-129 defines the bit fields of the PEX_OWBAR0–PEX_OWBAR3.

Table 15-129. PEX_OWBAR*n* Register Fields Description

Bits	Name	Description
31–12	BA	Base address. The CSB window base address. Represents the CSB-based address for the window. The actual address is a concatenation of the BAR field as most significant bits and 12 zeroes as least significant bits {BA[31:12], 0x000}.
11–0	—	Reserved. Must be zeros.

15.5.10.3 PCI Express Outbound Window Translation Address Register Low *n* (PEX_OWTARL0–PEX_OWTARL3)

PEX_OWTARL0–PEX_OWTARL3, shown in Figure 15-132, contain the lower base address of the PCI Express domain corresponding to this window. When this window is enabled and a CSB-based transaction hits its base address register, the address is translated to a PCI Express address, according to the PEX_OWTARL*n* and PEX_OWTARH*n* registers.

Offset 0xCA8, 0xCB8, 0xCC8, 0xCD8 — Access: Read/Write

	31–12	11–0
R / W	TAL	—
Reset	All zeros	

Figure 15-132. PCI Express Outbound Window Translation Address Register Low *n* (PEX_OWTARL0–PEX_OWTARL3)

Table 15-130 defines the bit fields of PEX_OWTARL*n*.

Table 15-130. PEX_OWTARL*n* Register Fields Description

Bits	Name	Description
31–12	TAL	Translation address low. The lower portion of the PCI Express address base. The actual address is a concatenation of the TA field as most significant bits and 12 zeroes as least significant bits {TAL[31:12], 0x000}. The complete 64 bits address on the PCI Express bus is built of {PEX_OWTARH[TAH], PEX_OWTARL[TAL], 0x000}.
11–0	—	Reserved.

15.5.10.4 PCI Express Outbound Window Translation Address Register High *n* (PEX_OWTARH0–PEX_OWTARH3)

PEX_OWTARH0–PEX_OWTARH3, shown in Figure 15-133, contains the higher base address of he PCI Express domain corresponding to this window. When this window is enabled and a CSB based transaction hits its base address register, the address is translated to a PCI Express address according to the PEX_OWTARL*n* and PEX_OWTARH*n* registers. This register should be used in 64-bit addressing. Otherwise it should contain all zeroes.

Offset 0xCAC, 0xCBC, 0xCCC, 0xCDC — Access: Read/Write

	31–0
R / W	TAH
Reset	All zeros

Figure 15-133. PCI Express Outbound Window Translation Address Register High *n* (PEX_OWTARH0–PEX_OWTARH3)

Table 15-131 defines the bit fields of PEX_OWTARH*n*.

Table 15-131. PEX_OWTARH*n* Register Fields Description

Bits	Name	Description
31–0	TAH	Translation address high. Higher portion of the PCI Express address base ([63:32]). The complete 64-bit address on the PCI Express bus is built of {PEX_OWTARH[TAH],PEX_OWTARL[TAL], 0x000}.

15.5.11 PCI Express EP Inbound Address Translation Registers

The following registers are used as the base address for the CSB domain to translate the address of an inbound transaction from the PCI Express domain. They are valid only in End Point (EP) mode and operate in conjunction with the base address registers in the PCI Express configuration space.

When a PCI Express inbound transaction hits a valid address window defined by the PCI Express configuration space base address registers and the respective translation register is enabled, the incoming address is translated to a CSB domain address and the transaction is forwarded to the CSB.

The correspondence between the PCI Express configuration space base address registers and the respective translation address registers is shown in Table 15-132.

Table 15-132. EP Inbound Base and Translation Address Registers Correspondence

Base Address Register (Configuration Space)	BAR Name	Type	TAR Address	TAR Name
0x010	BAR0	Window 0, 32-bit address	0xDE0	PEX_EPIWTAR0
0x014	BAR1	Window 1, 32-bit address	0xDE4	PEX_EPIWTAR1
0x018	BAR2	Window 2, 64-bit address, low portion	0xDE8	PEX_EPIWTAR2
0x01C	BAR3	Window 2, 64-bit address, high portion		
0x020	BAR4	Window 3, 64-bit address, low portion	0xDEC	PEX_EPIWTAR3
0x024	BAR5	Window 3, 64-bit address, high portion		

15.5.11.1 PCI Express EP Inbound Window Translation Address Register *n* (PEX_EPIWTAR0–PEX_EPIWTAR3)

PEX_EPIWTAR0–PEX_EPIWTAR3, shown in Figure 15-134, contain the CSB address to be mapped for a PCI Express inbound transaction hitting the respective BAR window. Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.

Offset 0xDE0, 0xDE4, 0xDE8, 0xDEC — Access: Read/Write

	31–12	11–1	0
R/W	TA		EN
Reset	All zeros		

Figure 15-134. PCI Express EP Inbound Window Translation Address Register *n* (PEX_EPIWTAR0–PEX_EPIWTAR3)

Table 15-133 defines the bit fields of PEX_EPIWTAR*n*.

Table 15-133. PEX_EPIWTAR*n* Register Fields Description

Bits	Name	Description
31–12	TA	Translation address. Contains the CSB base address to be mapped for a PCI Express inbound transaction hitting the respective BAR window. The actual address is a concatenation of the TA field as most significant bits and 12 zeroes as least significant bits {TA[31:12], 0x000}.
11–1	—	Reserved
0	EN	Enable. If set, indicates that the address mapping window is enabled.

15.5.12 PCI Express RC Inbound Address Mapping Registers

The registers discussed in this section control the inbound transactions attributes and address mapping from the PCI Express bus to the CSB domain. These registers are used only in RC mode, and they serve inbound transactions. When a PCI Express inbound transaction hits a valid address window defined by these registers and the respective translation register is enabled, the incoming address is translated to a CSB domain address and the transaction is forwarded to the CSB.

15.5.12.1 PCI Express RC Inbound Window Attributes Register *n* (PEX_RCIWAR0 –PEX_RCIWAR3)

PEX_RCIWAR0–PEX_RCIWAR3, shown in Figure 15-135, controls the mapping of a PCI Express inbound PIO transaction to a CSB transaction. This register is valid only in RC mode.

Offset 0xE60, 0xE70, 0xE80, 0xE90 — Access: Read/Write

	31–28	27–16
R/W	—	SIZE
Reset	All zeros	

	15–12	11–5	4	3	2–1	0
R/W	SIZE		NSNP	NSOV	TYPE	EN
Reset	All zeros					

Figure 15-135. PCI Express RC Inbound Window Attributes Register *n* (PEX_RCIWAR0–PEX_RCIWAR3)

Table 15-134 defines the bit fields of PEX_RCIWAR*n*.

Table 15-134. PEX_RCIWAR*n* Register Fields Description

Bits	Name	Description
31–28	—	Reserved
27–12	SIZE	PCI Express window size. Indicates the size of the window in bytes.The actual size is a concatenation of the SIZE field as most significant bits and 12 zeroes as least significant bits {SIZE[27:12], 0x000}.
11–5	—	Reserved. Must be zeros.
4	NSNP	No snoop. If the no-snoop override enable bit in this register (NSOV) is set, this bit defines the snooping behavior on the CSB domain for the inbound packet hitting this window. This applies only to memory transactions. 0 CSB snoop enabled 1 CSB snoop disabled
3	NSOV	No-snoop override. If set, the No-Snoop attribute in the packet is overridden by the No Snoop bit in this register (NSNP). Otherwise, the No-Snoop bit in the inbound packet defines the snooping behavior. 0 Snoop behavior is defined by the inbound packet 1 Snoop behavior is defined by the NSNP field
2–1	TYPE	Type. Indicates the type of the window to which the PCI Express transactions are mapped. 00 Reserved 01 Reserved 10 Prefetchable memory. Inbound read transactions are optimized for CSB performance. The address and size of the actual memory read transaction may differ from those of the original PCI Express packet, aligning the first and last segments of the data read from the memory to cache line boundaries. 11 Non-prefetchable memory. Inbound read transactions from the PCI Express bus access the exact address and size of the memory. This mode is not optimized for CSB performance.
0	EN	Enable. Must be set to enable this window.

15.5.12.2 PCI Express RC Inbound Window Translation Address Register *n* (PEX_RCIWTAR0–PEX_RCIWTAR3)

PEX_RCIWTAR0–PEX_RCIWTAR3), shown in Figure 15-136, contain the CSB address to be mapped for a PCI Express inbound transaction hitting the respective base address register of this window. These registers are valid only in RC mode. Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.

Offset 0xE64, 0xE74, 0xE84, 0xE94 — Access: Read/Write

	31–12	11–0
R W	TA	—
Reset	All zeros	

Figure 15-136. PCI Express RC Inbound Window Translation Address Register *n* (PEX_RCIWTAR0–PEX_RCIWTAR3)

Table 15-135 defines the bit fields of PEX_RCIWTAR*n*.

Table 15-135. PEX_RCIWTAR*n* Register Fields Description

Bits	Name	Description
31–12	TA	Translation address. Contains the CSB base address to be mapped for a PCI Express inbound transaction hitting the respective base address register of this window. The actual address is a concatenation of the TA field as most significant bits and 12 zeroes as least significant bits {TA[31:12], 0x000}.
11–0	—	Reserved. Must be zeros.

15.5.12.3 PCI Express RC Inbound Window Base Address Register Low *n* (PEX_RCIWBARL0–PEX_RCIWBARL3)

PEX_RCIWBARL0–PEX_RCIWBARL3, shown in Figure 15-137, contains the lower portion base address of the PCI Express domain corresponding to this window.

Offset 0xE68, 0xE78, 0xE88, 0xE98 Access: Read/Write

	31–12	11–0
R/W	BAL	—
Reset	All zeros	

Figure 15-137. PCI Express RC Inbound Window Base Address Register Low *n* (PEX_RCIWBARL0–PEX_RCIWBARL3)

Table 15-136 defines the bit fields of PEX_RCIWBARL*n*.

Table 15-136. PEX_RCIWBARL*n* Register Fields Description

Bits	Name	Description
31–12	BAL	Base address low. Lower portion of the PCI Express address base. Represents the PCI Express-based address for the window. The actual address is a concatenation of the BAL field as most significant bits and 12 zeroes as least significant bits {BAL[31:12], 0x000}.
11–0	—	Reserved. Must be zeros.

15.5.12.4 PCI Express RC Inbound Window Base Address Register High *n* (PEX_RCIWBARH0–PEX_RCIWBARH3)

PEX_RCIWBARH0–PEX_RCIWBARH3, shown in Figure 15-138, contain the higher-portion base address of the PCI Express domain corresponding to this window. This register should be used in 64-bit addressing. Otherwise, it should contain all zeroes.

Offset 0xE6C, 0xE7C, 0xE8C, 0xE9C Access: Read/Write

	31–0
R/W	
Reset	All zeros

Figure 15-138. CI Express RC Inbound Window Base Address Register High *n* (PEX_RCIWBARH0–PEX_RCIWBARH3)

Table 15-137 defines the bit fields of PEX_RCIWBARH*n*.

Table 15-137. PEX_RCIWBARH*n* Register Fields Description

Bits	Name	Description
31–0	BAH	Base address high. Higher portion of the PCI Express address base ([63:32]). The complete 64-bit address on the PCI Express bus is built of {PEX_RCIWBARH[BAH],PEX_RCIWBARL[BAL], 0b0000000000}.

15.6 Functional Description

The PCI Express protocol relies on a requestor/completer relationship in which one device requests that a target device perform an action, and the target device completes the task and responds. Usually, the requests and responses occur through a network of links, but to the requestor and to the completer, the intermediate components are transparent.

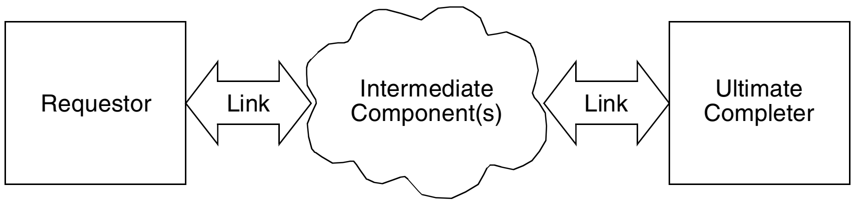


Figure 15-139. Requestor/Completer Relationship

Each PCI Express device is divided into two halves, transmit (Tx) and receive (Rx), and each of these halves is further divided into three layers—transaction, data link, and physical—as shown in Figure 15-140.

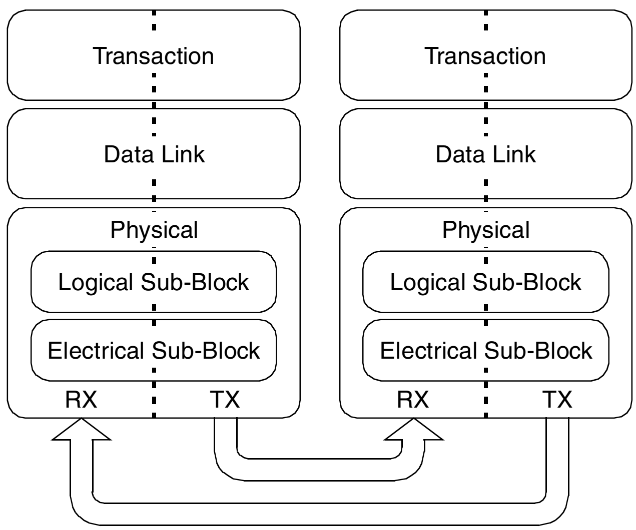


Figure 15-140. PCI Express High-Level Layering

Packets are formed in the transaction layer (TL) and data link layer (DLL), and each subsequent layer adds the necessary encoding and framing. As packets are received, they are decoded and processed by the same layers but in reverse order, so they may be processed by the layer or by the device application software, as shown in Figure 15-141.

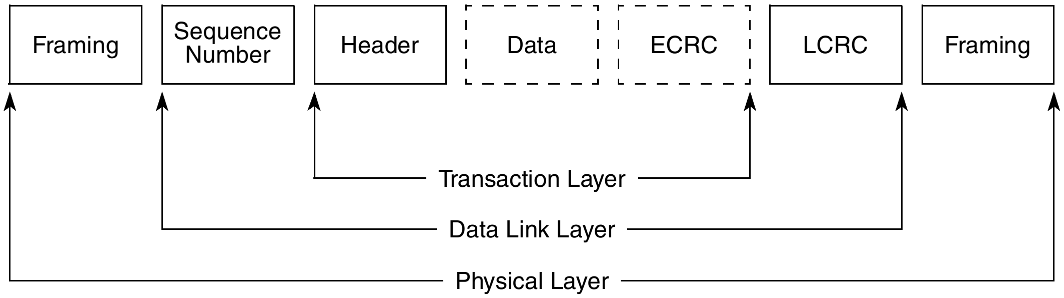


Figure 15-141. PCI Express Packet Flow

15.6.1 Architecture

15.6.1.1 Address Translation Windows (ATMUs)

The device includes four general purpose inbound and outbound address translation windows which are used to map between the PCI Express domain and the CSB domain (referred as local address space). The outbound windows are common for both Root Complex and End Point modes, and their configuration registers reside in the CSB bridge space. For Inbound windows there is a different set of configuration registers between Root Complex and End Point modes. EP inbound base address registers (BARs) reside in the standard configuration header space of the PCI Express core, while RC inbound base address registers (BARs) reside in the CSB bridge space. Since the CSB domain supports only 32 bit addressing, outbound base address registers and inbound translation registers are 32 bit only. Table 15-138 specifies the available combinations between base address registers and translation address registers.

Table 15-138. Address Translation Window Combinations

Window Number	Type	BAR Name	BAR Address	TAR Name	TAR Address
Outbound Windows—Common for Root Complex and End Point Modes					
0	64-bit address, low portion	PEX_OWBAR0	0x9CA4	PEX_OWTARL0	0x9CA8
	64-bit address, high portion			PEX_OWTARH0	0x9CAC
1	64-bit address, low portion	PEX_OWBAR1	0x9CB4	PEX_OWTARL1	0x9CB8
	64-bit address, high portion			PEX_OWTARH1	0x9CBC
2	64-bit address, low portion	PEX_OWBAR2	0x9CC4	PEX_OWTARL2	0x9CC8
	64-bit address, high portion			PEX_OWTARH2	0x9CCC
3	64-bit address, low portion	PEX_OWBAR3	0x9CD4	PEX_OWTARL3	0x9CD8
	64-bit address, high portion			PEX_OWTARH3	0x9CDC
Inbound Windows—End Point Mode					
0	32-bit address	BAR0	0x010	PEX_EPIWTAR0	0xDE0

Table 15-138. Address Translation Window Combinations

Window Number	Type	BAR Name	BAR Address	TAR Name	TAR Address
1	32-bit address	BAR1	0x014	PEX_EPIWTAR1	0xDE4
2	64-bit address, low portion	BAR2	0x018	PEX_EPIWTAR2	0xDE8
	64-bit address, high portion	BAR3	0x01C		
3	64-bit address, low portion	BAR4	0x020	PEX_EPIWTAR3	0xDEC
	64-bit address, high portion	BAR5	0x024		
Inbound Windows—Root Complex Mode					
0	64-bit address, low portion	PEX_RCIWBARL0	0x9E68	PEX_RCIWTAR0	0x9E64
	64-bit address, high portion	PEX_RCIWBARH0	0x9E6C		
1	64-bit address, low portion	PEX_RCIWBARL1	0x9E78	PEX_RCIWTAR1	0x9E74
	64-bit address, high portion	PEX_RCIWBARH1	0x9E7C		
2	64-bit address, low portion	PEX_RCIWBARL2	0x9E88	PEX_RCIWTAR2	0x9E84
	64-bit address, high portion	PEX_RCIWBARH2	0x9E8C		
3	64-bit address, low portion	PEX_RCIWBARL3	0x9E98	PEX_RCIWTAR3	0x9E94
	64-bit address, high portion	PEX_RCIWBARH3	0x9E9C		

The attributes of the outbound windows are controlled by the PEX_OWAR*n* registers and the attributes of the RC inbound windows are controlled by the PEX_RCIWAR*n* registers, residing in the CSB bridge address space. The attributes of the EP inbound windows are controlled by both the translation registers, PEX_EPIWTAR*n*, in the CSB bridge space, and by the PCI Express BAR configuration registers residing in the PCI Express core address space.

NOTE

For outbound transactions, both the PCI Express DMA and CSB Masters share the same set of windows.

15.6.1.2 PCI Express Transactions

Table 15-139 lists the PCI Express transactions supported by the device as an initiator and a target.

Table 15-139. PCI Express Transactions

PCI Express Transaction	MPC8378E/MPC8377E Support as an Initiator	MPC8378E/MPC8377E Support as a Target	Definition
Mrd	Yes	Yes	Memory Read Request
MRdLk	No	No	Memory Read Lock
MWr	Yes	Yes	Memory Write Request to memory-mapped PCI-Express space
IORd	Yes (RC only)	No	I/O Read request

Table 15-139. PCI Express Transactions (continued)

PCI Express Transaction	MPC8378E/MPC8377E Support as an Initiator	MPC8378E/MPC8377E Support as a Target	Definition
IOWr	Yes (RC only)	No	I/O Write Request
CfgRd0	Yes (RC only)	Yes	Configuration Read Type 0
CfgWr0	Yes (RC only)	Yes	Configuration Write Type 0
CfgRd1	Yes (RC only)	No	Configuration Read Type 1
CfgWr1	Yes (RC only)	No	Configuration Write Type 1
Msg	Yes	Yes	Message Request
MsgD	No	No	Message Request with data payload
Cpl	Yes	Yes	Completion without Data
CplD	Yes	Yes	Completion with Data
CplLk	No	No	Completion for Locked Memory Read without Data
CplDLk	No	No	Completion for Locked Memory Read with Data

15.6.1.3 Byte Swapping

Byte swapping is executed when either an inbound transaction (that is, a write or response) to the CSB or an outbound transaction with data (that is, a write) to PCI Express is performed.

15.6.1.4 Outbound Byte Swapping

Byte swapping is executed when an outbound transaction with data (that is, a write) to PCI Express is performed. Figure 15-142 shows the outbound data swapping.

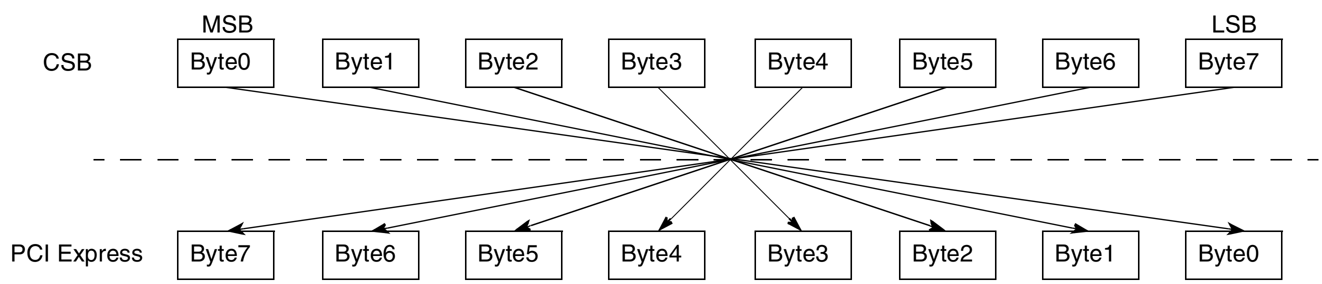


Figure 15-142. Outbound Byte Swapping

15.6.1.5 Inbound Byte Swapping

Byte swapping is executed when an inbound transaction with data (that is, a write or response) to the CSB is performed.

15.6.1.6 Transaction Ordering Rule

In general, transactions are serviced in the order that they were received. However, transactions can be reordered as they are sent due to stalled conditions such as the internal buffer full condition. Below are the ordering rules for sending the next outstanding request.

- A posted request can and will bypass all other transactions except another posted request.
- Completion can and will bypass non-posted and posted requests only if the relaxed ordering (RO) bit of the PCI Express packet's header is set.
- A non-posted request cannot bypass a posted or non-posted request but can bypass completion if the relaxed ordering bit is set.

Note that it is possible for one outbound configuration transaction to bypass another outbound configuration transaction due to CRS status and the ability for hardware to retry the transaction.

15.6.1.7 Memory Space Addressing

A PCI Express memory transaction can address a 32- or 64-bit memory space. The Fmt[0] field in the PCI Express header for a 32-bit address packet is 0 while a 64-bit address packet has a 1 indication. A memory read transaction has settings of 00000 for the Type[4:0] field in the PCI Express header and 0 for Fmt[1]. A memory write transaction has the Type[4:0] field in the PCI Express header as 00000 and Fmt[1] as 1. As an initiator, the controller is capable of sending a 32- or 64-bit memory packet depending on the window translation address. Any transaction from the CSB that has a translated address greater than 4G after going through the translation window is sent as a 64-bit memory packet. Otherwise, a 32-bit memory packet is sent. As a target device, the controller can decode a 32- or 64-bit memory packet using two 32-bit inbound windows and two 64-bit inbound windows. All inbound addresses are translated to the CSB address, which is 32 bits wide.

15.6.1.8 I/O Space Addressing

The PCI Express controller does not support I/O transactions as a target. As an initiator, the controller can send I/O transactions in RC mode by programming one of the outbound translation window's attributes to send I/O transactions. All I/O transactions access only a 32-bit address I/O space. An I/O read transaction has the Type[4:0] field in the PCI Express header as 00010 and the Fmt[1] as 0. An I/O write transaction has the Type[4:0] field in the PCI Express header as 00010 and the Fmt[1] as 1.

15.6.1.9 Configuration Space Access

To access the PCI Express controller's internal configuration header by MPC8379E itself, the only mechanism supported is the direct access via the CSB, since the whole internal configuration space is memory-mapped. This is true regardless the PCI Express controller is configured as RC or EP.

If the PCI Express controller is configured as RC,

- Inbound configuration transaction is not supported.
- Outbound configuration transaction to access downstream PCI Express devices is supported. The only mechanism can be used to initiate either Type 0 or Type 1 configuration cycle is via outbound ATMU windows.

If the PCI Express controller is configured as EP,

- Outbound configuration transaction is not supported. In other words, the PCI Express EP controller does not generate configuration transactions in EP mode.
- Inbound configuration transaction to access the PCI Express EP controller's configuration space is supported.

15.6.1.9.1 Outbound ATMU Configuration Transaction Generation (RC)

In RC mode, the PCI Express controller can generate both Type 0 and Type 1 configuration cycles to access downstream PCI Express devices via the outbound ATMU windows mechanism.

As RC the PCI Express controller configuration access mechanism utilizes a memory-mapped address space to access device configuration registers. To achieve this, software can program the TYPE field of the PEX_OWARn register to 0x0 in one of the outbound ATMU windows to perform a configuration access. The other bit fields of the PEX_OWARn register should be programmed as below:

- TC = 0x0;
- NSNP = 0;
- RLXO = 0;
- EN = 1;

The SIZE field of the PEX_OWARn register should be set to a value to correspond with the BA (base address) field of the base address register (PEX_OWBARn), normally based on the Bus Number(s) of the downstream PCI Express device(s) to be accessed. The base address registers, PEX_OWBARn, set the CSB address window for the configuration transactions. The translation address registers, PEX_OWTARLn, can be used to define the translated PCI Express address of the CSB-based configuration transaction.

Once the PCI Express outbound window attributes register (PEX_OWARn), base address register (PEX_OWARn) and translation address register (PEX_OWTARLn) are fully defined, a CSB-based memory transaction hitting the defined base address register will be converted to an external PCI Express configuration transaction cycle appeared on the downstream link of the PCI Express RC controller. In this case, the CSB memory address determines the configuration register accessed and the memory data returns the contents of the addressed register. Software must only issue 4-byte or less access to the ATMU configuration window and the access cannot cross a 4-byte boundary.

The mapping from the CSB local address to PCI Express configuration space is defined by the Outbound ATMU as in a memory transaction. The actual PCI Express configuration header will be formatted from the CSB address and the ATMU translation, defined by PEX_OWTARLn.

The formatted address defines the configuration transactions parameter, as shown in Table 15-140. Note that there is no byte swapping for the address itself, although the programming of the registers content do require byte swapping. The table also translates between the bit ordering commonly used by the Power PC terminology (0:31) and the bit ordering used by the PCI Express terminology (31:0).

Table 15-140. Configuration Address Mapping

CSB Address Bits Numbering	PCI Express Address Bits Numbering	PCI Express Configuration Space
0:7	31:24	Bus number
8:12	23:19	Device number
13:15	18:16	Function number
16:19	15:12	Reserved
20:23	11:8	Extended register number
24:29	7:2	Register number
30:31	1:0	Reserved

Note: It is the user's responsibility to set the reserved bit fields to zero (especially bits 15–12).

Note: The configuration cycle generation mechanism does not differentiate from internal or external configuration cycle. This means that any transaction which hits a configuration window will be passed to the PCI Express link with a relevant transaction type.

NOTE

The location of the configuration fields described in the table above are slightly different than the definition of a "flat memory addressing method" in the PCI Express specification, and are directly aligned to the fields in the configuration transaction header. The user software should take this difference into considerations.

The PCI Express RC controller initiates the Type 0 or Type 1 configuration cycle on its downstream link based on the following rules:

- If the bus number of the CSB-initiated transaction equals the secondary bus number from Type 1 header of RC's configuration space and the device number is 0, a Type 0 configuration cycle will be sent to the link.
- If the bus number of the CSB-initiated transaction does not equal the RC's primary bus number, and does not equal the secondary bus number (from RC's Type 1 header), and is less than or equal to the subordinate bus number (from RC's Type 1 header), a Type 1 configuration cycle will be sent to the link. Note that according to PCI and PCI Express base specifications, the relationship where the Secondary Bus Number ≤ Subordinate Bus Number must be ensured when configuring the two bus numbers within RC's Type 1 header.
- For all other cases, the PCI Express RC controller will issue a configuration cycle on the link whenever an outbound configuration window is hit on the CSB side, no matter what the parameters are. It is the software driver's responsibility to block transactions with unsupported bus, device, and function numbers and return "1"s for such reads. If the bus number in the CSB-initiated transaction equals the primary bus number of RC hitting the outbound configuration window, software error will occur which must be handled by the driver.

The following is an example showing how to configure the related registers of one of the MPC8379E outbound windows for configuration transaction generation purpose. Since MPC8379E's default 8 Mbyte boot ROM location can be configured at either 0x0000_0000 to 0x007F_FFFF (at bottom 8-MByte local address) or 0xFF80_0000 to 0xFFFF_FFFF (at top 8 Mbyte local address), the software must ensure that the base address is configured correctly in the OWBARn such that the outbound window does not overlap with the configured boot ROM location. To simplify the illustration, this example has the following assumption:

- The Boot ROM location is configured to locate within 0x0000_0000 to 0x007F_FFFF.
- The overall PCI Express system has a total of 16 buses to be configured.

Sixteen buses mean that the bus number can range from 0x00 to 0x0F. In general, if there are 2^n bus numbers to be configured, n number of address bits are needed to represent the variation of bus number ranging from 0 to $2^n - 1$. For this example, four address bits from CSB[4:7] are required to represent the bus number variation and therefore become the most significant four bits within the total 28 bit offset (CSB[4:31], including reserved bits) of the outbound window to be configured. The CSB[0:3] in this case will not participate in the bus number mapping process as shown in Table 15-140, where all eight bits of CSB[0:7] are mapped to PCI Express address bits [31:24] to represent the possible of 256 bus numbers within a very large system. In other words, in the example, CSB[0:3] become the four most significant bits of the base address of the outbound window to be configured and will be translated to a new "address" in the PCI Express space as defined by the corresponding PEX_OWTARL*n*.

Based on the above information, the most significant four bits of the base address of the outbound window came from CSB[0:3] must be unique within the total 4 Gbyte local CSB memory space. This essentially defines a window with size of 256 Mbytes, since the lower 28 bits are offset. For this example, assume a 256 Mbyte outbound window is therefore defined between 0x5000_0000 and 0x5FFF_FFFF in local CSB space. This yields the PEX_OWBAR*n*'s BA[31:12] to be 0x5000_0, with the actual base address of the outbound window as 0x5000_0000. The SIZE[31:12] field of the PEX_OWARn register is 0x1000_0 to reflect the actual size of this outbound window as 0x1000_0000 or 256 Mbytes.

Note that the final configuration transaction to be generated is based on the information gathered from two areas: the most significant four bits from the defined TAL (translation address low) bit field of PEX_OWTARL*n* and the lower order bits from the CSB[4:31] offsets directly mapped to PCI Express address bits [27:0]. As mention in the note section of Table 15-140, software should ensure all the reserved bits within CSB[4:31] are filled with zeros.

Since TAL[31:24] (total of eight bits) of PEX_OWTARLn could be used for mapping the bus number bit field for a general configuration transaction, while in this example the bus number to be configured ranges from 0x00 to 0x0F, the most significant four bits (TA[31:28]) are not needed for the bus number mapping and therefore must be configured as 0x0. The TA[27:12] bit field of PEX_OWTARLn is left as all zeros. This yields the PEX_OWTARL*n*'s TAL[31:12] to be 0x0000_0, with the actual translation address of the outbound window as 0x0000_0000. Since the size of this window is 256 Mbytes, the upper limit of the translation address is then locate at 0x0FFF_FFFF.

With the outbound window configured as above, if software intends to scan the PCI Express bus and attempts to probe the first bus immediately underneath the PCI Express RC, the software will need to issue a Configuration Transaction to read Register Number 0 (Vendor ID Register) at Bus Number/Device Number/Function Number of 1/0/0, assuming the RC's secondary bus number has been configured as 0x1 along with subordinate bus number being configured with a big number like 0xFF initially. As long as the LAW is configured to ensure that the local address space between 0x5000_0000 and 0x5FFF_FFFF is configured for PCI Express, a CSB-based memory read transaction to local address 0x5100_0000 initiated by software will be translated to a transaction hitting PCI Express RC controller with PCI Express address of 0x0100_0000. Upon receiving this CSB-based transaction, the RC controller checks the Type attribute of this outbound window and realizes the transaction is of a configuration type, instead of directly using the translated address as in usual memory transaction, the RC controller starts compose a configuration transaction with header information from this translation address based on the mapping defined in Table 15-140. The decode of the mapping process yields a type 0 configuration transaction to be generated on the PCI Express link with Bus Number=1, Device Number=0, Function Number=0, and Register Number=0.

Similarly, if the software intends to read the above EP's configuration space offset 0x440, it can generate a CSB-based memory transaction to address 0x5100_0440. Once the transaction hitting the above configured outbound window, the ATMU translates it into a transaction with PCI Express address 0x0100_0440. This RC controller, once receiving the transaction, will issue a Type 0 configuration transaction on the PCI Express link with Bus Number=1, Device Number=0, Function Number=0, Extended Register Number=0x4, and Register Number=0x40.

NOTE

In the example above the translation address register (PEX_OWTARLn) is set once. It is also possible to use a dynamic approach of updating the translation address register before every configuration access. In this method the PEX_OWBARn and the PEX_OWARn for the configuration window can be set for a relatively small address range, but the software needs to adjust the translation address register (PEX_OWTARn) for the configuration window to the desired parameters prior to issuing the configuration transaction.

The programming of the ATMU registers must guarantee that there is no overlap between address bits defined by the base address and size of the window, and address bits defined by the translation address. In other words, the bits in the lower portion of the PEX_OWTARLn covered by the base address must be zero.

15.6.1.9.2 EP Configuration Register Access

When the PCI Express controller is configured as an EP device it responds to remote host generated configuration cycles. This is indicated by decoding the configuration command along with type 0 access in the packet. A remote host can access up to 4096 bytes of the PCI Express configuration area. While in EP mode, the PCI Express controller does not support generating configuration accesses as a master. There is no configuration mechanism supported in EP mode using the ATMU window. If the outbound ATMU window is configured to issue a configuration transaction, all posted transactions hitting this window are ignored and all non-posted transactions will get a response with an error and can lead to unexpected results.

15.6.1.10 Messages

The following tables lists the messages and the actions that take place depending on whether RC or EP mode is configured. The actual events are logged in the PCI Express Root Error Status Register and in the CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR). See Section 15.4.5.10, "PCI

Express Root Error Status Register, Section 15.5.8.4, "CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER) and Section 15.5.8.8, "CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR) for further details.

Table 15-141 lists the messages and the actions that take place in RC mode.

Table 15-141. PCI Express RC Inbound Message Handling

Name	Code[7:0]	Routing[2:0]	Action
Assert_INTA	0010 0000	100	Set INTA event
Assert_INTB	0010 0001	100	Set INTB event
Assert_INTC	0010 0010	100	Set INTC event
Assert_INTD	0010 0011	100	Set INTD event
Deassert_INTA	0010 0100	100	Clear INTA event
Deassert_INTB	0010 0101	100	Clear INTB event
Deassert_INTC	0010 0110	100	Clear INTC event
Deassert_INTD	0010 0111	100	Clear INTD event
PM_Active_State_Nak	0001 0100	100	No action taken
PM_PME	0001 1000	000	Set PM_PME event
PME_Turn_Off	0001 1001	011	No action taken
PME_TO_Ack	0001 1011	101	Log entered_l23_state in PME and message detect register and generate interrupt to IPIC if enabled
ERR_COR	0011 0000	000	Set correctable error event
ERR_NONFATAL	0011 0001	000	Set non-fatal error event
ERR_FATAL	0011 0011	000	Set fatal error event
Unlock	0000 0000	000	No action taken
Set_Slot_Power_Limit	0101 0000	100	No action taken
Vendor_Defined Type 0	0111 1110		No action taken
Vendor_Defined Type 1	0111 1111		No action taken
Attention_Indicator_On	0100 0001	100	No action taken
Attention_Indicator_Blink	0100 0011	100	No action taken
Attention_Indicator_Off	0100 0000	100	No action taken
Power_Indicator_On	0100 0101	100	No action taken
Power_Indicator_Blink	0100 0111	100	No action taken

Table 15-141. PCI Express RC Inbound Message Handling (continued)

Name	Code[7:0]	Routing[2:0]	Action
Power_Indicator_Off	0100 0100	100	No action taken
Attention_Button_Pressed	0100 1000	100	Log in PME and message detect register if enabled. Send interrupt if enabled.

Table 15-142 lists the messages and the actions that take place in EP mode.

Table 15-142. PCI Express EP Inbound Message Handling

Name	Code[7:0]	Routing[2:0]	Action
Assert_INTA	0010 0000	100	No action taken
Assert_INTB	0010 0001	100	No action taken
Assert_INTC	0010 0010	100	No action taken
Assert_INTD	0010 0011	100	No action taken
Deassert_INTA	0010 0100	100	No action taken
Deassert_INTB	0010 0101	100	No action taken
Deassert_INTC	0010 0110	100	No action taken
Deassert_INTD	0010 0111	100	No action taken
PM_Active_State_Nak	0001 0100	100	No action taken
PM_PME	0001 1000	000	No action taken
PME_Turn_Off	0001 1001	011	1. Log in PME and message detect register if enabled. Send interrupt if enabled.
PM_TO_Ack	0001 1011	101	No action taken
ERR_COR	0011 0000	000	No action taken
ERR_NONFATAL	0011 0001	000	No action taken
ERR_FATAL	0011 0011	000	No action taken
Unlock	0000 0000	000	No action taken
Set_Slot_Power_Limit	0101 0000	100	No action taken
Vendor_Defined Type 0	0111 1110	—	No action taken
Vendor_Defined Type 1	0111 1111	—	No action taken
Attention_Indicator_On	0100 0001	100	Log in PME and message detect register if enabled. Send interrupt if enabled.
Attention_Indicator_Blink	0100 0011	100	Log in PME and message detect register if enabled. Send interrupt if enabled.
Attention_Indicator_Off	0100 0000	100	Log in PME and message detect register if enabled. Send interrupt if enabled.

Table 15-142. PCI Express EP Inbound Message Handling (continued)

Name	Code[7:0]	Routing[2:0]	Action
Power_Indicator_On	0100 0101	100	Log in PME and message detect register if enabled. Send interrupt if enabled.
Power_Indicator_Blink	0100 0111	100	Log in PME and message detect register if enabled. Send interrupt if enabled.
Power_Indicator_Off	0100 0100	100	Log in PME and message detect register if enabled. Send interrupt if enabled.
Attention_Button_Pressed	0100 1000	100	No action taken

15.6.2 Interrupts

Both INTx and Message Signaled Interrupt (MSI) generation and handling are supported; however there are subtle differences depending on whether the device is configured as an RC or EP device.

15.6.2.1 EP Interrupt Generation

Both hardware INTx messages generation and Hardware MSI generation are supported, for the interrupt events described in Section 15.5.7, "PCI Express Host Interrupt Registers." The INTx message and MSI mechanism are mutually exclusive. Only one of these two mechanisms should be enabled and used at a given time.

15.6.2.1.1 Hardware INTx Message Generation

Hardware INTx message is generated when any interrupt event occurs and the corresponding interrupt is enabled in PEX_HIER register, if the interrupt disable bit is cleared in the PCI Express EP's command register (see Section 15.4.1.3, "PCI Express Command Register") and the MSI interrupt mechanism is not enabled.

Only INTA message is supported by this device.

Note that MSI interrupt mechanism will be used as long as it is enabled, regardless the setting of the interrupt disable bit in the PCI Express EP's command register.

15.6.2.1.2 Software INTx Message Generation

Software INTx message generation is not supported.

15.6.2.1.3 Hardware MSI Generation

Host software must set up the MSI capability registers to enable MSI mode and put the correct MSI address and data values into the MSI capability registers prior to setting up various interrupt event enable bits in the PEX_HIER register to enable the generation of the correct MSI cycle to RC.

Note that the value being programmed by the host software into the MSI Data register of EP's Type 0 configuration space is a 16-bit base message data pattern, which is referred to as "base vector [15:0]" in the description below:

- If only one MSI message is desired, the EP software can directly use this “base vector” as the interrupt vector for MSI interrupt generation. In such case, there is no need to program any of the “vector registers” among PEX_HOPIVR, PEX_HIPIVR, PEX_HWDIVR, PEX_HRDIVR and PEX_HMIVR. The PEX_HIER register setting determines which event can trigger this single MSI interrupt message to RC. Please note that multiple interrupt events are allowed to share the same MSI interrupt vector.
- If multiple MSI messages are desired, the Multiple Message Capable bit field of EP’s MSI Message Control Register can be used to indicate how many MSI messages (in the power of two, up to 32 messages allowed per EP) the EP wants to use. During configuration stage, after examining the above desired value, the Host software will allocate the actual number of MSI messages for an EP to use by configuring the Multiple Message Enable bit field in the same register, in addition to programming the “base vector” in EP’s MSI Data Register. Once this is accomplished, the EP software can program the IVEC bit field of each individual “vector register” based on the number of MSI messages allocated by host. The IVEC value must be unique for the same EP and start from 0x00. At last, the EP software can set the corresponding interrupt event enable bits in the PEX_HIER to enable the Hardware MSI generation. The actual MSI data value for a given interrupt event used by the EP in its MSI message to RC is formed by the concatenation of the “base vector [15:5]” and the IVEC [4:0] value of the corresponding interrupt event.

As an example, if the value of the Multiple Message Enable bit field allocated by host software is 010b (4 MSI messages allocated) and the “base vector” being programmed by host software in EP’s MSI Data Register is 0x55A0 (lower-16bits little endian), the actual MSI data values of all possible MSI messages can be used by the EP are 0x0000_55A0, 0x0000_55A1, 0x0000_55A2, and 0x0000_55A3. The EP software only needs to program the four possible lower-order bits (0x00, 0x01, 0x02, and 0x03) as unique IVEC values in its “vector registers” among PEX_HOPIVR, PEX_HIPIVR, PEX_HWDIVR, PEX_HRDIVR and PEX_HMIVR. If both OPAIE and OPCIE bit fields are enabled in the PEX_HIER register, assuming PEX_HOPIVR register’s IVEC bit field is programmed as 0x03h, when any one of these two interrupt events occurs, the EP will use 0x0000_55A3 as the actual MSI data value in its MSI message sent to RC. Once receiving such MSI message, the device driver running at RC is responsible to issue a read to EP’s PCI Express Interrupt Status Register (PEX_HISR) to find out exactly which of the two interrupt events caused the interrupt.

15.6.2.1.4 Software MSI Generation

Host software needs to set up the MSI capability registers to enable MSI mode and put the correct values for the MSI address and data register. Next, local software must read the MSI address in the MSI capability register and configure the outbound ATMU window to map the translated address to the MSI address. Software determines the number of allocated messages in the MSI capability register and allocates the appropriate data values to use. A write to the MSI ATMU window with the appropriate data value generates the MSI transaction to the RC.

15.6.2.2 RC Handling of INTx Message and MSI Interrupt

15.6.2.2.1 INTx Message Handling

MSIs are the preferred interrupt signaling mechanism for the PCI Express. However, in RC mode, the PCI Express controller supports the INTx virtual-wire interrupt signaling mechanism (as described in the *PCI Express Base Specification*). When the controller receives an inbound INTx asserted message, it sets the appropriate INTA, INTB, INTC, or INTD bit in the CSB system miscellaneous interrupt status register (PEX_CSMISR). An interrupt is generated to the local host if the interrupts are enabled in the CSB system miscellaneous interrupt enable register (PEX_CSMIER) and the IPIC.

15.6.2.2.2 MSI Handling

An MSI interrupt cycle must hit into the IMMRBAR window (window 0) with the address offset that points to MIISR register in the IPIC. Note that the host software must configure the EP's MSI capability register so that an MSI cycle generated from the device is routed to the correct MIISR register in the IPIC and for the appropriate interrupt to be generated to the core.

15.6.2.3 Initial Credit Advertisement

To prevent overflowing of the link partner's receiver buffers and for compliance with ordering rules, the transmitter cannot send transactions unless it has enough credits to send. Each device maintains a flow control (FC) credit pool. The FC information is conveyed between two links by DLLPs during link training (initial credit advertisement). The transaction layer performs the FC accounting functions. It functions as the FC gate. One unit of FC is 4 DWs (16 bytes) of data.

Table 15-143. Initial Credit Advertisement

Credit Type	Initial Credit Advertisement
PH (memory write, message write)	4
PD (memory write, message write)	$(256 \div 16) \times 4 = 64$
NPH (memory read, I/O read, cfg read)	8
NPD (I/O write, cfg write)	2
CPLH (memory read completion, I/O R/W completion, cfg R/W completion)	Infinite
CPLD (memory read completion, I/O read completion, cfg read completion	Infinite

15.6.3 Mailbox

The Mailbox mechanism is useful when the device is in EP mode, and enables exchanging information between the remote PCI Express root complex and the local host using interrupts and data storage registers. There are sets of register for both inbound and outbound mailbox messages and interrupts.

15.6.3.1 Outbound Mailbox

The EP local host uses the outbound mailbox messages to signal the remote RC device across the PCI Express link. The local host, for example the e300 core, stores the required message in the PCI Express outbound mailbox data register (PEX_OMBDR) and then initiates an interrupt (MSI) to the remote PCI Express device. When the remote PCI Express RC device receives the interrupt it can perform a memory read from the mailbox data register and read the message. The following steps are required in order to use the outbound mailbox mechanism.

1. The RC should set the MSIE bit of the PCI Express MSI message control register (address 0x72 of the configuration space) to enable the generation of MSI.
2. The EP local host should set the OMBIE bit of the PCI Express host interrupt enable register (PEX_HIER) to enable interrupt generation to the PCI Express (MSI) at the event of setting the READY bit of the PCI Express outbound mailbox control register (PEX_OMBCR).
3. The EP local host should program the IVEC field of the PCI Express host miscellaneous interrupt vector register (PEX_HMIVR) with an appropriate vector value. This value, along with the value programmed in the EP's PCI Express MSI message control register's MME field determines the MSI message data sent to the RC. For example, if the MME field has a value of N, then the lower N bits of the MSI message data are replaced with the lower N bits of the PEX_HMIVR register.
4. The EP local host should program the MBD field of the PCI Express outbound mailbox data register (PEX_OMBDR) with the message to be read by the PCI Express remote device (user defined).
5. The EP local host should set the READY bit of the PCI Express outbound mailbox control register (PEX_OMBCR). This will generate an interrupt (MSI) to the PCI Express root complex.
6. The PCI Express RC, after receiving the MSI will perform a memory read to the EP's PCI Express outbound mailbox data register (PEX_OMBDR) and get the message content.
7. The PCI Express RC should perform a memory write to clear the READY bit of the EP's PCI Express outbound mailbox control register (PEX_OMBCR).
8. The EP can repeat steps 3–5 with an appropriate MSI and message data after verifying that the PEX_OMBCR[READY] is cleared.

15.6.3.2 Inbound Mailbox

The remote PCI Express RC device uses the inbound mailbox messages to signal the EP local host across the PCI Express link. The RC performs a memory write and stores the required message in the PCI Express inbound mailbox data register (PEX_IMBDR) and then initiates an interrupt to the local host by performing a memory write and setting the READY bit of the PCI Express inbound mailbox control register (PEX_IMBCR). When the local host detects the interrupt it can read the message from the mailbox data register. The following steps are required in order to use the outbound mailbox mechanism.

1. The EP local host should enable PCI Express interrupts by programming the integrated programmable interrupt controller (IPIC).
2. The EP local host should set the IMBIE bit of the CSB system miscellaneous interrupt enable register (PEX_CSMIER), to allow interrupt at the event of mailbox ready.
3. The PCI Express RC should perform a memory write and store the required message in the EP's PCI Express Inbound Mailbox Data Register (PEX_IMBDR).
4. The PCI Express RC should perform a memory write and set the READY bit of the EP's PCI Express inbound mailbox control register (PEX_IMBCR). This will issue an interrupt to the local host.

5. The local host can read the message content from the PCI Express inbound mailbox data register (PEX_IMBDR).
6. The local host should clear PEX_IMBCR[READY] and all the interrupt event and status bits in the relevant registers.
7. The RC can repeat steps 3–4 after verifying that the PEX_IMBCR[READY] is cleared.

15.6.4 Power Management

All device power states are supported except D3cold. In addition, all link power states are supported except the L2 and L3 states. Only L0s ASPM (active state power management) mode is supported if enabled by configuring the link control register bits 1–0 in configuration space. Note that there is no power saving in the controller when the device is put into a non-D0 state. The only power saving is the I/O drivers when the controller is put into a non-L0 link state.

Table 15-144. Power Management State Supported

Component D-State	Permissible Interconnect State	Action
D0	L0, L0s	In full operation.
D1	L1	All outbound traffic is stopped. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, a PM_Turn_Off message can be sent through the PCI Express power management command register.
D2	L1	All outbound traffic is stopped. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, a PM_Turn_Off message can be sent through the PCI Express power management command register.
D3hot	L1, L2/L3 ready	All outbound traffic is stopped. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, a PM_Turn_Off message can be sent through the PCI Express power management command register. Note that if a transition of D3 → D0 occurs, a reset is performed to the controller configuration space. In addition, link training restarts.
D3cold	Not supported	Not supported.

15.6.4.1 L2/L3 Ready Link State

The L2/L3 Ready link state is entered after the EP device is put into a D3hot state followed by a PME_Turn_Off/PME_TO_Ack message handshake protocol. Exiting this state requires a POR or a detection of a beacon or a WAKE# signal from the EP device. The PCI Express controller as an EP device does not support the generation of beacon, so the device can alternatively use one of the GPIO signals as an enable to an external tristate buffer to generate the WAKE# signal if the device needs to wake up from an L2/L3 Ready state. As an RC device, the WAKE# signal from the EP device can be connected to one of the external interrupt input pins to service the WAKE# request if needed.

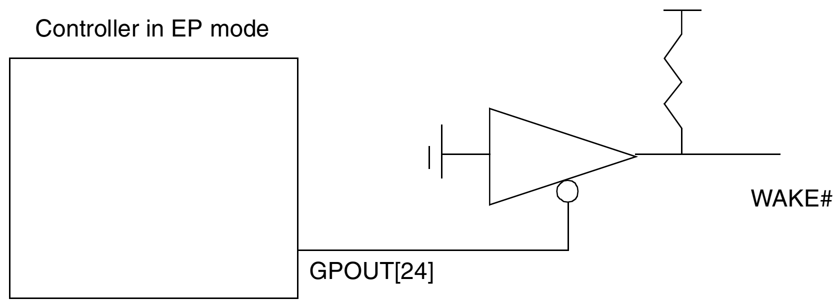


Figure 15-143. Example on How to Generate WAKE#

15.6.5 Hot Reset

When a hot reset condition occurs, the controller (in both RC and EP mode) initiates a cleanup of all outstanding transactions and goes into suspend mode. The RC controller then needs to be reset (issuing CBRST in PECR1/PECR2) to bring it back to the idle state followed by a reprogramming of all the CSB bridge registers (offset 0x800–0xFFC, such as the ATMU windows). The EP controller also needs to be reset, followed by a reprogramming of the entire configuration space and CSB bridge registers (offset 0x800–0xFFC, such as the ATMU windows). All configuration register bits that are non-sticky are reset. Link training takes place subsequently. The device is permitted to generate a hot reset condition on the bus when it is configured as an RC device by setting the secondary bus reset bit in the bridge control register in the configuration space. In EP mode, the device is not permitted to generate a hot reset condition; it can only detect a hot reset condition and initiate the cleanup procedure appropriately.

15.7 Initialization/Application Information

The following sections describe initialization sequences for root complex (RC) and end point (EP) modes.

15.7.1 Initialization Sequence

The following sequence must be followed. Note the specific stages for RC or EP modes.

1. The device performs its power-on reset sequence. The PCI Express controller and the SerDes PHY are held in reset (controlled by memory mapped registers).
2. Program the system configuration registers of the device, including some PCI Express controller related options (local memory windows, clock ratio, etc.) See the system configuration registers in Section 5.2.3.1, “Local Access Register Memory Map,” and the clock configuration registers in Section 4.5.2, “Clock Configuration Registers.”
3. Set the protocol to PCI Express, the number of lanes, and the reference clock in the SRDSCR4 register. See Section 19.3.5, “SerDesn Control Register 4 (SRDSnCR4).” In addition, set any SerDes electrical and functional parameters appropriate for PCI Express operation as described in Chapter 19, “SerDes PHY”.
4. Start the SerDes reset sequence by setting RST field (bit 0) in the SerDes reset control register (SRDSRSTCTL). See Section 19.3.6, “SerDesn Reset Control Register (SRDSnRSTCTL).”
5. Poll RDONE field (bit 1) in the SerDes reset control register (SRDSRSTCTL).

6. After RDONE is set, wait at least 1 ms.
7. Program the PCI Express control registers 1 and 2. See Section 5.3.2.11, “PCI Express Control Registers (PECR1 and PECR2).” Set the DEV_TYPE field, to select between EP or RC, take the PCI Express controller out of reset by setting bits [0:2] and optionally select the parameters in the PRI_DATA, PRI_DES, and PRI_PIO fields.
8. Configure the PCI Express core and CSB bridge control registers and address mapping windows to the desired values (that is, inbound/outbound windows).
9. Poll the Status Code field from the LTSSM State Status Register (see Section 15.4.6.1, “PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)”) to determine when link negotiation is done and link is up (that is, Status Code = 16 link is up).
10. For EP mode only: After system configuration is done, set the CFG_READY bit in the configuration ready register. See Section 15.4.6.12, “PCI Express Configuration Ready Register.”
11. For RC mode only: Set the bus master and memory space fields in the PCI Express command register (in the PCI Express configuration space) to allow inbound and outbound transactions. See Section 15.4.1.3, “PCI Express Command Register.”

NOTE

Only in RC mode the local host should perform this programming. When the device is in EP mode, it is expected that the remote PCI Express RC will do this operation by a configuration access.

12. The device is ready to generate or accept PCI Express transactions according to its mode.

15.8 DMA Functional Operation

Software uses the DMA engine to initiate memory transfers from the CSB subsystem to the PCI Express system and vice versa without using programmed input/output (PIO), where data is transferred by sending control data through the CPU. The DMA engine provides control registers to enable the transfers. The PCI Express CSB bridge supports two separate engines for read and write DMA operations, referred as RDMA and WDMA, respectively. The DMA engines use a descriptor-based programming model.

NOTE

In general, the DMA descriptors use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data. No byte swapping occurs when the registers are accessed from the PCI Express link.

15.8.1 DMA Descriptor Format

The DMA descriptor, shown in Figure 15-144, is five DW wide and consists of control and status fields. The software is responsible to prepare the descriptors at the local memory and program the relevant DMA control registers. The hardware updates the status register on completion of a DMA data transfer for a descriptor and a chain of descriptors, and report errors.

Offset *Anywhere in local memory* :

	159–128
R	Next descriptor pointer
W	
Reset	All zeros

	127–96
R	Destination address
W	
Reset	All zeros

	95–64
R	Source address
W	
Reset	All zeros

	63–37	36	35	34	33	32
R	—	PEX error	Bridge error	CSB error	Descriptor error	Done
W						
Reset	All zeros					

	31–30	29–12	11–8	7–4	3	2	1	0
R	—	Transfer length	Last Byte Enable	First Byte Enable	No snoop for CSB	No snoop for PEX	Next descriptor pointer valid	Valid
W								
Reset	All zeros							

Figure 15-144. DMA Descriptor Format

Table 15-145 defines the bit fields of a DMA descriptor.

Table 15-145. DMA Descriptor Bit Fields Description

Bits	Width	Attribute	Description
159–128		Control	Next descriptor pointer. Indicates the location of the next descriptor. Valid only if next_dp is set.
127–96	32	Control	Destination address. Software programs this field to indicate the destination address. For a write DMA, the destination address is a CSB address that is mapped to a PCI Express memory address using the outbound window address translation. For a read DMA, the destination address is the CSB address.
95–64	32	Control	Source address. Software programs this field to indicate the source address. For a write DMA, the source address is the CSB address. For a read DMA, the source address is a CSB address that is mapped to a PCI Express memory address using the outbound window address translation.
63–37	27	—	Reserved
36		Status	PCI Express error. Hardware sets this bit to indicate that a DMA data transfer corresponding to descriptor cannot complete due to a PCI Express error.

Table 15-145. DMA Descriptor Bit Fields Description (continued)

Bits	Width	Attribute	Description
35	1	Status	Bridge error. Hardware sets this bit to indicate that DMA data transfer corresponding to descriptor cannot complete due to Bridge error.
34	1	Status	CSB error. Hardware sets this bit to indicate that complete data cannot be fetched due to an CSB Error.
33	1	Status	Descriptor error. Hardware sets this bit to indicate that the next descriptor cannot be fetched due to an CSB error.
32	1	Status	Done. Hardware sets this bit after completing the transaction.
31–30	2	—	Reserved
29–12	18	Control	Transfer length. Software programs this field to indicate the length of transfers in DW or data payload size. A value of zero means that no data is transferred.
11–8	4	Control	Last byte enable indicates the byte enables of the last DW to be transmitted by DMA.
7–4	4	Control	First byte enable. Indicates the byte enables of first DW to be transmitted by DMA.
3	1	Control	Snoop for CSB transactions. Indicates the no snoop value to be used for CSB transactions during a payload data access.
2	1	Control	No snoop for PCI Express transactions. Indicates the no snoop value to be used in TLP header for PCI Express transactions.
1	1	Control	Next descriptor pointer—valid. Software sets this bit to indicate that a descriptor has a valid next descriptor pointer next_dp. When this bit is cleared, the address of the next descriptor is implicitly specified. This bit must be cleared for block descriptor-based memory.
0	1	Control	Valid. Software sets this bit to indicate that a descriptor is valid and has the information related to data transfer.

15.8.2 Write DMA

The PCI Express or CSB software can program the write DMA engines to send data from the CSB system to the PCI Express. After the control registers are programmed, the write DMA engine issues a CSB read request through the DMA read master. To improve system performance, the DMA request is segmented according to a natural aligned CSB address boundary of maximum transfer size (32 bytes).

The entire address space accessed by the DMA controller is prefetchable, so the CSB read request for DMA operation always reads all the bytes. All segments use the same ID and are posted without waiting for the previous read response.

When a response to a CSB read request is received, the segments are packed into the PCI Express memory write request according to the PCI Express MPS. All data requested by the DMA controller is processed as a single data stream as described in this section. For example, when a 256-byte request starting from address 0 is segmented to eight 32-byte CSB read requests, then it is segmented to two 128-byte PCI Express write requests (assuming a 128-byte MPS).

If all data can be packed into one PCI Express write request, no segmentation is performed. Otherwise, the first segment starts from the start address and ends at the MPS address boundary. Byte enables are set according to DMA control register settings. All other segments except the last one start from the MPS address boundary and end at the next boundary. That is, the size of a write request is equal to MPS. All bytes are enabled. Remaining data, if applicable from the last MPS address boundary to the PCI Express end address, is packed into the last segment. Byte enables are set according to DMA control register settings. When all segments get an CSB response with a status of OKAY, the DMA data transfer has completed successfully.

If any data within an CSB read response is aborted by the CSB slave (SLVERR response), all the data received before that point is packed and sent. Any remaining data returned from the CSB slave is discarded, and an error is logged.

If any segment gets a response of DECERR, all data received before that point is packed and sent. Any remaining data returned from the CSB slave is discarded, and an error is logged.

The CSB read master can issue multiple pending CSB read requests if the requests belong to a single DMA request. When all segments are successfully received (that is, they get an CSB response with a status of OKAY), the CSB read master starts processing the next DMA request.

15.8.3 Read DMA

The read DMA engines can be programmed by the PCI Express or CSB software to send data from the PCI Express system to the CSB system. After the control registers are programmed, the read DMA engine issues a PCI Express read request. The DMA request is segmented according to the PCI Express MRRS natural aligned address boundary. If all data can be requested in one read request, no segmentation is performed. Otherwise, the first segment starts from the start address and ends at the first MRRS boundary. All other segments except the last one start from the MRRS address boundary and end at the next boundary. That is, the length of the read request is equal to MRRS. Any remaining data, if applicable from the last MRRS address boundary to the end address, is requested in the last segment.

The entire address space accessed by the DMA controller is considered as prefetchable, so a PCI Express read request for DMA operation always enables all byte enables. All segments have a unique tag, so multiple read requests can be pending at any given time.

To improve system performance, when a PCI Express completion comes back, it is segmented according to the CSB maximum size natural aligned address boundary. If all data can be packed into one CSB write request, no segmentation is performed, and the write is performed according to the DMA control register settings. Otherwise, the first segment starts from the start address and ends at the first CSB address boundary. All other segments except the last one start from the CSB address boundary and end at the next boundary. That is, the size of the write request is equal to the CSB maximum size packet. All bytes within each data phase are enabled.

Any remaining data, if applicable from the last CSB address boundary to the PCI Express end address, is packed into the last segment.

Each completion is segmented as a single data stream according to these rules. For example, when a 256-byte request starting from address 0 is converted to two 128-byte PCI Express read requests (assuming a 128-byte MPS/MRRS), the bridge issues two read requests if a tag and a completion buffer are available. When any completion comes back, it is segmented as described in this section. No scatter gather is performed. Because PCI Express can return completions in any order, the bridge may not issue an CSB write request in address order.

When all segments get a response with a PCI Express completion that has a status of successful, the DMA data transfer has completed successfully. If any segment gets a response with a status other than successful, or a completion timeout occurs, the DMA data transfer completes with an error status after all requests complete either normally or abnormally. The data returned for prior requests is still processed and sent to the CSB accordingly.

15.8.4 Descriptor-Based DMA

Descriptor-based DMA operation has a specific format in which the host can store information about a data transfer, such as the source address for the data to be transferred, the destination address, the data transfer size, location of the next descriptor, and so on. The host can program a series of descriptors and store them in host local memory. The host also programs the DMA control register to indicate the location of the first descriptor. The DMA control registers are part of the bridge device-specific registers. After programming the control register and descriptors, the host is free to continue with its other functions. The DMA engine is responsible for fetching the descriptor from host memory and moving data from/to host memory.

Software can organize the descriptors in two different formats that are only for reference and do not affect the hardware functionality and requirements:

- Chain descriptor
- Block descriptor

15.8.4.1 Chain Descriptor

Chain descriptors form an *n*-way chain in which each descriptor implicitly or explicitly contains the address of the next descriptor. This enables the host to use memory efficiently to store the descriptors even when contiguous memory locations are not available. When the host needs to initiate another transfer, it adds another descriptor in its memory and modifies the pointer of the last descriptor in the chain to the location of the new descriptor. Figure 15-145 illustrates the chain descriptor organization in host memory.

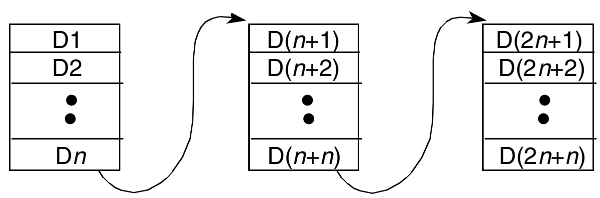


Figure 15-145. *n*-Way Chain Descriptor Organization in Host Memory

The number of ways, *n*, is software configurable. In this mode, *n* descriptors are written in contiguous memory locations. The implicit address of the next descriptor is the next memory location. The last descriptor in the contiguous block contains the explicit address pointer of the next set of *n* descriptors. Address 0x0 will never be part of the chain since it should close the chain. Non-contiguous valid descriptors are not supported. If the valid bit of a descriptor in the chain is not set, all of the succeeding descriptors should also have the valid bit as zero.

The software need to follow the following sequence on receiving a chain transfer done interrupt:

1. Software receives an interrupt for chain transfer done.

2. Software polls the main memory and waits for descriptor done bit to be set (bit 32 in the descriptor). This guarantees that the final data has been written into memory for read DMA. For write DMA, it guarantees that the final data has been sent to PCI Express controller. If a read is sent to the same location, PCI Express controller guarantees that the outbound read will not bypass the outbound write.

The host can reuse the descriptor memory after the DMA/bridge logic processes it. The exact handshake between the hardware and software is described later in this chapter. The DMA registers are described in Section 15.5.5, “DMA Registers.”

When *n* > 1, the hardware uses this knowledge to prefetch descriptors in advance, thereby reducing possible holes in transmission. This might be useful for applications that request several small DMA transfer requests, such as an Ethernet traffic. For applications that request large data transfers, the effect on performance due to descriptor fetching is not significant.

15.8.4.2 Block Descriptors

Block descriptors are a special case of chain descriptor in which all the descriptors are part of a single array. A portion of host memory is reserved to store the descriptors. The starting address of this block is indicated in the DMA control register. This block of memory serves as a circular buffer. Software writes the first descriptor to the first address in the descriptor block address space. Subsequent descriptors are written in the consecutive locations until the last location in the descriptor space. After that, the next descriptor is again written into the first location. All descriptors except the one written into the last descriptor location in the block have an implicit address that points to the immediate next descriptor location in the block. The last descriptor contains an explicit address pointer to the first descriptor location of the block. Software must ensure that a descriptor is processed by hardware before overwriting it. Figure 15-146 illustrates the organization of the block descriptors in memory.

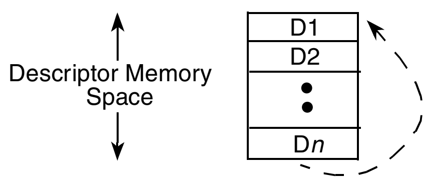


Figure 15-146. Block Descriptor Organization in Host Memory

The DMA can fetch prefetch a configurable number of descriptors in a single shot. This number depends on the chain organization (*n*) as well as the number of descriptor registers in the DMA engine.

15.8.4.3 Descriptor Format

For more information on descriptor format, refer to Section 15.8.1, “DMA Descriptor Format.” The fetched descriptors are stored in the bridge configuration space registers. Each time the DMA controller fetches a new set of descriptors, the register is updated to indicate the value/fields of the descriptors. After completion of a transaction specified by a descriptor, the status fields are updated in the descriptor registers in the configuration space. Also, the descriptor status is written back into host local memory and an interrupt is generated if enabled. When the last of *n* descriptors is processed, the next set of descriptors is fetched from memory.

15.8.4.4 Software-Hardware Handshake

Hardware and software communicate through descriptors, control registers, and interrupts. The control register programmed by software indicates to hardware the location of the first descriptor. The descriptors programmed by software in host memory or directly in the descriptor register provide information to the hardware about the impending data transfer. After the initial descriptor location and other DMA parameters are programmed in the DMA control registers, software sets the 'start' bit in the control register to trigger the DMA controller to initiate the operation. When it detects that the 'start' bit is set, the DMA controller uses these programmed DMA parameters to initiate a descriptor fetch and the corresponding data transfer, and then it clears the "start" bit.

The DMA controller executes the transfer according to the instructions given in the descriptor and then updates both the descriptor and DMA status register to indicate the status of the transfer operation. Descriptor status is updated in host local memory. An interrupt is also generated to the host, if enabled.

At any time, software can parse the done bit in the descriptor chain located in host local memory to determine the point to which the DMA controller has executed and also if the descriptors were successfully completed. The status register also gives details about transfer status, including details about errors if any occurred.

If the DMA encounters an unprogrammed descriptor (ready = 0) in the *n* descriptor array that it fetched, it first executes any remaining prefetched valid descriptors and then sends an interrupt to the host, if enabled. The transfers can resume in one of two ways:

- The DMA controller automatically fetches the same set of descriptors again after the time indicated in PEX_DMA_DSTMR[DSRT] (see Section 15.5.2.2, "PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR)"). If the descriptor is ready now, it continues execution or else checks again later.
- Software can force immediate resuming of the transfers by disabling and then re-enabling the DMA.

When the DMA controller completes the data transfer for the last descriptor in the chain (null descriptor), it updates the descriptor and status register. An interrupt is generated, if enabled. The DMA engine moves into the IDLE state until software re-triggers it by setting the 'start' bit in the control register.

Chapter 16
SATA Controller

16.1 Overview

The serial ATA controller is a high-performance SATA solution incorporating some of the latest SATA-IO extensions. The SATA may also be referred to as a host bus adapter (HBA). The SATA controller is designed to operate in a system that supports command queuing and, in particular, a switching scheme based on a frame information structure (FIS) using port multipliers.

FIS-based switching requires the SATA controller to maintain in hardware a context for each command it has queued at the devices that are attached to it. FIS-based switching also requires the SATA controller to maintain a queue per attached device ensuring that the command issue order for each device is maintained. It can be used in SATA controllers, as well as storage area network (SAN), network attached storage (NAS), and RAID (redundant array of independent/inexpensive disks) devices.

SATA controller has the following features:

- Designed to comply with *Serial ATA 2.5 Specification*
- Supports speeds: 1.5 Gbps (first-generation SATA), 3 Gbps (second-generation SATA and eSATA)
- Supports advanced technology attachment packet interface (ATAPI) devices
- Contains high-speed descriptor-based DMA controller
- Supports native command queuing (NCQ) commands
- Supports port multiplier operation
- Supports hot plug including asynchronous signal recovery

Figure 16-1 shows a block diagram of SATA.

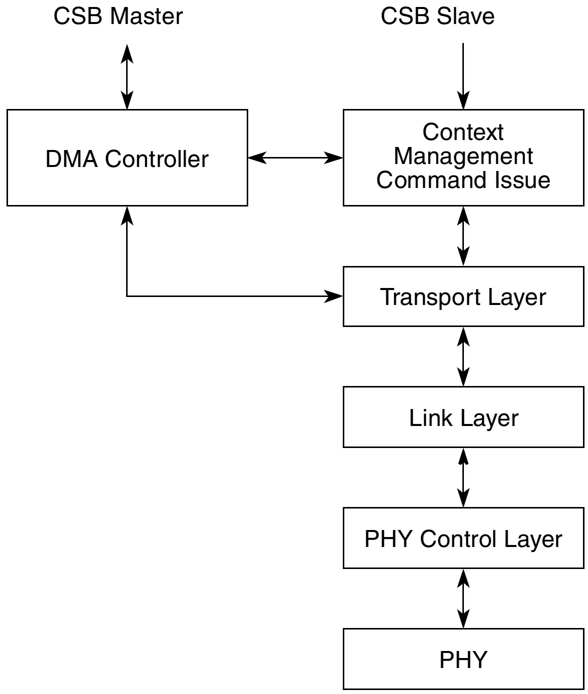


Figure 16-1. SATA Block Diagram

There are four layers in the SATA architecture: application, transport, link, and PHY. The application layer is responsible for overall ATA command execution, including controlling command block register accesses. The transport layer is responsible for placing control information and data to be transferred between the host and device in a packet/frame, known as a frame information structure (FIS). The link layer is responsible for taking data from the constructed frames, inserting control characters and moving data to the PHY layer. The PHY layer is responsible for 8B/10B encoding/decoding, then transmitting and receiving the encoded information as a serial data stream on the wire.

16.2 Command Operation

The SATA controller maintains a queue consisting of up to 16 commands. These commands can be distributed to a single attached device or, if the system contains a port multiplier, over each of the attached devices. It is possible to queue queued commands and non-queued commands into the SATA controller, provided the host software does not break protocol to any particular device (it is illegal to issue a non-queued command to a device that still has a queued command active as per ATAPI/ATA protocol).

16.2.1 Command Issue

When the host software is preparing to issue a command, it first builds a command descriptor as shown in Figure 16-28. The format of the command FIS is defined in the Serial ATA 2.5 standard shown in Figure 16-29. The software is also responsible for the creation of a scatter/gather list for data movement. This list should be defined to exactly match the transfer length as programmed into the command FIS. If the 16 entries are not sufficient, then an extended entry (ext) can be used to refer to an alternate table. When the descriptor is built, the host software locates a free command slot within the SATA controller by examining the command queue register. To issue the command, the host software programs the address of the command descriptor and the attributes into the appropriate command header locations and then issues the command by writing the PMP and setting the appropriate CQ bit in the command queue register.

16.2.2 Command Service

After a command is issued, the SATA controller takes ownership of the command descriptor, transferring the command FIS to the targeted device when required, servicing the data transfer using the scatter/gather list provided and transferring the status back into the command descriptor (if programmed).

16.2.3 Command Completion Interrupt Timing

When a command completes, it is possible to enable the SATA controller to generate an interrupt. Associated with some commands there will be a command completion status FIS. The SATA controller will always transfer the status FIS to memory whether it indicates an error or good command completion.

16.2.4 DMA Context (Read Data)

When receiving FIS's from attached devices, the SATA controller has to support interleaving from various devices. Data FIS's from device 0 could be interleaved with data FIS's for device 1. In order to accomplish this, the SATA controller maintains in hardware a context for each command which is pushed onto and pulled from the DMA controller when needed to service the transfers.

16.2.5 DMA Context (Write Data)

When the SATA controller receives an FIS indicating that the next operation to a particular device should be a data write transfer, the SATA controller will lock the interface by forcing the link layer to transition to X_RDY immediately and not go through idle SYNC. This will mean that write transfers will not have to be interleaved, which simplifies the transmit data path and eliminates the need for a complex scheduler.

16.2.6 DMAT Primitive Processing

The SATA controller supports the reception of the DMAT primitive. When the SATA controller receives a DMAT primitive from the device, it will perform the following actions.

The DMA controller will complete the current read burst and transfer the data to the transport layer FIFO. The DMA controller signals an EOF on the last data of the burst, which causes the link layer to insert the CRC and EOF. The context for this transfer is returned to the context store. Once this action is completed, the device can terminate the transfer or re-initiate the transfer as per Serial ATA Revision 2.5 Section 9.4.4.

16.3 Command Layer Overview

The function of the SATA command layer is to allow host software queue commands. It then manages the command issue and service using context to complete the queued commands.

16.3.1 SATA Memory Map/Register Definition

Table 16-1 shows the memory map for the SATA registers. The offsets to the memory map table are defined for all four SATA hosts. That is, SATA1 starts at 0x1_8000 address offset, SATA2 at 0x1_9000, SATA3 at 0x1_A000, and SATA4 at 0x1_B000. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

NOTE

All registers (except SYSPR) described in this section and descriptors described in Section 16.3.6, "Command Header," and Section 16.3.7, "Command Descriptor," use little-endian byte ordering. Software running on the local processor in big-endian mode must byte-swap the data.

In this table, and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- 'R/W', 'R', and 'W' (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- 'w1c' indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- 'Mixed' indicates a combination of access types.

Table 16-1. SATA Register Summary

Offset	Register	Access	Reset Value	Section/Page
SATA1—Block Base Address: 0x1_8000				
SATA Command Registers				
0x000	CQR—Command queue register	R/W	0x0000_0000	16.3.2.1/16-5
0x008	CAR—Command active register	R	0x0000_0000	16.3.2.2/16-6
0x010	CCR—Command completed register	w1c	0x0000_0000	16.3.2.3/16-7
0x018	CER—Command error register	w1c	0x0000_0000	16.3.2.4/16-7
0x020	DE—Device error register	w1c	0x0000_0000	16.3.2.5/16-8
0x024	CHBA—Command header base address	R/W	0x0000_0000	16.3.2.6/16-9
0x028	HStatus—Host status register	w1c	0x2000_0000	16.3.2.7/16-9
0x02C	HControl—Host control register	Mixed	0x0000_0100	16.3.2.8/16-12
0x030	CQPMP—Port number queue register	R/W	0x0000_0000	16.3.2.9/16-13
0x034	SIG—Signature register	R	0xFFFF_FFFF	16.3.2.10/16-14
0x038	ICC—Interrupt coalescing control register	R/W	0x0100_0000	16.3.2.11/16-14
SATA1 Superset Registers				
0x100	SStatus—SATA interface status register	R	0x0000_0000	16.3.3.1/16-15

Table 16-1. SATA Register Summary (continued)

Offset	Register	Access	Reset Value	Section/Page
0x104	SError—SATA interface error register	w1c	0x0000_0000	16.3.3.2/16-16
0x108	SControl—SATA interface control register	R/W	0x0000_0300	16.3.3.3/16-18
0x10C	SNotification—SATA interface notification register	w1c	0x0000_0000	16.3.3.4/16-19
SATA1 Control Status Registers				
0x140	TransCfg—Transport layer configuration	R/W	0x0800_0016	16.3.4.1/16-20
0x144	TransStatus—Transport layer status	R	0x0000_0000	16.3.4.2/16-21
0x148	LinkCfg—Link layer configuration	R/W	0x0000_FF34	16.3.4.3/16-21
0x14C	LinkCfg1—Link layer configuration1	R/W	0x0000_0000	16.3.4.4/16-22
0x150	LinkCfg2—Link layer configuration2	R/W	0x0000_0000	16.3.4.5/16-23
0x154	LinkStatus—Link layer status	R	0x0000_0000	16.3.4.6/16-23
0x158	LinkStatus1—Link layer status1	R	0x0000_0000	16.3.4.7/16-24
0x15C	PhyCtrlCfg1—PHY control configuration1	R/W	0x0000_3800	16.3.4.8/16-26
0x160	CommandStatus—Link layer command status	R	0x0000_0000	16.3.4.9/16-27
0x164–0x17C	Reserved	—	—	—
SATA1 System Control Registers				
0x410	SYSPR—System priority register	R/W	0x0000_0000	16.3.5.1/16-28
0x40C–0xFFF	Reserved	—	—	—
SATA2—Block Base Address: 0x1_9000				
SATA2 has the same memory-mapped registers that are described for SATA1 from 0x1_8000 to 0x1_8FFF except the offsets are from 0x1_9000 to 0x1_9FFF.				
SATA3—Block Base Address: 0x1_A000				
SATA3 has the same memory-mapped registers that are described for SATA1 from 0x1_8000 to 0x1_8FFF except the offsets are from 0x1_A000 to 0x1_AFFF.				
SATA4—Block Base Address: 0x1_B000				
SATA4 has the same memory-mapped registers that are described for SATA1 from 0x1_8000 to 0x1_8FFF except the offsets are from 0x1_B000 to 0x1_BFFF.				

16.3.2 Command Registers

16.3.2.1 Command Queue Register (CQR)

Before queuing a command into the SATA controller, the CQR (shown in Figure 16-2) is first examined to detect a free command queue (CQ) slot. A free CQ slot is indicated by a 0 in a bit position. To queue a command, the bit corresponding to the CQ slot to use is set. At this point the SATA controller takes ownership of the command header space and command descriptor associated with the command slot. While the command is queued in the SATA controller or at the device, the command queue bit remains 1. When the command completes, this bit is cleared to 0 by the hardware. For a device error, the CQR holds the command queue bits at 1 for each command queued or issued to the device in error. When the host software clears the device error, the hardware in turn clears each of the commands queued.

Offset 0x1_8000 — Access: Read/Write

	31–16	15–0
R/W	—	CQ*n*
Reset	All zeros	

Figure 16-2. Command Queue Register (CQR)

Table 16-2 describes the CQR fields.

Table 16-2. CQR Field Descriptions

Bit	Name	Description
31–16	—	Reserved
15–0	CQ*n*	Command *n* queue bit

16.3.2.2 Command Active Register (CAR)

When a command is issued from the SATA controller to the device, the command is marked as active by the hardware setting the appropriate command active bit of the CAR (shown in Figure 16-3). Once a command completes, the hardware clears the appropriate bit of the CAR.

For a device error, the CAR holds the command active bits at 1 for each command issued to the device in error. When the host software clears the device error, the hardware in turn clears each of the commands queued.

Offset 0x1_8008 — Access: Read only

	31–16	15–0
R/W	—	CA*n*
Reset	All zeros	

Figure 16-3. Command Active Register (CAR)

Table 16-3 describes the CAR fields.

Table 16-3. CAR Field Descriptions

Bit	Name	Description
31–16	—	Reserved
15–0	CA*n*	Command *n* active bit

16.3.2.3 Command Completed Register (CCR)

When a command completes, the hardware sets the command completed bit for that command in the CCR (shown in Figure 16-4) to a 1. The hardware also clears both the command queue and the command active bit for that command. When the software needs to acknowledge the reception of the command complete, it can do so in two ways:

- Writing a 1 to the command complete bit
- Issuing a command to the command slot

An interrupt coalescing scheme runs on the CCR. When the register contains a value other than 0x0000_0000, an interrupt coalescing timer runs. Each time a command completion is acknowledged, the timer is reset. When the timer times out, an interrupt is generated.

Offset 0x1_8010 Access: w1c

	31–16	15–0
R W	—	CC*n*
Reset	All zeros	

Figure 16-4. Command Completed Register (CCR)

Table 16-4 describes the CCR fields.

Table 16-4. CCR Field Descriptions

Bit	Name	Description
31–16	—	Reserved
15–0	CC*n*	Command *n* completed bit

16.3.2.4 Command Error Register (CER)

When a device errors a command by setting the error bit in the status register, this is detected by the SATA controller as a single device error. The associated command completing due to error is indicated by the hardware setting the command error bit for that command in the CER (shown in Figure 16-5). For safe operation under both command queuing and non-queuing operation, all commands queued into the SATA controller and at the device are considered aborted. The queue for that device is stopped. The values of the registers CQR, CAR, and CCR will allow the host software to know which commands have completed without error and those that were queued at the SATA controller and at the device.

When the host software clears the device error (by writing 1 to DER), the software is also responsible to clear CER by writing a 1 to the command error bit for the command that was in error. After the error condition at the device has been cleared, the host application software can reissue the commands to the SATA controller, which were aborted on the reception of the single device error.

Offset 0x1_8018 Access: Read only

	31–16	15–0
R/W	—	CE*n*
Reset	All zeros	

Figure 16-5. Command Error Register (CER)

Figure 16-5 describes the CER fields.

Table 16-5. CER Field Descriptions

Bit	Name	Description
31–16	—	Reserved
31–0	CE*n*	Command *n* error bit

16.3.2.5 Device Error Register (DER)

When a single device error is detected, the device that issued the error is indicated by the hardware setting the device error bit to a 1 in the DER (shown in Figure 16-6). The procedure as outlined in the command error register applies to the queues and to restarting the device.

The host application software acknowledges the device in error by clearing the device error bit. The device error is cleared by writing a 1 to the appropriate device error bit. When this action is performed, the queue to the device in error is cleared and is ready to have commands queue.

While a device is in error, no command can be queued for that device.

Offset 0x1_8020 Access: w1c

	31–16	15–0
R/W	—	DE*n*
Reset	All zeros	

Figure 16-6. Device Error Register (DER)

Table 16-6 describes the DER fields.

Table 16-6. DER Field Descriptions

Bit	Name	Description
31–16	—	Reserved
15–0	DE*n*	Device *n* error bit

16.3.2.6 Command Header Base Address Register (CHBA)

The CHBA is shown in Figure 16-7. This holds the address in memory of where the command header block is located. It must be written as part of the host software initialization process. After the SATA controller hardware is brought online, the SATA controller takes ownership of this register. The address in this register should not be changed while the SATA controller is online.

Offset 0x1_8024 Access: Read/Write

	31–2	1–0
R / W	Command Header Base	—
Reset	All zeros	

Figure 16-7. Command Header Base Address Register (CHBA)

Table 16-7 describes the CHBA fields.

Table 16-7. CHBA Field Descriptions

Bit	Name	Description
31–2	CHBA	Command header base address
1–0	—	Reserved, should be cleared.

16.3.2.7 Host Status Register (HStatus)

HStatus, shown in Figure 16-8, holds the status of the SATA controller as well as the interrupt sources. When an event occurs, the interrupt bit is set regardless of the status of the associated interrupt enable bit. The interrupt signal from the SATA controller is gated with the associated interrupt enable register. For all interrupt bits other than the interrupt on command complete bit, when software has processed the interrupt condition, it acknowledges the interrupt by writing a 1 to the interrupt source bit. This action will clear the interrupt signal if there are no other outstanding interrupts in HStatus.

The interrupt on command complete requires special processing. This bit is set as a result of the programmed interrupt coalescing algorithm running on the register CCR contents. For the interrupt on command complete bit, the command(s) that have completed to cause this interrupt need to be cleared by clearing the command N completed bit of the CCR. When the number or staleness of the CCR falls below the programmed interrupt coalescing algorithm, the interrupt on command complete bit clears.

Offset 0x1_8028 Access: w1c

Bits	31	30	29	28–19	18	17–16
R/W	HS_ON	HS_OFF	BE	—	ME	—
Reset	0	0	1	0 0 0 0 0 0 0 0 0 0	0	0 0

Bits	15–14	13	12	11	10	9	8	7–6	5	4	3	2	1	0
R/W	—	DUE	DOE	CET	CER	FOT	FOR	—	FE	PR	SIGU	SNTFU	DE	CC
Reset	All zeros													

Figure 16-8. Host Status Register (HStatus)

Table 16-8 describes the HStatus fields.

Table 16-8. HStatus Field Descriptions

Bit	Name	Description
31	HS_ON	Online/offline. This bit indicates if the SATA controller is online or offline. 0 Offline. The SATA controller is non-operational and the PHY is held in reset. 1 Online. The SATA controller is operational.
30	HS_OFF	Going offline. This bit indicates that the SATA controller is going offline it is waiting for the commands queued within the SATA controller or active at the device to complete. 0 Host is not in process going offline 1 Host is in process going offline
29	BE	BIST error. When the protocol is placed into BIST this bit maps the BIST error. 0 Indicates the link layer is passing BIST 1 Indicates that the link layer is not passing BIST When the protocol is not in BIST this bit will assert high and can be ignored.
28–24	—	Reserved
23–19	—	Reserved
18	ME	SATA controller master error. Indicates if the host received an error on the CSB master interface during the access to memory. 0 No error response is received when a transfer was made into the memory 1 Error response is received during the transfer into the memory
17–16	—	Reserved
15–14	—	Reserved
13	DUE	Data underrun. 0 No underrun encountered (data was retrieved from external memory in time to send a complete FIS) 1 The SATA controller encountered an underrun condition while sending the FIS

Table 16-8. HStatus Field Descriptions (continued)

Bit	Name	Description
12	DOE	Data overrun. 0 No overrun condition encountered 1 The SATA controller encountered an overrun condition while receiving the FIS
11	CET	CRC error Tx. When set, this bit indicates that one or more CRC errors occurred in Tx data path.
10	CER	CRC error Rx. When set, this bit indicates that one or more CRC errors occurred in Rx data path.
9	FOT	FIFO overflow Tx. When set, this bit indicates that Tx FIFO is in overflow condition while sending FIS.
8	FOR	FIFO overflow Rx. When set, this bit indicates that Rx FIFO is in overflow condition while receiving FIS.
7–6	—	Reserved
5	FE	Fatal error. When set, this bit indicates that fatal error occurred in SATA controller. In this state, the interrupt will be generated if FATAL_INT is set in the host control register. Write ‘1’ to clear the interrupt source.
4	PR	PHY ready. When set, this bit indicates that PHY READY signal was changed. In this state, the interrupt will be generated if PHYRDY_INT is set in the host control register. Write ‘1’ to clear the interrupt source.
3	SIGU	Signature update. When set, this bit indicates that the signature is updated in the host signature register. In this state, the interrupt will be generated if SIG_INT is set in the host control register. Write ‘1’ to clear the interrupt source.
2	SNTFU	SNotification update. When set, this bit indicates that the SNotification register has at least one bit set. In this state, the interrupt will be generated if SNTFY_INT is set in the host control register. Write ‘1’ to clear the interrupt source.
1	DE	Device error. When set, this bit indicates that the DE register has at least one bit set. In this state, the interrupt will be generated if DE_INT is set in the host control register. Write ‘1’ to clear the interrupt source.
0	CC	Command complete. When set, this bit indicates that the register CCR has at least one bit set. In this state, the interrupt will be generated if CC_INT is set in the host control register. Write ‘1’ to clear the interrupt source.

16.3.2.7.1 Error Processing

On single device error:

1. Examine the register DER to determine which device is in error state. There might be multiple devices in error.
2. Examine the register CER to determine which command was in error. The software knows which command belongs to which device.
3. Examine the status location of the descriptor of the command in error and determine the reason for the error.
4. If needed, the software should send commands to the device to clear down the error condition on device or for further examination of the device's status.
5. Clear the DER*n* bit by writing 1 to bit *n*, where *n* indicates the device in error. This will also clear out the outstanding commands for that device.

6. Clear the CER*n* bit by writing 1 to bit *n*, where *n* indicates the associated command in error. After that, the software can reissue command to the device if needed.

On fatal error:

1. Read the error register and other registers to determine how many commands are outstanding and how many have completed without error.
2. Bring the SATA controller offline. When this happens all queues within the SATA controller will be cleared.
3. Perform what corrective action the software determines is necessary.
4. Bring the SATA controller online. This will cause an out-of-bounds (OOB) to be run at the PHY level which will clear down any attached device.

16.3.2.8 Host Control Register (HControl)

HControl, shown in Figure 16-9, is written to control the operation of the SATA controller. To enable an interrupt, the associated bit must be set; to disable the interrupt, the associated bit must be cleared.

Offset 0x1_802C — Access: Mixed

	31	30	29–16
R/W	HC_ON	HC_OFF (R only)	—
Reset	All zeros		

	15–11	10	9	8–6	5	4	3	2	1	0
R/W	—	SNOOP_EN	PM_EN	—	FATAL_INT	PHYRDY_INT	SIG_INT	SNTFY_INT	DE_INT	CC_INT
Reset	0 0 0 0 0	0	0	1 0 0	0	0	0	0	0	0

Figure 16-9. Host Control (HControl) Register

Table 16-9 describes the HControl fields.

Table 16-9. HControl Field Descriptions

Bit	Name	Description
31	HC_ON	Online/offline. 0 Offline. Bring the SATA controller offline and place the PHY in reset 1 Online. Bring the SATA controller online
30	HC_OFF	Offline request status. 1 The SATA controller is currently completing an operation and will go offline when the operation completes. When this bit is set, the SATA controller can be forced to go offline by aborting its current operation by writing 0 to the HC_ON bit.
29–15	—	Reserved
14–12	—	Reserved. Reset value must be preserved when writing to the register.
11	—	Reserved

Table 16-9. HControl Field Descriptions (continued)

Bit	Name	Description
10	SNOOP_EN	Snoop enable during header fetch. 0 Snoop not enabled during command header fetch. 1 Snoop enabled during command header fetch.
9	PM_EN	Port multiplier attached. This bit is used to indicate if the HBA is attached to a port multiplier. This bit is set or cleared by software. 0 This SATA controller is directly attached to a SATA device. The SATA controller hardware does not auto-detect the presence of a port multiplier; this is to allow for future changes in signature type for the port multiplier. 1 A port multiplier is attached to the SATA controller.
8–6	—	Reserved. Reset value must be preserved when writing to the register.
5	FATAL_INT	Enable interrupt on fatal error.
4	PHYRDY_INT	Enable interrupt on PHY ready change.
3	SIG_INT	Enable interrupt signature update.
2	SNTFY_INT	Enable interrupt on SNotify register update.
1	DE_INT	Enable interrupt on single device error.
0	CC_INT	Enable interrupt on command complete.

16.3.2.8.1 Bringing the SATA Controller Online/Offline

This HC_ON bit in HControl allows the host software to bring the SATA controller online or offline. The SATA controller online status should only be changed when there are no commands queued in the SATA controller or at any attached device.

When the host application wishes to bring the SATA controller offline it clears the HC_ON control bit. This acts as a request to the SATA controller to go offline. The SATA controller will signal it has completed this operation by clearing the HS_ON bit in the HStatus register. If any commands are outstanding at SATA controller or device then the SATA controller will wait for the operation to complete before going offline.

It the host application wishes to bring the SATA controller offline regardless of the queue status, it clears the HC_ON bit while the HS_OFF bit of the HStatus register is set.

When the host application wishes to bring the SATA controller online, it sets the HC_ON control bit. This acts as a request to the SATA controller to go online. The SATA controller will signal it has completed this operation by setting to 1 the HS_ON status bit.

16.3.2.9 Port Number Queue Register (CQPMP)

When queuing a command into the SATA controller, the CQPMP, shown in Figure 16-10, is written with the value of the PMP field that addresses the device to which the command will be issued. If the device is directly attached (that is, there is no port multiplier in the system), then this register is not required and should be cleared.

Offset 0x1_8030 — Access: Read/Write

	31–4	3–0
R/W	—	CQPMP
Reset	All zeros	

Figure 16-10. Port Number Queue Register (CQPMP)

Table 16-10 describes the CQPMP fields.

Table 16-10. CQPMP Field Descriptions

Bit	Name	Description
31–4	—	Reserved
3–0	CQPMP	Command queue port multiplier field

16.3.2.10 Signature Register (SIG)

The 32-bit SIG register, shown in Figure 16-11, contains the initial signature of an attached device when the first D2H register FIS is received from that device.

Offset 0x1_8034 — Access: Read only

	31–24	23–16	15–8	7–0
R/W	LBA_HIGH	LBA_MID	LBA_LOW	SEC_CNT
Reset	All ones			

Figure 16-11. Signature Register (SIG)

Table 16-11 describes the SIG register fields.

Table 16-11. SIG Register Field Descriptions

Bit	Name	Description
31–24	LBA_HIGH	LBA high register
23–16	LBA_MID	LBA mid register
15–8	LBA_LOW	LBA low register
7–0	SEC_CNT	Sector count register

16.3.2.11 Interrupt Coalescing Control Register (ICC)

When a command completes, the SATA controller sets the corresponding bit in the command completed register. The interrupt coalescing scheme runs on the SIG register, shown in Figure 16-12. The scheme runs in two ways:

- If the number of completed commands exceeds the threshold, then the interrupt will be signaled.

- If any command complete bit has been set in the register for a number of HCLK ticks equal to the programmed count, then the interrupt will be set. This timer will be reset each time a command completion is acknowledged by the application layer software.

Offset 0x1_8038 Access: Read/Write

	31–29	28–24	23–19	18–0
R/W	—	ITC	—	ITTCV
Reset	0 0 0	0 0 0 0 1	0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 16-12. Interrupt Coalescing Control Register (ICC)

Table 16-12 describes the ICC fields.

Table 16-12. ICC Field Descriptions

Bit	Name	Description
31–29	—	Reserved
28–24	ITC	Interrupt threshold count. The number of command completions that will raise the interrupt. 00000 Implied no threshold and the interrupt will be signaled based on the threshold timer. 00001, 01111 The number of command complete bits which, if set, will cause the interrupt to be signaled.
23–19	—	Reserved
18–0	ITTCV	Interrupt threshold timer compare value. The number of AHB ticks for which a command complete bit has to be set before the interrupt will be signaled. A value of 0 indicates that whenever a command complete bit is set the interrupt should be signaled.

16.3.3 SATA Superset Registers

Serial ATA provides an additional block of registers to control the interface and to retrieve interface state information.

16.3.3.1 SATA Interface Status Register (SStatus)

SStatus, shown in Figure 16-13, is a 32-bit read-only register that conveys the current state of the interface and host adapter. The register conveys the interface state at the time it is read and is updated continuously and asynchronously by the host adapter. Writes to this register have no effect.

Offset 0x1_8100 Access: Read only

	31–12	11–8	7–4	3–0
R/W	—	IPM	SPD	DET
Reset	All zeros			

Figure 16-13. SATA Interface Status Register (SStatus)

Table 16-13 describes the SStatus fields.

Table 16-13. SStatus Field Descriptions

Bit	Name	Description
31–12	—	Reserved
11–8	IPM	Interface power management state. Indicates the current interface power management state. 0000 Device not present or communication not established 0001 Interface in active state 0010 Interface in partial power management state 0110 Interface in slumber power management state All other values reserved
7–4	SPD	Speed. Indicates the negotiated interface communication speed established. 0000 No negotiated speed (device not present or communication not established) 0001 First-generation communication rate negotiated 0010 Second-generation communication rate negotiated All other values reserved
3–0	DET	Detection. Indicates the interface device detection and PHY state. 0000 No device detected and PHY communication not established 0001 Device presence detected but PHY communication not established 0011 Device presence detected and PHY communication established 0100 PHY in offline mode as a result of the interface being disabled or running in a BIST loopback mode All other values reserved

16.3.3.2 SATA Interface Error Register (SError)

SError, shown in Figure 16-14, is a 32-bit register that conveys supplemental interface error information to complement the error information available in the shadow register block error register. The register represents all the detected errors accumulated since the last time the SError register was cleared (whether recovered by the interface or not). Set bits in the error register are explicitly cleared by a write operation to the SError register or by a reset operation. The error bits that have been set in this register are cleared by writing a 1 to the corresponding field. Host software should clear the interface SError register at appropriate checkpoints in order to best isolate error conditions and the commands they impact.

Bits 31–16 of this register represent the DIAG decode bits, which contain diagnostic error information, for use by diagnostic software in validating correct operation or isolating failure modes. Bits 15–0 represent the ERR decode bits, which contain information for use by the host software in determining the appropriate response to the error condition.

Offset 0x1_8104 Access: w1c

DIAG Decode (bits 31–16) | ERR Decode (bits 15–0)

	31–28	27	26	25	24	23	22	21	20	19	18	17	16	15–12	11	10	9	8	7–2	1	0
R	—	A	X	F	—	S	H	C	D	B	W	IN	N	—	E	—	C	T	—	M	ITG
W		w1c	w1c	w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c		w1c	w1c		w1c	w1c
Reset	All zeros																				

Figure 16-14. SATA Interface Error Register (SError)

Table 16-14 describes the SError field descriptions.

Table 16-14. SError Field Descriptions

Bit	Name	Description
DIAG Decode		
31–28	—	Reserved bit for future use. Should be cleared.
27	A	Port selector presence detected. This bit is set when COMWAKE is received while the host is in state HP2: HR_AwaitCOMINIT. On power-up reset this bit is cleared. The bit is cleared when the host writes a 1 to this bit location.
26	X	Exchanged. When set to 1 this bit indicates that device presence has changed since the last time this bit was cleared. The means by which the implementation determines that the device presence has changed is vendor specific. This bit may be set anytime a PHY reset initialization sequence occurs as determined by reception of the COMINIT signal, whether in response to a new device being inserted, to a COMRESET having been issued, or to power-up.
25	F	Unrecognized FIS type. When set to 1, this bit indicates that since the bit was last cleared, one or more FIS's were received by the transport layer with good CRC, but they had a type field that was not recognized.
24	—	Reserved.
23	S	Link sequence error. When set to 1, this bit indicates that one or more link state machine error conditions were encountered since the last time this bit was cleared. The link layer state machine defines the conditions under which the link layer detects an erroneous transition.
22	H	Handshake error. When set to 1, this bit indicates that one or more R_ERRP handshake responses were received in response to frame transmission. Such errors may be the result of a CRC error detected by the recipient, of a disparity or 10b/8b decoding error, or of other error conditions leading to a negative handshake on a transmitted frame.
21	C	CRC error. When set to 1, this bit indicates that one or more CRC errors occurred with the link layer since the bit was last cleared.
20	D	Disparity error. When set to 1, this bit indicates that incorrect disparity was detected one or more times since the last time the bit was cleared.
19	B	10b to 8b decode error. When set to 1, this bit indicates that one or more 10-bit to 8-bit decoding errors occurred since the bit was last cleared.
18	W	COMWAKE detected. When set to 1, this bit indicates that a COMWAKE signal was detected by the PHY since the last time this bit was cleared.
17	IN	PHY internal error. When set to 1, this bit indicates that the PHY detected some internal error since the last time this bit was cleared.
16	N	PHYRDY change. When set to 1, this bit indicates that the PHYRDY signal changed state since the last time this bit was cleared.
ERR Decode		
15–12	—	Reserved bit for future use; should be cleared.
11	E	E Internal error. The host bus adapter experienced an internal error that caused the operation to fail and may have put the host bus adapter into an error state. Host software should reset the interface before retrying the operation. If the condition persists, the host bus adapter may suffer from a design issue rendering it incompatible with the attached device.

Table 16-14. SError Field Descriptions (continued)

Bit	Name	Description
10	—	Reserved
9	C	Non-recovered persistent communication or data integrity error. A communication error that was not recovered occurred that is expected to be persistent. Because the error condition is expected to be persistent, the operation need not be retried by the host software. Persistent communications errors may arise from faulty interconnect with the device, from a device that has been removed or has failed, or a number of other causes.
8	T	Non-recovered transient data integrity error: A data integrity error occurred that was not recovered by the interface. Because the error condition is not expected to be persistent, the operation should be retried by the host software.
7–2	—	Reserved
1	M	Recovered communications error. Communications between the device and host were temporarily lost but were re-established. This can arise from a device temporarily being removed, from a temporary loss of PHY synchronization, or from other causes, and may be derived from the PHYRDY*n* signal between the PHY and link layers. No action is required by the host software, because the operation ultimately succeeded. However, the host software may elect to track such recovered errors to gauge overall communications integrity and potentially step down the negotiated communication speed.
0	ITG	Recovered data integrity error. A data integrity error occurred that was recovered by the interface through a retry operation or other recovery action. This can arise from a noise burst in the transmission, a voltage supply variation, or other causes. No action is required by host software, because the operation ultimately succeeded. However, the host software may elect to track such recovered errors to gauge overall communications integrity and potentially step down the negotiated communication speed.

16.3.3.3 SATA Interface Control Register (SControl)

SControl, shown in Figure 16-15, is a 32-bit read-write register that provides the interface by which software controls SATA interface capabilities. Writes to the SControl register result in an action being taken by the host adapter or interface. Reads from the register return the last value written to it.

Offset 0x1_8108 Access: Read/Write

Bits	31–16	15–12	11–8	7–4	3–0
R/W	—	SPM	IPM	SPD	DET
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0

Figure 16-15. SATA Interface Control Register (SControl)

Table 16-15 describes the SControl fields.

Table 16-15. SControl Field Descriptions

Bit	Name	Description
31–16	—	Reserved, should be cleared.
15–12	SPM	Select power management. Used to select a power management state. A non-zero value written to this field will cause the power management state specified to be initiated. A value written to this field is treated as a one-shot. 0000 No power management state transition requested 0001 Transition to the partial power management state initiated 0010 Transition to the slumber power management state initiated 0100 Transition to the active power management state initiated All other values reserved
11–8	IPM	Interface power management. The enabled interface power management states can be invoked via the SATA interface power management capabilities. 0000 No interface power management state restrictions 0001 Transitions to the partial power management state disabled 0010 Transitions to the slumber power management state disabled 0011 Transitions to both the partial and slumber power management states disabled All other values reserved
7–4	SPD	Speed. Highest allowed communication speed the interface is allowed to negotiate when interface communication speed is established. 0000 No speed negotiation restrictions 0001 Limit speed negotiation to a rate not greater than first-generation communication rate 0010 Limit speed negotiation to a rate not greater than second-generation communication rate All other values reserved
3–0	DET	Detection. Controls the host adapter device detection and interface initialization. 0000 No device detection or initialization action requested 0001 Perform interface communication initialization sequence to establish communication. This is functionally equivalent to a hard reset and results in the interface being reset and communications re-initialized. Upon a write to the SControl register that sets the DET field to 0001, the host interface should transition to the HP1: HR_Reset [Delete space after state and should remain in that state until the DET field is set to a value other than 0001 by a subsequent write to the SControl register. 0100 Disable the SATA interface and put PHY in offline mode All other values reserved

16.3.3.4 SATA Interface Notification Register (SNotification)

SNotification, shown in Figure 16-16, is a 32-bit, write-one-to-clear register that conveys the devices that have sent the host a set device bits FIS with the notification bit. When the host receives a set device bits FIS with the notification bit set to 1, the host should set the bit in SNotification corresponding to the value of the PM port field in the received FIS. For example, if the PM port field is set to 7 then the host should clear bit 7 by writing a 1 to it. Next, the host should generate an interrupt if the I bit of the set device bits FIS is set to 1 and interrupts are enabled.

In this register, bits previously set are explicitly cleared by a write operation or by a power-on-reset operation. If the register is not cleared due to a COMRESET, the software is responsible for clearing the register as appropriate.

Offset 0x1_810C — Access: w1c

	31–16	15–0
R W	—	Notify*n*
Reset	All zeros	

Figure 16-16. SATA Interface Notification Register (SNotification)

Table 16-16 describes the SNotification fields.

Table 16-16. SNotification Field Descriptions

Bit	Name	Description
31–16	—	Reserved, should be cleared.
15–0	Notify*n*	Represents whether a particular device with the corresponding PM port number *n* has sent a set device bits FIS to the host with the notification bit set.

16.3.4 Control Status Registers

16.3.4.1 Transport Layer Configuration Register (TransCfg)

TransCfg, shown in Figure 16-17, controls the configuration of the transport layer.

Offset 0x1_8140 — Access: Read/Write

	31–16	15–5	4–0
R W	DFIS_SIZE	—	RX_WATER_MARK
Reset	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	1 0 1 1 0

Figure 16-17. Transport Layer Configuration Register (TransCfg)

Table 16-17 describes the TransCfg fields.

Table 16-17. TransCfg Field Descriptions

Bit	Name	Description
31–16	DFIS_SIZE	Data FIS framing length words. Determines the maximum length each data FIS should be.
15–5	—	Reserved
4–0	RX_WATER_MARK	This sets the number of locations in the 58-deep Rx FIFO that can be used before the transport layer instructs the link layer to transmit HOLDS to the transmitting end. Note that it can take some time for the HOLDs to get to the other end, and that in the interim there must be enough room in the FIFO to absorb all data that could arrive. An initial value of 22 is recommended.

16.3.4.2 Transport Layer Status Register (TransStatus)

TransStatus, shown in Figure 16-18, can be read to determine the status of the transport layer.

Offset 0x1_8144 — Access: Read only

	15–8	7–0
R/W	TX_SM	RX_SM
Reset		

Figure 16-18. Transport Layer Status Register (TransStatus)

Table 16-18 describes the TransStatus fields.

Table 16-18. TransStatus Field Descriptions

Bit	Name	Description
15–8	TX_SM	Indicates the state of Tx state machine.
7–0	RX_SM	Indicates the state of Rx state machine.

16.3.4.3 Link Layer Configuration Register (LinkCfg)

LinkCfg, shown in Figure 16-19, controls the configuration of the link layer.

Offset 0x1_8148 — Access: Read/Write

	31–27	26	25–16
R/W	—	POE	PRT
Reset	All zeros		

	15–8	7	6	5	4	3	2	1	0
R/W	AR	EPNRT	S4A	RX_SCR_EN	TX_SCR_EN	TX_PRIM_JUNK	TX_CONT_EN	RX_BAD_CRC	TX_BAD_CRC
Reset	1 1 1 1 1 1 1 1	0	0	1	1	0	1	0	0

Figure 16-19. Link Layer Configuration Register (LinkCfg)

Table 16-19 describes the LinkCfg fields.

Table 16-19. LinkCfg Field Descriptions

Bit	Name	Description
31–27	—	Reserved
26	POE	Primitive override enable. When set, this bit enables the replacement of a single primitive, as specified by CFG_PRIM/CFG_CD, when the link layer state machine is in the CFG_PRIM_OVR_STATE state. This bit has to be toggled from a 0 to a 1 to enable this feature.

Table 16-19. LinkCfg Field Descriptions (continued)

Bit	Name	Description
25–16	PRT	PHY ready timer. These ten bits specify the timeout value of the PHY_READY timer. If EN_PHY_TO is set, the link layer will count down on every rising edge of scanTxClk, as long as PHY_READY is de-asserted. When the counter reaches 0, a PHY_RESET will be issued to the PHY to try and re-establish communications with the far end. The timer is initially loaded with a value equal to the concatenation of {PHY_READY_TIMER, 9b0_0000_0000}.
15–8	AR	Align insertion rate. The SATA specification requires that the link layer send a pair of ALIGN primitives at least every 254 words of data. This is achieved by setting ALIGN_RATE to ‘11111111’. However, for test purposes it is possible to send ALIGNs at a higher rate. This can be achieved by setting ALIGN_RATE to a lower value (that is, ALIGN_RATE-1); words will be sent by the link layer between each set of ALIGN primitive pairs. **Note:** If SEND_4_ALIGNS is set, one should not set the ALIGN_RATE to be four or less. If SEND_4_ALIGNS is not set, one should not set the ALIGN_RATE to be two or less.
7	EPNRT	Enable PHY not ready timer. If PHY_READY is de-asserted for a length of time, as specified by CFG_PHY_READY_TIMER, then this bit, when asserted, enables the link layer to re-issue a PHY_RESET, thereby re-initiating OOB.
6	S4A	Send four ALIGNs. When asserted, four ALIGN primitives are transmitted at the specified rate, instead of the normal two ALIGNS.
5	RX_SCR_EN	Rx scramble enable. If this bit is asserted, then descrambling of the receive data is enabled as per the SATA specification.
4	TX_SCR_EN	Tx scramble enable. If this bit is asserted, then scrambling of the transmit data is enabled as per the SATA specification.
3	TX_PRIM_JUNK	TX prim junk. If this bit is de-asserted, then scrambled junk data is sent after a CONT primitive, as per the SATA specification. If this bit is asserted, then the single character 0xDEADBEEF is sent continuously instead. This is to aid debug.
2	TX_CONT_EN	TX CONT. If this bit is asserted, then the transmission of CONT primitives is enabled. If de-asserted, then long sequences of repeated primitives can be sent by the link layer.
1	RX_BAD_CRC	RX bad CRC. When a rising edge is detected on this bit, it causes a bad CRC to be detected for the current frame. This bit has to be toggled from a 0 to a 1 to enable this feature.
0	TX_BAD_CRC	Tx bad CRC. A bad CRC (inverted value of the correct CRC) value will be transmitted for one FIS only by the link layer when a rising edge is detected on this signal. This bit has to be toggled from a 0 to a 1 to enable this feature.

16.3.4.4 Link Layer Configuration Register1 (LinkCfg1)

LinkCfg1, shown in Figure 16-20, controls the configuration of the link layer.

Offset 0x1_814C — Access: Read/Write

	31–7	6	5–0
R W	—	CD	PRIM_OVR_STATE
Reset	All zeros		

Figure 16-20. Link Layer Configuration Register1 (LinkCfg1)

Table 16-20 describes the LinkCfg1 fields.

Table 16-20. LinkCfg1 Field Descriptions

Bit	Name	Description
31–7	—	Reserved
6	CD	This bit specifies whether the data used during the primitive override should be a data character or a primitive. For example, if CD = 1, PRIM_OVR_STATE = L_SendEOF and PRIM = WTRM, then a WTRM primitive will be inserted into the datastream instead of an EOF (whenever a rising edge is seen on PRIM_OVR_EN). If CD = 0, then a normal data character (as specified by PRIM) is inserted into the datastream instead of the EOF.
5–0	PRIM_OVR_STATE	Prim override state. These 6 bits are used in the primitive override debug functionality. When the link layer detects a positive edge on PRIM_OVR_EN, it overrides the next primitive that would be inserted during the PRIM_OVR_STATE, with the data specified by the PRIM and CD configuration bits.

16.3.4.5 Link Layer Configuration Register2 (LinkCfg2)

LinkCfg2, shown in Figure 16-21, controls the configuration of the link layer.

Offset 0x1_8150 Access: Read/Write

	31–0
R W	PRIM
Reset	All zeros

Figure 16-21. Link Layer Configuration Register1 (LinkCfg1)

Table 16-21 describes the LinkCfg2 fields.

Table 16-21. LinkCfg2 Field Descriptions

Bit	Name	Description
31–0	PRIM	This 32-bit bus specifies the data to be used in the overriding primitive debug logic, described in the definition of LinkCfg1 register.

16.3.4.6 Link Layer Status Register (LinkStatus)

LinkStatus, shown in Figure 16-22, indicates the status of the link layer.

Offset 0x1_8154 Access: Read only

	31–6	5–0
R W	—	LINK_STATE
Reset	All zeros	

Figure 16-22. Link Layer Status Register (LinkStatus)

Table 16-22 describes the LinkStatus fields.

Table 16-22. LinkStatus Field Descriptions

Bit	Name	Description
31–6	—	Reserved
5–0	LINK_STATE	Current value of the link layer state machine at the time the LinkStatus register is read. L_Reset = 0 L_Idle = 1 HL_SendChkRdy = 2 DL_SendChkRdy = 3 L_TPMPartial = 4 L_TPMSlumber = 5 L_RcvWaitFifo = 6 L_PMOff = 7 L_PMDeny = 8 L_NoCommErr= 9 L_NoComm = 10 L_SendAlign = 11 L_SendSOF = 12 L_SendData = 13 WAIT_FOR_SYNC = 14 L_SendCRC = 15 L_SendHold = 16 L_RcvHold = 17 L_SendEOF = 18 L_Wait = 19 L_ChkPhyRdy = 20 BIST1 = 42 L_NoCommPower = 21 L_WakeUp1 = 22 L_WakeUp2 = 23 L_RcvChkRdy = 24 L_RcvData = 25 L_BadEnd = 26 L_RcvEOF = 27 L_SendHoldA = 28 L_Hold = 29 L_GoodCRC = 30 L_GoodEnd = 31 L_PMOff_2 = 32 L_PMOff_3 = 33 L_PMOff_4 = 34 WAIT_PMACK_SENT_1 = 35 WAIT_PMACK_SENT_2 = 36 WAIT_PMACK_SENT_3 = 37 WAIT_PMACK_SENT_4 = 38 WAIT_PMACK_SENT_5 = 39 WAIT_PMACK_SENT_6 = 40 BIST0 = 41

16.3.4.7 Link Layer Status Register1 (LinkStatus1)

LinkStatus1, shown in Figure 16-23, indicates the status of the link layer.

Offset 0x1_8158 Access: Read only

	31–24	23–16	15–8	7–0
R W	KEGC	PIEGC	CEGC	DEGC
Reset	All zeros			

Figure 16-23. Link Layer Status Register1 (LinkStatus1)

Table 16-23 describes the LinkStatus1 fields.

Table 16-23. LinkStatus1 Field Descriptions

Bit	Name	Description
31–24	KEGC	Kchar error gray count. The number of words received from the PHY, where one or more control character errors have been detected. A value of 255 indicates an error count of 255 or more as this counter does not wrap around to 0. The count value is updated with its current value each time the Status1 register is read. The count is represented in gray code.
23–16	PIEGC	PHY internal error gray count. The number of words received from the PHY, where one or more internal errors have been detected. A value of 255 indicates an error count of 255 or more as this counter does not wrap around to 0. The count value is updated with its current value each time the Status1 register is read. The count is represented in gray code.
15–8	CEGC	Code error gray count: The number of words received from the PHY, where one or more code errors have been detected. A value of 255 indicates an error count of 255 or more as this counter does not wrap around to 0. The count value is updated with its current value each time the Status1 register is read. The count is represented in gray code.
7–0	DEGC	Disparity error gray count. The number of words received from the PHY, where one or more disparity errors have been detected. A value of 255 indicates an error count of 255 or more as this counter does not wrap around to 0. The count value is updated with its current value each time the Status1 register is read. The count is represented in gray code.

Sample C code to convert gray counts to binary:

```
int Gray2Binary(int gray)
{
int ish;
unsigned long ans,idiv;
ish=1; This is the more complicated direction: In hierarchical stages, starting with a one-bit
right shift, cause each bit to be XORed with all more significant bits.
Ans=gray;
for (;;)
    {
    ans ^= (idiv=ans >> ish);
    if (idiv <= 1 || ish == 16) return ans;
    ish <<= 1; Double the amount of shift on the next cycle.
    }
}
```

16.3.4.8 PHY Control Configuration Register1 (PhyCtrlCfg1)

PhyCtrlCfg1, shown in Figure 16-24, controls the configuration of the link layer.

Offset 0x1_815C   Access: Read/Write

	31–16
R/W	—
Reset	All zeros

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R/W	—			ENDEC_EN[1]	—			FPRFTI		—			LPB_EN			—
Reset	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0

1 Reset value must be preserved when writing to the register.

Figure 16-24. PHY Control Configuration Register1 (PhyCtrlCfg1)

Table 16-24 describes the PhyCtrlCfg1 fields.

Table 16-24. PhyCtrlCfg1 Field Descriptions

Bit	Name	Description
31–13	—	Reserved
12	ENDEC_EN	Encode decode enable. When asserted high, it enables the PCS to operate in 8 bit mode, and to enable 8B/10B encoding and decoding. When negated low, the PCS is configured to operate in 10-bit mode; the 8B/10B encoder/decoders are bypassed and it is assumed this is done elsewhere.
11–9	—	Reserved. Reset value must be preserved when writing to the register.
8–7	FPRFTI	Force PHY ready, force Tx idle. This pair of signals determines how phyRdy is driven, how the output buffer IDLE condition is controlled and how disparity errors in ALIGN primitives should be tolerated during OOB. The IDLE condition is defined in SATA as both traces of the transmit differential pair being driven to common mode. • frcPhyRdy = 0 • frcTxIdle = 0 In this mode phyRdy and Tx buffer IDLE control driven by OOB state machine. Disparity errors in ALIGN primitives are not tolerated during OOB. • frcPhyRdy = 0 • frcTxIdle = 1 In this mode phyRdy and Tx buffer IDLE control driven by OOB state machine. Disparity errors in ALIGN primitives are tolerated during OOB. • frcPhyRdy = 1 • frcTxIdle = 0 In this mode phyRdy is asserted high and Tx buffer IDLE control is forced off, causing the output buffer to be enabled. Tolerance of disparity errors in ALIGN primitives is of no consequence, because OOB is bypassed. • frcPhyRdy = 1 • frcTxIdle = 1 In this mode phyRdy is asserted high and Tx buffer IDLE control is forced on, causing the output buffer to the IDLE condition. Tolerance of disparity errors in ALIGN primitives is of no consequence as OOB is bypassed.
6–4	—	Reserved

Table 16-24. PhyCtrlCfg1 Field Descriptions (continued)

Bit	Name	Description
3–1	LPB_EN	Loopback enable. These bits control both loopback modes and power management modes. 000 No loopback and in normal power mode 001 Far end re-timed (parallel) loopback enabled 010 Near end analog (serial) loopback enabled 011 Invalid 100 Invalid 101 goPartial. This encoding results in the OOB state machine entering the partial state. Note that in the PCS, partial and slumber have the same effect. 110 goSlumber. This encoding results in the OOB state machine entering the slumber state. Note that in the PCS, partial and slumber have the same effect. 111Invalid **Note:** This field is available only for SATA1.
0	—	Reserved

16.3.4.9 Link Layer Command Status Register (CommandStatus)

CommandStatus, shown in Figure 16-25, indicates the status of the command layer.

Offset 0x1_8160 Access: Read only

	31–24	23–20	19–16
R	—	DMATCmdNo	DMATPMPNo
W			
Reset	All zeros		

	15–12	11–8	7–0
R	RecPMPLocal	CSState	CMState
W			
Reset	All zeros		

Figure 16-25. Link Layer Command Status Register (CommandStatus)

Table 16-25 describes the CommandStatus fields.

Table 16-25. CommandStatus Field Descriptions

Bit	Name	Description
31–24	—	Reserved
23–20	DMATCmdNo	
19–16	DMATPMPNo	
15–12	RecPMPLocal	
11–8	CSState	CSSIdle = 0x0 CSSNPS = 0x4 CSSPF = 0x7 CSSGP = 0x1 CSSSC = 0x5 CSSTO = 0x8 CSSPCA = 0x2 CSSNCS = 0x6 CSSWCI = 0x9

Table 16-25. CommandStatus Field Descriptions (continued)

Bit	Name	Description
7–0	CMState	CMIdle = 0x00 CMFatalError = 0x01 CMWE = 0x02 CMRF = 0x03 CMSNDF = 0x04 CMWSNDF = 0x05 CMWSNDFWUCA = 0x06 CMCIWNE = 0x07 CMCIWIF = 0x08 CMCISNDS = 0x09 CMCIWDMAC = 0x0A CMRUF = 0x0B CMRUFUS = 0x0C CMRUFWUS = 0x0D CMRWSU = 0x0E CMRUFWMW = 0x0F CMSDB = 0x10 CMSDBWSN = 0x11 CMSDBCleanACK = 0x12 CMSDBCCT = 0x13 CMSDBPRT = 0x14 CMSDBNT = 0x15 CMSDBLT = 0x16 CMSDBFCC = 0x17 CMSDBNCC = 0x18 CMSDBLCC = 0x19 CMSDBWFT = 0x1A CMRDMAS = 0x1B CMRDMASTC = 0x1C CMRDMASTNC = 0x1D CMRDMASTLC = 0x1E CMRDMASTT = 0x1F CMRDMASTL = 0x20 CMRDMASWFTF = 0x21 CMRDMASDW = 0x22 CMPIOS = 0x23 CMPIOSWFTF = 0x24 CMPIOSDW = 0x25 CMSATAPI = 0x26 CMWFC = 0x27 CMDF = 0x28 CMDFWD = 0x29 CMDFWDW = 0x2A CMDFWCRC = 0x2B CMDFBSY = 0x2C CMDMAA = 0x2D CMDMAAWFTF = 0x2E CMDMAADW = 0x2F CMSD = 0x30 CMDC = 0x31 CMWDC = 0x32 CMWU = 0x33 CMWRFD = 0x34 CMWCC = 0x35 CMRUNF = 0x36 CMRUNFC = 0x37 CMFatalErrorUpdate = 0x38

16.3.5 System Control Registers

16.3.5.1 System Priority Register (SYSPR)

SYSPR, shown in Figure 16-26, can be used to control various settings that affect the system response to DMA operations. Note that the bit ordering is 0–31, rather than 31–0 of the other registers in this chapter.

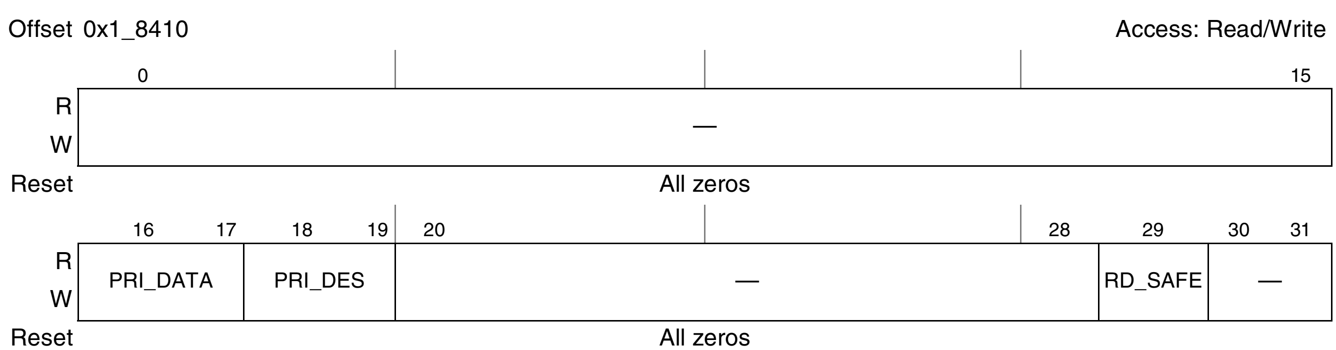


Figure 16-26. System Priority Register (SYSPR)

Table 16-26 describes the SYSPR fields.

Table 16-26. SYSPR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–17	PRI_DATA	Priority. This field will be used to present priority level for CSB arbitration for SATA controller's dma requests. Bits 16-17 will be used when the request belongs to data transfer. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
18–19	PRI_DES	Priority. This field will be used to present priority level for CSB arbitration for the SATA controller's DMA requests. Bits 18–19 will be used when the request belongs to descriptor fetch. 00 Level 0 (lowest priority) 01 Level 1 10 Level 2 11 Level 3 (highest priority)
10–28	—	Reserved
29	RD_SAFE	Read safe. This bit should be set only if the target of the read DMA operation is a well behaved memory that is not affected by the read operation and which will return the same data if read again from the same location. This means that unaligned reading operation can be rounded up to enable more efficient read operations. 0 It is not safe to read more bytes that were intended. 1 It is safe to read more bytes that were intended.

16.3.6 Command Header

Each entry in the command header table consists of the structure shown in Figure 16-27.

NOTE

In this chapter, “word” refers to 4 bytes or 32 bits.

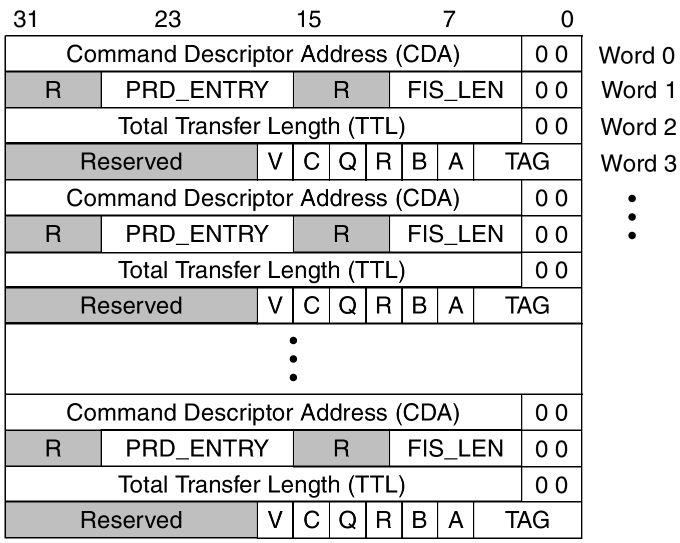


Figure 16-27. Command Header

Table 16-27 shows word 0—data base address—of the command header.

Table 16-27. Word 0—Data Base Address

Bit	Name	Description
31–2	CDA	Command descriptor base address. Indicates the 32-bit physical address of the command descriptor block. The block must be word-aligned, indicated by bits 1–0 being reserved.
1–0	—	Reserved

Table 16-28 shows word 1—FIS_LEN—of the command header.

Table 16-28. Word 1—FIS_LEN

Bit	Name	Description
31–22	—	Reserved
21–16	PRD_ENTRY	Number of PRD entries including indexed entries.
7	—	Reserved
6–2	FIS_LEN	FIS length. This is a 5-bit word count of the total length of the control or vendor-specific FIS to transfer.
1–0	—	Reserved

Table 16-29 shows word 2—data base address—of the command header.

Table 16-29. Word 2—Data Base Address

Bit	Name	Description
31–2	TTL	Total transfer length. This is a 30-bit word count of the total length of the data transfer. It is used to detect overruns/underruns between the transfer lengths programmed in the command and the PRDT.
1–0	—	Reserved

Table 16-30 shows word 3—description information—of the command header.

Table 16-30. Word 3—Description Information

Bit	Name	Description
31–12	—	Reserved
11	—	Reserved, should be 1.
10	V	Vendor BIST. When this bit is set, it indicates that the command is a Vendor BIST, thus FIS will loop back at the PHY local test.
9	C	Snoop enable during all descriptor read/write operations associated with this command.
8	Q	Queued. Command is an FPDMA queued command.
7	R	Reset. The command is a SRST or device reset.
6	B	BIST. The command will require the host to enter BIST mode.
5	A	ATAPI command. The command is an ATAPI command and thus will require that the host uses the ATAPI portion of the command descriptor to issue the command. The CFIS also has to be written with the packet command.
4–0	TAG	The 5-bit TAG assigned by software for command tracking. It is the same as the value written to the command register host-to-device.

16.3.7 Command Descriptor

As shown in Figure 16-28, each entry in the command list points to a structure called the command descriptor.

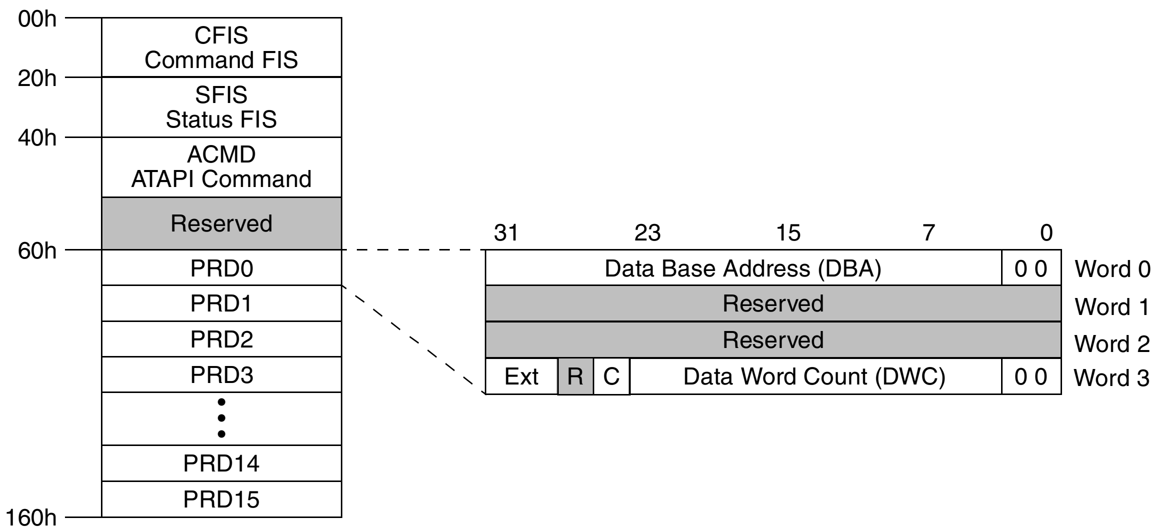


Figure 16-28. Command Descriptor

16.3.7.1 Command FIS Non-Queued Commands (CFIS)

Command FIS is a software constructed FIS. For data transfer operations, this is the H2D Register FIS format as specified in the Serial ATA 2.5 standard. The SATA controller fetches this from memory and sends the appropriate amount of data to the attached port. If a port multiplier is attached, this field must have the port multiplier port number in the FIS itself. CFIS lengths are two to eight words and must be in word granularity. A typical command FIS is shown in Figure 16-29.

Register FIS host-to-device

	31–24	23–16	15–12	11–8	7–0
0	Features	Command	C R R R	PM Port	FIS Type (27h)
1	Device	LBA High	LBA Mid		LBA Low
2	Features (exp)	LBA High (exp)	LBA Mid (exp)		LBA Low (exp) (0)
3	Control	Reserved (0)	Sector Count (exp)		Sector Count
4	Reserved (0)	Reserved (0)	Reserved (0)		Reserved (0)

Figure 16-29. Register Host-to-Device

16.3.7.2 Command FIS First Party DMA Commands NCQ

Figure 16-30 shows register host-to-device first party DMA commands NCQ. The shaded components show where this FIS differs for the non-NCQ register host-to-device FIS.

Register FIS host to device—Read/write FPDMA queued

	31 — 24	23 — 16	15 — 12	11 — 8	7 — 3, 2 — 0
0	Features Sector Count 7:0	Command	C R R R	PM Port	FIS Type (27h)
1	Device	LBA High	LBA Mid		LBA Low
2	Features (exp) Sector Count 15:8	LBA High (exp)	LBA Mid (exp)		LBA Low (exp) (0)
3	Control	Reserved (0)	Sector Count (exp) Reserved (0)		Sector Count TAG
4	Reserved (0)	Reserved (0)	Reserved (0)		Reserved (0)

Figure 16-30. Register Host-to-Device First Party DMA Commands NCQ

16.3.7.3 Status FIS (SFIS)

This FIS is created in hardware. For normal operations, this is the D2H register FIS format as specified in the Serial ATA 2.5 standard. SFIS lengths are two to eight words and must be of word granularity. A typical status FIS is shown in Figure 16-31.

Register FIS device to host

	31 — 24	23 — 16	15 — 12	11 — 8	7 — 0
0	Error	Status	R I R R	PM Port	FIS Type (34h)
1	Device	LBA High	LBA Mid		LBA Low
2	Reserved (0)	LBA High (exp)	LBA Mid (exp)		LBA Low (exp) (0)
3	Reserved (0)	Reserved (0)	Sector Count (exp)		Sector Count
4	Reserved (0)	Reserved (0)	Reserved (0)		Reserved (0)

Figure 16-31. Register Device-to-Host

16.3.7.4 ATAPI Command (ACMD)

This is a software constructed region of three or four words in length that contains the ATAPI command to transmit if the ‘A’ bit is set in the command header. The ATAPI command must be either 12 or 16 bytes in length. The length transmitted by the SATA IP is determined by the PIO setup FIS that is sent by the device requesting the ATAPI command.

16.3.7.5 Physical Region Descriptor Table (PRDT)

This is a software constructed table of addresses to use to complete the data transfer. Up to 16 structures can be supported in the current command descriptor. The format of the address entry is defined by the “Block vector structures for passing segmented data type of the IEEE Std. 1212.1-1993.” The total definable length supported in the 16 entries is 64 Mbytes.

Table 16-31 shows word 0—data base address—of the PRDT.

Table 16-31. Word 0—Data Base Address

Bit	Name	Description
31–2	DBA	Data base address. Indicates the 32-bit physical address of the data block. The block must be word aligned, indicated by bits 1–0 being “reserved, must be 00.”
1–0	—	Reserved

Table 16-32 shows word 3—description information—of the PRDT.

Table 16-32. Word 3—Description Information

Bit	Name	Description
31	EXT	If the extension flag is set to 1, then the DBA field contains the address of the extension segment, and the DWC field contains the size of this extension segment (this is called an “indirect descriptor”).
30–23	—	Reserved
22	C	Data snoop enable bit. When this bit is set, all data read/write operations associated with the PRD entry for this command will be snoopable.
21–2	DDC	Data word count. A 0-based value that Indicates the length, in words, of the data block. A maximum length of 4 Mbytes may exist for any entry. Bits 1–0 of this field must always be 0 to indicate that size is in words. A value of 0x0_0000 indicates a full 4 Mbytes transfer.
1–0	—	Reserved

16.3.8 Vendor-Specific BIST Operation

As part of the host self-diagnostic operation, a vendor-specific BIST mode is supported. This mode, in conjunction with a PHY that supports serial loopback, allows for the test of the SATA controller operation. The mode exercises the following paths:

- DMA controller FIS transmission
- Command layer FIS transmission
- Transport layer Tx FIFO FIS transmission
- Link layer FIS transmission
- PHY modes
- Link layer FIS reception
- Transport layer Rx FIFO and FIS reception
- Command layer FIS reception
- Command layer FIS reception
- Host DMA controller FIS reception

To run this self-test on SATA1, lane A, the software performs the following operations:

1. Initialize the SerDes as described in Section 16.7.1, “SerDes Initialization.”
2. Set the PhyCtrlCfg1[FPRFTI] field to 2'b10, meaning that frcPhyRdy = 1 and frcTxIdle = 0.

3. Follow steps 1 and 2 in Section 16.7.2, “SATA Controller Initialization Steps.”
4. Poll the SStatus[DET] till DET = 4’b0010, meaning that the PHY communication is established. In this state, SStatus[SPD] indicates the negotiated communication speed.
5. Modify the SRDS*n*CR1 control register to place the PHY into loopback mode (LBSEL=4'b0110).
6. Build the vendor-specific command header and descriptor in external memory as described in Table 16-33 and Table 16-34. Refer to Section 16.3.6, “Command Header,” and Section 16.3.7, “Command Descriptor,” for more details.

Table 16-33. Vendor BIST Test—Command Header

Word Number	Hexadecimal Value
Word 0	CDA
Word 1	0x0000_000C
Word 2	0x0000_0000
Word 3	0x0000_0400

Table 16-34. Vendor BIST Test—Command Descriptor

Word Number	Hexadecimal Value	Comments
Word 0	0x0001_0058	—
Word 1	0xAAAA_A034	AAAA_A is the first test pattern (can be any value)
Word 2	0xBBBB_B034	BBBB_B is the second test pattern (can be any value)
Word 3	Reserved	Reserved, must be all zeros
Word 4	Reserved	Reserved, must be all zeros
Word 5	Reserved	Reserved, must be all zeros
Word 6	Reserved	Reserved, must be all zeros
Word 7	Reserved	Reserved, must be all zeros

7. Initialize the CHBA register to point to the start of Command Header block.
8. Issue this command to the SATA controller, as described in steps 10 and 11 of Section 16.7.2, “SATA Controller Initialization Steps.”
9. When the SATA controller indicates that command has completed (refer to Step 12 in Section 16.7.2, “SATA Controller Initialization Steps”), the software examines the contexts of the command descriptor’s SFIS field (offset 0x20 from Command Descriptor Base). If the predefined test words are present, the test has passed. Note that the order of the received test words may be swapped.
10. For checking SATA controller 2 a similar sequence should be applied by pointing to SATA2 registers and programming appropriate values for Lane E in the SerDes registers.
11. For checking SATA controllers 3-4 a similar sequence should be applied by pointing to SATA3 and SATA4 registers, and programming the SerDes2 registers.

16.4 Transport Layer Architectural Overview

The function of the SATA transport layer is to interface between the command and link layers in the transmission and reception of FIS.

On the transmit path, the transport layer frames the FIS's placed into the Tx FIFO. The FIS's are framed based on a programmed length for non-data FIS and are a configurable length for data FIS. When the transport layer is instructed to send a non-data FIS, it employs a retry policy until the far end signals acceptance of the transmitted FIS.

On the reception path, the transport layer deframes the FIS's and places them into the Rx FIFO. When an FIS is received, the transport layer informs the command layer. For a non-data FIS, the FIS is considered received when the end-of-frame (EOF) is signaled by the link layer and the FIS has been received with a good CRC. For a short vendor-specific FIS, the FIS is considered as a non-data FIS. For a longer vendor-specific FIS, the FIS reception is signaled when the RX FIFO reaches its water mark. For a data FIS, the FIS is considered received when the first word (header) is written into the FIFO.

The receive FIFO is written with data contained in the FIS sent by the link layer. When the data is stable at the output of the receive FIFO, the command layer can take the data. If the command layer is not ready to accept the data, the data builds up in the receive FIFO. When the receive FIFO exceeds its threshold, the transport layer stalls the link layer, which will in turn send HOLD primitives to the far end. This threshold takes into consideration the latency involved in getting the far end to stop transmitting the data. This threshold is programmable to allow for the use of high-latency repeaters or retainers in between the host and device.

The transmit FIFO is written with data to be sent in the FIS transferred by the DMA controller. When the data is stable at the output of the transmit FIFO, the link layer can take the data. If the transmit FIFO cannot supply data to the link layer, the transport layer stalls the link layer, which will in turn send HOLD primitives to the far end.

16.5 Link Layer Overview

The function of the SATA link layer is to interface between the transport and physical layers in the transmission and reception of frames and primitives. The link layer utilizes the two unidirectional links provided by the SATA interface to maintain coordinated communication between the host and the device. Payload data can only be transmitted in one direction at a time. The link layer can work at either SATA first-generation (1.5 Gbps) or second-generation 2 (3 Gbps) speeds.

On transmit, the link layer first communicates with the peer far end link layer to determine if it is ready to receive. Assuming the far end link layer can receive data, the local link layer can then begin to take data in the form of words from its transport layer. It inserts start-of-frame (SOF) before the start of the data portion of a frame, calculates and inserts the CRC after the data portion of a frame, and inserts the EOF primitive at the end. The link layer scrambles the contents of the frame, including the calculated CRC, but excluding the SOF and EOF diameters and any other embedded primitives. The 8B/10B encoding of the data is done in the PHY layer. At the end of the transmission, the link layer reports transmission status to the transport layer.

On receive, the link layer first acknowledges its readiness to receive with its peer link layer. Then it awaits reception of the SOF primitive that marks the start of the received data. Following detection of the SOF primitive, the link layer proceeds to accept the incoming data. The 8B/10B decoding of the data is done in the PHY layer. Next, the link layer removes all primitives including the SOF and EOF diameters. It then descrambles the contents of the frame. The link layer also calculates the CRC on the incoming frame between the SOF and EOF delimiters, and compares this calculated value to the received value. Any mismatch is reported to the transport layer. During frame reception, disparity or code errors are reported to the command layer, and appropriate action is taken in the link layer. The descrambled and decoded receive data stream is passed to the transport layer as the frame is being received. Finally, at the end of the frame, the link layer reports reception status to the transport layer.

The link layer also partakes in flow control between the local and remote ends. The layer supports flow control actions based on the local FIFO status (located in the transport layer), or in response to receiving flow control messages from the remote end.

The transmit side of the link layer is also responsible for inserting a pair of ALIGN primitives every 254 words, or more frequently if programmed by the user.

16.5.1 Link Layer functionality

The link layer is composed of a number of functions:

- Link layer state machines
- Frame content scrambler and descrambler
- CRC generation and checking
- Bus interfaces to PHY and transport layer
- CONT primitive processing
- ALIGN insertion on transmit
- Debug functionality
- BIST support
- Link layer state machines

The four link layer state machines are described in the following sections.

16.5.1.1 Link Idle State Machine

The link idle state machine is responsible for detecting a transmit request from the transport layer or a frame reception request from the far end. The state machine arbitrates whether these two events coincide. The SATA specification defines that the host end always backs down in this case. Furthermore, this machine interprets power mode change requests from both the transport and PhyCtrl layers and initiates actions to enable the power mode change. Power mode change can only occur if the feature is enabled via the PhyCtrlCfg register LPB_EN bits. Finally, this state machine also detects the negation and assertion of PHY_READY from the PHY and notifies the transport layer of the change.

16.5.1.2 Transmit State Machine

This state machine is responsible for frame transmission to the PHY. The state machine places the SOF and EOF headers on each frame, calculates the CRC, and inserts it before the EOF delimiter. Between the SOF and CRC markers, the link layer accepts the current word from the transport layer and uses this as the next word of the frame. The link layer also inserts a pair of ALIGN primitives every 254 words of frame data. Finally, at the end of the frame transmission, the state machine waits for status from the far end link layer via received R_OK or R_ERR primitives. If the far end received the frame correctly, the local link layer signals TX_OK to the transport layer; otherwise, it signals TX_NOT_OK to the transport layer.

The transmit state machine also partakes in flow control actions, if necessary, during packet transmission. If the transport layer cannot supply a new word and the frame is not finished, the transmit state machine responds by sending HOLD primitives until the transport layer is ready with valid frame data. Also, during frame transmission, if the state machine detects a received HOLD primitive from the PHY layer, it interrupts the current frame transmission and sends HOLDA primitives to the PHY to be transmitted to the far end.

The current frame transmission can only be aborted by two events. The first is on reception of a DMAT primitive from the far end. In this case, the link layer state machine stops the current transfer and calculates and inserts the current CRC. This is a controlled termination. The second is when the transport layer wishes to send a control register frame signaled via TRANSMIT_CRF.

If at any point in the frame transmission process, the link layer detects error conditions, it signals these to the command layer. The errors can occur if the link layer detects the following conditions:

- PHY_READY negates
- SYNC primitive is received during frame transmission

16.5.1.3 Receive State Machine

This state machine is responsible for frame reception from the PHY layer. The state machine removes the SOF and EOF headers and other primitives from each frame, calculates the CRC, and compares it to the received CRC. Between the SOF and CRC markers, the link layer accepts the current word from the Phy layer and uses this as the next word of the frame, transferring it to the transport layer. At the end of the frame reception, if the calculated CRC is not the same as the received CRC, the link layer signals an error to the transport layer. This is done via RX_CRC_OK and RX_CRC_NOT_OK. During frame reception, if no errors are detected, the link layer transmits R_IP primitives to the far end peer link layer. Finally, at the end of the frame reception, the link layer sends the R_OK primitive if no error was detected during reception. If an error was detected, it sends a R_ERR primitive instead.

The receive state machine also partakes in flow control actions if necessary, during FIS reception. If the transport layer cannot accept a new word, (because its receive FIFO has reached its watermark level), and the FIS is not finished, the receive state machine responds by sending HOLD primitives on the back channel until such time as the transport layer is ready to accept FIS data again. Also, during FIS reception, if the state machine detects a received HOLD primitive from the far end, it responds by sending HOLDA primitives to the far end.

The current frame reception can be interrupted if the transport layer wishes to send a control register frame, signaled via TRANSMIT_CRF.

If at any point in the frame reception process, the link layer detects error conditions, it signals these to the command layer. The errors can occur if the link layer detects the following conditions:

- PHY_READY negates
- SYNC primitives is received during frame transmission
- WTRM primitive is received before EOF

16.5.1.4 Power Mode Change State Machine

This state machine is responsible for handling change of power mode requests. These requests can come from the command layer superset registers or the far end. This state machine responds by transmitting PMREQ_P/PMREQ_S primitives to the far end and waiting for PMACK primitives from it in response. Once PMACK is received, the state machine instructs the PHY layer to enter either a partial or slumber state.

A write to the SControl register SPM field or reception of a COMWAKE from the far end will initiate a resume to active power mode.

If the link layer receives a PMREQ_P/PMREQ_S primitive from a peer link layer and is enabled to perform power management modes (SControl IPM bits are cleared), it responds by sending at least four PMACK primitives. A write to the SControl register SPM field or reception of a COMWAKE from the far end will initiate a resume to active power mode.

If the link layer receives an XRDY primitive from the far end while it is in the partial or slumber state, it returns to idle and signals a link sequence error to the command layer, that is, SError[S] = 1.

16.5.1.5 Frame Content Scrambler and Descrambler

There are two separate scramblers used in the SATA controller, one for the data payload and the other for repeated primitive suppression. The contents of each word of data (excluding all primitives) between SOF and EOF must be scrambled before 8B/10B encoding. Scrambling is performed on word quantities according to the following polynomial:

G(X) = X16 + X15 + X13 + X4 + 1

The scrambler is initialized with a seed value of 0xFFFF at each SOF transmission and rolls over every 2048 words. Payload data is scrambled prior to transmission, by XORing the data to be transmitted with the output of this scrambler.

If a CONT primitive is transmitted, then the intervening data between the last CONT primitive and a subsequent primitive must be scrambled also. This scrambler uses the same polynomial as defined above for data payload scrambling and is reset to the initial value upon detection of a COMINIT or COMRESET event. If a CONT primitive is transmitted or received during a frame transfer, then the current data payload scrambler value at the last word is held.

When payload data is received by the link layer it is descrambled by XORing it with the output of its descrambler. The descrambler is re-seeded at the beginning of the received data payload, that is, at each SOF reception. The descrambler uses the same polynomial as the scrambler.

16.5.1.6 CRC Generator and Checker

A 32-bit CRC is calculated on the data contents of each frame and is inserted in the word before the EOF. The CRC covers all data bytes in the frame excluding any primitives such as SOF, EOF, HOLD, HOLDA, DMAT, SYNC, X_RDY, R_RDY, or ALIGNs.

The CRC generator works on word quantities. Any padding to the boundary is done in the transport layer. The polynomial used for the CRC is as follows:

G(X) = X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X + 1

The CRC is initialized with a seed value of 0x52325032 at each SOF.

The CRC generation or checking does not apply to primitives (as stated above) or to CONT'ed primitives. If a CONT primitive is transmitted or received, then the intervening data between the last CONT primitive and a subsequent primitive is not included in the CRC calculation for a frame. If this happens during a frame transfer, then the current CRC scrambler value at the last word is held.

16.5.1.7 8B/10B Encode and Decode

All data and primitives must be encoded prior to transmission on the line. The 8B/10B encode/decode occur in the PHY layer.

16.5.1.8 CONT Primitive Processing

Using the CONT primitive, the link layer is capable of replacing repetitive primitive streams with scrambled data. This reduces EMI emissions because primitives are not scrambled.

The link layer can transmit a CONT primitive at a point where it knows it must transmit a number of repeating primitives. After a CONT primitive has been transmitted, the link layer then transmits scrambled junk data to the PHY layer. The content of this junk data is disregarded. At the far end link layer, the reception of a CONT primitive will cause the last received valid primitive to be implied to be repeated until it receives the next valid non-ALIGN primitive. Transmission of a new valid primitive halts the current CONT processing; reception of a new valid non-ALIGN primitive halts the current CONT processing.

This action can occur on transmit and receive. The link layer supports both the transmission and reception of CONT primitives.

16.5.1.9 ALIGN Insertion

The link layer is responsible for ALIGN insertion and removal at a fixed frequency. A pair of ALIGN primitives are inserted into the transmit data stream every 254 words. At the receive end, the ALIGN primitives are stripped from the incoming data stream in the link layer.

For diagnostic purposes, the rate of ALIGNs can be increased as much as two ALIGNS per one word; for example, ALIGN, ALIGN, data, ALIGN, ALIGN. In addition, the SEND_4_ALIGNS bit can be set to instruct the link layer to send four ALIGNs at a time instead of two.

16.5.1.10 Debug Functionality

There are a number of useful features designed into the link layer to aid debug, as follows:

- The align insertion rate can be increased using the ALIGN_RATE register field in the command layer.
- Four error counters can be monitored by issuing register reads to the command layer: the disparity error counter, the code error counter, the PHY internal error counter, and the control character error counter.
- A number of configuration bits in the command layer can be used to override normal primitive insertion. For example,
  - — Set PRIM_OVR_STATE = (L_SendHold state (16))
  - — Set PRIM = 0xb5b5957c, that is, a SYNC primitive
  - — During the transfer, set PRIM_OVRD_EN = 1
- When the link layer detects a rising edge on PRIM_OVRD_EN, it will insert one SYNC primitive into the datastream in place of the HOLD, when the LINK_STATE reaches the L_SendHold state. Only one HOLD primitive will be overridden; the PRIM_OVRD_EN must be cleared and written to again to force another override to occur.

16.5.1.11 BIST Support

The transmit and receive subblocks of the link layer contain logic to support BIST activate FIS functionality.

When a BIST activate FIS is either received or transmitted successfully by the transport layer, it issues a request to the link layer to enter BIST mode. This forces the link layer to enter a BIST state in its state machine as soon as it receives a SYNC primitive from the far end. In the BIST state, the link layer transmits a data sequence as specified by the two BIST data patterns in the BIST activate FIS. The link layer also monitors the incoming data from the PHY to detect the BIST data pattern is as specified in the BIST activate FIS. When it detects the correct data sequence, the HStatus[BIST_Err] is deasserted. The BIST_Err bit will stay deasserted unless an error occurs in the datastream from the far end.

16.6 PHY Control Layer Overview

The PHY control layer operates between the PHY and link layers. On receive, the PHY control layer converts the 16-bit parallel data from the PHY to a 32-bit word, which it presents to the link layer. The PHY control layer aligns the control word of the SATA primitive to the lowest word position of the word. The PHY control layer takes in the per-byte error signals and the per-byte control/data bits output by the PHY and converts them into 4-bit buses, with each bit of the bus corresponding to a byte in the word.

On transmit, the PHY control layer takes in the 32-bit transmit data from the link layer and converts it to 16 bits of data which it presents to the PHY. The control/data bit from the link layer (which is always assumed to be associated with the lowest byte position of the transmit word) is also passed onto the PHY with the appropriate word.

16.7 Initialization/Application Information

The typical initialization sequence is divided into two main parts and should be performed in this order:

- SerDes initialization
- SATA controller initialization

16.7.1 SerDes Initialization

The purpose of the SerDes initialization is to configure the SerDes to operate in SATA mode. For details, refer to Section 19.4, "Initialization Sequence and Reset."

16.7.2 SATA Controller Initialization Steps

These steps bring the SATA controller online, synchronize the SATA controller with the attached device, and issue typical command for execution.

1. Write HControl[HC_ON] = 1 to bring the SATA controller online.
2. Poll the HStatus[HS_ON] till HS_ON = 1, indicating that the controller is online.
3. Poll the SStatus[DET] till DET = 4'b0011 meaning that the device presence is detected and PHY communication is established. In this state, SStatus[SPD] indicates the negotiated communication speed.
4. To read the device's signature, poll HStatus[SIG_UPD] till it goes up. Read the signature from the SIG register.
5. Initialize the CHBA register to point to the command header block.
6. Build a command header block in memory. Refer to Section 16.3.6, "Command Header."
7. Build a command descriptor block in memory. Refer to Section 16.3.7, "Command Descriptor."
8. Build a number of PRD tables in memory as defined by the PRD_NUM field in the command header. Refer to Section 16.3.7.5, "Physical Region Descriptor Table (PRDT)."
9. Initialize the CQPMP register with the device's PM number. If the port multiplier is not used, clear this field.
10. Poll the CQR[CQ*n*] to determine which command can be issued.
11. After CQ*n* is determined, write 1 to CQR[CQ*n*] to issue this command and start execution.
12. Poll the CCR[CC*n*] till CC*n* goes up, indicating that the command is completed.

The following example presents the structure of descriptors to issue the ReadDMA command:

- Build the command header in memory. See Table 16-35, below.

Table 16-35. Read DMA Command—Command Header

Word Number	Hexadecimal Value	Description
Word 0	CDA	Pointer to memory where command descriptor begins
Word 1	0x0002_0014	Two PRD tables contain the data, FIS length = 20 bytes

Table 16-35. Read DMA Command—Command Header

Word Number	Hexadecimal Value	Description
Word 2	0x0000_0200	Length of data associated with this command = 0x200
Word 3	0x0000_0000	Tag = 0

- Build command descriptor in memory. See Table 16-36, below.

Table 16-36. Read DMA Command—Command Descriptor

Word Number	Hexadecimal Value	Description
Word 0	0x00C8_8027	Command = Read DMA
Word 1	0x0000_0010	LBA = 24’h10
Word 2	0x0000_0000	LBA(exp) = 24’h0
Word 3	0x0000_0001	Sector Count = 1
Word 4	0x0000_0000	Reserved

- Build two PRD entries in memory. See Table 16-37.

Table 16-37. Read DMA Command—PRD Entries

Word Number	Hexadecimal Value	Description
Word 0	PRD1	Address of first portion of data.
Word 1	0x0000_0000	Reserved
Word 2	0x0000_0000	Reserved
Word 3	0x0000_0100	PRD1 contains 0x100 bytes of data
Word 4	PRD2	Address of second portion of data.
Word 5	0x0000_0000	Reserved
Word 6	0x0000_0000	Reserved
Word 7	0x0000_0100	PRD2 contains 0x100 bytes of data

- Fill the PRD1 and PRD2 with user-defined data; see Figure 16-32, below.

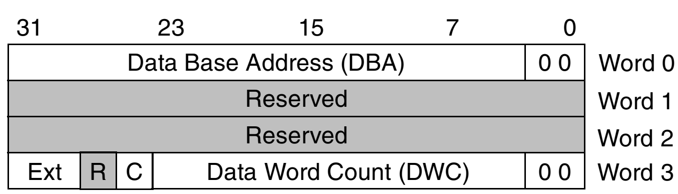


Figure 16-32. PRD Entry

Chapter 17
Security Engine (SEC) 3.0

This chapter describes the functionality of Freescale's integrated security engine (SEC 3.0). It addresses the following topics:

The SEC 3.0 is designed to off-load computationally intensive security functions, such as key generation and exchange, authentication, and bulk encryption from the processor core of the SoC. It is optimized to process all cryptographic algorithms associated with IPsec, IKE, SSL/TLS, iSCSI, SRTP, 802.11i, WiMAX, 3G, A5/3 for GSM and EDGE, and GEA3 for GPRS. The SEC 3.0 is derived from integrated security cores found in other members of the PowerQUICC II and PowerQUICC III families.

The security engine includes eight different execution units (EUs). Where data flows in and out of an EU, each has buffer FIFOs of at least 256 bytes. EU types and features include the following:

- AESU—Advanced Encryption Standard unit
  - Implements the Rijndael symmetric key cipher per U.S. National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 197.
  - Modes providing data confidentiality: ECB, CBC, CCM, Counter, GCM,CBC-RBP, OFB-128, and CFB-128.
  - Modes providing data authentication: CCM, GCM, CMAC (OMAC1), and XCBC-MAC.
  - 128-, 192-, or 256-bit key lengths (only 128-bit keys in XCBC-MAC)
  - ICV (integrity check vector) checking in CCM, GCM, CMAC (OMAC1), and XCBC-MAC mode
  - XOR operations on 2–6 sources for RAID
- AFEU—ARC4 execution unit
  - Implements a stream cipher compatible with the RC4 algorithm
  - 8- to 128-bit programmable key
- CRCU—Cyclical redundancy check unit

  - Implements CRC32C as required for iSCSI header and payload checksums, CRC32 as required for IEEE 802 packets, as well as for programmable CRC polynomials
  - ICV checking
- DEU—Data Encryption Standard execution unit
  - DES, 3DES
  - Two key (K1, K2, K1) or Three Key (K1, K2, K3)
  - ECB, CBC, CFB-64 and OFB-64 modes for both DES and 3DES
- KEU—Kasumi execution unit
  - Implements cipher and authentication modes f8 and f9 used in 3G, A5/3 for GSM and EDGE, and GEA3 for GPRS
  - 128-bit confidentiality key and 128-bit integrity key
  - ICV checking for f9
- MDEU—Message digest execution unit
  - Implements SHA with 160-, 224-, 256-, 384-, or 512-bit message digest (as specified by the FIPS 180-2 standard)
  - Implements MD5 with 128-bit message digest (as specified by RFC 1321)
  - Implements HMAC computation with either message digest algorithm (as specified in RFC 2104 and FIPS-198)
  - Implements SSL MAC computation
  - ICV checking
- PKEU—Public key execution unit that supports the following:
  - RSA and Diffie-Hellman
    - Programmable field size up to 4096 bits
  - Elliptic curve cryptography
    - F_2m and F_p modes
    - Programmable field size up to 1023 bits
  - Run time equalization to protect against timing and power attacks
- RNGU—Random number generator unit
  - True Random Number Generator (TRNG)

In addition to the execution units, SEC 3.0 also includes:

- A context switching polychannel, permitting operation of up to four virtual channels, where each channel:
  - Supports a queue of commands (descriptor pointers) to be executed
  - Provides dynamic arbitration for needed crypto-execution units
  - Manages up to two execution units (one ciphering and one hashing), and configures for any required data transfers from one to another
  - Performs flow-control management of buffer FIFOs on the inputs and outputs of execution units

— Supports scatter/gather of input and output data (where the term data is used loosely, and includes keys, context, ICV values, etc.), enabling concatenation of multiple segments of memory when reading or writing data
— Masters data bursts on 32-byte boundaries to optimize bus throughput

- Master and slave interfaces, with DMA capability
  — 32- or 36-bit address/64-bit data
  — Master interface allows pipelined requests
  — DMA data blocks can start and end on any byte boundary

17.1 SEC Architecture Overview

The SEC can act as a master on the internal system bus, allowing it to offload the data movement bottleneck normally associated with slave-only cores. A host processor accesses the SEC through its device drivers using system memory for data storage. The SEC resides in the peripheral memory map of the processor. When an application requires cryptographic functions, it creates descriptors for the SEC which define the functions to be performed and the locations of the data (descriptors are introduced in Section 17.1.1, "Descriptor Overview", and discussed in detail in Section 17.3, "Descriptors"). With a single 64-bit write, the host processor can enqueue a descriptor pointer in the SEC. The SEC's bus-mastering capability then enables it to execute the entire cryptographic task, performing reads and writes on system memory as needed.

A block diagram of the SEC internal architecture is shown in Figure 17-1.

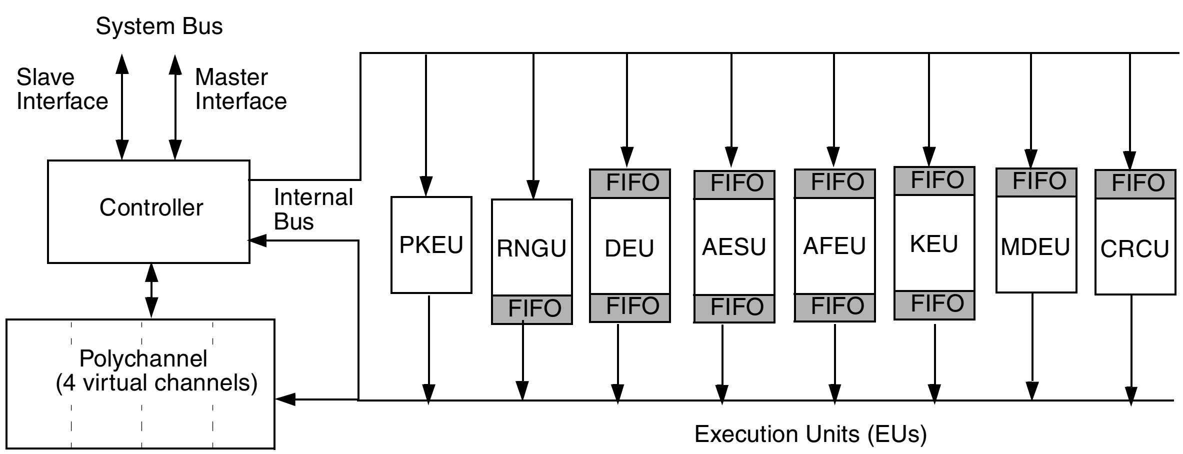


Figure 17-1. SEC Functional Modules

The SEC interfaces with the system buses through the controller (the controller is introduced in Section 17.1.3, "Controller Overview", and discussed in detail in Section 17.5, "Controller"). The slave interface permits an external device to perform 32- or 64-bit writes on any register or FIFO inside the SEC Some locations permit byte writes. Reads may be of any length. Using the master interface, the controller can transfer blocks of 64-bit words between system memory and SEC FIFOs or registers.

A typical SEC operation begins when a host processor writes a descriptor pointer to the fetch FIFO in one of the four SEC virtual channels. This write operation uses the slave interface (where the host is master and SEC is the slave). Following the write, the channel directs the sequence of operations using the master interface (where SEC is master). The channel uses the descriptor pointer to read the descriptor, then decodes the first word of the descriptor to determine the operation to be performed and the crypto-execution unit(s) needed to perform it (the execution units are introduced in Section 17.1.4, "Execution Units (EUs) Overview," and discussed in detail in Section 17.7, "Execution Units"). If necessary, the channel waits for the needed crypto-execution unit(s) to be free. Next, the channel requests the controller to transfer keys, context, and data from memory locations specified in the descriptor be sent to the appropriate execution units. The controller satisfies the requests through its master interface. Data is fed into the execution units through their registers and input FIFOs. The execution units read from their input FIFOs and write processed data to their output FIFOs and registers. The channel requests the controller to write data from the output FIFOs and registers back to system memory.

The channel can signal to the host that it is done with a descriptor by interrupt or by a writeback of the descriptor header into host memory. For more about this signaling, see Section 17.1.2, "Polychannel Overview."

Upon completion of a descriptor, the channel checks the next entry in its fetch FIFO, and (if non-empty) requests a read of the next descriptor.

For most packets, the entire payload is too long to fit in an execution unit's input or output FIFO, so the channel uses a flow control scheme for reading and writing data. The channel directs the controller to read bursts of input as necessary to keep refilling the input FIFO, until the entire payload has been fetched. Similarly, the channel directs the controller to write bursts of output whenever enough accumulates in the execution unit's output FIFO.

The polychannel can process up to four descriptors concurrently by implementing the four virtual channels (the polychannel is introduced in Section 17.1.2, "Polychannel Overview," and discussed in detail in Section 17.4, "Polychannel"). Channels arbitrate for use of execution units, and wait if the needed execution unit is currently reserved by another channel. Each channel has its own FIFO of descriptor pointers to fetch and execute, and its own internal storage. The four channels, however, time-share a single control and datapath unit, and hence they are referred to as virtual channels. A programmable priority scheme allows for round-robin or weighted priorities among these channels.

17.1.1 Descriptor Overview

All of the SEC's cryptographic functions are accessible through descriptors. This design facilitates easy use and integration with existing systems and software.

A descriptor specifies cryptographic functions to be performed, and contains reference address pointers to all necessary input data and to the locations where output data is to be written. Some descriptor types perform multiple functions to facilitate particular protocols. A sample descriptor is diagrammed in Table 17-1.Each descriptor contains eight dwords (64 bits each), consisting of the following:

- One dword of header—The header describes the required services and encodes information that indicates which EUs to use and which modes to set. It also indicates whether notification should be sent to the host when the descriptor operation is complete.

- Seven dwords containing pointers and lengths used to locate input or output data. Each pointer can either point directly to the data, or can point to a link table that lists a set of data segments to be concatenated.

Table 17-1. Example Descriptor

Field Name	Value	Description
Header	0x2053_1E08_0000_0000	Example header for IPsec ESP outbound using DES and MD-5
Length0 Extent0 Pointer0	16 0 (32 or 36-bit pointer)	Number of bytes in authenticate key Unused Pointer to authentication key
Length1 Extent1 Pointer1	16 0 (32 or 36-bit pointer)	Number of bytes in authentication-only data Unused Pointer to authentication-only data
Length2 Extent2 Pointer2	8 0 (32 or 36-bit pointer)	Length of input context (IV) Unused Pointer to input context
Length3 Extent3 Pointer3	8 0 (32 or 36-bit pointer)	Number of bytes in cipher key Unused Pointer to cipher key
Length4 Extent4 Pointer4	1500 0 (32 or 36-bit pointer)	Number of bytes of data to be ciphered Unused Pointer to input data to perform ciphering upon
Length5 Extent5 Pointer5	1500 12 (32 or 36-bit pointer)	Number of bytes of data after ciphering Number of bytes in authentication result (ICV) Pointer to location where cipher output is to be written, followed by ICV
Length6 Extent6 Pointer6	8 0 (32 or 36-bit pointer)	Length of output context (IV) Unused Pointer to location where altered context is to be written

For more information about descriptors, refer to Section 17.3, “Descriptors.”

17.1.2 Polychannel Overview

The polychannel block implements four channels for processing descriptors. Each channel contains the following addressable structures:

- A fetch FIFO, which holds a queue of pointers to descriptors waiting to be processed
- A configuration register, which allows the user a number of options for SEC event signaling
- A status register containing an indication of the last unfulfilled bus request
- A descriptor buffer memory used to store the active descriptor and other temporary data.

Whenever a channel is idle and its fetch FIFO is non-empty, the channel reads the next descriptor pointer from the fetch FIFO. Using this pointer, the channel fetches the descriptor and places it in its descriptor buffer. The channel’s processing of descriptors is described in more detail in Section 17.4.1.1, “Channel Descriptor Processing”

A channel can signal to the host that it is done with a descriptor by interrupt and/or by a writeback of the descriptor header into host memory. In the case of interrupt, there is an option to signal after every descriptor, or only after selected descriptors. In the case of writeback, the value written back is identical to the header that was read, with the exception that a DONE byte is set to 0xFF. The channels' done signaling is described in more detail in Section 17.4.1.3, "Channel Host Notification".

An EU operation can include generating an ICV and then comparing it against a received ICV. The result of the ICV checking can be signalled to the host either by interrupt or by a writeback of the descriptor header. If both are enabled, note that the occurrence of an error interrupt prevents the writeback from occurring. In the case of writeback, the user can opt to do it at end of every descriptor, or only at the end of descriptors that call for ICV checking.

In case of an error condition in a channel or its reserved EUs, the channel issues an interrupt to the host. The channel can be configured to either abort the current descriptor and proceed to the next one, or halt and wait for host intervention.

For more about configuring signaling see Section 17.4.4.1, "Channel Configuration Register (CCR)" and for detail on the writeback fields see Section 17.3.4, "Link Table Format."

Many security protocols involve both encryption and hashing of packet payloads. To accomplish this without requiring two passes through the data, channels can configure data flows through two EUs. In such cases, one EU is designated the "primary EU", and the other as the "secondary EU". The primary EU receives its data from memory through the controller, and the secondary EU receives its data by "snooping" the SEC buses.

There are two types of snooping:

- Input data can be fed to the primary EU and the same input data snooped by the secondary EU. This is called "in-snooping".
- Output data from the primary EU can be snooped by the secondary EU. This is called "out-snooping".

In the SEC, only MDEU and CRCU are used as secondary EUs.

For more information on the polychannel block, refer to Section 17.4, "Polychannel."

17.1.3 Controller Overview

The controller manages the master and slave interfaces to the system bus and the internal buses that connect all the various modules. It receives service requests from the host (through the slave interface) and from the channels, and schedules the required data transfers. The system bus interface and access to system memory are critical factors in performance, and the 64-bit master and slave interfaces of the SEC controller enable it to achieve performance unattainable on secondary buses.

The controller enables two modes of operation for the execution units: channel-controlled access and host-controlled access:

- In channel-controlled access (the SEC's normal operating mode), all interactions with EUs are directed by a channel executing a descriptor. The host is involved only in initially supplying the descriptor pointer and in handling results once descriptor processing is complete.

- In host-controlled access (intended primarily for debug purposes), the host moves data in and out of execution units directly through memory-mapped EU registers. No descriptor is involved.

For more information about the controller (including more details about channel-controlled and host-controlled access), refer to Section 17.5, “Controller.”

17.1.4 Execution Units (EUs) Overview

“Execution unit” (EU) is the generic term for the functional blocks that perform cryptographic computations. The EUs are compatible with many protocols, and can work together to perform high-level cryptographic tasks. The SEC’s execution units are as follows:

- PKEU for computing asymmetric key operations, including modular exponentiation (and other modular arithmetic functions) or ECC point arithmetic
- DEU for performing block cipher, symmetric key cryptography using DES and 3DES
- AESU for performing the Advanced Encryption Standard algorithm in various modes
- AFEU for performing RC-4 compatible stream cipher symmetric key cryptography
- MDEU for performing security hashing using MD-5, SHA-1, SHA-224, SHA-256, SHA-384 or SHA-512
- KEU for performing 3GPP confidentiality (f8) and integrity (f9) algorithms.
- CRCU for generating cyclical redundancy check values
- RNGU for random number generation

The following sections give an overview of the EUs. Operational details of each EU are given in Section 17.7, “Execution Units.”

17.1.4.1 Public Key Execution Unit (PKEU)

The PKEU is capable of performing many advanced mathematical functions to support both RSA and ECC public key cryptographic algorithms. ECC is supported in both F_2m (polynomial field) and F_p (prime field) modes.

To assist the host in performing its desired cryptographic functions, the PKEU supports functions with various levels of complexity. For example, at the highest level, the accelerator performs modular exponentiations to support RSA and performs point multiplies to support ECC. At a lower level, the PKEU can perform simple operations such as modular adds and multiplies. For more information about the unit’s operation, refer to Section 17.7.7, “Public Key Execution Units (PKEU).”

17.1.4.1.1 Elliptic Curve Operations

The PKEU has its own data and control units, including a general-purpose register file in the programmable-size arithmetic unit. The field or modulus size can be programmed to any value between 33 bits and 1024 bits in programmable increments of 8, with each programmable value i supporting all actual field sizes from $8i - 7$ to $8i$. The result is hardware supporting a wide range of cryptographic security. Larger field / modulus sizes result in greater security but lower performance.

Compared to RSA, elliptic curve cryptography provides greater security with smaller field sizes. For example, an elliptic curve field size of 160 is roughly equivalent to the security provided by 1024-bit RSA. A field size set to 224 roughly equates to 2048 bits of RSA security.

The PKEU contains routines implementing the atomic functions for elliptic curve processing, including point arithmetic and finite field arithmetic. The point operations (multiplication, addition and doubling) all involve one or more finite field operations which are addition, multiplication, inverse, and squaring. Point add and double each use all four finite field operations. Similarly, point multiplication uses all elliptic curve point operations as well as the finite field operations. All these functions are supported both in prime fields and polynomial fields.

17.1.4.1.2 Modular Exponentiation Operations

The PKEU is also capable of performing integer modulo arithmetic. This arithmetic is an integral part of the RSA public key algorithm; however, it can also play a role in the generation of ECC digital signatures (including ECDSA) and Diffie-Hellman key exchanges.

Modular arithmetic functions supported by the SEC's PKEU include the following (refer to Table 17-68 for a complete list):

- R^2 mod N
- $(A \times B) R^{-1}$ mod N
- $(A \times B) R^{-2}$ mod N
- (A + B) mod N
- (A – B) mod N

In the preceding list, the following notation is used:

- N is the modulus
- A and B are input parameters
- R is $2^{Sz'(N)}$, where Sz'(N) is the bit length of N rounded up to the nearest multiple of 32 (Note: R is referred to as "E" in public key descriptors)

The PKEU can perform modular arithmetic on operands up to 4096 bits in length. The modulus must be larger than or equal to 33 bits (5 bytes), or an error is returned. This is not seen as a limitation since no useful cryptographic applications exist for smaller moduli. The PKEU uses the Montgomery modular multiplication algorithm to perform core functions. The addition and subtraction functions help support known methods of the Chinese Remainder Theorem (CRT) for efficient implementation of the RSA algorithm.

17.1.4.2 Data Encryption Standard Execution Unit (DEU)

The DES Execution Unit (DEU) performs bulk data encryption/decryption, in compliance with the Data Encryption Standard algorithm (NIST FIPS 46-3). The DEU can also compute 3DES, an extension of the DES algorithm in which each 64-bit input block is processed three times. The SEC supports 2-key (K1=K3) or 3-key 3DES.

The DEU operates by permuting 64-bit data blocks with a shared 56-bit key and an initialization vector (IV). The SEC supports four modes of operation:

- Electronic Code Book (ECB)
- Cipher Clock Chaining (CBC)
- 64-bit Cipher Feedback Mode (CFB-64)
- 64-bit Output Feedback Mode (OFB-64).

For more information about the unit's operation, refer to Section 17.7.4, "Data Encryption Standard Execution Unit (DEU)."

17.1.4.3 Advanced Encryption Standard Execution Unit (AESU)

The AESU is used to accelerate bulk data encryption/decryption in compliance with the Advanced Encryption Standard algorithm Rijndael specified by NIST standard FIPS-197. The AESU executes on 128 bit blocks with a choice of key sizes: 128, 192, or 256 bits.

AES is a symmetric key algorithm, meaning the sender and receiver use the same key for encryption and decryption. The session key and IV are supplied to the AESU module prior to encryption. The processor supplies data to the module that is processed as 128 bit input.

AESU implements the following confidentiality modes from NIST Recommendation 800-38A:

- Electronic Codebook mode (ECB)
- Cipher Block Chaining mode (CBC)
- Output Feedback mode (OFB)
- 128-bit Cipher Feedback mode (CFB-128)
- Counter mode (CTR)

AESU also implements other NIST recommended modes providing authentication (two of which also provide confidentiality):

- Counter with CBC-MAC (CCM) per NIST recommendation 800-38C
- Galois Counter Mode (GCM) per NIST draft recommendation 800-38D
- Cipher-based MAC (CMAC) per NIST recommendation 800-38B.

  Note that CMAC is identical to OMAC1.

AESU modes also implement the following modes not sanctioned by NIST:

- LRW as specified by IEEE P1619 Draft 6
- CBC-RBP
- XCBC-MAC as specified by IETF RFC-3566

In all modes supporting authentication, the AESU hashes data to produce an integrity check vector (ICV). If a reference ICV is supplied to the AESU, it can do a bitwise check of the reference ICV against the one computed by the AESU.

For more information about the unit's operation, refer to Section 17.7.1, "Advanced Encryption Standard Execution Unit (AESU)."

17.1.4.4 Arc Four Execution Unit (AFEU)

The AFEU accelerates a bulk encryption algorithm compatible with the RC4 stream cipher from RSA Security, Inc. The algorithm is byte-oriented, meaning the data to be ciphered can be any number of bytes. The AFEU supports key lengths from 8 to 128 bits (in byte increments), providing a wide range of security strengths.

For more information, refer to Section 17.7.2, “ARC4 Execution Unit (AFEU).”

17.1.4.5 Message Digest Execution Unit (MDEU)

The MDEU computes a single message digest (or hash or integrity check) value for all the data presented on the input bus. The output size is determined by the specific algorithm, and is typically much smaller than the input size.

The MDEU is designed to support the following hashing algorithms:

- MD5 generates a 128-bit hash, and is specified in RFC 1321.
- SHA-1 is a 160-bit hash function, specified by the NIST FIPS 180-1 standard.
- SHA-224, SHA-256, SHA-384, and SHA-512 are 224-, 256-, 384-, and 512-bit hash functions respectively, specified by the NIST FIPS 180-2 standard.
- The MDEU also supports HMAC computations, as specified by the NIST FIPS-198 standard.

If a digest is supplied to the MDEU, it can do a bitwise check of this supplied digest against the one computed by the MDEU (ICV checking).

For more information about the unit’s operation, refer to Section 17.7.6, “Message Digest Execution Unit (MDEU).”

17.1.4.6 Kasumi Execution Unit (KEU)

The KEU (Kasumi Execution Unit) is a functional block capable of encrypting/decrypting and/or performing integrity checks on 64-bit blocks of data using a 128-bit key. The KEU is designed support the following cryptographic algorithms:

- f8 and f9, as defined in the ETSI/SAGE Specification Document 1 for the 3GPP standard
- A5/3 for GSM/EDGE
- GEA3 for GPRS

With the exception of f9, which is an authentication algorithm, KEU implements confidentiality algorithms. For f9, if the KEU is supplied with a MAC value, it is capable of performing a bitwise check of this original MAC against a f9 MAC generated by the KEU (ICV checking).

For more information about the unit’s operation, refer to Section 17.7.5, “Kasumi Execution Unit (KEU).”

17.1.4.7 Cyclical Redundancy Check Unit (CRCU)

The CRCU computes a single 32-bit cyclic redundancy code (checksum) from all data presented on the input bus.

The CRC algorithm treats a message stream of bits as coefficients of a massive polynomial and computes the remainder of the modulo two division by an order 32 divisor polynomial. The divisor polynomial is specific to the protocol and chosen to conform to certain mathematical properties to ensure that single bit errors can be detected. Cyclic redundancy codes are used to ensure data integrity over potentially unreliable channels. There are two major CRC protocol algorithms: CRC32 and CRC32C. The IEEE 802 standard defines the CRC32 algorithm, while iSCSI defines the CRC32C algorithm. Both protocols bit swap, byte swap, and then complement the calculated remainder to generate the checksum. The CRCU is designed to support the following check algorithms:

- CRC32 algorithm specified in the IEEE 802.1 standard.
- CRC32C algorithm specified in RFC3385.
- A programmable polynomial mode with no remainder bit mangling is also supported, which can be used to implement proprietary protocols.

The CRCU can perform ICV checking by computing a raw CRC across a message and previously-calculated CRC. Integrity is verified if the result matches the polynomial specific residue.

For more information about the unit's operation, refer to Section 17.7.3, "Cyclical Redundancy Check Unit (CRCU)."

17.1.4.8 Random Number Generator Unit (RNGU)

The RNGU is a functional block that generates 64-bit random numbers and stores them in an output FIFO.

Because many cryptographic algorithms use random numbers as a source for generating a secret value (a nonce), it is desirable to have a private RNG for use by the SEC. The anonymity of each random number must be maintained, as well as the unpredictability of the next random number. The FIPS-140 'common criteria'-compliant private RNG allows the system to develop random challenges or random secret keys. The secret key can thus remain hidden from even the high-level application code, providing an added measure of physical security.

For more information about the unit's operation, refer to Section 17.7.8, "Random Number Generator Unit (RNGU)."

17.2 Configuration of Internal Memory Space

Table 17-2 gives the base address map, and shows the blocks of addresses assigned to each SEC sub-block. All address gaps in Table 17-2 are reserved for future use. The 18-bit SEC address bus value is shown. These address values are offsets from the SoC's base address register (consult the SoC documentation for specific register name).

Table 17-2. SEC Address Map

Byte Address Offset (AD 17–0)	Module	Description	Type	Reference
0x3_1000–0x3_10FF	Controller	Arbiter/controller control register space	Controller	17.5/17-46
0x3_1100–0x3_11FF	Channel_1	Channel 1	Channels	17.4/17-32 Also see RCA bits in Table 17-21
0x3_1200–0x3_12FF	Channel_2	Channel 2		
0x3_1300–0x3_13FF	Channel_3	Channel 3		
0x3_1400–0x3_14FF	Channel_4	Channel 4		
0x3_1500–0x3_16FF	PolyChn	PolyChannel		17.4.3/17-35
0x3_1BF8	Controller	IP block revision register	Read only	17.5.4.5/17-54
0x3_2000–0x3_2FFF	DEU	DES/3DES execution unit	Crypto EU	17.7.4/17-108
0x3_4000–0x3_4FFF	AESU	AES execution unit		17.7.1/17-57
0x3_6000–0x3_6FFF	MDEU	Message digest execution unit		17.7.6/17-131
0x3_8000–0x3_8FFF	AFEU	Arc Four execution unit		17.7.2/17-88
0x3_A000–0x3_AFFF	RNGU	Random number generator unit		17.7.8/17-154
0x3_C000–0x3_CFFF	PKEU	Public key execution unit		17.7.7/17-145
0x3_E000–0x3_EFFF	KEU	Kasumi execution unit		17.7.5/17-117
0x3_F000–0x3_FFFF	CRCU	Cyclical Redundancy Check Unit		17.7.3/17-98

Table 17-3 shows the detailed system address map showing all functional registers.

All SEC registers are allocated 8 bytes (one dword), and the addresses listed in the table are all at 8-byte boundaries (addresses end in 0 or 8). It is possible, however, to access the registers by 4-byte words, or in some cases by byte. The "Write by" column in Table 17-3 distinguishes these cases. The column entries are interpreted as follows:

- Byte: This register can be written by byte (using any address), by word (using an address ending in 0, 4, 8, or C), or by dword (using an address ending in 0 or 8).
- Word: This register can be written by dword (using an address ending in 0 or 8), or by word (using an address ending in 0, 4, 8, or C), but not by byte.

Reads can always be done by byte, word, or dword.

Table 17-3. SEC Address Map

Byte Address Offset (AD 17–0)	Module	Register	Access	Write by	Reference
0x3_1008	Controller	Interrupt enable	R/W	byte[1]	17.5.4.2/1717-50
0x3_1010		Interrupt status	R	—	17.5.4.2/1717-53
0x3_1018		Interrupt clear	R/W	byte	17.5.4.3/1717-54
0x3_1020		Identification	R	—	17.5.4.4/1717-54
0x3_1028		EU assignment status	R	—	17.5.4.1/1717-50
0x3_1030		Master control	R/W	byte	17.5.4.6/1717-55
0x3_1108	Channel_1	Configuration register	R/W	word	17.4.4.1/1717-37
0x3_1110		Pointer status	R/W	word	17.4.4.2/1717-41
0x3_1140		Current descriptor pointer	R	—	17.4.4.3/1717-43
0x3_1148		Fetch FIFO	W	word	17.4.4.4/1717-44
0x3_1180–0x3_11BF		Descriptor buffer	R	—	17.4.5.1/1717-45
0x3_11C0–0x3_11DF		Gather Link Table	R	—	17.4.5.2/1717-45
0x3_11E0–0x3_11FF		Scatter Link Table	R	—	17.4.5.2/1717-45
0x3_1208	Channel_2	Configuration register	R/W	word	17.4.4.1/1717-37
0x3_1210		Pointer status	R/W	word	17.4.4.2/1717-41
0x3_1240		Current descriptor pointer	R	—	17.4.4.3/1717-43
0x3_1248		Fetch FIFO	W	word	17.4.4.4/1717-44
0x3_1280–0x3_12BF		Descriptor buffer	R	—	17.4.5.1/1717-45
0x3_12C0–0x3_12DF		Gather Link Table	R	—	17.4.5.2/1717-45
0x3_12E0–0x3_12FF		Scatter Link Table	R	—	17.4.5.2/1717-45
0x3_1308	Channel_3	Configuration register	R/W	word	17.4.4.1/1717-37
0x3_1310		Pointer status	R/W	word	17.4.4.2/1717-41
0x3_1340		Current descriptor pointer	R	—	17.4.4.3/1717-43
0x3_1348		Fetch FIFO	W	word	17.4.4.4/1717-44
0x3_1380–0x3_13BF		Descriptor buffer	R	—	17.4.5.1/1717-45
0x3_13C0–0x3_13DF		Gather Link Table	R	—	17.4.5.2/1717-45
0x3_13E0–0x3_13FF		Scatter Link Table	R	—	17.4.5.2/1717-45

Table 17-3. SEC Address Map (continued)

Byte Address Offset (AD 17–0)	Module	Register	Access	Write by	Reference
0x3_1408	Channel_4	Configuration register	R/W	word	17.4.4.1/1717-37
0x3_1410		Pointer status	R/W	word	17.4.4.2/1717-41
0x3_1440		Current descriptor pointer	R	—	17.4.4.3/1717-43
0x3_1448		Fetch FIFO	W	word	17.4.4.4/1717-44
0x3_1480–0x3_14BF		Descriptor buffer	R	—	17.4.5.1/1717-45
0x3_14C0–0x3_14DF		Gather Link Table	R	—	17.4.5.2/1717-45
0x3_14E0–0x3_14FF		Scatter Link Table	R	—	17.4.5.2/1717-45
0x3_1500	Poly-Channel	Fetch FIFO Enqueue Count	R/W	word	17.4.3.1/1717-35
0x3_1508		Descriptor Finished Count	R/W	word	17.4.3.1/1717-36
0x3_1510		Data Bytes In Count	R/W	word	17.4.3.1/1717-36
0x3_1518		Data Bytes Out Count	R/W	word	17.4.3.1/1717-37
0x3_1BF8	Controller	IP block revision	R	—	17.5.4.5/1717-54
0x3_2000	DEU	Mode register	R/W	word	17.7.4.1/1717-108
0x3_2008		Key size register	R/W	word	17.7.4.2/1717-109
0x3_2010		Data size register	R/W	word	17.7.4.3/1717-110
0x3_2018		Reset control register	R/W	word	17.7.4.4/1717-110
0x3_2028		Status register	R	—	17.7.4.5/1717-111
0x3_2030		Interrupt status register	R/W	word	17.7.4.6/1717-112
0x3_2038		Interrupt mask register	R/W	word	17.7.4.7/1717-114
0x3_2050		EU-Go	W	word	17.7.4.8/1717-116
0x3_2100		IV register	R/W	word	17.7.4.9/1717-116
0x3_2400		Key 1 register	W	byte	17.7.4.10/1717-116
0x3_2408		Key 2 register	W	byte	17.7.4.10/1717-116
0x3_2410		Key 3 register	W	byte	17.7.4.10/1717-116
0x3_2800–0x3_2FFF		Input FIFO / Output FIFO	R/W[2]	byte	17.7.4.11/1717-116

Table 17-3. SEC Address Map (continued)

Byte Address Offset (AD 17–0)	Module	Register	Access	Write by	Reference
0x3_4000	AESU	Mode register	R/W	word	17.7.1.2/1717-58
0x3_4008		Key size register	R/W	word	17.7.1.3/1717-62
0x3_4010		Data size register	R/W	word	17.7.1.4/1717-62
0x3_4018		Reset control register	R/W	word	17.7.1.5/1717-63
0x3_4028		Status register	R	—	17.7.1.6/1717-63
0x3_4030		Interrupt status register	R/W	word	17.7.1.7/1717-64
0x3_4038		Interrupt mask register	R/W	word	17.7.1.8/1717-66
0x3_4040		ICV size register	R/W	word	17.7.1.9/1717-68
0x3_4050		End of message register	W	word	17.7.1.10/1717-68
0x3_4100–0x3_415F		Context	R/W	byte	17.7.1.11/1717-69
0x3_4400–0x3_441F		Key registers	R/W	byte	17.7.1.12/1717-87
0x3_4800–0x3_4FFF		Input FIFO / Output FIFO	R/W [1]	byte	17.7.1.12/1717-88
0x3_6000	MDEU	Mode register	R/W	word	17.7.6.2/1717-132
0x3_6008		Key size register	R/W	word	17.7.6.4/1717-136
0x3_6010		Data size register	R/W	word	17.7.6.5/1717-136
0x3_6018		Reset control register	R/W	word	17.7.6.6/1717-137
0x3_6028		Status register	R	—	17.7.6.7/1717-137
0x3_6030		Interrupt status register	R/W	word	17.7.6.8/1717-139
0x3_6038		Interrupt mask register	R/W	word	17.7.6.9/1717-140
0x3_6040		ICV size register	W	word	17.7.6.10/1717-141
0x3_6050		End of message reigster	W	word	17.7.6.11/1717-142
0x3_6100–0x3_6147		Context registers	R/W	byte	17.7.6.12/1717-142
0x3_6400–0x3_647F		Key registers	W	byte	17.7.6.13/1717-145
0x3_6800–0x3_6FFF		Input FIFO	W [1]	byte	17.7.6.14/1717-145

Table 17-3. SEC Address Map (continued)

Byte Address Offset (AD 17–0)	Module	Register	Access	Write by	Reference
0x3_8000	AFEU	Mode register	R/W	word	17.7.2.1/1717-89
0x3_8008		Key size register	R/W	word	17.7.2.2/1717-89
0x3_8010		Data size register	R/W	word	17.7.2.3/1717-90
0x3_8018		Reset control register	R/W	word	17.7.2.4/1717-91
0x3_8028		Status register	R	—	17.7.2.5/1717-91
0x3_8030		Interrupt status register	R/W	word	17.7.2.6/1717-92
0x3_8038		Interrupt mask register	R/W	word	17.7.2.7/1717-94
0x3_8050		End of message register	W	word	17.7.2.8/1717-96
0x3_8100–0x3_81FF		Context memory	R/W	byte	17.7.2.9/1717-96
0x3_8200		Context memory pointers	R/W	byte	17.7.2.9/1717-96
0x3_8400–0x3_840F		Key registers	W	byte	17.7.2.10/1717-97
0x3_8800–0x3_8FFF (3_8E00)		Input FIFO / Output FIFO (special context address)	R/W [1]	byte	17.7.2.10/1717-97
0x3_A000	RNGU	Mode register	R/W	word	17.7.8.1/1717-155
0x3_A010		Data size register	R/W	word	17.7.8.2/1717-155
0x3_A018		Reset control register	R/W	word	17.7.8.3/1717-155
0x3_A028		Status register	R	—	17.7.8.4/1717-156
0x3_A030		Interrupt status register	R/W	word	17.7.8.5/1717-157
0x3_A038		Interrupt mask register	R/W	word	17.7.8.6/1717-158
0x3_A050		End of message register	W	word	17.7.8.7/1717-159
0x3_A400–0x3_A43F		Entropy registers	W	word	17.7.8.8/1717-160
0x3_A800–0x3_AFFF		Output FIFO	R [1]	—	17.7.8.8/1717-160

Table 17-3. SEC Address Map (continued)

Byte Address Offset (AD 17–0)	Module	Register	Access	Write by	Reference
0x3_C000	PKEU	Mode register	R/W	word	17.7.7.1/1717-146
0x3_C008		Key size register	R/W	word	17.7.7.2/1717-146
0x3_C010		Data size register	R/W	word	17.7.7.4/1717-148
0x3_C018		Reset control register	R/W	word	17.7.7.5/1717-148
0x3_C028		Status register	R	—	17.7.7.6/1717-149
0x3_C030		Interrupt status register	R/W	word	17.7.7.7/1717-150
0x3_C038		Interrupt mask register	R/W	word	17.7.7.8/1717-152
0x3_C040		ABSize	R/W	word	17.7.7.3/1717-147
0x3_C050		End of message register	W	word	17.7.7.9/1717-153
0x3_C200–0x3_C27F		Parameter memory A0	R/W	byte	17.7.7.10/1717-153
0x3_C280–0x3_C2FF		Parameter memory A1	R/W	byte	
0x3_C300–0x3_C37F		Parameter memory A2	R/W	byte	
0x3_C380–0x3_C3FF		Parameter memory A3	R/W	byte	
0x3_C400–0x3_C47F		Parameter memory B0	R/W	byte	
0x3_C480–0x3_C4FF		Parameter memory B1	R/W	byte	
0x3_C500–0x3_C57F		Parameter memory B2	R/W	byte	
0x3_C580–0x3_C5FF		Parameter memory B3	R/W	byte	
0x3_C800–0x3_C9FF		Parameter memory N	R/W	byte	
0x3_CA00–0x3_CBFF		Parameter memory E	W	byte	

Table 17-3. SEC Address Map (continued)

Byte Address Offset (AD 17–0)	Module	Register	Access	Write by	Reference
0x3_E000	KEU	Mode register	R/W	word	17.7.5.1/1717-118
0x3_E008		Key size register	R/W	word	17.7.5.2/1717-119
0x3_E010		Data size register	R/W	word	17.7.5.3/1717-119
0x3_E018		Reset control register	R/W	word	17.7.5.4/1717-121
0x3_E028		Status register	R	—	17.7.5.5/1717-122
0x3_E030		Interrupt Status register	R/W	word	17.7.5.6/1717-123
0x3_E038		Interrupt Mask register	R/W	word	17.7.5.7/1717-125
0x3_E048		Data out register (f9 MAC)	R	—	17.7.5.8/1717-126
0x3_E050		End of message register	W	word	17.7.5.9/1717-127
0x3_E100		IV_1 register	R/W	byte	17.7.5.10/1717-127
0x3_E108		ICV_In register	R/W	byte	17.7.5.11/1717-128
0x3_E110		IV_2 register (FRESH)	R/W	byte	17.7.5.12/1717-128
0x3_E118		Context_1 register	R/W	byte	17.7.5.13/1717-129
0x3_E120		Context_2 register	R/W	byte	17.7.5.13/1717-129
0x3_E128		Context_3 register	R/W	byte	17.7.5.13/1717-129
0x3_E130		Context_4 register	R/W	byte	17.7.5.13/1717-129
0x3_E138		Context_5 register	R/W	byte	17.7.5.13/1717-129
0x3_E140		Context_6 register	R/W	byte	17.7.5.13/1717-129
0x3_E400		Key data register_1 (CK-high)	R/W	byte	17.7.5.14/1717-129
0x3_E408		Key data register_2 (CK-low)	R/W	byte	17.7.5.14/1717-129
0x3_E410		Key dataregister_3 (IK-high)	R/W	byte	17.7.5.15/1717-130
0x3_E418		Key data register_4 (IK-low)	R/W	byte	17.7.5.15/1717-130
0x3_E800–0x3_EFFF		Input FIFO / Output FIFO	R/W [1]	byte	17.7.5.16/1717-130

Table 17-3. SEC Address Map (continued)

Byte Address Offset (AD 17–0)	Module	Register	Access	Write by	Reference
0x3_F000	CRCU	Mode register	R/W	word	17.7.3.2/1717-98
0x3_F008		Key size register	R/W	word	17.7.3.3/1717-99
0x3_F010		Data size register	R/W	word	17.7.3.4/1717-100
0x3_F018		Reset control register	R/W	word	17.7.3.5/1717-100
0x3_F020		Control	R/W	word	17.7.3.6/1717-101
0x3_F028		Status register	R	—	17.7.3.7/1717-101
0x3_F030		Interrupt status register	R/W	word	17.7.3.8/1717-102
0x3_F038		Interrupt mask register	R/W	word	17.7.3.9/1717-104
0x3_F040		ICV size register	R/W	word	17.7.1.9/1717-68
0x3_F050		End of message register	W	word	17.7.3.11/1717-106
0x3_F108		Context register	R/W	byte	17.7.3.12/1717-106
0x3_F400		Key register	R/W	byte	17.7.3.13/1717-107
0x3_F800–0x3_FFFF		Input FIFO	W [1]	byte	17.7.3.14/1717-108

[1] Byte accessibility is controlled by internal logic, particularly at FIFOs, to prevent unintended overwrites of partial words during writes, and to prevent unintended duplicate reads of partial data during reads. In addition, these bytes must be presented on the correct byte lanes for the intended destination.

[2] For the EU FIFOs, write operations anywhere in the address range enqueue to the input FIFO, and read operations anywhere in the address range dequeue from the output FIFO. See the referenced section for more detailed information.

17.3 Descriptors

The host processor maintains a record of current secure sessions and the corresponding keys and contexts of those sessions. Once the host has determined that a security operation is required, it creates a “descriptor” containing all the information the SEC needs to perform the security operation. The host creates the descriptor in main memory, then writes a pointer to the descriptor into the fetch FIFO of one of the SEC channels. The channel uses this pointer to read the descriptor into its descriptor buffer. Once it obtains the descriptor, the SEC uses its bus mastering capability to obtain inputs and write results, thus offloading data movement and encryption operations from the host processor.

Descriptors are only used in channel-controlled accesses to SEC, and not in host-controlled accesses. For more information about host-controlled access, see Section 17.5.1.1, “Host-Controlled Access”.

17.3.1 Descriptor Structure

SEC descriptors are designed so that a single descriptor supports the cryptographic computation of a single packet. SEC descriptors have a fixed length of 64 bytes, that is, eight 64-bit words (referred to as dwords). A descriptor consists of one header dword and seven “pointer dwords”, as seen in Figure 17-2.

Figure 17-2. Descriptor Format

	0 15	16	17 23	24 27	28 31	32 63
Header Dword	Descriptor Control					Descriptor Feedback
Pointer Dword 0	Length0	J0	Extent0	—	Eptr0	Pointer0
Pointer Dword 1	Length1	J1	Extent1	—	Eptr1	Pointer1
Pointer Dword 2	Length2	J2	Extent2	—	Eptr2	Pointer2
Pointer Dword 3	Length3	J3	Extent3	—	Eptr3	Pointer3
Pointer Dword 4	Length4	J4	Extent4	—	Eptr4	Pointer4
Pointer Dword 5	Length5	J5	Extent5	—	Eptr5	Pointer5
Pointer Dword 6	Length6	J6	Extent6	—	Eptr6	Pointer6

As shown in Figure 17-2, the first and second halves of the header dword are denoted as descriptor control and descriptor feedback fields, respectively. The descriptor control field of the header dword specifies the security operation to be performed, the execution unit(s) needed, and the modes for each execution unit. The descriptor feedback field is written to by the security engine upon completion of descriptor processing, when the "channel done writeback" feature is enabled. Further details about the header dword may be found in Section 17.3.2, "Descriptor Format: Header Dword".

The pointer dwords, all of which have the same format, contain pointer and length information for locating input or output parcels (such as keys, context, or text data). The large number of pointers provided in the descriptor allows for multi-algorithm operations that require fetching of multiple keys, as well as fetch and return of contexts. Any pointer dword that is not needed may be assigned a length of zero. Further details about the pointer dwords may be found in Section 17.3.3, "Descriptor Format: Pointer Dwords".

SEC descriptors include scatter/gather capability, which means that each pointer in a descriptor can be either a direct pointer to a contiguous parcel of data, or a pointer to a "link table" which is a list of pointers and lengths used to assemble the parcel. When a link table is used to read input data, this is referred to as a "gather" operation; when used to write output data, it is referred to as a "scatter" operation. Further details about scatter/gather capability may be found in Section 17.3.4, "Link Table Format".

17.3.2 Descriptor Format: Header Dword

Descriptors are created by the host to guide the SEC through required cryptographic operations. The header dword provides the primary indication of the operations to be performed, the mode for each operation, and internal addressing used by the controller and channel for internal data movement. The fields that must be supplied to SEC are shown in the "Field" rows of Figure 17-3 and described in Table 17-4. The SEC device drivers allow the host to create proper headers for each cryptographic operation.

SEC processing of a descriptor sometimes includes writing the original header dword back to system memory with certain fields modified. The modified fields are shown in the "Writeback" rows of Figure 17-3 and described in Table 17-5.

	0–3	4–7	8–11	12–15	16–23	24–28	29	30	31
Field	OP_0			OP_1		DESC_TYPE	—	DIR	DN
	EU_SEL0	MODE0		EU_SEL1	MODE1				
Writeback	DONE		—						

	32–34	35–36	37–42	43–44	45–47	48–63
Field	—					
Writeback	—	ICCR0	—	ICCR1	—	ID_TAG

Figure 17-3. Header Dword

Table 17-4. Header Dword Bit Definitions

Bits	Name	Description
	OP_0:	
0–3	EU_SEL0	Primary execution unit select: See Section 17.3.2.1, "Selecting Execution Units—EU_SEL0 and EU_SEL1," for possible values.
4–11	MODE0	Primary mode: Mode data used to program the primary EU. The mode data is specific to the chosen EU. This field is passed directly to bits 56–63 of the mode register in the selected EU. Refer to the EU-specific mode register sections (Section 17.7.1.2, "AESU Mode Register," Section 17.7.2.1, "AFEU Mode Register," Section 17.7.3.2, "CRCU Mode Register," Section 17.7.4.1, "DEU Mode Register," Section 17.7.5.1, "KEU Mode Register (KEUMR)," Section 17.7.6.2, "MDEU Mode Register," Section 17.7.7.1, "PKEU Mode Register," and Section 17.7.8.1, "RNGU Mode Register") for further info. Any bits of any use in any mode register beyond bits 56–63 are under control of the channel and not the MODE0 field.
	OP_1:	
12–15	EU_SEL1	Secondary EU select: See Section 17.3.2.1, "Selecting Execution Units—EU_SEL0 and EU_SEL1," for possible values.
16–23	MODE1	Secondary mode: Mode data used to program the primary EU. The mode data is specific to the chosen EU. This field is passed directly to bits 56–63 of the mode register in the selected EU. Refer to the EU-specific mode register sections (sections Section 17.7.3.2, "CRCU Mode Register," and Section 17.7.6.2, "MDEU Mode Register") for further info.
24–28	DESC_TYPE	Descriptor Type: This, along with the DIR field, determines the sequence of actions to be performed by the channel and selected EUs using the blocks of data listed in the rest of the descriptor. The attributes determined include the direction of data flow for each data block, which EU (primary or secondary) is accessed, what snooping options are used, and address offsets for internal EU accesses. See Section 17.3.2.2, "Selecting Descriptor Type—DESC_TYPE," for possible values.
29	—	Reserved
30	DIR	Direction: direction of overall data flow: 0 Outbound 1 Inbound This, along with the DESC_TYPE field, helps determine the sequence of actions to be performed by the channel and selected EUs.

Table 17-4. Header Dword Bit Definitions (continued)

Bits	Name	Description
31	DN	Done notification: 0 No done notification. 1 Signal "done" to the host on completion of this descriptor. This enables done notification if the NT bit is set in the channel configuration register (see Table 17-11). The done notification can take the form of an interrupt, a writeback in the DONE field of this header dword (see Table 17-5), or both, depending upon the states of the CDIE (channel done interrupt enable) and CDWE (channel done writeback enable) bits in the channel configuration register.

Table 17-5. Header Dword Writeback Bit Definitions

Bits	Name	Description
0–7	DONE	When "channel done writeback" is enabled, then at the completion of descriptor processing this byte is written with the value 0xFF. "Channel done writeback" is enabled by programming the CDWE, NT, and CDIE fields in the channel configuration register (see Table 17-11).
8–34	—	Reserved
35–36	ICCR0	Integrity check comparison result from primary: These bits are supplied by the primary EU when descriptor processing is complete. 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved
37–42	—	Reserved
43–44	ICCR1	Integrity check comparison result from secondary: These bits are supplied by the secondary EU (if any) when descriptor processing is complete. 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved
45–47	—	Reserved
48–63	ID_TAG	Identification Tag. This value is copied from the ID_TAG field written by the host into the fetch FIFO (see Section 17.4.4.4, "Fetch FIFO Enqueue Register (FFER)").

17.3.2.1 Selecting Execution Units—EU_SEL0 and EU_SEL1

Table 17-6 shows the values for EU_SEL0 and EU_SEL1 in the descriptor header. The following rules govern the choices for these fields:

1. EU_SEL0 values of "No EU selected" or "Reserved" result in an "Unrecognized header" error condition during processing of the descriptor header.
2. The only valid choices for EU_SEL1 are "No EU selected", CRCU, or MDEU. Any other choice results in an "Unrecognized header" error condition.
3. If EU_SEL1 is CRCU or MDEU, then EU_SEL0 must be DEU, AESU, AFEU, or KEU. All other values of EU_SEL0 result in an "Unrecognized header" error condition.

Table 17-6. EU_SEL0 and EU_SEL1 Values

Value (binary)	Selected EU
0000	No EU selected
0001	AFEU
0010	DEU
0011	MDEU-A
1011	MDEU-B
0100	RNGU
0101	PKEU
0110	AESU
0111	KEU
1000	CRCU
others	Reserved
1111	Reserved for header writeback

The designators MDEU-A and MDEU-B both refer to the same physical MDEU. If MDEU-B is selected, then the channel configures MDEU to perform SHA-224, SHA-256, SHA-384, and SHA-512. If MDEU-A is selected, then the channel configures MDEU to perform SHA-160, SHA-224, SHA-256, or MD5. This configuration is achieved automatically; the channel sets bit 51 of the MDEU mode register as it inserts the MODE0 (or MODE1) value into the MDEU mode register. For further information see Section 17.7.6.2, "MDEU Mode Register."

17.3.2.2 Selecting Descriptor Type—DESC_TYPE

Table 17-7 shows the permissible values for the DESC_TYPE field in the descriptor header. Descriptor types from the SEC1.0, which have "0" in the last bit, are listed first, followed by new SEC 2.x/3.x types, which have "1" in the last bit.

Table 17-7. Descriptor Types

Value (binary)	Descriptor Type	Notes
0000_0	aesu_ctr_nonsnoop	AESU CTR nonsnooping
0001_0	common_nonsnoop	Common, nonsnooping, non-PKEU, non-AFEU
0010_0	hmac_snoop_no_afeu	Snooping, HMAC, non-AFEU
0011_0	—	Reserved
0100_0	—	Reserved
0101_0	common_nonsnoop_afeu	Common, nonsnooping, AFEU
0110_ 0	—	Reserved

Table 17-7. Descriptor Types (continued)

Value (binary)	Descriptor Type	Notes
0111_0	—	Reserved
1000_0	pkeu_mm	PKEU-Montgomery Multiplication
1001_0	—	Reserved
1010_0	—	Reserved
1011_0	—	Reserved
1100_0	hmac_snoop_aesu_ctr	AESU CTR hmac snooping [2]
1101_0	—	Reserved
1110_0	—	Reserved
1111_0	—	Reserved
0000_1	ipsec_esp	IPsec ESP mode encryption and hashing
0001_1	802.11i_aes_ccmp	CCMP encryption and hashing, suitable for 802.11i
0010_1	srtp	SRTP encryption and hashing
0011_1	pkeu_build	pkeu_build Elliptic Curve Cryptography
0100_1	pkeu_ptmul	pkeu_ptmul Elliptic Curve Cryptography
0101_1	pkeu_ptadd_dbl	pkeu_ptadd_dbl Elliptic Curve Cryptography
0110_1	—	Reserved
0111_1	—	Reserved
1000_1	tls_ssl_block	TLS/SSL generic block cipher
1001_1	tls_ssl_stream	TLS/SSL generic stream cipher
1010_1	raid_xor	XOR 2-6 sources together
1011_1	ipsec_aes_gcm	IPsec ESP mode using AES GCM encryption and hashing
1100_1	dbl_crc	Do two CRC operations
others	—	Reserved

[1] Type 0000_0 is for AES-CTR operations. Type 0001_0 also supports AES-CTR, however to use AES-CTR with 0001_0, the user must prepend zeros to the AES-Context before loading the AES Context Registers.

[2] Type 1100_0 is for AES-CTR operations with HMAC. Type 0010_0 also supports AES-CTR with HMAC, however to use AES-CTR with 0010_0, the user must prepend zeros to the AES-Context before loading the AES Context Registers.

For more about descriptor types and the data used for each type, see Section 17.3.5, “Descriptor Types.”

17.3.3 Descriptor Format: Pointer Dwords

The descriptor contains seven “pointer dwords” which define where in memory the SEC should access its input and output parcels. The pointer dwords are numbered 0 to 6 as shown in Figure 17-2. The channel determines how it uses each of the pointer dwords based on the descriptor type and direction fields in the header (see Table 17-4).

The pointer dword bit fields as shown in Figure 17-4, and are described in Table 17-8.

Access: Read/Write

	0–15	16	17–23	24–27	28–31
R/W	LENGTH	J	EXTENT	—	EPTR
Reset	All zeros				

	32–63
R/W	POINTER
Reset	All zeros

Figure 17-4. Pointer Dword

Table 17-8. Pointer Dword Field Definitions

Bits	Name	Description
0-15	LENGTH	Length: A number of bytes in the range 0 to 65535. The use of this field depends on the descriptor type and direction fields in the header dword. A value of zero may cause the channel to skip this dword.
16	J	Jump: Determines whether to “jump” to a link table whenever the POINTER field in this same dword is used. 0 The POINTER field points to data. 1 The POINTER field points to a link table, and scatter/gather is enabled.
17-23	EXTENT	Extent: A number of bytes in the range 0 to 127. The use of this field depends on the “Descriptor Type” and “Direction” fields in the header dword.
24-27	—	Reserved
28-31	EPTR	Extended Pointer: Concatenated as the top 4 bits of the pointer when EAE is high (see the EAE bit in Table 17-11).
32-63	POINTER	Pointer: A memory address.

On occasion, a descriptor field may not be applicable to the requested service. With seven pointer dwords, it is possible that not all these dwords are required to specify the input and output parameters (for instance, some operations do not require context.) Wherever a particular field is not used, it should be set to zero.

The channel proceeds linearly through the descriptor, fetching LENGTH data beginning at location POINTER. If the EAE (extend address enable) bit is set in the channel configuration register (see Table 17-11), then the four EPTR bits are concatenated with the POINTER field to form a 36-bit pointer address.

If all the data of LENGTH is found contiguously beginning at POINTER, then the Jump bit is not set. Otherwise, POINTER indicates the location of a link table (scatter-gather list). For more details, see Section 17.3.4, “Link Table Format”.

LENGTH and EXTENT fields normally specify the sizes of parcels: often (but not always) the size of the parcel located at the address contained in the matching POINTER field[1]. However, in some cases the POINTER field is zero, and the LENGTH and/or EXTENT fields simply specify values to be written to an EU. The specific use of these fields in each channel depends on the descriptor type and direction fields in the descriptor's header dword (see Table 17-4).

The RAID-XOR descriptor type does not support scatter/gather capability. However, scatter/gather is available for all pointer dwords for all other descriptor types (provided the J bit is set).

17.3.4 Link Table Format

Link tables implement scatter/gather capability. For "gather" operations, a link table specifies a list of "memory segments" that are to be concatenated in the process of assembling parcels. For "scatter" operations, a link table specifies a list of memory segments into which the output data should be written. Scatter or gather of a parcel may be specified by a single link table or by a chain of link tables that are linked together with pointers, as shown in Figure 17-6.

A link table may contain any number of dword entries. Link table entry format is shown in Figure 17-5, and the field definitions are given in Table 17-9.

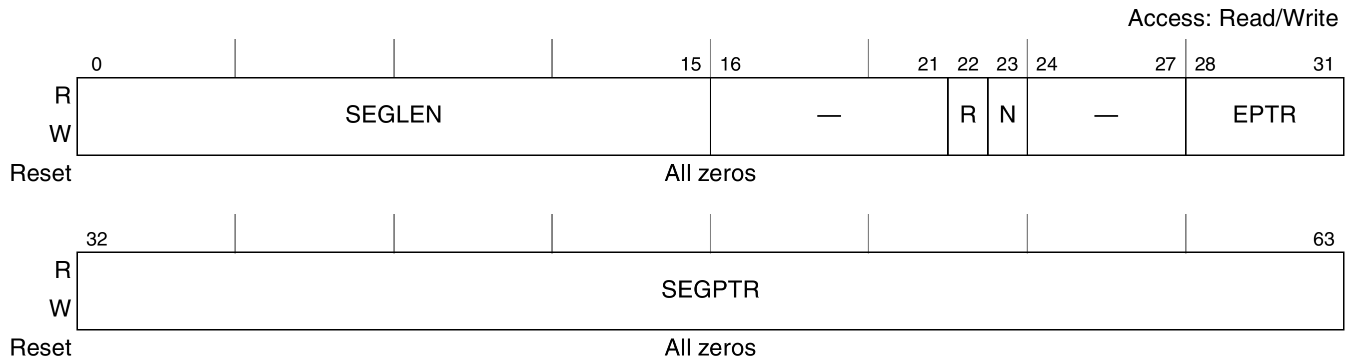


Figure 17-5. Link Table Entry

Table 17-9. Link Table Field Definitions

Bits	Name	Description
0-15	SEGLEN	Length: 0-15 When N=0, SEGLEN is in the range 1 to 65535, specifying the number of bytes in the memory segment pointed to by SEGPTR. A value of 0 causes the SGZL error bit to be set in the Channel Status (see Section 17.4.4.2, "Channel Status Register (CSR)"). When N=1, SEGLEN must be 0.
16-21	—	Reserved

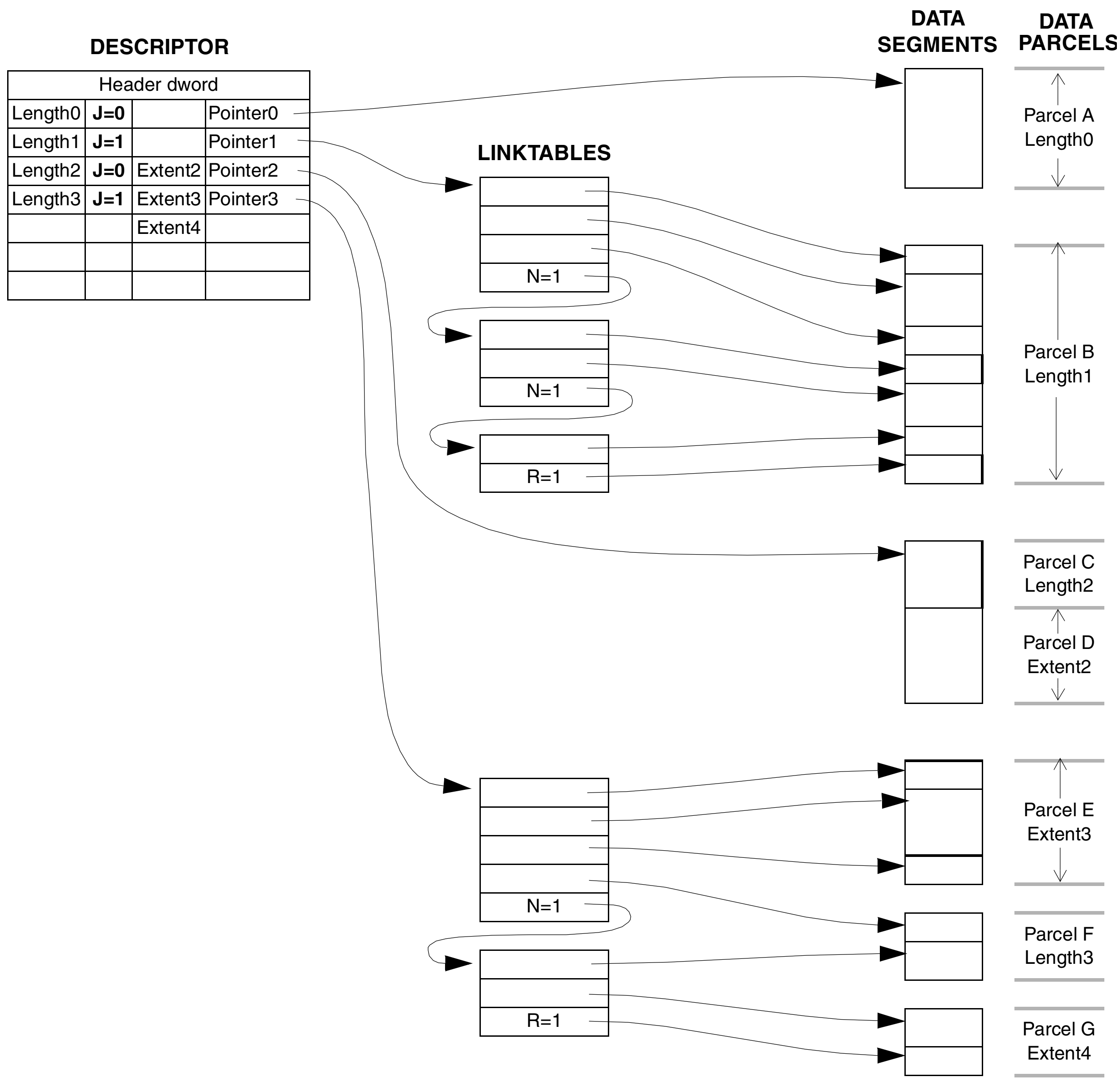
1. Sometimes an EXTENT field refers to data in a pointer which is not in the same dword. For example, with the CCMP descriptor type (see Table 17-10), the length of the CRC check field appears in Extent0, but the field that is Extent0 bytes in length is referred to either by Pointer4 or Pointer5, depending on the direction bit in the descriptor's header dword.

Table 17-9. Link Table Field Definitions (continued)

Bits	Name	Description
22	R	Return: When N=0: 0 No special action. 1 Indicates the last entry in the chain of link tables. If this entry does not specify the right number of bytes to complete the last parcel, a G-STATE or S-STATE error is set in the Channel Status Register (see Section 17.4.4.2, "Channel Status Register (CSR)"). When N=1, ignored.
23	N	Next: 0 No special action. 1 Indicates the last dword in the current link table. The SEGPTR field is the address of the next link table in the chain.
24-27	—	Reserved
28-31	EPTR	Extended Pointer: Concatenated as the top 4 bits of the segment pointer when EAE is high (see the EAE bit in Table 17-11).
32-63	SEGPTR	Segment pointer: A memory address.

There are two kinds of link table entries: "regular" entries or "next" entries (which have the N bit cleared or set, respectively). Each "regular" entry specifies a memory segment by means of a 36-bit starting address (SEGPTR) and a 16-bit length (SEGLEN). A "next" entry is used at the end of a link table to specify that the list of memory segments is continued in another link table. In a "next" entry, the N bit is set, the SEGPTR field gives the address of the next link table, and the SEGLEN field must be cleared. A chain of link tables may contain any number of link tables. Whether the list of memory segments is in a single link table or split into several link tables, the last entry in the last link table is a "regular" entry with the R (return) bit set. The R bit signifies the end of link table operations so that the channel returns to the descriptor for its next pointer (if any).

The construction and use of link tables is illustrated in Figure 17-6.



This figure illustrates various ways that a descriptor (table in upper left) may specify parcels:
The first pointer dword in the descriptor (following the header dword) specifies Parcel A using the simplest method—the parcel is specified directly through Pointer0 and Length0.
The next pointer dword uses a chain of link tables to specify Parcel B. Since J=1, Pointer1 is used as the address of a link table. The link table specifies several “regular” entries specifying data segments to be concatenated. The last word of the link table is a “next” entry indicating that the list continues in the next link table. The last entry in the last link table of the chain has the R bit set.
The last two cases illustrate how one pointer in a descriptor can be used to specify multiple parcels. Pointer2 and Length2 specify Parcel C, then Parcel D follows immediately afterwards, with length specified by Extent2. Pointer3 is used for three parcels (E, F, and G), this time using link tables.

Figure 17-6. Descriptors, Link Tables, and Parcels

As shown in Figure 17-6, in some cases a single parcel is accessed through a given POINTER, and the chain of link tables specifies only that parcel (this is the most common situation). In other cases, the descriptor POINTER is used multiple times to access a sequence of parcels, and the chain of link tables must supply data for the entire sequence.

17.3.4.1 Example of Link Table Operation

To further clarify the link table's operation, we explain in detail the case where the fourth pointer dword in the descriptor in Figure 17-6 is used to access parcels. We suppose that the descriptor type is such that Pointer3 is used to access successive parcels of size Extent3, Length3, and Extent4 respectively (refer to Table 17-10 for the significance of POINTER, EXTENT, and LENGTH fields in various descriptor types).

Since the J3 bit is set, Pointer3 is used as the address of a link table and not a data address. The channel begins by reading the first four dwords of the link table starting at Pointer3 into an internal "gather table buffer".

Using the first entry of the gather table buffer, the channel starts accessing the parcel by reading SEGLEN bytes beginning at SEGPTR. If the required parcel size (specified by 'Extent3' in the pointer dword) is greater than this first segment length, the channel moves on to the next entry of the gather table buffer, and reads SEGLEN bytes starting at SEGPTR. This process continues as long as there are more bytes to be read in the parcel. If all the link table entries in the channel's gather table buffer have been exhausted, then the channel reads the next four dwords of the link table into its gather table buffer. If a gather table buffer entry is encountered in which the N bit is set, the channel uses the SEGPTR field in that word to find the next link table in the chain.

Now assume that the channel accesses its next parcel using Pointer3 again, this time with length given by Length3. In this case the channel continues to the next line of the link table, and begins reading the memory segment specified there. As before, the channel concatenates memory segments from as many link table entries as necessary to obtain the required number of bytes (Length3).

Similarly, the next parcel is obtained by using Pointer3 yet again, this time with length given by Extent4.

Assume that for the current descriptor type, the Extent4 parcel is the last one to be accessed through Pointer3. Then the link table entry that supplies the last memory segment for Extent4 has the R bit set, signifying that this is the last entry in the chain of link tables.

NOTE

The link table or chain of link tables accessed through a descriptor pointer must specify enough memory segments to hold precisely *all the data that will be accessed through that pointer*. This means that the combined lengths of the parcels associated with that pointer (where each parcel length is specified by a particular LENGTH or EXTENT field in the descriptor) must equal the combined lengths of the link table memory segments (SEGLEN fields). Otherwise the channel sets the error state in the SGLM bit of the channel status register (see Section 17.4.4.2, "Channel Status Register (CSR)").

17.3.5 Descriptor Types

Table 17-10 shows in summary form how the pointer dwords are used with the various descriptor types. Detailed information about each descriptor type is given in the remainder of this chapter. Additional explanation of the use of certain descriptor types, can be found in the SEC 3.0 Descriptor Programmer’s Guide.

As in Table 17-7 above, older descriptor types which end in 0 are listed first, followed by newer types which end in 1.

Table 17-10. Descriptor Format Summary

Descriptor Type	field type	Pointer Dword0	Pointer Dword1	Pointer Dword2	Pointer Dword3	Pointer Dword4	Pointer Dword5	Pointer Dword6
0000_0 aesu_ctr_ nosnoop	Length	reserved	Cipher Context In	Cipher Key	Main Data In	Data Out	Cipher Context Out	reserved
	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
0001_0 common_ nosnoop for DES, KEU f8, RNGU, AES-CCM	Length	reserved	Context In	Key	Main Data In	Data Out	Context Out (incl. ICV out)	reserved
	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
0001_0 common_ nosnoop for MDEU	Length	reserved	Context In	Key	Main Data In	ICV In	Context Out	reserved
	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
0001_0 common_ nosnoop for AES-XCBC, AES-CMAC	Length	reserved	Context In	Key	Main Data In	reserved	Context Out	ICV Out
	Extent	reserved	reserved	reserved	reserved	ICV In	reserved	reserved
0001_0 common_ nosnoop for KEU f9	Length	reserved	Context In	Key	Main Data In	reserved	Context Out	ICV Out
	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
0001_0 common_ nonsnoop for CRCU	Length	reserved	Context In	Key	Main Data In	reserved	Context Out	reserved
	Extent	reserved	reserved	reserved	reserved	ICV In	reserved	reserved
0010_0 hmac_snoop _no_afeu	Length	Hash Key	Hash-only Header	Cipher Key	Cipher Context In	Main Data In	Data Out	ICV Out
	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
0101_0 common_ nosnoop_ afeu	Length	reserved	Context In (via In FIFO)	Cipher Key	Main Data In	Data Out	Context Out (via Out FIFO)	reserved
	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved

Table 17-10. Descriptor Format Summary (continued)

Descriptor Type	field type	Pointer Dword0	Pointer Dword1	Pointer Dword2	Pointer Dword3	Pointer Dword4	Pointer Dword5	Pointer Dword6
1000_0	Length	“N” In	“B” In	“A” In	“E” In	“B” Out	reserved	reserved
pkeu_mm	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
1100_0 hmac_snoop_ aesu_ctr	Length	Hash Key	Hash-only Header	AES Key	AES Context In	Main Data In	Data Out	ICV Out
	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
0000_1	Length	HMAC Key	Hash-only Header	Cipher IV In	Cipher Key	Main Data In	Data Out	Cipher IV Out
ipsec_esp	Extent	reserved	reserved	reserved	reserved	ICV In	ICV Out	reserved
0001_1	Length	CRC-only Header	AES Context In	AES Key	Hash-only Header	Main Data In	Data Out	AES Context Out
802.11i AES ccmp	Extent	CRC In/Out (FCS)	reserved	reserved	reserved	MIC In	MIC Out	reserved
0010_1	Length	HMAC Key	AES Context In	AES Key	Main Data In	Data Out	HMAC Out	AES Context Out
srtp with ICV Check	Extent	reserved	reserved	reserved	Hash-only Header	Hash-only Trailer	reserved	reserved
0010_1	Length	HMAC Key	AES Context In	AES Key	Main Data In	HMAC In	Data Out	AES Context Out
srtp without ICV Check	Extent	reserved	reserved	reserved	Hash-only Header	Hash-only Trailer	HMAC Out	reserved
0011_1	Length	“A0” In	“A1” In	“A2” In	“A3” In	“B0” In	“B1” In	"Build" Out
pkeu_build	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
0100_1	Length	“N” In	“E” In	“Build” In	“B1” Out	“B2” Out	“B3” Out	reserved
pkeu_ptmul	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
0101_1	Length	“N” In	“Build” In	“B2” In	“B3” In	“B1” Out	“B2” Out	“B3” Out
pkeu_ptadd_ dbl	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
1000_1 outbound	Length	MAC Key	Cipher IV In	Cipher Key	Main Data In	Cipher-only Trailer	Data Out	Cipher IV Out
tls_ssl_ block	Extent	reserved	reserved	reserved	Hash-only Header	ICV Out	reserved	reserved
1000_1 inbound	Length	MAC Key	Cipher IV In	Cipher Key	reserved	Main Data In	Data Out	Cipher IV Out
tls_ssl_ block	Extent	reserved	reserved	reserved	Hash-only Header	ICV In	ICV Out	reserved

Table 17-10. Descriptor Format Summary (continued)

Descriptor Type	field type	Pointer Dword0	Pointer Dword1	Pointer Dword2	Pointer Dword3	Pointer Dword4	Pointer Dword5	Pointer Dword6
1001_1 outbound	Length	MAC Key	Cipher IV In	Cipher Key	Main Data In	reserved	Data Out	Cipher IV Out
tls_ssl_ stream	Extent	reserved	reserved	reserved	Hash-only Header	ICV Out	reserved	reserved
1001_1 inbound	Length	MAC Key	Cipher IV In	Cipher Key	reserved	Main Data In	Data Out	Cipher IV Out
tls_ssl_ stream	Extent	reserved	reserved	reserved	Hash-only Header	ICV In	ICV Out	reserved
1010_1	Length	Source F Data In	Source E Data In	Source D Data In	Source C Data In	Source B Data In	Source A Data In	Data Out
raid_xor	Extent	reserved	reserved	reserved	reserved	reserved	reserved	reserved
1011_1	Length	AES Context In	AAD In	Nonce Part 2 In	AES Key In	Main Data In	Data Out	Cipher Context Out
ipsec_aes_ gcm	Extent	reserved	reserved	reserved	Nonce Part 1 In	AES ICV In	AES ICV Out	CRC ICV In/Out
1100_1	Length	Header In	Payload In	reserved	reserved	reserved	reserved	reserved
dbl_crc	Extent	Header ICV	Payload ICV	Header ICV Out	Payload ICV Out	reserved	reserved	reserved
others		reserved						

17.4 Polychannel

The polychannel is the main control unit in the SEC. It implements four independent channels.

Each cryptographic task performed by the SEC is managed by a channel and makes use of one or more of the SEC's execution units (EUs). Control information and data pointers for a given task are stored in the form of a descriptor (see Section 17.3.1, "Descriptor Structure") placed in system memory. A descriptor determines what EUs are used, how they are configured, where to fetch needed data, and where to store the results.

The following subsections describe the operation (including descriptor processing, arbitration, and host notification), registers, and interrupts of the polychannel.

17.4.1 Channel Operation

17.4.1.1 Channel Descriptor Processing

To invoke a cryptographic task, the host constructs a descriptor, selects a channel, and writes a pointer to the descriptor into the selected channel's fetch FIFO. Each fetch FIFO can store up to 24 pointers.

Typical operations performed by a channel to process a descriptor are:

1. Analyze the descriptor header to determine the cryptographic services required, and arbitrate for the appropriate EUs. If required EUs are already reserved by another channel, wait for the EUs to be available. When available, reserve them.
2. Set the mode register in each reserved EU(s) for the required EU function.
3. Fetch "parcels" (up to 64K–1 bytes long) from system memory using pointers from the descriptor buffer, and place them in either an EU input FIFO or EU registers, as appropriate. The term "parcel" refers here to any input or output of an EU algorithm, such as a key, hash result, input context, output context, or text data. "Context" refers to either an IV (initialization vector) or other internal EU state that can be read out or loaded in. "Text data" refers to plaintext or ciphertext to be operated on. Each parcel transfer may involve using link tables to gather input data that has been split into multiple segments in system memory.
4. Take data accumulated in the EU output FIFO and write it to system memory using pointers from the descriptor buffer. This may again involve using link tables to scatter output data into multiple segments in system memory.
5. If the data size is greater than EU FIFO size, continue fetching input data and writing output data to memory as needed.
6. After writing the last input data to each EU's input FIFO, write to the end of message register in the EU.
7. Wait for EU(s) to complete processing of text data.
8. Unload final results from output FIFOs and context registers and write them to external memory using pointers from the descriptor buffer. This may again involve using link tables to scatter output data into multiple segments in system memory.
9. Reset and release the EUs.
10. If enabled, then notify the host of descriptor completion (see Section 17.4.1.3, "Channel Host Notification").

17.4.1.2 Channel Arbitration

All channels share a set of common resources, including the EUs and the SEC's bus master interface (managed by the SEC controller). When multiple channels are used in parallel, arbitration may be required to determine which channel is serviced. The different arbitration schemes are described in Section 17.5.2, "Arbitration Algorithms."

Generally speaking, no arbitration for use of the controller/bus master interface is required. The channels within the polychannel execute one at a time, so individual channels do not experience contention when requests to the controller. In effect, when a channel wins arbitration for use of the polychannel, it wins use of the controller as well.

The same is not true of EUs. Once the controller has assigned an EU to a channel, that channel owns the EU for the duration of descriptor processing. The maximum amount of data that can be processed by a single descriptor is 64 Kbytes, which prevents a channel from owning an EU for an unbounded length of time.

While one channel owns a particular EU, it is possible for two or more other channels to request access to the same EU; in this case, an arbitration scheme determines which channel is granted next access. EU arbitration schemes are similar to channel arbitration mentioned above, and are described in Section 17.5.2, "Arbitration Algorithms."

If a channel needs two EUs, a primary and a secondary, it requests them one at a time. Sometimes a channel reserves one EU and then has to wait for some other channel(s) to finish before obtaining the second requested EU. Though such waiting may occur, the requests are always eventually satisfied. Deadlock is avoided through the following design rules:

1. The channel always requests the secondary EU first.
2. In cases where both a primary and secondary are used, the choices for primaries and secondaries are distinct sets. Primaries are AESU, AFEU, DES, and KFEU, and the secondaries are MDEU and CRCU.

17.4.1.3 Channel Host Notification

When a channel completes operation on a descriptor, it can notify the host that it is done through interrupt and/or through a writeback of the descriptor header dword. In case the descriptor operation is not completed or completed with a known error, the host may be notified by an error interrupt. The error interrupts, done interrupts and header writeback are described as follows:

- Error interrupts are always enabled at the channel level, but can be masked at the controller level. For more details concerning these interrupts, see Section 17.4.2.2, "Channel Error Interrupt."
- Done interrupts are enabled on a per-channel basis by programming the channel's configuration register. For programming details, refer to Section 17.4.2.1, "Channel Done Interrupt," and Section 17.4.4.1, "Channel Configuration Register (CCR)."
- Independently of the done interrupt, channels can inform software of their completion status via header writebacks. Like done interrupts, writebacks are enabled on a per-channel basis by programming the CCR. If enabled, then upon completion the channel writes 0xFF to the DONE byte in the original descriptor header (see Table 17-5), allowing software to poll for completion of a specific descriptor. The CCR can also be programmed so that the channel writes back a status code indicating whether an integrity checking EU has encountered a mismatch between the received ICV and the recalculated ICV. Table 17-5 shows the specific bytes in the descriptor header that are updated in this case.

  For more details on programming the CCR for writeback, see Section 17.4.4.1, "Channel Configuration Register (CCR)."

NOTE

The done and status writebacks are not performed should the channel signal any error during processing. For example, there are no writebacks in case of a failing, unmasked ICV check in an EU.

17.4.2 Channel Interrupts

Active channels can assert done and error interrupts to the controller. As with all SEC interrupt events, channel done and error interrupts are reflected in the controller's interrupt status register. Channel do not have internal interrupt masks, but the controller can be programmed to disable channel interrupts through its interrupt enable register. For more details on interrupt types and disablement, see Section 17.5.4.2, "Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)."

17.4.2.1 Channel Done Interrupt

Channel done interrupt generation depend on the setting of the CDIE (channel done interrupt enable) and NT (notification type) bits in the channel configuration register (see Section 17.4.4.1, "Channel Configuration Register (CCR)"). If both CDIE and NT are set, the channel generates an interrupt event after every successfully completed descriptor; If CDIE is set and NT is cleared, an interrupt is generated after each successfully completed descriptor with the DN (done notification) bit set in the descriptor's header word. If the EU(s) signal any error during processing, the channel done interrupt is not generated.

Even if multiple channel done interrupt events are generated by a channel before the first can be cleared by the host, the interrupt events are not lost. The controller keeps count of the backlog of channel done interrupts from each channel (see Section 17.5.3, "Controller Interrupts").

17.4.2.2 Channel Error Interrupt

The channel error interrupt is generated when an error condition occurs during descriptor processing. The error could be a bus error for a transaction requested by the channel; or it could be in one of the EUs reserved by the channel, or in the channel itself. The channel error interrupt is asserted as soon as the error condition is detected. The type of error condition is reflected in the ERROR field of the channel status register (CSR).

For most error types, the error causes the corresponding channel to halt. Any EUs reserved by the halted channel continue to be reserved until the channel reset occurs. Other channels continue normal processing, though they may be held up if they need an EU that is reserved by a halted channel.

Handling of errors depends on the error type. Details of each error type are given in Table 17-15. For some types, the host must clear the source of the error before restarting the channel. If the channel is halted, the host restarts it by setting the no-pop-reset, continue or reset bits of the CCR (see Section 17.4.4.1, "Channel Configuration Register (CCR)").

17.4.3 Polychannel Registers

The polychannel has several aggregate performance counters, which are common to all channels; plus a set of channel-specific registers, descriptor buffers, and link tables which are duplicated for each channel. The following subsections describes the format and function of all of these objects in the SEC's memory.

17.4.3.1 Traffic Counters

The SEC maintains several counters, which are described in the following subsections.

17.4.3.1.1 Fetch FIFO Enqueue Counter

The fetch FIFO enqueue counter, shown in Figure 17-7, counts the total number of descriptor addresses that have been enqueued to the channel fetch FIFOs.

If the FETCH_FIFO_ENQ_COUNT field is 0x1111_1111, then adding another entry to the FIFO clears the register and causes the FFE_CNT bit (if enabled) to be set in the controller's interrupt status register (see Section 17.5.4.2, "Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)").

Offset Channel 0x3_1500 Access: Read/Write

	0–31	32–63
R W	—	Fetch_FIFO_ENQ_COUNT
Reset	All zeros	

Figure 17-7. Fetch FIFO Enqueue Counter

17.4.3.1.2 Descriptor Finished Counter

The descriptor finished counter, shown in Figure 17-8, indicates the total number of descriptors that have successfully completed processing. It does not count descriptors that halt due to error.

When the DESCRIPTOR_FINISHED_COUNT field reaches 0x1111_1111, then the next completed descriptor clears the counter and causes the DF_CNT bit (if enabled) to be set in the controller's interrupt status register (see Section 17.5.4.2, "Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)").

Offset Channel 0x3_1508 Access: Read/Write

	0–31	32–63
R W	—	Descriptor_Finished_Count
Reset	All zeros	

Figure 17-8. Descriptor Finished Counter

17.4.3.1.3 Data Bytes In Counter

The data bytes in counter, shown in Figure 17-9, indicates the total number of bytes written into a primary EU input FIFO. If other parcels such as context or ICV are placed in the input FIFO, they are not counted. When a secondary EU is used, data going only to the secondary EU (such as a hash-only region or authentication data) is counted, but the data used by both EUs is not double counted.

If this counter reaches all 1s, at the next count it rolls over to all 0s and the interrupt enable register's DI_CNT bit is set (see Section 17.5.4.2, "Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)").

If this counter is read by software in 32-bit increments, then the least significant 32 bits must be read first, followed by the most significant 32 bits. If this counter is written by software in 32 bit increments, then the most significant 32 bits must be written first, followed by the least significant 32 bits. Note that 32 bit reads and writes must not be interleaved (that is, read low, write low, read high, write high is not allowed). These restrictions are required to maintain counter coherency.

Offset Channel 0x3_1510 Access: Read/Write

	0–31	32–63
R/W	Data_Bytes_In_Count	
Reset	All zeros	

Figure 17-9. Data Bytes In Counter

17.4.3.1.4 Data Bytes Out Counter

The data bytes out counter, shown in Figure 17-10, indicates the total number of payload bytes read from an EU output FIFO. If other parcels such as context or ICV are read from the output FIFO, they are not counted. In no case is data counted twice by the same counter.

If this counter reaches all 1s, at the next count it rolls over to all 0s and the interrupt enable register’s DO_CNT bit is set (see Section 17.5.4.2, “Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)”).

If this counter is read by software in 32-bit increments, then the least significant 32 bits must be read first, followed by the most significant 32 bits. If this counter is written by software in 32 bit increments, then the most significant 32 bits must be written first, followed by the least significant 32 bits. Note that 32 bit reads and writes must not be interleaved (that is, read low, write low, read high, write high is not allowed). These restrictions are required to maintain counter coherency.

Offset Channel 0x3_1518 Access: Read/Write

	0–31	32–63
R/W	Data_Bytes_Out_Count	
Reset	All zeros	

Figure 17-10. Data Bytes Out Counter

17.4.4 Channel Registers

The channel registers are replicated for each of the 4 channels in the polychannel.

17.4.4.1 Channel Configuration Register (CCR)

This register contains bits that allow the user to configure and reset the channel. The CCR fields are shown in Figure 17-11, and described in Table 17-11.

Offset Channel 1: 0x3_110C
Channel 2: 0x3_120C
Channel 3: 0x3_130C
Channel 4: 0x3_140C
Channel 1: 0x3_110C
Channel 2: 0x3_120
Channel 3: 0x3_130C
Channel 4: 0x3_140C

Access: Read/Write

	0–28	29	30	31
R/W	—	NPR	CON	R
Reset	All zeros			

	32–50	51	52–54	55	56	57	58	59	60	61	62	63
R/W	—	PBS	—	BS	IWSE	—	EAE	CDWE	AWSE	NT	CDIE	—
Reset	All zeros											

Figure 17-11. Channel Configuration Register (CCR)

Table 17-11. Channel Configuration Register Fields

Bits	Name	Description
0–28	—	Reserved, should be set to zero.
29	NPR	No-Pop-Reset[1]. 0 No special action. 1 Causes the same channel reset actions as the CON bit, except that the fetch FIFO is left unchanged, such that the channel picks up by re-fetching the previous descriptor. This permits debug of a descriptor in-place without having to rewrite the descriptor pointer into the fetch FIFO. • If the NPR bit is set while the channel is requesting an EU assignment from the controller, the channel cancels its request. • If the NPR bit is set after the channel has been assigned one or more EUs, the channel requests a write from the controller to set the software reset bit of each reserved EU. The channel then releases the EU(s).
30	CON	Continue bit[1]. 0 No special action. 1 Causes the same channel reset actions as bit R, except that the fetch FIFO and bits 32-63 of the CCR register are not cleared. After the reset sequence is complete, this bit automatically returns to 0 and the channel resumes normal operation, servicing the next descriptor pointer in the fetch FIFO, if any. • If the CON bit is set while the channel is requesting an EU assignment from the controller, the channel cancels its request. • If the CON bit is set after the channel has been assigned one or more EUs, the channel requests a write from the controller to set the software reset bit of each reserved EU. The channel then releases the EU(s).

Table 17-11. Channel Configuration Register Fields (continued)

Bits	Name	Description
31	R	Reset channel[1]. 0 No special action. 1 Causes a software reset of the channel. All channel registers are cleared. Other actions depend on the state of the channel when the bit is set: • If the R bit is set while the channel is requesting an EU assignment from the controller, then the channel cancels its request. • If the R bit is set after the channel has been assigned one or more EUs, the channel requests a write from the controller to set the software reset bit of each reserved EU. The channel then releases the EU(s). After the reset sequence is complete, the channel returns to the idle state, the R bit is cleared automatically, and normal operation is resumed.
32–48	—	Reserved, should be set to zero.
49	FCC	Fast clock counting. 0 Watchdog timer counts normally 1 Watchdog timer counts in an accelerated fashion (force-assert several selected bits in timer) to assist with functional testing.
50	WGN	Watchdog go now. 0 Watchdog timer disabled 1 Watchdog timer enabled
51	PBS	Permit byte summing. 0 Bytes written to EU input FIFOs and read from EU output FIFOs are not counted in the data bytes counters 1 Bytes written to EU input FIFOs and read from EU output FIFOs are counted in the data bytes counters
52–54	—	Reserved, should be set to zero.
55	BS	Burst size. The SEC accesses long text-parcels in main memory through bursts of programmable size. 0 Burst size is 64 bytes 1 Burst size is 128 bytes
56	IWSE	ICV writeback status enable. 0 No special action. 1 If the descriptor calls for ICV checking, then at the completion of descriptor processing, the channel writes back to the descriptor header the DONE, ICCR0, and ICCR1 fields (see Table 17-5).[2]
57	—	Reserved, should be set to zero.
58	EAE	Extend address enable. This bit determines whether the channel uses a 36-bit address bus or a 32-bit address bus. 0 Channel's address bus is 32 bits. 1 Channel's address bus is 36 bits.
59	CDWE	Channel done writeback enable. 0 Channel done writeback disabled. 1 Channel done writeback enabled. Upon successful completion of descriptor processing, if the NT bit is cleared (for global notification), or if the DN (done notification) bit is set in the header word of the descriptor, then the channel notifies the host by writing back the descriptor header with the DONE field shown in Table 17-5. This enables the host to poll the memory location of the original descriptor header to determine if that descriptor has been completed.[2]

Table 17-11. Channel Configuration Register Fields (continued)

Bits	Name	Description
60	AWSE	Always writeback status enable. 0 No special action. 1 At the completion of processing each descriptor, the channel writes back to the descriptor header the DONE, ICCR0, and ICCR1 fields (see Table 17-5). In this case, IWSE has no effect.[2]
61	NT	Notification type. This bit controls when the channel generates channel done notification. Channel done notification can take the form of an interrupt and/or modified header writeback, depending on the state of the CDIE and CDWE control bits. 0 Global notification: The channel generates channel done notification (if enabled) at the end of each descriptor. 1 Selected notification: The channel generates channel done notification (if enabled) at the end of every descriptor with the DN bit set in the descriptor header.
62	CDIE	Channel done interrupt enable. 0 Channel done interrupt disabled 1 Channel done interrupt enabled. Upon successful completion of descriptor processing, if the NT bit is cleared (for global notification), or if the DN (done notification) bit is set in the header word of the descriptor, then a channel done interrupt is asserted to notify the host.[2] Refer to Section 17.4.4, "Channel Registers," for a complete description of channel done interrupt operation.
63	—	Reserved, should be set to zero.

[1] WARNING: When using reset bits R, CON and NPR: the configuration register must be polled to confirm completion of the multi-cycle reset sequence. The length of time required for this reset sequence depends on several factors and should be considered indeterminate. Completion is indicated by the self-clearing of the asserted reset bit. Failure to ensure completion of reset prior to writing to the channel may result in a channel hang condition.

[2] WARNING: The done interrupt, done writeback, and status writeback do not occur if an EU produces an error interrupt to the channel. In particular, if the ICV check error interrupt is enabled in the EU (see the ICE bit in the EU's interrupt mask register), and the ICV check finds a mismatch, then the channel produces an error interrupt but no channel done interrupt or writebacks.

Table 17-12 shows the CCR and descriptor header bit settings for different descriptor header writeback options; and Table 17-13 shows the bit settings for different done interrupt generation options.

Table 17-12. Writeback Options

AWSE CCR bit 60	CDWE CCR bit 59	IWSE CCR bit 56	NT CCR bit 61	DN Header bit 63	Writeback Action for a Descriptor completing without error
1	x	x	x	x	write back header fields DONE, ICCR0, ICCR1
0	1	x	1	0	no writeback performed
0	1	x	1	1	write back header field DONE
0	1	x	0	x	write back header field DONE
0	x	1	x	x	if the descriptor header indicates ICV checking in AESU, CRCU, KEU, or MDEU, then write back header fields DONE, ICCR0, and ICCR1.

Table 17-13. Done Interrupt Options

NT CCR bit 61	DN Header bit 63	CDIE CCR bit 62	Done Interrupt action by channel to controller for a descriptor completing without error
x	x	0	never assert done interrupt
0	x	1	assert done interrupt
1	0	1	never assert done interrupt
1	1	1	assert done interrupt

17.4.4.2 Channel Status Register (CSR)

CSR contains status fields and counters which provide status information regarding the channel’s processing of the current descriptor. This register is intended for debug use.

Figure 17-12 shows the channel status register fields, which are described in Table 17-14.

The multiple state-machine architecture of the channel makes it difficult to completely determine the channel’s status. The channel should be considered idle only if GET_STATE, PUT_STATE, and MAIN_STATE and FF_LEVEL are all cleared.

Offset Channel_1 0x01110
Channel_2 0x01210
Channel_3 0x01310
Channel_4 0x01410
Channel 1: 0x3_1114
Channel 2: 0x3_1214
Channel 3: 0x3_1314
Channel 4: 0x3_1414

Access: Read/Write

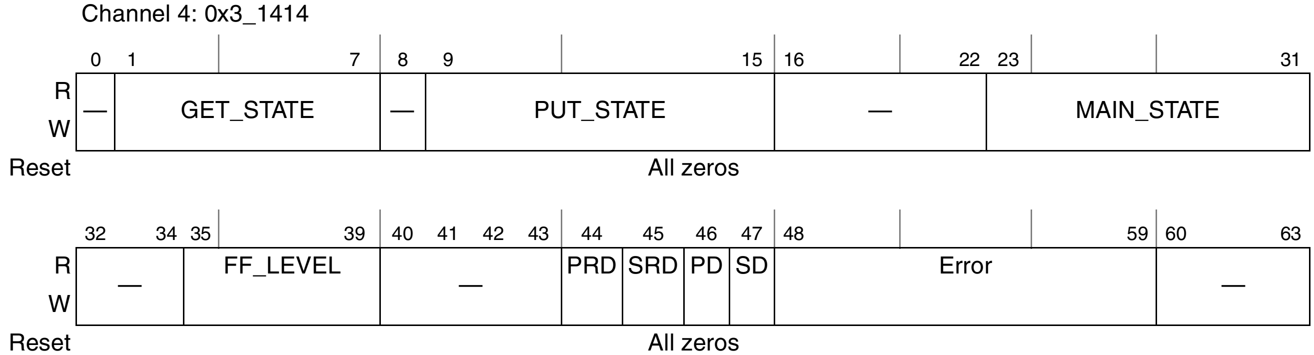


Figure 17-12. Channel Status Register (CSR)

Table 17-14. Channel Status Register Field Descriptions

Bits	Name	Description
0	—	Reserved.
1–7	GET_STATE	Get state machine state. This field reflects the state of the get state machine when it last went to sleep, or the state captured when an error occurred. For debug purposes only.

Table 17-14. Channel Status Register Field Descriptions

Bits	Name	Description
8	—	Reserved.
9–15	PUT_STATE	Put state machine state. This field reflects the state of the put state machine when it last went to sleep, or the state captured when an error occurred. For debug purposes only.
16–22	—	Reserved.
23–31	MAIN_STATE	Main state machine state. This field reflects the state of the main state machine when it last went to sleep, or the state captured when an error occurred. For debug purposes only.
32–34	—	Reserved, should be set to zero.
35–39	FF_LEVEL	Fetch FIFO level. This five-bit counter indicates how many pointers are currently stored in the fetch FIFO.
40-43	—	Reserved, should be set to zero.
44	PRD	Primary EU reset done. This bit reflects the state of the reset done signal from the assigned primary EU. 0 The assigned primary EU reset done signal is inactive. 1 The assigned primary EU reset done signal is active, indicating its reset sequence has completed and it is ready to accept data.
45	SRD	Secondary EU reset done. This bit reflects the state of the reset done signal from the assigned secondary EU. 0 The assigned secondary EU reset done signal is inactive. 1 The assigned secondary EU reset done signal is active, indicating its reset sequence has completed and it is ready to accept data.
46	PD	Primary EU done. This bit reflects the state of the done interrupt from the assigned primary EU. 0 The assigned primary EU done interrupt is inactive. 1 The assigned primary EU done interrupt is active, indicating the EU has completed processing and final values are available from EU registers. If the EU has an output FIFO, then all text data output has been placed in the output FIFO. If the EU provides context out through the output FIFO, then the context is placed in the output FIFO *after* the PD bit is asserted.
47	SD	Secondary EU done. The SEC_DONE bit reflects the state of the done interrupt from the assigned secondary EU. 0 The assigned secondary EU done interrupt is inactive. 1 The assigned secondary EU done interrupt is active, indicating the EU has completed processing and final values are available from EU registers.
48–59	Error	Error bits for the channel. See Figure 17-15.
60–63	—	Reserved.

Table 17-15 lists the errors corresponding to each bit in the CSR's Error field. Multiple bits may be set simultaneously. Whenever an error field bit is set a channel error interrupt is generated, and in most cases the channel is halted. For some error types, the host must take action to clear the error bit before restarting the channel, as described in Table 17-15. For information about restarting the channel, see the description of the R and CON bits in Section 17.4.4.1, "Channel Configuration Register (CCR)".

Table 17-15. Channel Status Register Error Field Definitions

CSR Bit #	Name	Error
48	DOF	Double Fetch FIFO write overflow error. This bit is set when the channel fetch FIFO is full, SOF is set, and another write has been made to the fetch FIFO. This error halts the channel. To clear this error, the host must write a ‘1’ to this bit.
49	SOF	Single Fetch FIFO write overflow error. This bit is set when the channel fetch FIFO is full and another write has been made to the fetch FIFO. The channel continues processing, but the descriptor pointer is lost. To clear this error, the host must write a ‘1’ to this bit.
50	MDTE	Master Data Transfer Error. When the SEC, while acting as a bus master, detects an error, the controller passes this error to the channel. This error halts the channel. Restarting the channel clears this bit.
51-52		Reserved
53	IDH	Illegal descriptor header. Possible causes of an illegal descriptor header are: • Invalid primary EU indicated by op0 field in descriptor header. • Invalid secondary EU indicated by op1 field in descriptor header. This error halts the channel. Restarting the channel clears this bit.
54		Reserved
55	EUE	EU error. An EU assigned t o t hi s channel has generated an error interrupt. This error may also be reflected in the controller's interrupt status register. This error halts the channel. To clear this error, the host must clear the error source in the EU that produced the error.
56	WDT	Watchdog timeout. The main state machine stayed asleep too long. This timer runs only after EUs have been reserved, and does not run if the primary EU is the RNGU or PKEU. The timeout interval is controlled by the FCC field of the Channel Configuration Register. This error halts the channel. Restarting the channel clears this bit.
57	SGLM	Scatter/Gather Length Mismatch. Indicates the total data size covered by a gather link table did not match the total data size from the main descriptor. This error halts the channel. Restarting the channel clears this bit.
58	RSI	RAID Size Incorrect. The channel was provided with a descriptor of type RAID_XOR with data sizes not permitted. To clear this error, the host must write a ‘1’ to this bit.
59	RSG	RAID Scatter Gather Error. The channel was provided with a descriptor of type RAID_XOR with a j bit set. Use of scatter/gather is not permitted with RAID_XOR type descriptors. To clear this error, the host must write a ‘1’ to this bit.

17.4.4.3 Current Descriptor Pointer Register (CDPR)

The CDPR reflects the value of the head end of the fetch FIFO, which contains the address of the descriptor which the channel is currently processing.

CPDR fields are shown in Figure 17-13, and described in Table 17-16.

Offset Channel 1: 0x3_1140, Channel 2: 0x3_1240, Channel 3: 0x3_1340, Channel 4: 0x3_1440 — Access: Read only

	0–15	16–27	28–31	32–63
R	ID_TAG	—	EPTR	CUR_DES_PTR_ADRS
W				
Reset	All zeros			

Figure 17-13. Current Descriptor Pointer Register

Table 17-16. Current Descriptor Pointer Register Fields

Bits	Name	Description
0–27	—	Reserved, must be cleared.
28-31	EPTR	Extended Pointer: Concatenated as the top 4 bits of the CUR_DES_PTR_ADRS when EAE is high (see the EAE bit description in Table 17-11).
32–63	CUR_DES_PTR_ADRS	Current Descriptor Pointer Address. Pointer to system memory location of the current descriptor. This field reflects the starting location in system memory of the descriptor currently loaded into the DB. This value is updated whenever the channel requests a fetch of a descriptor from the controller. The value from the fetch FIFO is transferred to the current descriptor pointer register immediately after the fetch is completed. This address is used as the destination for writeback of the modified header dword, if header writeback notification is enabled.

17.4.4.4 Fetch FIFO Enqueue Register (FFER)

Each channel contains a fetch FIFO to store a queue of pointers to descriptors which the channel will process. A pointer is added to the queue by writing to the FFER.

The register is shown in Figure 17-14, and the fields are described in Table 17-17.

Offset Channel 1: 0x3_1148, Channel 2: 0x3_1248, Channel 3: 0x3_1348, Channel 4: 0x3_1448

Access: Write only

	0–27	28–31	32–63
R			
W	—	EPTR	CUR_DES_PTR_ADRS
Reset	All zeros		

Figure 17-14. Fetch FIFO Enqueue Register (FFER)

Table 17-17. Fetch FIFO Enqueue Register Field Descriptions

Bits	Name	Description
0–27	—	Reserved, must be cleared.
28-31	EPTR	Extended Pointer: Concatenated as the top 4 bits of the FETCH_ADR when EAE is high (see the EAE bit description in Table 17-11).
32–63	FETCH_ADR	Fetch Address. Pointer to system memory location of the first byte of descriptor to be processed.

In channel-driven access, the host CPU creates a descriptor in memory containing all relevant mode and location information for the SEC, then launches the descriptor by writing its address to the fetch FIFO enqueue register.

The fetch FIFO can hold up to 24 descriptor pointers at a time. When the current descriptor's processing is finished, the next fetch FIFO entry is read and the descriptor located at FETCH_ADR is launched.

NOTE

When extended addresses are enabled (by setting the EAE bit in channel configuration register), then the FFER's EPTR field must be written before or concurrently with the FETCH_ADR field. This is necessary because writing the least significant byte (bits 56–63) is the "trigger" which causes the FFER contents to be added to the FIFO.

17.4.5 Channel Buffers and Tables

Besides the registers described in Section 17.4.4, "Channel Registers," each channel has memory allocated for descriptors and scatter/gather link table entries (described in Section Section 17.3, "Descriptors"). The following subsections describe these features.

17.4.5.1 Descriptor Buffer (DB)

The descriptor buffer (DB) provides read-only access to the descriptor currently being processed by the channel. All descriptors are 8 dwords long. For descriptor format, see Figure 17-2. The address ranges of each channel's DB are shown in Table 17-3.

Note that the DB is working storage and the channel may modify the contents of the DB during processing. In debug scenarios, it may be useful to read the contents of the DB to determine if a well formed descriptor is being fetched by the channel. Potential causes of malformed descriptors in the DB include:

- The descriptor is built incorrectly
- The descriptor is fully or partially overwritten by some other system bus master before the SEC can fetch the descriptor
- The descriptor is not built at the address written to the fetch FIFO

17.4.5.2 Scatter and Gather Link Tables (SLT, GLT)

A pointer dword in the descriptor buffer (DB) refers to a Gather Link Table (GLT) or a Scatter Link Table (SLT) if the J bit in the dword is set. As a channel works on a DB pointer entry, the GLT/SLT is loaded into channel memory. Reads from the GLT/SLT are enabled for debug purposes.

Figure 17-15 summarizes the entry format and address ranges for gather and scatter link table entries.

Offset Channel 1: 0x3_11c0-0x3_11df (Gather); 0x3_11e0-0x3_11ff(Scatter)
Channel 2: 0x3_12c0-0x3_12df (Gather); 0x3_12e0-0x3_12ff (Scatter)
Channel 3: 0x3_13c0-0x3_13df (Gather); 0x3_13e0-0x3_13ff (Scatter)
Channel 4: 0x3_14c0-0x3_14df (Gather); 0x3_14e0-0x3_14ff (Scatter)

Access: Read/Write

	0–15	16–21	22	23	24–27	28–31
R W	SEGLEN	—	R	N	—	EPTR
Reset	Not reset					

	32–63
R W	SEGPTR
Reset	Not reset

Figure 17-15. Gather/Scatter Link Table Entry Format and Memory Ranges

17.5 Controller

All transfers between the hosts and the EUs are moderated by the controller. Some of the main functions of the controller are as follows:

- Accept and execute commands from the slave system bus to read or write memory-mapped locations (up to 64 bits) anywhere in the SEC.
- Accept and execute requests from the polychannel to transfer blocks of bytes among system memory, EUs, and the channels.
- Arbitrate between channels when they contend for EUs and bus access
- Realign read and write data to the proper byte alignment
- Monitor interrupts from channels and pass them to the host

The remainder of this section discusses the controller’s bus management, arbitration, interrupts, and registers.

17.5.1 Bus Transfers

As shown in Figure 17-1, the SEC has an internal bus and connects to the SoC’s system bus. The internal bus is a private 64-bit slave bus, with the controller block as the sole master. The SoC’s system bus actually refers to two buses: a slave bus and a master bus, for which SEC’s controller operates as slave and master, respectively. All accesses to SEC over the system bus go through the controller.

As mentioned in Section 17.1.3, “Controller Overview”, there are two modes of access to the SEC, depending on whether the SEC’s controller is slave or master. These two modes of access (host-controlled and channel-controlled) are discussed in the following subsections.

17.5.1.1 Host-Controlled Access

For host-controlled access, the host uses the SoC’s slave bus to access the controller as a slave, and the controller relays the read or write accesses over the internal bus to the appropriate registers and FIFOs of the EUs. When a write command is received from the system bus, the controller takes the data and sends it to whichever internal location is indicated by the address. For a read, the controller goes to the internal location, fetches the requested data from the specified address (if allowed), and returns it over the system bus.

Host-controlled access is much more CPU-intensive than channel-controlled access, and requires a great deal of familiarity with the EU and controller registers and procedures. If host-controlled access is used, it is recommended that only a single EU be operated at a time. Snooping is not available through this interface.

NOTE

Host-controlled access of execution units is provided primarily for system debug purposes. The SEC contains no mechanism to arbitrate between host and channel accesses to EUs. Simultaneous use of an execution unit by a channel and a host is liable to force the execution unit into an error condition.

17.5.1.2 Channel-Controlled Access

Channel-controlled access is the SEC's normal operating mode. The controller performs data transfers based on information from the channels' descriptors. The controller can queue up to four requests. The controller dequeues requests and performs the required transfer. Most transfers involve not only the internal bus, but also the SoC's master bus with the controller as bus master.

When the SEC performs a read or write transaction as master, in some cases the intended target (for instance, system memory) may terminate the transaction due to an error. Once the transaction is posted to the SoC's target queue, it is the SoC's responsibility to either complete the transaction or signal an error. An error in an SEC-initiated transaction is also reported by the SEC through the channel interrupt status register (ISR). The host is able to determine which channel generated the interrupt by checking the ISR for the channel ERROR bit.

17.5.1.2.1 Channel Controlled Read—Detailed Description

A detailed description for a system bus read with controller as master is as follows:

1. Channel asserts bus read request to the controller
2. Channel furnishes external read address, internal write address, and transfer length
3. Controller asserts request to the system bus through the master interface
4. Controller waits for system bus read to begin
5. When bus read begins, controller receives data from the master interface and performs a write to the appropriate internal address supplied by the channel. Data may be realigned byte-wise by the controller if either:
   - the external read address was not on an 8-byte boundary, or
   - the internal write address was not on an 8-byte boundary.
6. Transfer continues until the bus read is completed and the controller has written all data to the appropriate internal address. The master interface continues making bus requests until the full data length has been read.

17.5.1.2.2 System Bus Master Write—Detailed Description

A detailed description for a system bus write with controller as master is as follows:

1. Channel asserts its bus write request to the controller.
2. Channel furnishes internal read address, external write address, and transfer length.
3. Controller performs a read from the appropriate internal address supplied by the channel, loads the write data into its FIFO, asserts a request to the system bus through the master interface, and waits for the system bus to become available.
4. When the system bus becomes available, controller writes data from its FIFO to the master interface.

17.5.2 Arbitration Algorithms

This section applies to both arbitration for use of the polychannel, and arbitration for use of execution units. Control fields for both are in the master control register (Section 17.5.4.6, “Master Control Register (MCR)”), as follows:

- CHN3_BUS_PR_CNT and CHN4_BUS_PR_CNT control polychannel arbitration
- CHN3_EU_PR_CNT and CHN4_EU_PR_CNT control EU arbitration

In this section we refer to generic control fields CHN3_*xx*_PR_CNT and CHN4_*xx*_PR_CNT, where “*xx*” refers to either “BUS” or “EU”.

If both CHN3_*xx*_PR_CNT and CHN4_*xx*_PR_CNT are zero (the default), the arbitration is round-robin (see Section 17.5.2.1); otherwise a weighted priority scheme is used (see Section 17.5.2.2).

17.5.2.1 Round-Robin Arbitration

In round-robin arbitration, requesting channels are granted access in rotating numerical order: 1, 2, 3, 4, 1, 2, ... etc.

17.5.2.2 Weighted Priority Arbitration

In the weighted priority scheme, the priority is as follows:

- Channel 1—Highest priority
- Channel 2—Second highest priority, unless CHN3_*xx*_PR_CNT or CHN4_*xx*_PR_CNT has expired
- Channel 3—Third priority, unless CHN4_*xx*_PR_CNT expired
- Channel 4—Lowest priority, until CHN4_*xx*_PR_CNT expired

Initially, the priority is fixed from highest to lowest as channel 1, channel 2, channel 3, and channel 4, in that order. When channel 3 has lost arbitration the number of times specified in CHN3_*xx*_PR_CNT, channel 3 replaces channel 2 as the second-highest priority in the next round of arbitration. Likewise, when channel 4 has lost arbitration the number of times specified in CHN4_*xx*_PR_CNT, channel 4 replaces channel 2 as the second-highest priority in the next round of arbitration. These rules prevent channels 3 and 4 from being locked out.

Channel 1 always has the highest priority, but cannot make back-to-back requests. It follows that the second highest priority channel wins arbitration either immediately, or after one win for channel 1.

Note that the SEC does not dynamically adjust its own transaction priorities. System software, however, can adjust SEC transaction priority in real time, with the change in priority taking effect immediately.

17.5.3 Controller Interrupts

17.5.3.1 Controller Interrupt Conditions and Interrupt Generation

All interrupt outputs from other SEC blocks are fed to the controller as interrupt conditions. In addition, the controller itself detects some interrupt conditions. The controller maintains an interrupt status register (ISR) with bits corresponding to all of these possible interrupt conditions. If an interrupt condition occurs and the corresponding bit of the interrupt enable register (IER) is set, then the associated ISR bit is set, indicating the presence of a pending interrupt.

A channel can generate frequent interrupts, especially if it is configured to interrupt at the completion of each descriptor. To make sure that the host receives the right number of interrupts, each channel done interrupt has a special “queuing” feature. If multiple channel done interrupts are generated before the first is cleared, then the additional interrupts are counted by the controller. Each time the host clears a channel interrupt, the count is decremented. If the host clears the channel interrupt and the count reaches zero, the channel done interrupt is negated. If the count does not reach zero, the controller negates the interrupt for one cycle and then re-asserts it.

Up to 15 interrupts can be queued for each channel. If the count of queued interrupts for any channel exceeds 15, then that channel’s done overflow bit is set in the channel’s ISR (if the corresponding IER bit is set), and the channel done interrupt is asserted.

17.5.3.2 Blocking of Interrupts

Interrupt conditions from the channels and controller can only be blocked through the controller’s IER, as described in Section 17.5.3.1. However, the EU interrupt conditions may be blocked at two different levels. There is an interrupt mask register in each EU which can block particular interrupt conditions before they reach the EU’s interrupt status register. In addition, interrupts from EUs can be individually blocked by bits of the controller’s IER before they reach the controller’s ISR. For normal operation, interrupts from EUs are typically disabled in the controller’s IER, but they still reach the channel, and the channel produces done or error interrupts to the host as needed.

17.5.3.3 Interrupt Handling

To handle an interrupt, the host must read the ISR to determine the source. If necessary, the host may read the interrupt status registers of other blocks to ascertain the cause. In some cases, the host may need to take action to clear the root cause of the interrupt. Once the appropriate action is taken, the host can clear the ISR bit by setting the corresponding bit of the interrupt clear register (ICR). If the cause of the interrupt condition has not been cleared, or if there is another interrupt condition from the same source, then the ISR bit clears for a cycle and then goes high again, and the interrupt signal to the host remains high. If the ISR bit is successfully cleared and no other interrupt conditions are present, the controller negates its interrupt signal. If any interrupts are still pending in the ISR, the interrupt remains asserted.

17.5.4 Controller Registers

The controller registers are described in detail in the following sections.

17.5.4.1 EU Assignment Status Register (EUASR)

The EUASR indicates which EUs are reserved by a particular channel. When an EU is already assigned, it is inaccessible to any other channel.

The EAUSR fields are displayed in Figure 17-16. The register has a four-bit field for each EU which indicates the EU's assigned channel. The field values and corresponding channel assignments are shown in Table 17-18.

Offset 0x3_1028 Access: Read only

	0–3	4–7	8–11	12–15	16–19	20–23	24–27	28–31
R	—	AFEU	—	MDEU	—	AESU	—	DEU
W								
Reset	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0

	32–35	36–39	40–43	44–47	48–51	52–55	56–59	60–63
R	—	—	CRCU	—	—	PKEU	—	RNG
W								
Reset	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0	1 1 1 1	0 0 0 0	1 1 1 1	0 0 0 0

Figure 17-16. EU Assignment Status Register (EUASR)

Table 17-18. Channel Assignment Value

Value	Channel
0x0	No channel assigned
0x1	Channel 1
0x2	Channel 2
0x3	Channel 3
0x4	Channel 4
0xA–0xE	Undefined
0xF	Unavailable

17.5.4.2 Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (IER, ISR, ICR)

The SEC controller generates the interrupt outputs from all possible interrupt sources. These outputs are enabled, displayed, and cleared by the IER, ISR, and ICR, respectively. These three registers share a common set of bit fields, which are shown in Figure 17-17. The corresponding interrupt sources are described in Table 17-19.

The IER, ISR, and ICR are described in more detail in the following subsections.

	0–7	8	9	10	11	12–14	15
Field	—	FFE_CNT	DF_CNT	DI_CNT	DO_CNT	—	ITO
Subfield							
Reset	0x0000						
R/W	R/W(Interrupt Enable) R(Interrupt Status) W(Interrupt Clear)						
Addr	0x3_1008(Interrupt Enable) 0x3_1010(Interrupt Status) 0x31018(Interrupt Clear)						

	16–19	20	21	22	23	24	25	26	27	28	29	30	31
Field	—	DONE Overflow				CHN_4		CHN_3		CHN_2		CHN_1	
Subfield		CH4	CH3	CH2	CH1	Err	Dn	Err	Dn	Err	Dn	Err	Dn
Reset	0x0000												
R/W	R/W(Interrupt Enable) R(Interrupt Status) W(Interrupt Clear)												
Addr	0x3_100A(Interrupt Enable) 0x3_1012(Interrupt Status) 0x3101A(Interrupt Clear)												

	32–35	36	37	38	39	40–41	42	43	44–45	46	47
Field	—	CRCU		KEU		—	PKEU		—	RNG	
Subfield		Err	Dn	Err	Dn		Err	Dn		Err	Dn
Reset	0x0000										
R/W	R/W(Interrupt Enable) R(Interrupt Status) W(Interrupt Clear)										
Addr	0x3_100C(Interrupt Enable) 0x3_1014(Interrupt Status) 0x3101C(Interrupt Clear)										

	48–49	50	51	52–53	54	55	56–57	58	59	60–61	62	63
Field	—	AFEU		—	MDEU		—	AESU		—	DEU	
Subfield		Err	Dn		Err	Dn		Err	Dn		Err	Dn
Reset	0x0000											
R/W	R/W(Interrupt Enable) R(Interrupt Status) W(Interrupt Clear)											
Addr	0x3_100E(Interrupt Enable) 0x3_1016(Interrupt Status) 0x3101E(Interrupt Clear)											

Figure 17-17. Interrupt Enable, Interrupt Status, and Interrupt Clear Registers

Table 17-19. Field Names in Interrupt Enable, Interrupt Status, and Interrupt Clear Registers

Bits	Name	Description
0–7	—	Reserved
8	FFE_CNT	Fetch FIFO enqueue count rollover 0 No rollover. 1 Fetch FIFO enqueue counter rolled over to zero (see Section 17.4.3.1.1, "Fetch FIFO Enqueue Counter").

Table 17-19. Field Names in Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (continued)

Bits	Name	Description
9	DF_CNT	Descriptor Finished Count Rollover 0 No rollover. 1 The Descriptor Finished Counter rolled over to zero (see Section 17.4.3.1.2, "Descriptor Finished Counter").
10	DI_CNT	Data In Count Rollover 0 No rollover. 1 The Data In Counter rolled over to zero (see Section 17.4.3.1.2, "Descriptor Finished Counter").
11	DO_CNT	Data Out Count Rollover 0 No rollover. 1 The Data Out Counter rolled over to zero (see Section 17.4.3.1.2, "Descriptor Finished Counter").
12–14	—	Reserved
15	ITO	Internal Time Out 0 No internal time out 1 an internal time out was detected **Note:** Internal time out is an indication that a channel or EU has failed to respond to a slave read or write within 16 cycles, which would only occur in an impending hang condition. Assertion of this interrupt indicates the SEC controller has completed the transaction to avoid a hang—however the 'completed' transaction does not result in a successful read or write, and the interrupt advises the system that the slave transaction was unsuccessful.
20–23	Done Overflow	Done Overflow (one bit for each channel—CH1 to CH4) 0 No done overflow 1 Done overflow error. Indicates that more than 15 Done interrupts were queued from the associated channel without a corresponding interrupt clear from the host.
24–31	Err and Dn bits for channels (CHN_1 to CHN_4)	Err 0 No error detected. 1 Error detected. Indicates that channel status register must be read to determine exact cause of the error. Dn 0 Not DONE. 1 DONE bit indicates that the corresponding channel has completed a descriptor.

Table 17-19. Field Names in Interrupt Enable, Interrupt Status, and Interrupt Clear Registers (continued)

Bits	Name	Description
36–37, 38-39, 42–43, 46–47, 50–51, 54–55, 58–59, 62–63	Err and Dn bits for execution units (CRCU,KEU, PKEU, RNG, AFEU, MDEU, AESU,DEU)	Err 0 No error detected. 1 Error detected. Indicates that execution unit status register must be read to determine exact cause of the error. Dn 0 Not Done 1 DONE bit indicates that the corresponding EU has completed its operation. This means that final values are available from EU registers. For EUs with output FIFOs, it means that all text data output has been placed in the output FIFO. For EUs that provide context out through the output FIFO, the EU places the context in the output FIFO after asserting PRI_DONE.
0–9, 16–19, 32–35, 40–41, 44–45, 48–49, 52–53, 56–57, 60–61	—	Reserved, must be cleared.

17.5.4.2.1 Interrupt Enable Register (IER)

Interrupt sources can be individually enabled by setting the corresponding IER bits (see Table 17-19 for the correspondence between IER bits and interrupt sources). If an IER bit is set, the corresponding interrupt source value is captured in the corresponding interrupt status register (ISR) bit. If an IER bit is cleared, the corresponding ISR bit remains cleared.

At reset, all IER bits are cleared, so all interrupts are disabled.

NOTE

For normal operation the IER should be programmed with the value 0x0031_0fff_0000_0000, which enables all channel interrupts and disables interrupts from the EUs. The EU interrupt bits are provided as a convenience during debug: during normal operation, an EU error causes the channel using that EU to generate the appropriate interrupt to the host.

17.5.4.2.2 Interrupt Status Register (ISR)

Each ISR bit shows the status of a corresponding interrupt source (see Table 17-19 for the correspondence between ISR bits and interrupt sources). However, if the corresponding IER bit is cleared, then the ISR bit remains cleared.

ISR bits are cleared either by reset, or by setting the corresponding bits in the ISR or interrupt clear register.

17.5.4.3 Interrupt Clear Register (ICR)

The ICR provides a means of clearing the interrupt status register (ISR). Setting an ICR bit clears the corresponding bit in the ISR, and negates the interrupt output signal (assuming that particular ISR bit is the only interrupt source). When set, an ICR bit is cleared automatically on the following cycle.

NOTE

If the cause of an interrupt is not removed, then the ISR bit is set (and corresponding interrupt output signal asserted) a few cycles after it has been cleared using the ICR.

For this reason, the ICR is ineffective in clearing the RNG Done bit (bit 47) in the ISR. The user should use the IER to mask the RNG Done interrupt. To determine whether a descriptor-based RNG request is complete, the user should rely on Channel Done interrupts.

17.5.4.4 ID Register

The read-only ID register, displayed in Figure 17-18, contains the same value as the IP block revision register (see Section 17.5.4.5 below). This register provides the IP block revision information at a legacy location for software convenience.

Offset 0x3_1020 — Access: Read only

Bits	0–63
R	ID
W	
Reset	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 17-18. ID Register

17.5.4.5 IP Block Revision Register

The read-only IP block revision register, displayed in Figure 17-19, contains a 64-bit value that uniquely identifies the version of the SEC.

Offset 0x 3_1BF8 — Access: Read only

Bits	0		31	32			63
R	IP_ID	IP_MJ	IP_MN	—	IP_INT	—	IP_CFG
W							
Reset	All zeros						

Figure 17-19. IP Block Revision Register

Table 17-20 describes the fields of the IP block revision register.

Table 17-20. IP Block Revision Register Fields

Bits	Name	Description
0–15	IP_ID	IP block identifier. This field value is currently set as 0x0030
16-23	IP_MJ	IP major revision number. This field value is currently set as 0x03.
24-31	IP_MN	IP minor revision number. This field value is currently set as 0x00
32-39	—	Reserved
40-47	IP_INT	IP block integration options. Field value depends on the options of the specific SoC
48-55	—	Reserved
56-63	IP_CFG	IP block configuration options. Field value depends on the options of the specific SoC

17.5.4.6 Master Control Register (MCR)

The MCR, shown in Figure 17-20, controls certain functions in the controller and provides a means for software to reset the SEC. Table 17-21 describes the MCR fields.

Offset 0x3_1030 Access: Read/Write

	0–21	22–23	24–29	30	31
R/W	—	PRIORITY	—	GIH	SWR
Reset	All zeros				

	32–39	40–47	48–55	56–63
R/W	CHN3_EU_PR_CNT	CHN4_EU_PR_CNT	CHN3_BUS_PR_CNT	CHN4_BUS_PR_CNT
Reset	All zeros			

Figure 17-20. Master Control Register

Table 17-21. Master Control Register Fields

Bits	Name	Description
0–21	—	Reserved
22-23	Priority	Priority on Master Bus. The setting of these bits determines the transaction priority level the SEC asserts to the SoC's internal arbiter. The SEC does not dynamically alter its priority level based on system congestion or SEC utilization; however, software may change the SEC priority level in real time. 00 Lowest Priority (default) 01 Next Lowest Priority 10 Next Highest Priority 11 Highest Priority
24-29	—	Reserved

Table 17-21. Master Control Register Fields (continued)

Bits	Name	Description
30	GIH	Global Inhibit. Setting this bit indicates that SoC master bus transfers are defined as not snoopable and results in lowering the snoop attribute of bus requests generated by the external gasket (see note following table). 0 SoC master bus transfers are defined as snoopable (default) 1 SoC master bus transfers are defined as not snoopable
31	SWR	Software Reset. Setting this bit causes a global software reset. Upon completion of the reset, this bit is automatically cleared. 0 Do not reset 1 Global reset
32–39	CHN3_EU_PR_CNT	Channel 3 EU Priority Count. In weighted priority arbitration, this field gives the number of times that Channel 3 is denied access for a requested EU before its priority is elevated (see Section 17.5.2.2, "Weighted Priority Arbitration"). **Note:** If both CHN3_EU_PR_CTR and CHN4_EU_PR_CTR are zero, the controller assigns EU's on a pure round-robin basis. If either of these counters is zero and the other is non-zero, then the zero is interpreted as 256.
40–47	CHN4_EU_PR_CNT	Channel 4 EU Priority Count. In weighted priority arbitration, this field gives the number of times that Channel 4 is denied access for a requested EU before its priority is elevated (see Section 17.5.2.2, "Weighted Priority Arbitration"). **Note:** If both CHN3_EU_PR_CTR and CHN4_EU_PR_CTR are zero, the controller assigns EU's on a pure round-robin basis. If either of these counters is zero and the other is non-zero, then the zero is interpreted as 256.
48–55	CHN3_BUS_PR_CNT	Channel 3 Bus Priority Count. In weighted priority arbitration, this field gives the number of times that Channel 3 is denied access to the polychannel before its priority is elevated (see Section 17.5.2.2, "Weighted Priority Arbitration"). **Note:** If both CHN3_BUS_PR_CTR and CHN4_BUS_PR_CTR are zero, the controller assigns the polychannel on a pure round-robin basis. If either of these counters is zero and the other is non-zero, then the zero is interpreted as 256.
56–63	CHN4_BUS_PR_CNT	Channel 4 Bus Priority Counter. In weighted priority arbitration, this field gives the number of times that Channel 4is denied access to the polychannel before its priority is elevated (see Section 17.5.2.2, "Weighted Priority Arbitration"). If both CHN3_BUS_PR_CTR and CHN4_BUS_PR_CTR are zero, the controller assigns the polychannel on a pure round-robin basis. If either of these counters is zero and the other is non-zero, then the zero is interpreted as 256.

NOTE

By default, All SEC memory transactions are snooped by the coherency module of the MPC85*xx*. This is part of the wiring of the SEC interface and requires no user intervention. Bit 30 in the MCR is used to inhibit cache snooping of SEC transactions in non-MPC85*xx* situations.

17.6 Power Saving Mode

The SEC may be disabled by setting DEVDISR[SEC] in the SoC. The clocks to the SEC are active by default. The SEC should not be enabled/disabled during normal operation.

SEC disablement is delayed if the disable request is made while descriptors are being processed. Once notified of the disable request, the SEC channels complete their current tasks, and then are forced to idle (with no additional reads from the fetch descriptor FIFO). Once all channels are idle, then SEC permits disablement.

17.7 Execution Units

Execution unit (EU) is the term used for a functional block that performs the mathematical manipulations required by cryptographic processing. The following execution units are used in the SEC (covered here in alphabetical order):

- Advanced Encryption Standard Execution Unit (AESU) implementing the Rijndael symmetric key cipher.
- ARC4 Execution Unit (AFEU)
- Cyclical Redundancy Check Unit (CRCU)
- Data Encryption Standard Execution Unit (DEU)
- Kasumi (f8/f9) Execution Unit (KEU)
- Message Digest Execution Unit (MDEU)
- Public Key Execution Unit (PKEU)
- Random Number Generator Unit(RNGU)

Working together, the EUs can perform high-level cryptographic tasks, such as IPsec Encapsulating Security Protocol (ESP) and digital signature. The remainder of this chapter provides details about these execution units, including modes of operation, status and control registers, and FIFOs.

17.7.1 Advanced Encryption Standard Execution Unit (AESU)

This section contains details about the Advanced Encryption Standard Execution Unit (AESU), including modes of operation, status and control registers, and FIFOs.

NOTE

Most of the registers described in this section are not accessed by the host under normal operation. They are documented here mainly for debug purposes. Normally the AESU is used through channel-controlled access, so that most reads and writes of AESU registers are directed by the SEC channels. Driver software performs host-controlled register accesses only on a few registers for initial configuration and error handling.

17.7.1.1 ICV Checking in AESU

For CCM, GCM, CMAC (OMAC1), and XCBC-MAC cipher modes, the AESU includes an ICV checking feature which can generate an ICV and compare it to another supplied ICV.

There are two methods for returning the pass/fail result of ICV checking to the host:

- The ICV check result can be sent to the host by a writeback of EU status fields into host memory. This is enabled as follows:
  - — Set either the IWSE or AWSE bit in the channel configuration register (see Section 17.4.4.1, "Channel Configuration Register (CCR)")
  - — Set the ICE bit in the interrupt mask register (Section 17.7.1.8, "AESU Interrupt Mask Register").

  In this case the normal done signaling (by interrupt or writeback) is undisturbed.
- The ICV checking result can be sent to the host by interrupt. This is enabled as follows:
  - — Clear the ICE bit in the interrupt mask register
  - — Clear both IWSE and AWSE bits in the channel configuration register.

  In this case, then the normal done signaling (by interrupt or writeback) occurs if there is no ICV mismatch. If an ICV mismatch occurs, then an error interrupt is sent to the host, but no channel done interrupt or writeback.

17.7.1.2 AESU Mode Register

The AESU mode register contains 11 non-reserved bits which are used to program the AESU. The mode register is cleared when the AESU is reset or re-initialized. Setting a reserved AESU mode register bit generates a data error. If the mode register is modified during processing, a context error is generated.

Figure 17-21 shows the AESU mode register, and Table 17-22 describes its fields. In normal operation, the register's values are set by the descriptor header (see Section 17.3.2, "Descriptor Format: Header Dword").

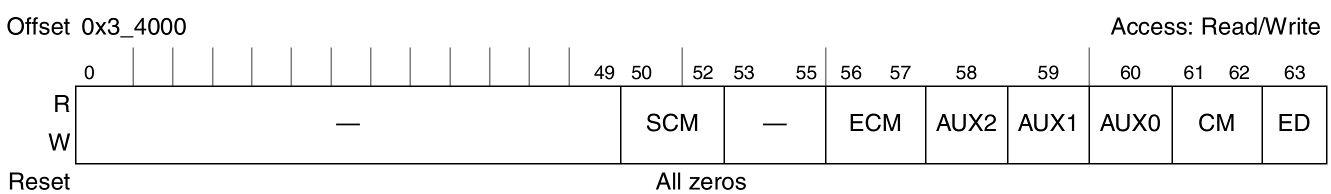


Figure 17-21. AESU Mode Register

Table 17-22. AESU Mode Register Field Descriptions

Bits	Name	Description
0–49	—	Reserved
50–52	SCM	Sub-Cipher Mode. Specifies additional options specific to particular cipher modes. • XOR cipher mode: specifies the number of sources to be XORed together. Valid values are 2-6. For all other cipher modes, this field must be 0.

Table 17-22. AESU Mode Register Field Descriptions (continued)

Bits	Name	Description
53–55	—	Reserved, must be cleared.
56–57	ECM	Extended Cipher Mode. Used in combination with bits 61:62 (Cipher Mode) to select the cipher mode for AES operation. See Table 17-23 on page 17-61 for mode bit combinations.
58	AUX2	AUX2 Mode. Definition depends upon the value of the 4 Cipher Mode (CM) and Extended Cipher Mode (ECM) bits:
	AUX2 = Finalize MAC for CCM and GCM modes	• CCM and GCM Cipher Modes (ECM=1X, CM=0X): Generate Final MAC Bit—Processes final message block and generates final MAC tag at the end of message processing. 0 = Do not generate final MAC tag 1 = Generate final MAC tag after CCM/GCM processing is complete. Note that for GCM, when message processing is split into multiple descriptors, it must be AUX1=1 when AUX2=1.
	AUX2 = LRW Tweak Control	• LRW Cipher Mode (ECM=10, CM=00): Context Switched Bit—Enables passing initial tweak value (T) to the cipher and skips its computation, thus allowing proper continuation of the interrupted processing after context switch by restoring the saved content of the Context Registers 1-2 (T—tweak) and 3-4 (I—Index). 1 = Skip computing tweak for the first data block and use the one provided in Context Registers 1-2. Also, use the index provided in Context Registers 3-4 for the computation of the next tweak. 0 = Ignore content of Context Registers 1-2 and compute the initial tweak using index provided in Context Registers 3-4
	AUX2 = ICV Bit	• XCBC-MAC and CMAC Cipher Modes (ECM=10, 01, CM=10): ICV Bit—Enables XCBC-MAC with ICV and CMAC with ICV Cipher Modes 0 = XCBC-MAC or CMAC cipher mode 1 = XCBC-MAC with ICV or CMAC with ICV cipher mode
	AUX2 = Enable RBP	• CBC, CBC-RBP Cipher Modes (ECM=00, CM=01): RBP Bit—Enables CBC-RBP 0 = CBC cipher mode 1 = CBC-RBP cipher mode

Table 17-22. AESU Mode Register Field Descriptions (continued)

Bits	Name	Description
59	AUX1	AUX1 Mode. Definition depends upon the value of the 4 Cipher Mode and Extended Cipher Mode bits:
	AUX1 = Initialize CCM	• CCM Cipher Mode (ECM=10, CM=00): Initialize Mode Bit—Initializes AESU for new message 0 = Do not initialize (context is loaded by host) 1 = Initialize new message with nonce/initialization vector
	AUX1 = Generate Final GHASH	• GCM Cipher Mode (ECM=10, CM=01): Generate Final GHASH Bit—Enables completion of GHASH computation by signaling that the last iteration of GHASH should be performed. This last iteration performs XOR of the current (intermediate) GHASH result with the concatenation of additional authenticated data (AAD) and ciphertext bit lengths in case of GHASH(H, AAD, ciphertext), or with the concatenation of 0^{64} and the bit length of IV in case of GHASH(H, {}, IV). As an exception, this bit should be cleared if the whole message (IV+AAD+text data) together with the generation of the final MAC is processed with one descriptor since in that case the generation of final GHASH is implied. Incidentally, whenever AUX1=1 in GCM cipher mode, the total bit lengths of AAD, text data or IV must be provided in context registers 9-10. 0 = Do not perform the last iteration in GHASH(H, AAD, ciphertext) or GHASH(H, {}, IV) unless the message is processed and the final MAC computed in 1 descriptor. 1 = Generate the final result of GHASH(H, AAD, ciphertext) or GHASH(H, {}, IV)—implies that the message processing is split into multiple descriptors.
	AUX1 = Use Context for XCBC-MAC derived keys	• XCBC-MAC Cipher Mode (ECM=10, CM=10): Load Keys—Do not compute K1, K2 and K3, but instead use the keys loaded in the Key Data Registers (K1), and Context Registers 5-6 (K2) and 7-8 (K3). 0 = Compute K1=E(K, 16{01}), K2=E(K, 16{02}), K3=E(K, {03}) and write K1 to Context Registers 3-4, K2 to 5-6, and K3 to 7-8. 1 = Load keys: K1= [Key Data Reg 1-2], K2= [Reg 5-6], K3=[Reg 7-8]
	AUX1 = Use Context for CMAC derived keys	• CMAC Cipher Mode (ECM=01, CM=10): Load Keys—Do not compute E(K, 0^{128}) to derive K1 and K2, but instead use the value loaded in Context Registers 3-4. This is useful after a context switch. Deriving K1 and K2 does not incur any timing penalty. 0 = Compute E(K, 0^{128}) and write it to Context Registers 3-4 1 = Load E(K, 0^{128}) and preserve it in Context Registers 3-4
60	AUX0	AUX0 Mode. Definition depends upon the value of the 4 Cipher Mode and Extended Cipher Mode bits, and Encrypt/Decrypt bit:
	AUX0 = GCM GHASH Only	• GCM Cipher Mode (ECM=10, CM=01) and Encrypt (ED=1): Specifies GHASH mode—performs GHASH on AAD and ciphertext 0 = Perform GCM encryption 1 = Compute GHASH(H, AAD, ciphertext)
	AUX0 = Finalize Mac for XCBC-MAC and CMAC modes	• XCBC-MAC, CMAC Cipher Modes (ECM=10, 01, CM=10): Do Not Generate Final MAC Bit—Does not generate final MAC tag at the end of message processing (used only when splitting a message into multiple descriptors) 0 = Generate final MAC tag by XORing the final data block with K2/K3 (for XCBC-MAC) or K1/K2 (for CMAC) before encryption 1 = Do not generate final MAC tag by XORing final data block before encryption. This enables message processing to be interrupted on the block boundary and later continued after a context switch.

Table 17-22. AESU Mode Register Field Descriptions (continued)

Bits	Name	Description
61–62	CM	Cipher Mode. Used in combination with bits 56:57 (Extended Cipher Mode) to select the cipher mode for AES operation. See Table 17-23 for mode bit combinations.
63	ED	Encrypt/Decrypt. If set, AESU operates the encryption algorithm; if cleared, AESU operates the decryption algorithm. 0 Perform decryption 1 Perform encryption **Note:** This bit is ignored in CTR, SRT, CMAC, and XCBC-MAC cipher modes.

Table 17-23 shows the AESU field settings corresponding to different AES cipher modes.

Table 17-23. AES Cipher Modes

Cipher Mode	ECM (56:57)	AUX2 (58)	CM (61:62)
ECB	00	X	00
CBC	00	X	01
CBC-RBP	00	1	01
OFB	00	X	10
CTR	00	X	11
LRW	01	X	01
CMAC	01	X	10
CMAC with ICV	01	1	10
SRT[1]	01	X	11
CCM	10	X	00
GCM	10	X	01
XCBC-MAC	10	0	10
XCBC-MAC with ICV	10	1	10
CFB128	10	X	11
CCM with ICV	11	X	00
GCM with ICV	11	X	01
XOR	11	X	11
Reserved	all others		

1 SRT is not a new AES cipher mode, it is an AESU method of performing AES counter mode with reduced context loading overhead specifically for performing SRTP. It should be used with descriptor type 0010_1 'srtp' (but may also be used with descriptor type 0010_0 for IPsec with AES counter mode). See the section on "Context for SRT Cipher Mode" for more information on how SRT cipher mode reduces context loading overhead.

17.7.1.3 AESU Key Size Register

The AESU key size register, shown in Figure 17-22, is used to specify the number of bytes in the key (16, 24, or 32). Any key data beyond the number of bytes specified in the key size register is ignored. This register is cleared when the AESU is reset or re-initialized. If a key size other than 16, 24, or 32 bytes (or other than 16 bytes, in XCBC-MAC cipher mode) is specified, an illegal key size error is generated. If the key size register is modified during processing, a context error is generated.

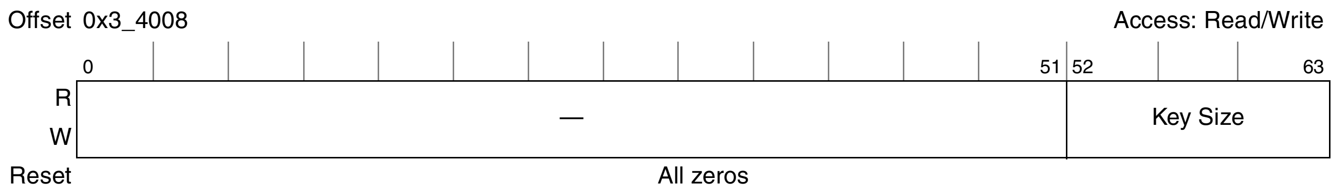


Figure 17-22. AESU Key Size Register

17.7.1.4 AESU Data Size Register

The AESU data size register, shown in Figure 17-23, is used to specify the number of bits (not bytes) of plaintext/ciphertext to be processed in the current descriptor. The number of data size register bits used by the SEC, and the acceptable values for these bits, vary depending on the AES cipher mode selected as specified in Table 17-24.

Writing to this register signals the AESU to start processing data from the input FIFO as soon as it is available. If the value of data size is modified during processing, a context error is generated. The register is cleared when the AESU is reset or re-initialized.

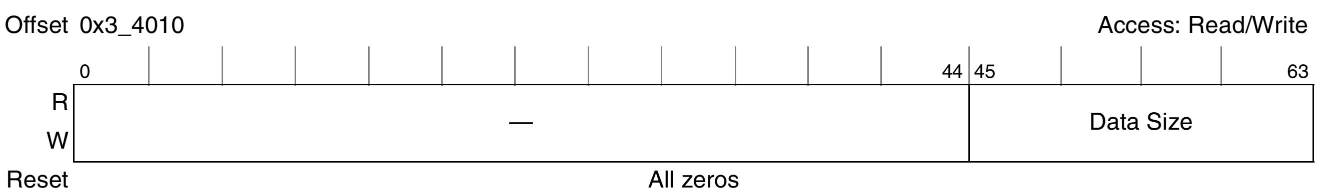


Figure 17-23. AESU Data Size Register

Table 17-24. Use of Data Size Register

AESU Cipher Mode	Register bits used by SEC (others are don't cares)	Legal Values (data size in bits)
ECB, CBC	lowest 7 bits [57:63]	must be a multiple of 128
OFB, CMAC, SRT, CCM, XCBC-MAC, CFB128		must be a multiple of 8
LRW	all bits	must be a multiple of 8, minimum 128
GCM	all bits	any value
XOR	all bits	must be a multiple of 256

17.7.1.5 AESU Reset Control Register

The AESU reset control register has three self-clearing bits, where each bit corresponds to a different type of AESU reset. Figure 17-24 shows the AESU reset control register, and Table 17-25 describes its fields.

Offset 0x3_4018 — Access: Read/Write

	0–60	61	62	63
R/W	—	RI	MI	SR
Reset	All zeros			

Figure 17-24. AESU Reset Control Register

Table 17-25. AESU Reset Control Register Field Descriptions

Bits	Names	Description
0–60	—	Reserved
61	RI	Reset Interrupt. Setting this bit resets the AESU's done and error interrupts, and resets the state of the AESU interrupt status register. 0 Do not reset 1 Reset interrupt logic
62	MI	Module Initialization. The same as software reset (including the initialization routine: see SR bit description below), except that the interrupt mask register remains unchanged. 0 Do not reset 1 Reset most of AESU
63	SR	Software reset. Functionally equivalent to hardware reset, but applies only to AESU. All registers and internal state are returned to their defined reset states. The RESET_DONE bit in the AESU status register indicates when this initialization routine is complete 0 Do not reset 1 Full AESU reset

17.7.1.6 AESU Status Register

The AESU status register is a read-only register that reflects the state of six status outputs. Writing to this location results in an address error being reflected in the AESU interrupt status register.

Figure 17-25 shows the AESU status register, and Table 17-26 describes its fields.

Offset 0x3_4028 — Access: Read only

	0–39	40–47	48–55	56–57	58	59–60	61	62	63
R	—	OFL	IFL	—	HALT	ICCR	EI	DI	RD
W									
Reset	All zeros								

Figure 17-25. AESU Status Register

Table 17-26. AESU Status Register Field Descriptions

Bits	Name	Description
0–39	—	Reserved
40-47	OFL	The number of dwords currently in the output FIFO
48-55	IFL	The number of dwords currently in the input FIFO
56-57	—	Reserved
58	HALT	Halt. Indicates that the AESU has halted due to an error. 0 AESU not halted 1 AESU halted **Note:** Because the error causing the AESU to stop operating may be masked before reaching the interrupt status register, the AESU interrupt status register is used to provide a second source of information regarding errors preventing normal operation.
59-60	ICCR	Integrity Check Comparison Result 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved **Note:** A passed or failed result is generated only if the cipher mode with ICV checking is selected
61	EI	Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 AESU is not signaling error 1 AESU is signaling error
62	DI	Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 AESU is not signaling done 1 AESU is signaling done
63	RD	Reset Done. This status bit, when high, indicates that AESU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel. 0 Reset in progress 1 Reset done **Note:** This bit resets to 0 but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation.

17.7.1.7 AESU Interrupt Status Register

The AESU interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the AESU interrupt mask register is zero (see Section 17.7.1.8, “AESU Interrupt Mask Register”). If an AESU interrupt mask register bit is set, the corresponding AESU interrupt status bit is always zero regardless of the error status.

If the AESU interrupt status register is non-zero, the AESU halts and the AESU error interrupt signal is asserted to the controller (see Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the

AESU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 17-15) and generates a channel error interrupt to the controller.

Interrupt status register bits can be set by writes from the host, but only if the corresponding bit is cleared in the interrupt mask register. Bits masked by the interrupt mask register bits are always zero.

The AESU interrupt status and interrupt mask registers can be cleared by programming the AESU reset control register, as described in Section 17.7.1.5, “AESU Reset Control Register”.

The AESU interrupt status register fields are shown in Figure 17-26. These fields are described in Table 17-27.

Offset 0x3_4030 — Access: Read only

	0–48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	63
R		ICE	—	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	—	IFO	OFU	—	ICE
W																	
Reset	All zeros																

Figure 17-26. AESU Interrupt Status Register

Table 17-27. AESU Interrupt Status Register Field Descriptions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity Check Error: 0 No error detected 1 Integrity check error detected. An ICV check was performed and the supplied ICV did not match the one computed by the AESU.
50	—	Reserved
51	IE	Internal Error. An internal processing error was detected while the AESU was processing. 0 No error detected 1 Internal error **Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the interrupt mask register or by resetting the AESU.
52	ERE	Early Read Error. An AESU context register was read while the AESU was processing. 0 No error detected 1 Early read error
53	CE	Context Error. An AESU key register or the key size register, data size register, mode register, or IV register was modified while AESU was processing 0 No error detected 1 Context error
54	KSE	Key Size Error. An inappropriate value (not 16, 24 or 32 bytes) was written to the AESU key size register. 0 No error detected 1 Key size error

Table 17-27. AESU Interrupt Status Register Field Descriptions (continued)

Bits	Name	Description
55	DSE	Data Size Error (DSE): A value was written to the AESU data size register that is not a proper size. See Section 17.7.1.4, "AESU Data Size Register." 0 No error detected 1 Data size error
56	ME	Mode Error. Indicates that invalid data was written to a register or a reserved mode bit was set. 0 Valid Data 1 Reserved or invalid mode selected
57	AE	Address Error. An illegal read or write address was detected within the AESU address space. 0 No error detected 1 Address error
58	OFE	Output FIFO Error. The AESU output FIFO was detected non-empty upon write of AESU data size register. 0 No error detected 1 Output FIFO non-empty error
59	IFE	Input FIFO Error. The AESU input FIFO was detected non-empty upon generation of done interrupt. 0 No error detected 1 Input FIFO non-empty error
60	—	Reserved
61	IFO	Input FIFO Overflow. The AESU Input FIFO was pushed while full. 0 No error detected 1 Input FIFO has overflowed **Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input. When operated through host-controlled access, the AESU cannot accept FIFO inputs larger than 256 bytes without overflowing.
62	OFU	Output FIFO Underflow. The AESU Output FIFO was read while empty. 0 No error detected 1 Output FIFO has underflow error
63	—	Reserved

17.7.1.8 AESU Interrupt Mask Register

The AESU interrupt mask register, controls the setting of bits in the AESU interrupt status register, as described in Section 17.7.1.7, "AESU Interrupt Status Register". If an AESU interrupt mask register bit is set, then the corresponding interrupt status register bit is always zero.

As shown in Figure 17-27, the interrupt mask register has the same field designations as the interrupt status register. Table 17-28 describes the AESU interrupt mask register fields.

Offset 0x3_4038 Access: Read/Write

	0–48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
R/W	—	ICE	—	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	—	IFO	OFU	—
Reset	1000															

Figure 17-27. AESU Interrupt Mask Register

Table 17-28. AESU Interrupt Mask Register Field Descriptions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity Check Error. The supplied ICV did not match the one computed by the AESU. 0 Integrity check error enabled. 1 Integrity check error disabled **Note:** ICE should not be enabled if using EU status writeback (see bits IWSE and AWSE in Section 17.4.4.1, "Channel Configuration Register (CCR)").
50	—	Reserved
51	IE	Internal Error. An internal processing error was detected while the AESU was processing. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early Read Error. An AESU context register was read while the AESU was processing. 0 Early read error enabled 1 Early read error disabled
53	CE	Context Error. An AESU key register or the key size register, data size register, mode register, or IV register was modified while the AESU was processing. 0 Context error enabled 1 Context error disabled
54	KSE	Key Size Error. An inappropriate value (non 16, 24 or 32 bytes) was written to the AESU key size register 0 Key size error enabled 1 Key size error disabled
55	DSE	Data Size Error. Indicates that the number of bits to process is out of range. 0 Data size error enabled 1 Data size error disabled
56	ME	Mode Error. Indicates that invalid data was written to a register or a reserved mode bit was set. 0 Mode error enabled 1 Mode error disabled
57	AE	Address Error. An illegal read or write address was detected within the AESU address space. 1 Address error disabled 0 Address error enabled

Table 17-28. AESU Interrupt Mask Register Field Descriptions (continued)

Bits	Name	Description
58	OFE	Output FIFO Error. Indicates the AESU Output FIFO was detected non-empty upon write of AESU data size register 0 Output FIFO non-empty error enabled 1 Output FIFO non-empty error disabled
59	IFE	Input FIFO Error. Indicates he AESU Input FIFO was detected non-empty upon generation of done interrupt 0 Input FIFO non-empty error enabled 1 Input FIFO non-empty error disabled
60	—	Reserved
61	IFO	Input FIFO Overflow. Indicates the AESU Input FIFO was pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled
62	OFU	Output FIFO Underflow. Indicates the AESU output FIFO was read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	—	Reserved

17.7.1.9 AESU ICV Size Register

The ICV size register, shown in Figure 17-28, is used in AES hashing modes CMAC and GCM to specify the number of most significant bytes in the received MAC tag supplied in context registers 3–4. AES truncates the computed MAC in context registers 1–2 to the same number of bytes, and writes zeros in the remaining LSB's. It follows that the received MAC can be padded to 16 bytes with arbitrary data (not necessarily zeros) when written into context registers 3–4. Acceptable values for ICV size are 8, 10, 12, 14 and 16 bytes in CMAC, or 8, 12, and 16 bytes in GCM. All other sizes are interpreted as 16.

In XCBC-MAC cipher mode, the ICV size register is not used. The received MAC (written to context registers 9-10) is always truncated to the most significant 12 bytes, as defined in the XCBC-MAC-96 for IPsec specification. The computed MAC written at the end of processing to Context Registers 1-2 is a full 16-byte MAC.

In CCM mode with ICV, the ICV size register is not used. Instead, the tag size is encoded within one of the CCM formatting flags.

Offset 0x3_4040 Access: Read/Write

	0–56	57–63
R W	—	ICV Size
Reset	All zeros	

Figure 17-28. AESU ICV Size Register

17.7.1.10 AESU End of Message Register

The AESU end of message register, shown in Figure 17-29, is used to signal to the AESU that the final message block has been written to the input FIFO (in channel-driven access, this signaling is done automatically). The AESU will not process the last block of data in its input FIFO until this register is written. Once the end of message register is written, the AESU processes any remaining data in the input FIFO and generates the done interrupt.

The value written to this register does not matter: ordinarily, zero is written. A read of this register always returns a zero value.

Offset 0x3_4050 — Access:Write only

	0–63
R	
W	AESU End of Message
Reset	All zeros

Figure 17-29. AESU End of Message Register

17.7.1.11 AESU Context Registers

There are twelve 64-bit context data registers that allow the host to read/write the contents of the context used to process a message. The context must be written prior to the key data. If the context registers are written during message processing, a context error is generated. All context registers are cleared when an initialization or a hard or soft reset is performed.

If a message is processed through the AESU in two separate operations (that is, using two descriptors), then the context must be read from the SEC at the end of the first operation and then restored at the beginning of the second operation.

Context is always read and restored as a contiguous subset of the twelve context registers ending with the highest numbered register used in that cipher mode. For example, when restoring context in CTR cipher mode (which uses context registers 5–7, as shown in Table 17-33), context registers 1–7 must be written (where registers 1–4 must be filled with zeros).

Context register assignments for cipher modes for confidentiality, data integrity, and combined confidentiality and integrity are described in the following subsections.

17.7.1.11.1 Context for Confidentiality Cipher Modes

The context registers for the different cipher modes which provide confidentiality only are summarized in Table 17-33. The registers are described in more detail in the following subsections.

Table 17-29. AESU Context Registers for Confidentiality Modes

<table>
<tr><th rowspan="2">Context Register (byte address)</th><th colspan="5">Confidentiality-only Cipher Mode</th></tr>
<tr><th>ECB</th><th>CBC / CBC-RBP / OFB / CFB128</th><th>CTR</th><th>LRW</th><th>SRT</th></tr>
<tr><td>1 (0x34100)</td><td rowspan="2">—</td><td rowspan="2">IV*</td><td rowspan="2">0</td><td rowspan="2">I*</td><td rowspan="2">Initial Counter Value*</td></tr>
<tr><td>2 (0x34108)</td></tr>
<tr><td>3 (0x34110)</td><td rowspan="2">—</td><td rowspan="2">—</td><td rowspan="2">0</td><td rowspan="2">Key 2</td><td>Counter Modulus*</td></tr>
<tr><td>4 (0x34118)</td><td>—</td></tr>
</table>

Table 17-29. AESU Context Registers for Confidentiality Modes (continued)

Context Register (byte address)	Confidentiality-only Cipher Mode				
	ECB	CBC / CBC-RBP / OFB / CFB128	CTR	LRW	SRT
5 (0x34120)	—	—	Initial Counter Value*	Tweak	—
6 (0x34128)					
7 (0x34130)	—	—	Counter Modulus Exponent*	—	—

Notes:
Context Registers 8 through 12 are not used for these modes
* Must be written at start of new message, except if zero
— don't care

Context for ECB Mode

ECB does not use any context registers.

Context for CBC, CBC-RBP, OFB, and CFB128 Cipher Modes

In CBC, CBC-RBP, OFB, and CFB128 cipher modes, the first two context data registers allow the host to read/write the contents of the initialization vector (IV) as follows:

- Context register 1 holds the least significant bytes of the initialization vector (bytes 1–8).
- Context register 2 holds the most significant bytes of the initialization vector (bytes 9–16).

The IV must be written prior to the message data. If the IV registers are written during message processing, or the mode is not set, a context error is generated.

The IV registers may only be read after processing has completed, as indicated by the assertion of the done interrupt (DI) bit in the AESU status register (see Section 17.7.1.6, "AESU Status Register"). If the IV registers are read prior to the assertion of DI, then an early read error is generated.

Context for Counter (CTR) Cipher Mode

In counter cipher mode, a random 128-bit initial counter value is incremented modulo 2^M with each block processed. The running counter is encrypted and XORed with the plaintext to derive the ciphertext, or with the ciphertext to recover the plaintext. The modulus exponent M can be set between 8 and 128, in multiples of 8.

As shown in Table 17-33, in CTR mode context registers 5–6 hold the initial counter value, and context register 7 holds the modulus exponent M .

Context for LRW Cipher Mode

Like GCM Cipher Mode, LRW requires Galois Field arithmetic computations. In the case of LRW, the GF multiplier built into the AESU mode logic computes a tweak value T for every block index I as follows:

T = Key2 x I.

The irreducible polynomial used (relative to AES data and key byte ordering) is .

Like ECB cipher mode, LRW does not use any historical value in computing the encryption of any block. However LRW does XOR plaintext with the tweak value T prior to AES encryption, and the AES encryption result is again XORed with the same tweak value in order to generate actual ciphertext.

LRW cipher mode uses context registers 1-2 for the Index (I), registers 3-4 for the Key 2, and registers 5-6 for the Tweak (T). The contents of context registers will be replaced with the running T and I as the calculation proceeds.

Context for SRT Cipher Mode

As mentioned in the footnote to Table 17-23, SRT is not a new AES cipher mode but rather an AESU method of performing AES-CTR cipher mode with reduced context loading overhead specifically for performing SRTP. As with CTR cipher mode, a random 128-bit initial counter value is incremented modulo 2^M with each block processed. The running counter is encrypted and XORed with the plaintext to derive the ciphertext, or with the ciphertext to recover the plaintext. The modulus exponent M can be set between 8 and 128 in multiples of 8.

As shown in Table 17-33, in SRT mode context registers 1–2 hold the initial counter value, and context register 3 holds the modulus exponent M .

17.7.1.11.2 Context for Data Integrity Cipher Modes

The context registers for the different cipher modes which provide data integrity only are summarized in Table 17-30. The registers are described in more detail in the following subsections.

Table 17-30. AESU Context Registers for Integrity Modes

Context Register # (byte address)	Cipher Mode providing only Data Integrity		
	XCBC-MAC	GCM-GHASH	CMAC (OMAC1)
1 (0x34100)	Computed MAC	Computed MAC	Computed MAC
2 (0x34108)			
3 (0x34110)	Received MAC*		Received MAC*
4 (0x34118)			
5 (0x34120)	Key 1		E(K, 0^{128})
6 (0x34128)			
7 (0x34130)	Key 2	len(AAD)^T	
8 (0x34138)			
9 (0x34140)	Key 3	H	
10 (0x34148)			

Table 17-30. AESU Context Registers for Integrity Modes (continued)

Context Register # (byte address)	Cipher Mode providing only Data Integrity		
	XCBC-MAC	GCM-GHASH	CMAC (OMAC1)
11 (0x34150)		len(AAD)[C]	

Notes:
Context register 12 is unused for these modes
* Used only in ICV mode—must be written at start of new message for ICV checking
[C] Length of data processed with current descriptor (in bits)
[T] Length of total data (in bits)

Context and Operation for XCBC-MAC Cipher Mode

XCBC-MAC cipher mode is an authentication-only mode of AES. Normal CBC-MAC runs AES in CBC cipher mode and assigns the final ciphertext result as the MAC. XCBC-MAC supports only 16-byte keys and extends the normal CBC-MAC as follows:

1. 3 keys are precomputed
   a. K1 = AES-Encrypt (K, $\{0x01\}^{16}$). [1]
   b. K2 = AES-Encrypt (K, $\{0x02\}^{16}$).
   c. K3 = AES-Encrypt (K, $\{0x03\}^{16}$).
2. Compute C_{n-1} = AES-CBC (P_1, 0, K1) ... AES-CBC (P_{n-1}, C_{n-2}, K1)
3. If $|P_n|$ = block size (128 bits)
   then: MAC = AES-CBC ($P_n \oplus K2$, C_{n-1}, K1)
   else: MAC = AES-CBC ($(P_n || 10^i) \oplus K3$, C_{n-1}, K1)

In XCBC-MAC cipher mode, AUX0=1 means that the final data block is not XORed with K2 or K3, so that message processing can be interrupted and later continued after a context switch. AUX1=1 disables computation of keys K1, K2, and K3, and instead expects these keys to be placed in key registers 5–6 (K1), context registers 7–8 (K2) and 9–10 (K3). If AUX1=0, computed keys are placed into context registers 5–10. AUX2=1 enables XCBC-MAC with ICV. In XCBC-MAC with ICV, the received MAC is supplied in context registers 3–4 and compared to the computed MAC in context registers 1–2.

Operation of the AESU in XCBC-MAC cipher mode requires the following steps (note these steps are performed automatically in channel-driven access):

1. Reset
2. Program the mode register as follows:
   a. Set the cipher mode to XCBC-MAC (encode/decode bit is ignored)
   b. Set AUX0 = 1 if processing of the message is going to be interrupted and later continued after a context switch. Set AUX0 = 0 if this is the last (or only) part of the message so that the final MAC can be generated.

1.Notation: $\{01\}^{16}$ means the byte 0x01 repeated 16 times

c. Set AUX1 = 1 if keys K1, K2 and K3 are loaded to key registers 5–6, context registers 7–8 and 9–10, respectively. Otherwise, set AUX1=0, and put K into key registers 1–2, so that keys K1, K2, and K3 can be computed and written to context registers 5–6, 7–8, and 9–10, respectively.

d. Set AUX2 = 1 if using XCBC-MAC with ICV.

3. Load Key (K if AUX1=0; K1 if AUX1=1)
4. Load Context
5. Set Key size
6. Set data size
7. While available:
   a. Load data blocks
8. Write to the end of message register
9. Read MAC from context registers 1-2
10. For XCBC-MAC with ICV, check ICCR bits in the status register

Context and Operation for GCM-GHASH Cipher Mode

GCM-GHASH denotes the authentication part of GCM cipher mode, and is described in Section 17.7.1.11.3, "Context for Confidentiality and Data Integrity Cipher Modes".

Context and Operation for CMAC (OMAC1) Cipher Mode

CMAC cipher mode is an authentication-only mode of AES. CMAC may be specified using the following notation:

- E(K,L) denotes the AES-encrypt function;
- xtime(L) is defined as follows, where L is a 128-bit vector with L[127] as most significant bit:
  — If L[127]=0, then xtime(L)=L<<1 (where '<<' denotes bitwise left shift)
  — Else xtime(L) = (L<<1) XOR 0x87.

Using this notation, the specification of CMAC is as follows:

1. Two keys are precomputed as follows:
   a. $K1 = xtime(E(K, \{0\}^{128}))$.[1]
   b. K2 = xtime(K1).
2. Compute C_{n-1} = AES-CBC (P_1, 0, K) ... AES-CBC (P_{n-1}, C_{n-2}, K)
3. If $|P_n|$ = block size (128 bits)
   then: MAC = AES-CBC ($P_n \oplus K1$, C_{n-1}, K)
   else: MAC = AES-CBC ($(P_n || 10^i) \oplus K2$, C_{n-1}, K)

1.Notation: $\{0\}^{128}$ means the bit 0 repeated 128 times

In CMAC cipher mode the received MAC is placed in context registers 3–4, and the computed MAC is put in registers 1–2 if AUX0=1. Context registers 5-6 are used to provide E(K, $\{0\}^{128}$) if AUX1=1, so that K1 and K2 can be computed after context switch without a time penalty. The computed value of E(K, $\{0\}^{128}$) is always stored in context registers 5-6 to be available for saving context in case of context switching.

Operation of the AESU in CMAC cipher mode requires the following steps (note these steps are performed automatically in channel-driven access):

1. Reset
2. Program the AESU mode register as follows:
   a. Set cipher mode to CMAC (encode/decode bit is ignored)
   b. Set AUX0 = 1 if processing of the message is going to be interrupted and later continued after a context switch. Set AUX0 = 0 if this is the last (or only) part of the message so that the final MAC can be generated.
   c. Set AUX1 = 1 for keys K1 and K2 to be derived from E(K, $\{0\}^{128}$) that is loaded into context registers 5–6. Otherwise, set AUX1=0, and CMAC computes E(K, $\{0\}^{128}$).
   d. Set AUX2 = 1 if using CMAC with ICV.
3. Load key
4. Load context
5. Set key size
6. Set ICV size for computed/received MAC (8, 10, 12, 14 or 16 bytes, default is 16)—ignored if AUX0 = 1
7. Set data size
8. While available:
   a. Load data blocks
9. Write to the end of message register
10. Read MAC from context registers 1-2
11. For CMAC with ICV, check ICCR bits in the status register

17.7.1.11.3 Context for Confidentiality and Data Integrity Cipher Modes

The context registers for the different cipher modes which provide both confidentiality and data integrity are summarized in Table 17-31. The registers are described in more detail in the following subsections.

Table 17-31. AESU Context Registers for Modes Providing Confidentiality and Integrity

Context Register # (byte address)	Cipher Mode providing Confidentiality and Integrity	
	CCM	GCM
1 (0x34100)	IV* / MAC	Computed MAC
2 (0x34108)		

Table 17-31. AESU Context Registers for Modes Providing Confidentiality and Integrity (continued)

Context Register # (byte address)	Cipher Mode providing Confidentiality and Integrity	
	CCM	GCM
3 (0x34110)	Encrypted MAC** / Decrypted MAC / Encrypted Counter	Received MAC**
4 (0x34118)		
5 (0x34120)	Counter*	Counter
6 (0x34128)		
7 (0x34130)	Counter Modulus Exponent* (header size/ MAC size)***	len(AAD)^T
8 (0x34138)	—	len(IV)^T
9 (0x34140)	—	Y_0
10 (0x34148)		
11 (0x34150)	—	len(AAD)^C
12 (0x34158)		len(IV)^C

* Must be written at start of new message, except if zero
** Needed only in ICV mode—must be written at start of new CCM decryption
*** The Header and MAC Sizes are internally constructed by the AES engine; then, that information is included inside context register 7 for context switching purposes
^{C} length of data processed with current descriptor (in bits)
^{T} length of total data (in bits)
— don't care

Context for CCM Cipher Mode

The SEC AESU is capable of performing single pass encryption and MAC generation. The host is required to order the CCM context in such a way that the context can be fetched as a contiguous string into the context registers, prior to encryption/MAC generation or decryption/MAC validation. The context register contents for CCM cipher mode is summarized in Figure 17-30 and further described below.

NOTE

AES-CCM mode does not support zero-length AAD and zero-length payload simultaneously. Either the AAD length or the payload length must be at least 1 byte.

		Context Registers						
		1	2	3	4	5	6	7
Encrypt (outbound)	Inputs	IV		0		Initial Counter		Counter Modulus Exponent
	Outputs	MAC	0	MIC	0			
Decrypt (inbound)	Inputs	IV		MIC	0	Initial Counter Value		Counter Modulus Exponent
	Outputs	Computed MAC	0	Decrypted MAC	0			

Figure 17-30. AESU CCM Context Registers

Context and Operation for CCM Encryption/MAC Generation

The context for CCM encryption/MAC generation (shown in Figure 17-30) is as follows:

- Registers 1–2 contain the session-specific 128-bit initialization vector (from memory)
- Registers 3–4 contain 128 bits of zero padding
- Registers 5–6 contain the session specific initial counter value (from memory)
- Register 7 contains the counter modulus exponent.

  Several current standards require a counter modulus exponent of 128 for CCM cipher mode. However, in order to support possible new standards the counter modulus exponent in AESU is a programmable field, which must be generated and stored along with other session-specific information for loading into the AESU context register prior to CCM encryption.

Using the session-specific key and context described above, operation of the AESU for CCM encryption/MAC generation requires the following steps (note these steps are performed automatically in channel-driven access):

1. Initialize the IV, and encrypt with the symmetric key.
2. In CBC fashion, take the output of step 1, hash with the first block of plaintext, and encrypt with the symmetric key.
3. Continue as in step 2 until the final block of plaintext has been processed. The result of the encryption of the final block of plaintext with the symmetric key is the MAC tag. The full 128 bits of MAC data is written to context registers 1–2, for use in the next phase of CCM processing. Once the MAC tag has been generated (step 3), the MAC tag along with the plaintext is encrypted with the AESU operating in counter cipher mode.
4. The first item to be encrypted in counter cipher mode is the counter (initial counter value) from context registers 5–6. The counter is encrypted with the symmetric key, and the result is hashed with the MAC tag (retrieved from context registers 1–2) to produce the MIC (encrypted MAC), which is then stored in context registers 3–4. At the completion of CCM encrypt processing, this MIC is output to memory (per the descriptor pointer) for the host to append to the 802.11i frame.

The AESU writes the full 128-bit MIC out to memory. The host must only append the most significant 64 bits to the frame as the MIC.

5. The counter value is incremented, then encrypted with the symmetric key. The result is hashed with the first block of plaintext to produce the first block of cipher text. The ciphertext is placed in the AESU output FIFO.
6. The counter continues to be incremented, and encrypted with the symmetric key, with the result hashed with each successive block of plaintext, until all plaintext has been converted to ciphertext. The SEC controller manages FIFO reads and writes, fetching plaintext and writing ciphertext per the pointers provided in the descriptor. When all ciphertext and the MIC has been output, the CCM encrypt operation is complete.

Context and Operation for CCM Decryption/MAC Regeneration

The context for CCM decryption/MAC regeneration (shown in Figure 17-30) is as follows:

- Registers 1–2 contain the session-specific 128-bit initialization vector (from memory)
- Registers 3–4 contain the MIC (from the received frame) plus 64 bits of zero padding
- Registers 5–6 contain the session-specific initial counter value (from memory)
- Register 7 contains the counter modulus exponent

  Several current standards require a counter modulus exponent of 128 for CCM cipher mode. However, in order to support possible new standards the counter modulus exponent in AESU is a programmable field, which must be generated and stored along with other session-specific information for loading into the AESU context register prior to CCM encryption.

Using the session-specific key and context described above, operation of the AESU for CCM decryption and MAC regenerationrequires the following steps (note these steps are performed automatically in channel-driven access):

1. Initialize the IV, and encrypt with the symmetric key. Simultaneously, the counter (Initial Counter Value) from Context Registers 5-6 is encrypted with the symmetric key. The result is hashed with the encrypted MAC (from Context Register 3-4), and the resulting original MAC is written to Context Reg 3-4, overwriting the encrypted MAC.

   Strictly speaking, the counter is encrypted with the symmetric key: however the AESU should be set for “decrypt” to perform the counter and CBC processes in the correct order.
2. The 802.11 frame header is hashed with the encrypted IV. (The AESU automatically determines the header length.) Simultaneously, the counter is incremented, and is then encrypted with the symmetric key. The result is then hashed with the first block of ciphertext to produce the first block of plaintext. The plaintext is placed in the AESU output FIFO, while simultaneously, in CBC fashion, a copy of the first block of plaintext is hashed with the output of encryption of the 802.11 frame header. The output is encrypted with the symmetric key.
3. As each ciphertext block is converted to plaintext, the plaintext is CBC encrypted. When the final plaintext block has been processed, the CBC MAC (MAC tag) is written to context registers 1–2. The first 64 bits of the MAC tag are compared to the MAC tag recovered in step 1.

NOTE

For both encrypt and decrypt operations, if the 802.11 frame is being processed as a whole (not split across multiple descriptors), the “Initialize” (AUX1) and “Final MAC” (AUX2) bits should be set in the AESU mode register.

Options and Operation for GCM Cipher Mode

Galois counter mode (GCM) uses AES counter mode to achieve data confidentiality. Authentication is achieved by computing a GHASH message authentication code (GMAC) through performing repetitive multiplication-accumulate functions in a Galois field.

Normally, the initialization vector (which is provided through the input FIFO) is 96 bits. If it is 96 bits, then the initialization vector (IV) is padded with the value ($\{0\}^{31}1$)[1]; otherwise the IV is hashed using the GHASH (H, {}, IV) function, where H represents E($\{0\}^{128}$, K), E stands for encryption operation, and K represents the key used. The resulting value Y_0 (the padded IV or the GHASHed IV) is provided as the initial counter value to counter mode AES. The result of encrypting Y_0 is denoted E(Y_0, K), and is used to generate the final MAC tag.

Data is encrypted or decrypted by XORing input data with the pseudorandom key stream generated by counter mode AES, starting with the *second* pseudorandom key block. The initial counter value Y_0 is incremented modulo 2^{32}.

GCM cipher mode can optionally be used to perform only the authentication part (GHASH (H, AAD, ciphertext), where ‘AAD’ denotes ‘additional authenticated data’): this special sub-mode is called GCM-GHASH in this document. GCM-GHASH is implemented by setting AUX0 and specifying the appropriate encryption operation. The format of the context registers for GCM-GHASH mode is shown in Table 17-36.

GCM cipher mode also has option of automatically verifying that the received and computed MAC tags are identical. This cipher mode is called GCM with ICV and can be specified by setting AESU mode register bits 56, 57 and 62 to 1, and bit 61 to 0. GCM with ICV context format is shown in Table 17-35.

Messages (IV+AAD+text data) are fed in through the input FIFO, and are always processed in the following order: IV, AAD, text data, followed by the final MAC computation (where “text data” refers to plaintext or ciphertext to be operated on). The whole message, however, does not have to be processed in one GCM execution. It can be split and processed with multiple descriptors in multiple GCM runs separated by resets of the AESU block. The boundaries can be set at the end of any full block (16 bytes) of the stream IV+AAD+text data. Hence, any of the individual components (IV, AAD, or text data) can be split into multiple descriptors. Refer to Table 17-32 for proper AUX mode specification in this case and to Table 17-33 through Table 17-36 for proper context formatting under the different GCM options (encrypt, decrypt, GCM with ICV, or GCM-GHASH). It should be noted that in case of a late arrival of the MAC tag on the receiving side, the final MAC can be computed and verified against the received MAC in a separate descriptor after the rest of the message (IV+AAD+text data) has already been processed.

1. Notation: $\{0\}^{31}1$ is defined to mean a string of thirty-one bits of 0 followed by a single bit of 1.

Operation of the AESU in GCM cipher mode requires the following steps (note these steps are performed automatically in channel-driven access):

1. Reset.
2. Set cipher mode to GCM or GCM with ICV and specify encrypt, decrypt in the AESU mode register. To perform GCM-GHASH (only GHASH (H, AAD, ciphertext) is computed) set AUX0 and specify encrypt. Set AUX2 and AUX1 bits according to Table 17-32.
3. Load key
4. Load (restore) context as needed (see Table 17-33 to Table 17-36).
5. Set key size
6. Set the size of the computed/received MAC (8, 12 or 16 bytes, default is 16)
7. Set data size
8. While available:
   a. Load IV into the input FIFO (1 or multiple blocks up to 2^{64} bits in total)
   b. Load AAD into the input FIFO (0 or multiple blocks up to 2^{64} bits in total)
   c. Load plaintext (for encryption) or ciphertext (for decryption) blocks into the input FIFO
   d. Unload ciphertext (for encryption) or plaintext (for decryption) blocks from the output FIFO
9. Write to the end of message register
10. Unload final ciphertext (for encryption) or plaintext (for decryption) blocks
11. Read (Save) context registers if another segment of the message is processed later
12. Read final GCM MAC from context registers 1-2, if AUX2 bit was set in mode register
13. For GCM with ICV, check ICCR bits in the AESU status register

AESU Mode Register Auxiliary Bit Settings for GCM Cipher Modes

Table 17-32 shows the significance of the AUX bits (bits 58–60) in the AESU mode register, under different operating conditions.

Table 17-32. GCM Cipher Mode Auxiliary Bit Definitions

Auxiliary Bit	Definitions	
	0	1
AUX2 (bit 58)	Do not compute MAC	Compute MAC
AUX1 (bit 59)	One of the following cases: Descriptor contains the whole message (IV+AAD+text data) Descriptor contains the whole IV and no or part of AAD or text data Descriptor contains a non-final part of IV, AAD, text data (IV, AAD or text data split between descriptors) Descriptor contains the final part of AAD or text data but no MAC is computed	One of the following cases: Descriptor contains the final part of IV (IV split between descriptors)—len(IV)^T needed Descriptor contains the final part of text data and the final MAC is computed (AUX2=1) (text data split between descriptors)—len(AAD)^T, len(text data)^T needed Descriptor contains the whole text data but no or part of AAD and the final MAC is computed—len(AAD)^T, len(text data)^T needed Descriptor contains the final part of AAD and the final MAC is computed—len(AAD)^T, len(text data)^T needed Descriptor computes only MAC (based on restored context) but does not contain either IV, AAD or text data —len(AAD)^T, len(text data)^T needed
AUX0 (bit 60) and Encrypt	--	GHASH-only mode
AUX0 (bit 60) and Decrypt	The key is to be unrolled	The key is already unrolled

AUX0 has different use depending on whether encryption or decryption is specified. For decryption, it determines whether the provided key should be first unrolled before processing starts, while in case of encryption it should generally be set to 0 unless GCM-GHASH cipher mode is desired. AUX2 determines whether the final MAC tag is to be computed or not. If AUX2 is set to 1, $E(K, Y_0)$ and the last iteration of the GHASH(H, AAD, ciphertext) is going to be performed and then XORed to give the MAC tag. Hence, if the message is split into multiple descriptors, only the last one should have AUX2=1 for proper MAC tag computation. AUX1 is used to resolve the issues related to the splitting of messages into multiple descriptors. Table 17-32 shows the proper settings of AUX1 for several scenarios of message splitting. In general, whenever the final GHASH iteration needs to be computed (either for GHASH(H, {}, IV) or GHASH(H, AAD, ciphertext)), and the current length is not equal to total length for either IV, AAD, or text data, then AUX1 should be set to 1. Consequently, an AUX1 value of 1 also indicates that the context registers 9-10 need to provide the total length of IV, AAD, or text data for this to be accomplished.

Context for GCM Cipher Modes

Table 17-33 to Table 17-36 describe the proper usage of context registers in case of encryption, decryption, GCM with ICV, and GCM-GHASH cipher mode settings, respectively. The context is in each case described in terms of the input context required for starting new GCM processing or continuation of processing after context switch, and in terms of the results (output) stored in context registers after GCM execution run is completed. The tables are followed by verbal descriptions of the different registers for the different options.

Table 17-33. GCM Encryption Context

Context Register	GCM Encrypt (Outbound)				
	Mode Register (ECM = 10, AUX0 = 0, CM = 01, ED = 1)				
	AUX1 Value			AUX2 Value	
	Inputs			Outputs	
	AUX1 = 0	AUX1 = 1		AUX2 = 0	AUX2 = 1
		last AAD or text data segment, or MAC only	last IV segment		
1	MAC (Computed)			—	MAC (Computed)
2					
3	—			—	
4					
5	Y_i (Counter)			—	
6					
7	—	len(AAD)^{T*}	—	—	
8	—	len(text data)^{T*}	len(IV)^{T}	—	
9	Y_0 (Initial Counter)			—	
10					
11	len(AAD)^{C*}			—	
12	len(IV)^{C*}			—	

* Must be written at the start of a new message, except if zero

^{C} length of data processed with current descriptor (in bits)

^{T} length of total data (in bits)

-- don't care

Table 17-34. GCM Decryption Context

Context Register	GCM Decrypt (Inbound)				
	Mode Register (ECM = 10, AUX0 = 0 or 1, CM = 01, ED = 0)				
	AUX1 Value			AUX2 Value	
	Inputs			Outputs	
		AUX1 = 1			
	AUX1 = 0	last AAD or text data segment, or MAC only	last IV segment	AUX2 = 0	AUX2 = 1
1	MAC (Computed)			—	
2					
3	—			—	MAC (Computed)
4					
5	Yi (Counter)			—	
6					
7	—	len(AAD)^{T*}	—	—	
8	—	len(text data)^{T*}	len(IV)^{T*}	—	
9	Y_0 (Initial Counter)			—	
10				—	
11	len(AAD)^{C*}			—	
12	len(IV)^{C*}				

* Must be written at the start of a new message, except if zero

^{C} length of data processed with current descriptor (in bits)

^{T} length of total data (in bits)

-- don’t care

Table 17-35. GCM with ICV Context

Context Register	GCM with ICV (Inbound)				
	Mode Register (ECM = 11, AUX0 = 0 or 1, CM = 01, ED = 0)				
	AUX1 Value			AUX2 Value	
	Inputs			Outputs	
	AUX1 = 0	AUX1 = 1		AUX2 = 0	AUX2 = 1
		last AAD or text data segment, or MAC only	last IV segment		
1	MAC (Computed)			—	MAC (Computed and truncated to icv_size most significant bytes)
2					
3	MAC (Received)			—	
4					
5	Y_i (Counter)			—	
6					
7	—	len(AAD)^{T*}	—	—	
8	—	len(text data)^{T*}	len(IV)^{T}	—	
9	Y_0 (Initial Counter)			—	
10				—	
11	len(AAD)^{C*}			—	
12	len(IV)^{C*}				

* Must be written at the start of a new message, except if zero

C length of data processed with current descriptor (in bits)

T length of total data (in bits)

-- don't care

Table 17-36. GCM-GHASH Context

Context Register	GCM-GHASH (Only GHASH Computed)				
	Mode Register (ECM = 10, AUX0 = 10, CM = 01, ED = 1)				
	AUX1 Value			AUX2 Value	
	Inputs			Outputs	
	AUX1 = 0	AUX1 = 1: last AAD or text data segment, or MAC only	AUX1 = 1: last IV segment	AUX2 = 0	AUX2 = 1
1	MAC (Computed)			—	MAC (Computed)
2					
3	—			—	
4					
5	—			—	
6					
7	—	len(AAD)[T*]	—	—	
8	—	len(text data)[T*]	len(IV)[T]	—	
9	H[*]			—	
10					
11	len(AAD)[C]			—	
12	—			—	

* Must be written at the start of a new message, except if zero

C length of data processed with current descriptor (in bits)

T length of total data (in bits)

-- don’t care

The context registers may be described as follows:

- Registers 1–2 contain the intermediate MAC value. This needs to be provided only when switching context during additional authenticated data (AAD) and/or text data processing (AAD+text data stream split into multiple descriptors).

  On the output side, these registers contain either the intermediate MAC tag in case of context switching (requires AUX2=0) or the final MAC tag at the end of processing (if AUX2=1). If AUX2=0 on the last descriptor processing a particular message, then these registers contain the partially computed GHASH(H, AAD, ciphertext), where the last GHASH iteration is not computed.

In the case of GCM with ICV, the final MAC tag written here as the result of GCM processing is truncated to 8, 12, or 16 (no truncation) bytes as defined in ICV size register. Note that any size from 1 to 16 bytes can be specified in ICV size register but any value other than 8 or 12 automatically defaults to 16 bytes.

- Registers 3–4 contain the received MAC tag, in case of inbound processing using GCM with ICV. This can be a 8, 12 or 16-byte block as specified by the ICV size register.
- Registers 5–6 contain the counter value Y_i, which is required only if restoring the context to continue processing a message. Note that the same value read when saving context should be written to these registers when restoring the context, since it is automatically incremented after every processed block.

  In the case of GCM-GHASH, these registers are not used.
- Register 7 contains the total length of the additional authenticated data (AAD) in bits. This is the total AAD length irrespective of whether AAD is split in multiple descriptors. It is required when AUX1=1 and the current descriptor processes the last segment of AAD or text data. It is also required if the whole message is already processed and the current descriptor only computes the final MAC tag.
- Register 8 contains the total length of the plaintext/ciphertext or IV in bits. This is required only when AUX1=1 (see Table 17-32). If the current descriptor processes the last segment of the IV, then total IV length should be provided; otherwise, the total length of text data should be provided.
- Registers 9–10 contain the initial counter value Y_0. Normally, this value is a result of the IV stream processing and needs to be provided only if the message is split into multiple descriptors and for those descriptors that come after IV processing is complete. Otherwise, the value provided here is ignored and overwritten with computed Y_0.

  In case of GCM-GHASH cipher mode setting, the constant H from GHASH(H, AAD, ciphertext) should be provided in these registers. Note that in the general case this may not be to be equal to $E(K, \{0\}^{128})$ where K is a key as defined for GCM.
- Register 11 contains the length (in bits) of the AAD part processed in the current descriptor. If the current descriptor does not process AAD, then the register should be zero. If AAD is not split into multiple descriptors, then this field should contain the total AAD length. The value written here should be divisible by 128 for all AAD segments except for the last one, which can be any number of bits. Note, however, that the actual AAD stream supplied to the AES engine through the FIFOs has to be zero-padded to an integral number of 16-byte blocks.
- Register 12 contains the length (in bits) of the IV part processed in the current descriptor. Similar remarks apply for IV in register 12 as for AAD in register 11.

  In case of GCM-GHASH, this register is not used.

Example of Context in GCM Encryption

For illustrative purposes we consider the case of a GCM encrypt operation that generates the final MAC tag, where the whole message is small enough to be processed with one descriptor. AESU mode register bits 56-63 (ECM, AUX, CM, and ED) should be set to 10_100_01_1. Only context registers 11–12 must be written with the bit lengths of AAD and IV, respectively. The bit length of text data should be written to the data size register, up to a size of 2^{19} bits. IV, AAD, and text data in that order should be sent through the input FIFO and the result (text data) read from the output FIFO as available. At the end, the final MAC tag is read from context registers 1–2.

17.7.1.12 AESU Key Registers

The format of the AESU key registers is shown in Figure 17-31. These registers may hold 16, 24, or 32 bytes of key data, with the first 8 bytes of key data written to key 1.Any key data written to bytes beyond the key size (as specified in the key size register) is ignored. The key data registers are cleared when the AESU is reset or re-initialized. If these registers are modified during message processing, a context error is generated.

The key registers may be read when changing context in decrypt mode. To resume processing, the value read must be written back to the key registers and the "restore decrypt key" bit must be set in the mode register. This eliminates the overhead of expanding the key prior to starting decryption when switching context.

	0–63	
Field	Key 1U Register	Key 1U
Reset	0	
R/W	R/W	
Addr	AESU 0x3_4400	

Field	Key 1L Register	Key 1L
Reset	0	
R/W	R/W	
Addr	AESU 0x3_4408	

Field	Key 2U Register	Key 2U
Reset	0	
R/W	R/W	
Addr	AESU 0x3_4410	

Field	Key 2L Register	Key 2L
Reset	0	
R/W	R/W	
Addr	AESU 0x3_4418	

Figure 17-31. AESU Key Registers

17.7.1.12.1 AESU FIFOs

AESU uses an input FIFO/output FIFO pair to hold data before and after the encryption process. Normally, the channels control all access to these FIFOs. For host-controlled operation, a write to anywhere in the AESU FIFO address space enqueues data to the AESU input FIFO, and a read from anywhere in the AESU FIFO address space dequeues data from the AESU output FIFO.

Writes to the input FIFO go first to a staging register which can be written by byte, word (4 bytes), or dword (8 bytes). When all 8 bytes of the staging register have been written, the entire dword is automatically enqueued into the FIFO. If any byte is written twice between enqueues, it causes an error interrupt of type AE from the EU. When writing the last portion of data, it is not necessary to write all 8 bytes. Any last bytes remaining in the staging register are automatically padded with zeros and forced into the input FIFO when the AESU end of message register is written.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword have been read, that dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between dequeues, it causes an error interrupt of type AE from the EU.

Overflows and underflows caused by reading or writing the AESU FIFOs are reflected in the AESU interrupt status register.

The AESU fetches data 128 bits at a time from the input FIFO. During processing, the input data is encrypted or decrypted and the results are placed in the output FIFO. The output size is the same as the input size.

The input FIFO may be written any time the number of dwords currently in the input FIFO (as indicated by the IFL field of the AESU status register) is less than 32. There is no limit on the total number of bytes in a message. The number of bits in the final message block must be set in the data size register.

The output FIFO may be read any time the OFR signal is asserted (as indicated in the AESU status register). This indicates that the number of bytes in the output FIFO is at or above the threshold specified in the mode register.

17.7.2 ARC4 Execution Unit (AFEU)

This section contains details about the ARC4 execution unit (AFEU), including modes of operation, status and control registers, S-box memory, and FIFOs.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the AFEU is used through channel-controlled access, which means that most reads and writes of AFEU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

17.7.2.1 AFEU Mode Register

As shown in Figure 17-32, the AFEU mode register contains three bits which are used to program the AFEU. The mode register is cleared when the AFEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error is generated.

Offset 0x3_8000 — Access: Read/Write

	0–60	61	62	63
R/W	—	CS	DC	PP
Reset	All zeros			

Figure 17-32. AFEU Mode Register

Table 17-37 describes AFEU Mode Register fields.

Table 17-37. AFEU Mode Register Field Descriptions

Bits	Name	Description
The following bits are described for information only. They are not under direct user control.		
0–55	—	Reserved
The following bits are controlled through the MODE0 field of the descriptor header.		
56–60	—	Reserved
61	CS	Context Source. If set, this causes the context to be moved from the input FIFO into the S-box prior to starting encryption/decryption. Otherwise, context should be directly written to the context registers or context should be generated automatically through key permutation. Context source is only checked if the prevent permute bit is set. 0 Context not from FIFO (written directly to context register addresses) 1 Context from input FIFO
62	DC	Dump Context. If set, this causes the context to be moved from the S-box to the output FIFO following assertion AFEU's done interrupt. 0 Do not dump context 1 After cipher, dump context
63	PP	Prevent Permute. Normally, AFEU receives a key and uses that information to randomize the S-box. If reusing a context from a previous descriptor, this bit should be set to prevent AFEU from re-performing this permutation step. 0 Perform S-box permutation 1 Do not permute

17.7.2.2 AFEU Key Size Register

As displayed in Figure 17-33, this value indicates the number of bytes of key memory that should be used in performing S-box permutation. Any key data beyond the number of bytes in the key size register is ignored. This register is cleared when the AFEU is reset or re-initialized. If the key size specified is less than 1 or greater than 16, a key size error is generated. If the key size register is modified during processing, a context error is generated. Note: Although the AFEU supports key lengths as short as 1 byte, a 1 byte key offers little security. Most applications of ARC4 specify keys of 5-16 bytes.

Offset 0x3_8008 Access: Read/Write

	0–51	52–63
R	—	Key Size
W		
Reset	All zeros	

Figure 17-33. AFEU Key Size Register

NOTE

The device driver must create properly formatted descriptors for situations requiring a key permute prior to ciphering. When using host-controlled access (typically for debug), the user must set the AFEU mode register to perform 'permute with key', then write the key data to AFEU Key Registers, then write the key size to the key size register. The AFEU starts permuting the memory with the contents of the key registers immediately after the key size is written.

17.7.2.3 AFEU Context/Data Size Register

The AFEU context/data size register (shown in Figure 17-34), specifies the number of bits of context or data to be processed by the AFEU.

In channel-driven access, the necessary writes to this register are performed automatically, based on information contained in the descriptors.

In host-driven access, the correct order of operations for an AFEU operation with context loading is as follows:

1. Write the AFEU mode register, with 'Context Source' and 'Prevent Permute' set.
2. Write the 259 bytes of previously saved S-Box (256 bytes) and counters (3 bytes) to the AFEU input FIFO.
3. Write 2072 (bits) to the AFEU context/data size register
4. Begin writing the data to the AFEU Input FIFO. If the total data size is > 256 bytes, monitor the input FIFO level (IFL) in the AFEU Status Register to avoid overflowing the Input FIFO. Use the Output FIFO Level (OFL) to avoid underflowing the Output FIFO.
5. After writing the final data to the Input FIFO, write the data size (in bits) to the AFEU context/data size register. The data size written must be an integral number of bytes (bits 61:63 must be zero) or the AFEU will generate a data size error. The AFEU performs additional checking on bits 57:60 to determine the number of bytes of data from the final Input FIFO write to permute with the S-Box.

This register is cleared when the AFEU is reset or re-initialized, shown in Figure 17-34.

Offset 0x3_8010 Access: Read/Write

	0–51	52–63
R W	—	Data Size
Reset	All zeros	

Figure 17-34. AFEU Context/Data Size Register

17.7.2.4 AFEU Reset Control Register

This register, as shown in Figure 17-35, allows 3 levels reset that effect the AFEU only, as defined by 3 self-clearing bits. It should be noted that the AFEU executes an internal reset sequence for hardware reset, SW_RESET, or module initialization, which performs proper initialization of the S-box. To determine when this is complete, observe the RESET_DONE bit in the AFEU status register.

Offset 0x3_8018 Access: Read/Write

	0–60	61	62	63
R W	—	RI	MI	SR
Reset	All zeros			

Figure 17-35. AFEU Reset Control Register

Table 17-38 describes AFEU reset control register fields.

Table 17-38. AFEU Reset Control Register Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	RI	Reset Interrupt. Writing this bit active high causes AFEU interrupts signaling done and error to be reset. It further resets the state of the AFEU interrupt status register. 0 Do not reset 1 Reset interrupt logic
62	MI	Module initialization resets everything reset by SR, with the exception of the AFEU interrupt mask register. 0 Do not reset 1 Reset most of AFEU
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for AFEU. All registers and internal state are returned to their defined reset state. On negation of SW_RESET, the AFEU enters a routine to perform proper initialization of the S-box. 0 Do not reset 1 Full AFEU reset

17.7.2.5 AFEU Status Register

This status register, shown in Figure 17-36, reflect the state of AFEU internal signals.

The AFEU status register is read only. Writing to this location results in address error being reflected in the AFEU interrupt status register.

Offset 0x3_8028 Access: Read only

Bits	0–39	40–47	48–55	56–57	58	59–60	61	62	63
R		OFL	IFL	—	HALT	—	EI	DI	RD
W									
Reset	All zeros								

Figure 17-36. AFEU Status Register

Table 17-39 describes AFEU Status Register fields.

Table 17-39. AFEU Status Register Field Descriptions

Bits	Name	Description
0–39	—	Reserved
40-47	OFL	The number of dwords currently in the output FIFO
48-55	IFL	The number of dwords currently in the input FIFO
56-57	—	Reserved
58	HALT	Halt. Indicates that the AFEU has halted due to an error. 0 AFEU not halted 1 AFEU halted **Note:** Because the error causing the AFEU to stop operating may be masked before reaching the interrupt status register, the AFEU interrupt status register is used to provide a second source of information regarding errors preventing normal operation.
59–60	—	Reserved
61	EI	Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 AFEU is not signaling error 1 AFEU is signaling error
62	DI	Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 AFEU is not signaling done 1 AFEU is signaling done
63	RD	Reset Done. This status bit, when high, indicates that AFEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel. 0 Reset in progress 1 Reset done **Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation.

17.7.2.6 AFEU Interrupt Status Register

The interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the AFEU interrupt mask register is zero (see Section 17.7.2.7, “AFEU Interrupt Mask Register”).

If the AFEU interrupt status register is non-zero, the AFEU halts and the AFEU error interrupt signal is asserted to the controller (see Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). In addition, if the AFEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 17-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the AFEU reset control register.

The definition of each bit in the AFEU interrupt status register is shown in Figure 17-37.

Offset 0x3_8030 — Access: Read/Write

	0–50	51	52	53	54	55	56	57	58	59	60	61	62	63
R/W	—	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	—	IFO	OFU	—
Reset	All zeros													

Figure 17-37. AFEU Interrupt Status Register

Table 17-40 describes AFEU interrupt status register fields.

Table 17-40. AFEU Interrupt Status Register Fields

Bits	Names	Description
0-50	—	Reserved
51	IE	Internal Error. An internal processing error was detected while performing encryption. 0 No error detected 1 Internal error
52	ERE	Early Read Error. The AFEU context memory or control was read while the AFEU was performing encryption. 0 No error detected 1 Early read error
53	CE	Context Error. The AFEU mode register, key register, key size register, data size register, or context memory is modified while AFEU processes data. 0 No error detected 1 Context error
54	KSE	Key Size Error. A value outside the bounds 1–16 bytes was written to the AFEU key size register 0 No error detected 1 Key size error
55	DSE	Data Size Error. A value that is not a multiple of 8 bits was written to the AFEU data size register: 0 No error detected 1 Data size error
56	ME	Mode Error. An illegal value was detected in the mode register. Note: writing to reserved bits in mode register is likely source of error. 0 No error detected 1 Mode error

Table 17-40. AFEU Interrupt Status Register Fields (continued)

Bits	Names	Description
57	AE	Address Error. An illegal read or write address was detected within the AFEU address space. 0 No error detected 1 Address error
58	OFE	Output FIFO Error. The AFEU output FIFO was detected non-empty upon write of AFEU data size register. 0 No error detected 1 Output FIFO non-empty error
59	IFE	Input FIFO Error. The AFEU Input FIFO was detected non-empty upon generation of done interrupt 0 Input FIFO non-empty error enabled 1 Input FIFO non-empty error disabled
60	—	Reserved
61	IFO	Input FIFO Overflow. The AFEU input FIFO was pushed while full. 1 Input FIFO has overflowed 0 No error detected **Note:** When operated through channel-controlled access, the SEC implements flow control, which prevents input FIFO overflow—hence FIFO size is not a limit to data input in this case. When operated through host-controlled access, the AFEU cannot accept FIFO inputs larger than 256 bytes without overflowing.
62	OFU	Output FIFO Underflow. The AFEU output FIFO was read while empty. 0 No error detected 1 Output FIFO has underflow error
63	—	Reserved

17.7.2.7 AFEU Interrupt Mask Register

The interrupt mask register, shown in Figure 17-38, controls the result of detected errors. For a given error (as defined in Section 17.7.2.6, "AFEU Interrupt Status Register"), if the corresponding bit in this register is set, the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

Offset 0x3_8038 Access: Read/Write

	0–50	51	52	53	54	55	56	57	58	59	60	61	62	63
R W	—	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	—	IFO	OFU	—

Figure 17-38. AFEU Interrupt Mask Register

Table 17-41 describes AFEU interrupt mask register fields.

Table 17-41. AFEU Interrupt Mask Register

Bits	Names	Description
0–50	—	Reserved
51	IE	Internal Error. An internal processing error was detected while performing encryption. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early Read Error. The AFEU Register was read while the AFEU was performing encryption. 0 Early read error enabled 1 Early read error disabled
53	CE	Context Error. An AFEU key register, the key size register, data size register, mode register, or context memory was modified while AFEU was performing encryption. 0 Context error enabled 1 Context error disabled
54	KSE	Key Size Error. A value outside the bounds 1–16 bytes was written to the AFEU key size register 0 Key size error enabled 1 Key size error disabled
55	DSE	Data Size Error. An inconsistent value was written to the AFEU data size register: 0 Data Size error enabled 1 Data size error disabled
56	ME	Mode Error. An illegal value was detected in the mode register. 0 Mode error enabled 1 Mode error disabled
57	AE	Address Error. An illegal read or write address was detected within the AFEU address space. 0 Address error enabled 1 Address error disabled
58	OFE	Output FIFO Error. The AFEU Output FIFO was detected non-empty upon write of AFEU data size register 0 Output FIFO non-empty error enabled 1 Output FIFO non-empty error disabled
59	IFE	Input FIFO Error. The AFEU Input FIFO was detected non-empty upon generation of done interrupt. 0 Input FIFO non-empty error enabled 1 Input FIFO non-empty error disabled
60	—	Reserved
61	IFO	Input FIFO Overflow. The AFEU Input FIFO was pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled
62	OFU	Output FIFO Underflow. The AFEU Output FIFO was read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	—	Reserved

17.7.2.8 AFEU End of Message Register

The end of message register in the AFEU, displayed in Figure 17-39, is used to signal the AFEU that all data to be processed has been written to the input FIFO (in channel-driven access, this signaling is done automatically). Before this register is written, the AFEU does not process the last block of data in its input FIFO. Once the end of message register is written, the AESU processes any remaining data in the input FIFO and generates the done interrupt. If the DC(dump context) bit in the AFEU mode register is set, the context is written to the output FIFO following the last message word.

Any value written to the end of message register has the same effect. A read of the AFEU end of message register always returns a zero value.

Addr 0x3_8050		Access: Write only
	0	63
R		
W	AFEU End of Message	
Reset	All zeros	

Figure 17-39. AFEU End of Message Register

17.7.2.9 AFEU Context

An ARC4 encryption session begins with an initial key permutation to generate the initial S-box state. After that each subsequent message in the same session makes use of the S-box state and also modifies the S-box state.

To implement this behavior using the AFEU, the first descriptor of a session must perform a key permute operation. If there are additional messages in the same session, then at the end of each descriptor execution the S-box state must be dumped out as context so that the next message of the session can re-load that context.

AFEU context consists of two parts:

- AFEU context memory — a 256-byte SRAM that holds the current S-box contents
- AFEU context memory pointer register — holds the internal context pointers that are updated with each byte of message processed. These pointers correspond to the values of I, J, and Sbox[I+1] in the ARC4 algorithm.

There is no standard data format for S-box state information. To ensure proper AFEU operation, the AFEU context should only be written with data that was read from the AFEU context during a previous operation.

17.7.2.9.1 Writing AFEU Context

In the default mode of operation, the key and key size are provided to the AFEU. The initial memory values in the S-box are permuted with the key to create new S-box values, which are used to encrypt the plaintext.

If the “prevent permute” (PP) mode bit is set in the AFEU mode register (see Section 17.7.2.1, “AFEU Mode Register”), then the AFEU does not require a key, but instead requires context to set the S-box state before processing data. This mode is used to resume processing in an ARC4 session using a previously computed S-box. In this case, the steps in the processing are as follows:

1. Write the context to the AFEU through the context registers or through the input FIFO (as selected by the CS mode bit).
2. Write the context length to the context/data length register (see Section 17.7.2.3, “AFEU Context/Data Size Register”).
3. Write the message data size to the context/data register.
4. Write the message data.

If the context registers are written during message processing or hen the PP bit is not set, a context error is generated.

For more information about writing AFEU context through the input FIFO, see Section 17.7.2.10.1, “AFEU FIFOs.”

17.7.2.9.2 Reading AFEU Context

Once message processing is complete and the output data has been read, the AFEU S-box state can be read out either through the context registers or through the output FIFO, as selected by the “dump context” (DC) mode bit (see Section 17.7.2.1, “AFEU Mode Register”).

Valid context data can only be read after AFEU has completed processing (as indicated by the done interrupt, as reflected in the “DI” bit of the AFEU Status Register, Section 17.7.2.5, “AFEU Status Register”). Reading context data before the module is done generates an error interrupt.

For more information about reading AFEU context through the output FIFO, see Section 17.7.2.10.1, “AFEU FIFOs.”

17.7.2.10 AFEU Key Registers

AFEU uses two write-only key registers to seed the initial permutation of the AFEU S-box, in conjunction with the AFEU key size register. Any key data beyond the key size (specified in the key size register) is ignored. AFEU permutes starting with the first byte of key register 0, and uses as many bytes from the two key registers as necessary to complete the permutation. Reading either of these memory locations generates an address error interrupt.

17.7.2.10.1 AFEU FIFOs

AFEU uses an input FIFO/output FIFO pair to hold data before and after the ciphering process. Normally, the channels control all access to these FIFOs. For host-controlled operation, a write to anywhere in the AFEU FIFO address space enqueues data to the AFEU input FIFO, and a read from anywhere in the AFEU FIFO address space dequeues data from the AFEU output FIFO.

When context is written to the input FIFO (see Section 17.7.2.9.1, “Writing AFEU Context”), the first context write must be in the address range 3_8E00-3_8E07. Similarly, when context is read from the output FIFO (see Section 17.7.2.9.1, “Writing AFEU Context”), the first context read must be in the address range 3_8E00-3_8E07. This causes any incomplete data word remaining in the output FIFO to be cleared out so that the context can be read.

Writes to the input FIFO go first to a staging register which can be written by byte, word (4 bytes), or dword (8 bytes). When all 8 bytes of the staging register have been written, the entire dword is automatically enqueued into the FIFO. If any byte is written twice between enqueues, it causes an error interrupt of type AE from the EU. When writing the last portion of data, it is not necessary to write all 8 bytes. Any last bytes remaining in the staging register are automatically padded with zeros and forced into the input FIFO when the AFEU end of message register is written.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword have been read, that dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between dequeues, it causes an error interrupt of type AE from the EU.

Overflows and underflows caused by reading or writing the AFEU FIFOs are reflected in the AFEU interrupt status register.

17.7.3 Cyclical Redundancy Check Unit (CRCU)

This section contains details about the cyclical redundancy check unit (CRCU), including modes of operation, status and control registers, and FIFO.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the CRCU is used through channel-controlled access, which means that most reads and writes of CRCU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

17.7.3.1 ICV Checking in CRCU

This EU includes an ICV checking feature, that is, it can verify a message/CRC pair by calculating a raw CRC and comparing it to the polynomial specific residue. The pass/fail result of this check can be returned to the host either by interrupt by a writeback of EU status fields into host memory, but not by both methods at once.

To signal the ICV checking result by status writeback, turn on either the IWSE bit or AWSE bit in the channel configuration register (see Section 17.4.4.1, “Channel Configuration Register (CCR)”), and mask the CICV fail (CICVF) bit in the interrupt mask register (see Section 17.7.3.9, “CRCU Interrupt Mask Register”). In this case the normal done signaling (by interrupt or writeback) is undisturbed.

To signal the ICV checking result by interrupt, unmask the CICVF bit in the interrupt mask register and turn off the IWSE and AWSE bits in the channel configuration register. If there is no CRC mismatch, then the normal done signaling (by interrupt or writeback) occurs. When there is an CRC mismatch, there is an error interrupt to the host, but no done interrupt or writeback.

17.7.3.2 CRCU Mode Register

The mode register (shown in Figure 17-40) is used to program the function of the CRCU and is generally the first register written. A context error is generated if this register is written after processing has begun.

Addr 0x3_F000 Access: Read/Write

	0–55	56	57	58–61	62–63
R/W	—	RAW	CICV	—	ALG
Reset	All zeros				

Figure 17-40. CRCU Mode Register

Table 17-42 describes the CRCU mode register fields:

Table 17-42. CRCU Mode Register Field Definitions

Bits	Name	Description
0–55	—	Reserved
56	RAW	RAW Raw output. Controls whether CRC result bit manipulations are performed. 0 Perform bit manipulations. The CRC result is bit-swapped, byte-swapped, and then complemented before it is available in the Context Register. This is compliance with the IEEE 802 and iSCSI CRC implementations. 1 Do not perform bit manipulations. Provide raw CRC result in the Context Register.
57	CICV	ICV (CRC) Check. Selects whether a comparison between the computed CRC and the residue should be performed. 0 Disabled. 1 Enable ICV check. Compares a CRC calculated across the message and received ICV against the stored residue. The comparison is performed prior to the bit manipulations controlled by the RAW bit.
58–61	—	Reserved
63	ALG	Algorithm. Selects the CRC algorithm mode for the CRCU. 00 IEEE 802 mode. The CRC32 algorithm is performed using the polynomial 0x04C11DB7 and the residue 0xC704DD7B is used for ICV checking. 01 iSCSI mode. The CRC32c algorithm is performed using the polynomial 0x1EDC6F41 and the residue 0x1C2D19ED is used for ICV checking. 10 Static custom mode. The CRC remainder is computed using the Control Register bits 32-63 as the polynomial and bits 0-31 as the residue. 11 Dynamic custom mode. The CRC remainder is computed using the Key Register bits 32-63 as the polynomial and bits 0-31 as the residue.

17.7.3.3 CRCU Key Size Register

The key size register stores the number of valid bytes in the dynamic polynomial written to the key register (Section 17.7.3.13, "CRCU Key Register"). The reset value is zero, and an illegal key size error is generated if a value other than zero, four or eight is written to the register. It is not necessary to write to this register, as the polynomial size is clearly fixed by the algorithm. A context error is generated if this register is written after processing has begun.

Offset 0x3_F008 Access: Read/Write

	0–51	52–63
R W	—	Key Siz
Reset	All zeros	

Figure 17-41. CRCU Key Size Register

17.7.3.4 CRCU Data Size Register

The data size register is written with the number of bits of data to be processed. Writing to this register puts the CRCU module into a busy state and starts data processing. This register may be written multiple times while data processing is in progress. The actual values written are ignored, although an error is generated if the value is not a multiple of 8 bits.

Offset 0x3_F010 Access: Read/Write

	0–51	52–63
R W	—	Data Size
Reset	All zeros	

Figure 17-42. CRCU Data Size Register

17.7.3.5 CRCU Reset Control Register

The reset control register controls the reset/re-initialization of the block, as shown in Figure 17-43.

Offset 0x3_F018 Access: Read/Write

	0–60	61	62	63
R W		RI	MI	SR
Reset	All zeros			

Figure 17-43. CRCU Reset Control Register

Table 17-43 describes CRCU reset control register fields.

Table 17-43. CRCU Reset Control Register Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	RI	Reset Interrupt. Writing this bit active high causes CRCU interrupts signaling done and error to be reset. It further resets the state of the CRCU interrupt status register. 0 Do not reset 1 Reset interrupt logic

Table 17-43. CRCU Reset Control Register Field Descriptions (continued)

Bits	Name	Description
62	MI	Module initialization resets everything reset by SR, with the exception of the CRCU interrupt mask register. 0 Do not reset 1 Reset most of CRCU
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for CRCU. All registers and internal state are returned to their defined reset state. On negation of SW_RESET, the CRCU enters a routine to perform proper initialization of the S-box. 0 Do not reset 1 Full CRCU reset

17.7.3.6 CRCU Control Register

The CRCU control register stores the coefficients of the residue and static polynomial used in custom CRC computations. Figure 17-44 shows the bit position of each coefficient.

Offset 0x020 Access: Read/Write

	0	1	2	3	4	5	6	7	8–23	24	25	26	27	28	29	30	31
R W	r^{31}	r^{30}	r^{29}	r^{28}	r^{27}	r^{26}	r^{25}	r^{24}	—	r^7	r^6	r^5	r^4	r^3	r^2	r^1	r^0

	32	33	34	35	36	37	38	39	40–55	56	57	58	59	60	61	62	63
R W	g^{31}	g^{30}	g^{29}	g^{28}	g^{27}	g^{26}	g^{25}	g^{24}	—	g^7	g^6	g^5	g^4	g^3	g^2	g^1	g^0

Figure 17-44. CRCU Control Register

In Figure 17-44 r^n (respectively g^n) represents the *n*'th residue (respectively polynomial) coefficient. The reset value of this register corresponds to the IEEE 802 CRC32 residue and polynomial coefficients. This register is static in that it is only reset by performing a software reset, and not by an EU reinitialization. This allows a platform-specific custom polynomial to be written to the register once and used many times. A context error is generated if this register is written after processing has begun. A polynomial error is generated if a value is written to this register which does not have a one in bit 0 (representing g^0).

17.7.3.7 CRCU Status Register

The CRCU status register provides general information on the status of the CRCU, as shown in Figure 17-45. A read of the status register captures a snapshot of CRCU's operating state at a particular moment in time. Of notable interest to the user are the three interrupt flags (error interrupt, done interrupt, and reset done). Writes to this register are ignored.

Offset 0x3_F028 Access: Read only

	0–47	48–55	56–57	58	59–60	61	62	63
R		IFL	—	HALT	ICCR	EI	DI	RD
W								
Reset	All zeros							

Figure 17-45. CRCU Status Register

Table 17-44 shows the CRCU status register bit definitions.

Table 17-44. CRCU Status Register Bit Definitions

Bits	Field Name	Description
0–47	—	Reserved
48–55	IFL	Input FIFO Level: The number of dwords currently in the input FIFO
56–57	—	Reserved
58	HALT	Indicates when the CRCU core has halted due to an error. 0 CRCU not halted 1 CRCU core halted (Must be reset/re-initialized) **Note:** Because the error causing the CRCU to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation.
59–60	ICCR	Integrity Check Comparison Result 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved **Note:** A passed or failed result is generated only if ICV checking is enabled and the algorithm selected is f9.
61	EI	Error interrupt.Reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 CRCU is not signaling error 1 CRCU is signaling error
62	DI	Done Interrupt: Reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 Processing not done 1 All bytes processed
63	RD	Reset Done: Indicates when the CRCU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel. 0 Reset in progress 1 Reset done

17.7.3.8 CRCU Interrupt Status Register

The CRCU interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the CRCU interrupt mask register is zero (see Section 17.7.3.9, “CRCU Interrupt Mask Register”). If a CRCU interrupt mask register bit is set, the corresponding CRCU interrupt status bit is always zero regardless of the error status.

If the CRCU interrupt status register is non-zero, then the CRCU halts and the CRCU error interrupt signal is asserted to the controller (see Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the CRCU is being operated in channel-driven mode, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 17-15) and generates a channel error interrupt to the controller.

Interrupt status register bits can be set by writes from the host, but only if the corresponding bit is cleared in the interrupt mask register. Bits masked by the interrupt mask register bits are always zero.

The CRCU interrupt status and interrupt mask registers can be cleared by programming the CRCU reset control register, as described in Section 17.7.3.5, "CRCU Reset Control Register."

Offset 0x3_F030 — Access: Read/Write

	0–48	49	50	51	52	53	54	55	56	57	58–60	61	62–63
R/W	—	ICE	PE	IE	ERE	CE	KSE	DSE	ME	AE	—	IFO	—
Reset	All zeros												

Figure 17-46. CRCU Interrupt Status Register

Table 17-45. CRCU Interrupt Status Register Bit Definitions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity Check (CRC) Error: 0 No error detected 1 Integrity check error detected. An ICV (CRC) check was performed and the supplied ICV (CRC) did not match the one computed by the CRCU.
50	PE	Polynomial Error. 0 No error. 1 An invalid polynomial has been written to the Key or Control Register.
51	IE	Internal Error. Indicates the CRCU has been locked up and requires a reset before use. 0 No internal error detected 1 Internal error detected **Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by resetting the CRCU.
52	ERE	Early Read Error. The CRCU context was read before the CRCU completed the requested operation. 0 No error detected 1 Early read error
53	CE	Context Error. The CRCU mode, key size, control, context, or key register was modified after processing had begun. 0 No error detected 1 Context error
54	KSE	Key Size Error. A value other than 0 or 4 has been written to the CRCU key size register. 0 No error detected 1 Key size error
55	DSE	Data Size Error. An non-byte-multiple size has been written to the CRCU data size register. 0 No error detected 1 Data size error

Table 17-45. CRCU Interrupt Status Register Bit Definitions (continued)

Bits	Name	Description
56	ME	Mode Error. Bit is set if any of these error conditions is detected: • any reserved bit of the Mode register is set • the ALG field of the Mode register contains an illegal value 0 No error detected 1 Mode error
57	AE	Address Error. An illegal read or write address was detected within the CRCU address space. 0 No error detected 1 Address Error
58–60	—	Reserved
61	IFO	Input FIFO Overflow. The CRCU Input FIFO was pushed while full. 0 No overflow detected 1 Input FIFO has overflowed **Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input size. When operated through host-controlled access, the CRCU cannot accept FIFO inputs larger than 256 bytes without overflowing.
62–63	—	Reserved

17.7.3.9 CRCU Interrupt Mask Register

The CRCU interrupt mask register, controls the setting of bits in the CRCU interrupt status register, as described in Section 17.7.3.8, "CRCU Interrupt Status Register." If a CRCU interrupt mask register bit is set, then the corresponding interrupt status register bit is always zero.

Masking an error bit allows for a hardware error condition to go potentially undetected. Therefore, extreme care should be taken when masking errors, as invalid results may be produced. It is recommended that errors only be masked during debug operation. This register may be reset by resetting the CRCU.

The interrupt mask register's bit fields are shown in Figure 17-47, and defined in Table 17-46.

Offset 0x3_F038 Access: Read/Write

Bits	0–48	49	50	51	52	53	54	55	56	57	58–60	61	62–63
R/W	—	ICE	PE	IE	ERE	CE	KSE	DSE	ME	AE	—	IFO	—

Figure 17-47. CRCU Interrupt Mask Register

Table 17-46. CRCU Interrupt Mask Register Bit Definitions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity Check Error. The supplied ICV(CRC) did not match the one computed by the CRCU. 0 Integrity check error enabled. WARNING: Do not enable this if using EU status writeback (see bits IWSE and AWSE in Section 17.4.4.1, “Channel Configuration Register (CCR)”). 1 Integrity check error disabled
50	PE	Polynomial Error. 0 Polynomial error enabled 1 Polynomial error disabled
51	IE	Internal Error. An internal processing error was detected while performing hashing. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early Read Error. The CRCU register was read while the CRCU was performing hashing. 0 Early read error enabled 1 Early read error disabled
53	CE	Context Error. The CRCU key register, the key size register, the data size register, or the mode register, was modified while the CRCU was performing hashing. 0 Context error enabled 1 Context error disabled
54	KSE	Key Size Error. A value outside the bounds was written to the CRCU key size register 0 Key size error enabled 1 Key size error disabled
55	DSE	Data Size Error. An inconsistent value was written to the CRCU data size register: 0 Data size error enabled 1 Data size error disabled
56	ME	Mode Error. An illegal value was detected in the mode register. 0 Mode error enabled 1 Mode error disabled
57	AE	Address Error. An illegal read or write address was detected within the CRCU address space. 0 Address error enabled 1 Address error disabled
58–60	—	Reserved
61	IFO	Input FIFO Overflow. The CRCU input FIFO was pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled
62–63	—	Reserved

17.7.3.10 CRCU ICV Size Register

The CRCU ICV size register (shown in Figure 17-48) is word readable/writable for compatibility with the MDEU. Values written to this location are always ignored, and reads from this location always returns zero. A context error is generated if this register is written after processing has begun.

	0–63
Field	—
Reset	0
R/W	R/W
Addr	CRCU 0x3_F040

Figure 17-48. CRCU ICV Size Register

17.7.3.11 CRCU End of Message Register

The CRCU end of message register (shown in Figure 17-49) is used to indicate that all data has been written to the CRCU (in channel-driven access, this signaling is done automatically). A write to this register is required to complete a CRC32 operation. The CRCU starts processing message data as soon as the data size register is written and data becomes available in the FIFO, but it does not process a remaining partial word or perform an ICV check until this register is written.

Any value written to this register has the same effect. Reading this register returns a zero value.

Offset 0x3_F050 Access: Write only

	0–63
R	
W	
Reset	All zeros

Figure 17-49. CRCU End of Message Register

17.7.3.12 CRCU Context Register

This register can be written with an intermediate CRC result or desired initial state prior to processing any data. Once processing is complete, the CRC result is available from this register. The reset state of this register is all ones, as this allows the CRC32 algorithm to detect bit errors in the leading zeros of a message. Figure 17-50 shows the bit position of each term in the written context value. A context error is generated if this register is written after processing has begun. An early read error is generated if this register is read while the module is busy.

Offset 0x3_F108 Access: Write only

	32	33	34	35	36	37	38	39	40–55	56	57	58	59	60	61	62	63
R									—								
W	X^{31}	X^{30}	X^{29}	X^{28}	X^{27}	X^{26}	X^{25}	X^{24}	—	X^{7}	X^{6}	X^{5}	X^{4}	X^{3}	X^{2}	X^{1}	X^{0}
Reset	1	1	1	1	1	1	1	1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1	1	1	1	1	1	1	1

Figure 17-50. CRCU Context Register (Write)

If the CRCU is in the default output mode (RAW bit is 0), this register when read holds the CRC remainder after it has been bit swapped, byte swapped, and complemented. This sequence of operations is described in the protocol specifications and generates a result which can be written directly to the end of a frame or command. Figure 17-51 shows the value in each bit position.

Offset 0x3_F108 Access: Read only

	32	33	34	35	36	37	38	39	40–55	56	57	58	59	60	61	62	63
R	x^{24}	x^{25}	x^{26}	x^{27}	x^{28}	x^{29}	x^{30}	x^{31}	—	x^0	x^1	x^2	x^3	x^4	x^5	x^6	x^7
W																	
Reset	All zeros																

Figure 17-51. CRCU Context Register (Read-Default Mode)

If the CRCU is in the raw output mode (RAW bit is 1), this register when read holds the unmodified CRC remainder. This form is the one used internally to match against the polynomial specific residue when performing ICV checking. Figure 17-52 shows the value in each bit position.

Offset 0x3_F108 Access: Read only

	32	33	34	35	36	37	38	39	40–55	56	57	58	59	60	61	62	63
R	x^{31}	x^{30}	x^{29}	x^{28}	x^{27}	x^{26}	x^{25}	x^{24}	—	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
W																	
Reset	All zeros																

Figure 17-52. CRCU Context Register (Read-Raw Mode)

17.7.3.13 CRCU Key Register

The CRCU key register stores the polynomial and residue for the dynamic custom mode as set in the mode register (see Section 17.7.3.2, "CRCU Mode Register"). Figure 17-53 shows the bit position of each coefficients. The reset value of this register is all zeros with a one in bit position 0. This register is dynamic, in that it is reset by performing a re-initialize or a software reset. This allows a custom polynomial to be used for specific processing without changing the platform-specific static custom polynomial stored in the control register (see Section 17.7.3.6, "CRCU Control Register"). A residue does not need to be programmed unless ICV checking is being performed. A context error is generated if this register is written after processing has begun. A polynomial error is generated if a value is written to this register which does not have a one in bit position 0.

Offset 0x3_F400 Access: Read/Write

	32	33	34	35	36	37	38	39	40–55	56	57	58	59	60	61	62	63
R/W	r^{31}	r^{30}	r^{29}	r^{28}	r^{27}	r^{26}	r^{25}	r^{24}	—	r^7	r^6	r^5	r^4	r^3	r^2	r^1	r^0

	32	33	34	35	36	37	38	39	40–55	56	57	58	59	60	61	62	63
R/W	g^{31}	g^{30}	g^{29}	g^{28}	g^{27}	g^{26}	g^{25}	g^{24}	—	g^7	g^6	g^5	g^4	g^3	g^2	g^1	g^0

Figure 17-53. CRCU Key Register

17.7.3.14 CRCU FIFO

Words written to this address range are pushed onto the CRCU input FIFO, thereby buffering them for processing. Partial words and misaligned data can be written to this address and it is automatically realigned based on a big endian byte order.

17.7.4 Data Encryption Standard Execution Unit (DEU)

This section contains details about the Data Encryption Standard execution unit (DEU), including modes of operation, status and control registers, and FIFOs.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the DEU is used through channel-controlled access, which means that most reads and writes of DEU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

17.7.4.1 DEU Mode Register

The DEU mode register contains 3 bits which are used to program DEU operation.

The mode register is cleared when the DEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error is generated.

Offset 0x3_2000 Access: Read/Write

	0–59	60–61	62	63
R/W	—	CM	TS	ED
Reset	All zeros			

Figure 17-54. DEU Mode Register

Table 17-47 describes DEU mode register fields.

Table 17-47. DEU Mode Register Field Descriptions

Bits	Name	Description
The following bits are described for information only. They are not under direct user control.		
0–55	—	Reserved
The following bits are controlled through the MODE0 field of the descriptor header.		
56–59	—	Reserved
60–61	CM	Cipher Mode: Used to define the mode of DEU operation. See Table 17-48 for mode bit combinations.

Table 17-47. DEU Mode Register Field Descriptions (continued)

Bits	Name	Description
62	TS	Triple/Single DES. If set, DEU operates the Triple DES algorithm; if not set, DEU operates the single DES algorithm. 0 Single DES 1 Triple DES
63	ED	Encrypt/decrypt. If set, DEU operates the encryption algorithm; if not set, DEU operates the decryption algorithm. 0 Perform decryption 1 Perform encryption

Table 17-48. DEU Cipher Modes

Mode	CM (60:61)
ECB	00
CBC	01
CFB-64	10
OFB-64	11

17.7.4.2 DEU Key Size Register

This value indicates the number of bytes of key memory that should be used in encrypting or decrypting. If the DEU mode register is set for single DES, any value other than 8 bytes automatically generates a key size error in the DEU interrupt status register. If the mode bit is set for triple DES, any value other than 16 bytes (112 bits for 2-key triple DES (K1=K3) or 24 bytes (168 bits for 3-key triple DES) generates an error. Triple DES always uses K1 to encrypt, K2 to decrypt, K3 to encrypt.

Offset 0x3_2008 Access: Read/Write

	0–51	52–63
R/W	—	Key Size
Reset	All zeros	

Figure 17-55. DEU Key Size Register

Table 17-49 shows the legal values for DEU key size.

Table 17-49. DEU Key Size Register Field Descriptions

Bits	Name	Description
0–51	—	Reserved
52–63	Key Size	8 bytes = 0x08 (only legal value if mode is single DES.) 16 bytes = 0x10 (for 2 key 3DES, K1 = K3) 24 bytes = 0x18 (for 3 key 3DES)

17.7.4.3 DEU Data Size Register (DEUDSR)

The DEUDSR, shown in Figure 17-56, is used to verify that the data to be processed by the DEU is divisible by the DES algorithm block size of 64 bits. The DEU does not automatically pad messages out to 64-bit blocks; therefore, any message processed by the DEU must be divisible by 64 bits or a data size error will occur.

In channel-driven operation, the full message length (data size) to be encrypted or decrypted by the DEU is copied from the descriptor to the DEUDSR; however, only bits 58–63 are checked to determine if there is a data size error. If bits 58–63 are all zeros, the message is evenly divisible into 64-bit blocks. In host-driven operation,the user must write the data size to the DEUDSR. If bits 58–63 are not all zero, then a data size error occurs.This register is cleared when the DEU is reset or re-initialized.

Offset 0x3_2008 Access: Read/Write

	0–51	52–63
R W	—	Data Size
Reset	All zeros	

Figure 17-56. DEU Data Size Register

17.7.4.4 DEU Reset Control Register

This register, shown in Figure 17-57, allows three levels reset of just DEU, as defined by the three self-clearing bits:

	0–60	61	62	63
Field	—	RI	MI	SR
Reset	0			
R/W	R/W			
Addr	DEU 0x3_2018			

Figure 17-57. DEU Reset Control Register

Table 17-50 describes DEU reset control register fields.

Table 17-50. DEU Reset Control Register Field Descriptions

Bits	Names	Description
0–60	—	Reserved
61	RI	Reset interrupt. Writing this bit active high causes DEU interrupts signaling done and error to be reset. It further resets the state of the DEU interrupt status register. 0 Do not reset 1 Reset interrupt logic

Table 17-50. DEU Reset Control Register Field Descriptions (continued)

Bits	Names	Description
62	MI	Module initialization is nearly the same as software reset, except that the interrupt mask register remains unchanged. this module initialization includes execution of an initialization routine, completion of which is indicated by the RESET_DONE bit in the DEU status register 0 Do not reset 1 Reset most of DEU
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for DEU. All registers and internal state are returned to their defined reset state. Upon negation of SW_RESET, the DEU enters a routine to perform proper initialization of the parameter memories. The RESET_DONE bit in the DEU status register indicates when this initialization routine is complete 0 Do not reset 1 Full DEU reset

17.7.4.5 DEU Status Register

This status register, displayed in Figure 17-58, contains 6 fields which reflect the state of DEU internal signals. The DEU status register is read only. Writing to this location results in address error being reflected in the DEU interrupt status register.

Offset 0x3_2028 Access: Read only

	0–39	40–47	48–55	56–57	58	59–60	61	62	63
R	—	OFL	IFL	—	HALT	—	EI	DI	RD
W									
Reset	All zeros								

Figure 17-58. DEU Status Register

Table 17-51 describes the DEU status register's bit settings.

Table 17-51. DEU Status Register

Bits	Name	Description
0–39	—	Reserved
40-47	OFL	The number of dwords currently in the output FIFO
48-55	IFL	The number of dwords currently in the input FIFO
56-57	—	Reserved
58	HALT	Halt. Indicates that the DEU has halted due to an error. 0 DEU not halted 1 DEU halted **Note:** Because the error causing the DEU to stop operating may be masked before reaching the interrupt status register, the DEU interrupt status register is used to provide a second source of information regarding errors preventing normal operation.
59-60	—	Reserved

Table 17-51. DEU Status Register (continued)

Bits	Name	Description
61	EI	Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). 0 DEU is not signaling error 1 DEU is signaling error
62	DI	Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). 0 DEU is not signaling done 1 DEU is signaling done
63	RD	Reset done. This status bit, when high, indicates that DEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel. 0 Reset in progress 1 Reset done **Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation.

17.7.4.6 DEU Interrupt Status Register

The DEU interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the DEU interrupt mask register is zero (see Section 17.7.4.7, "DEU Interrupt Mask Register").

If the DEU interrupt status register is non-zero, the DEU halts and the DEU error interrupt signal is asserted to the controller (see Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). In addition, if the DEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 17-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the DEU reset control register.

The definition of each bit in the DEU interrupt status register is shown in Figure 17-59.

Offset DEU 0x3_2030 Access: Read only

	0–49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
R	—	KPE	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	IFU	IFO	OFU	OFO
W															
Reset	All zeros														

Figure 17-59. DEU Interrupt Status Register

Table 17-52 describes DEU interrupt status register fields.

Table 17-52. DEU Interrupt Status Register Field Descriptions

Bits	Name	Description
0-49	—	Reserved
50	KPE	Key Parity Error. Defined parity bits in the keys written to the key registers did not reflect odd parity correctly. (Note that key register 2 and key register 3 are checked for parity only if the appropriate DEU mode register bit indicates triple DES. Also, key register 3 is checked only if key size reg = 24. Key register 2 is checked only if key size reg = 16 or 24.) 0 No error detected 1 Key parity error
51	IE	Internal Error. An internal processing error was detected while performing encryption. 0 No error detected 1 Internal error **Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the interrupt mask register or by resetting the DEU.
52	ERE	Early Read Error. The DEU IV register was read while the DEU was performing encryption. 0 No error detected 1 Early read error
53	CE	Context Error. A DEU key register or the key size register, data size register, mode register, or IV register was modified while DEU was performing encryption. 0 No error detected 1 Context error
54	KSE	Key Size Error. An inappropriate value (8 being appropriate for single DES, and 16 and 24 being appropriate for triple DES) was written to the DEU key size register 0 No error detected 1 Key size error
55	DSE	Data Size Error (DSE): A value was written to the DEU data size register that is not a multiple of 64 bits if ECB, CBC, or CFB mode is selected. If OFB mode is selected, any data size value is permitted. 0 No error detected 1 Data size error
56	ME	Mode error. An illegal value was detected in the mode register. Note: writing to reserved bits in mode register is likely source of error. 0 No error detected 1 Mode error
57	AE	Address error. An illegal read or write address was detected within the DEU address space. 0 No error detected 1 Address error
58	OFE	Output FIFO error. The DEU output FIFO was detected non-empty upon write of DEU data size register. 0 No error detected 1 Output FIFO non-empty error
59	IFE	Input FIFO error. The DEU input FIFO was detected non-empty upon generation of done interrupt. 0 No error detected 1 Input FIFO non-empty error

Table 17-52. DEU Interrupt Status Register Field Descriptions (continued)

Bits	Name	Description
60	IFU	Input FIFO Underflow. The DEU input FIFO was read while empty. 0 No error detected 1 Input FIFO has had underflow error
61	IFO	Input FIFO Overflow. The DEU input FIFO was pushed while full. 0 No error detected 1 Input FIFO has overflowed **Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input. When operated through host-controlled access, the DEU cannot accept FIFO inputs larger than 256 bytes without overflowing.
62	OFU	Output FIFO Underflow. The DEU output FIFO was read while empty. 0 No error detected 1 Output FIFO has underflow error
63	OFO	Output FIFO Overflow. The DEU output FIFO was pushed while full. 0 No error detected 1 Output FIFO has overflowed

17.7.4.7 DEU Interrupt Mask Register

The DEU interrupt mask register, controls the setting of bits in the DEU interrupt status register, as described in Section 17.7.4.6, “DEU Interrupt Status Register”. If a DEU interrupt mask register bit is set, then the corresponding interrupt status register bit is always zero.

Masking an error bit allows for a hardware error condition to go potentially undetected. Therefore, extreme care should be taken when masking errors, as invalid results may be produced. It is recommended that errors only be masked during debug operation. This register may be reset by resetting the DEU.

Offset 0x3_2038 Access: Read/Write

	0–48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
R/W	—	KPE	IE	ERE	CE	KSE	DSE	ME	AE	OFE	IFE	IFU	IFO	OFU	OFO	KPE
Reset	1000															

Figure 17-60. DEU Interrupt Mask Register

Table 17-53. DEU Interrupt Mask Register Field Descriptions

Bits	Name	Description
0-49	—	Reserved
50	KPE	Key Parity Error. The defined parity bits in the keys written to the key registers did not reflect odd parity correctly. (Note that key register 2 and key register 3 are only checked for parity if the appropriate DEU mode register bit indicates triple DES. 0 Key parity error enabled 1 Key parity error disabled

Table 17-53. DEU Interrupt Mask Register Field Descriptions

Bits	Name	Description
51	IE	Internal Error. An internal processing error was detected while performing encryption. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early Read Error. The DEU IV Register was read while the DEU was performing encryption. 0 Early read error enabled 1 Early read error disabled
53	CE	Context Error. A DEU key register,or the key size register, the data size register, the mode register, or IV register was modified while DEU was performing encryption. 0 Context error enabled 1 Context error disabled
54	KSE	Key Size Error. An inappropriate value (8 being appropriate for single DES, and 16 and 24 being appropriate for Triple DES) was written to the DEU key size register 0 Key size error enabled 1 Key size error disabled
55	DSE	Data Size Error (DSE). A value that is not a multiple of 64 bits was written to the DEU data size register if ECB, CBC, or CFB mode is selected. If OFB mode is selected, any data size value is permitted. 0 Data size error enabled 1 Data size error disabled
56	ME	Mode Error. An illegal value was detected in the mode register. 0 Mode error enabled 1 Mode error disabled
57	AE	Address Error. An illegal read or write address was detected within the DEU address space. 0 Address error enabled 1 Address error disabled
58	OFE	Output FIFO Error. The DEU output FIFO was detected non-empty upon write of DEU data size register 0 Output FIFO non-empty error enabled 1 Output FIFO non-empty error disabled
59	IFE	Input FIFO Error. The DEU input FIFO was detected non-empty upon generation of done interrupt 0 Input FIFO non-empty error enabled 1 Input FIFO non-empty error disabled
60	IFU	Input FIFO Underflow. The DEU input FIFO was read while empty. 0 Input FIFO Underflow error enabled 1 Input FIFO Underflow error disabled
61	IFO	Input FIFO Overflow. The DEU input FIFO was pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled **Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input. When operated through host-controlled access, the DEU cannot accept FIFO inputs larger than 256 bytes without overflowing.

Table 17-53. DEU Interrupt Mask Register Field Descriptions

Bits	Name	Description
62	OFU	Output FIFO Underflow. The DEU output FIFO was read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	OFO	Output FIFO Overflow. The DEU output FIFO was pushed while full. 0 Output FIFO Overflow error enabled 1 Output FIFO Overflow error disabled

17.7.4.8 DEU End of Message Register

The DEU end of message register, shown in Figure 17-61, is used to signal to the DEU that the final message block has been written to the input FIFO (in channel-driven access, this signaling is done automatically). The DEU will not process the last block of data in its input FIFO until this register is written. Once the end of message register is written, the DEU processes any remaining data in the input FIFO and generates the done interrupt.

The value written to this register does not matter. A read of this register always returns a zero value.

Offset 0x3_2050 Access: Write only

	0 — 63
R W	—
Reset	All zeros

Figure 17-61. DEU End of Message Register

17.7.4.9 DEU IV Register

For CBC mode, the initialization vector is written to and read from the DEU IV register. The value of this register changes as a result of the encryption process and reflects the context of DEU. Reading this memory location while the module is processing data generates an error interrupt.

17.7.4.10 DEU Key Registers

The DEU uses three write-only key registers, K1, K2, and K3, to perform encryption and decryption. In Single DES mode, only K1 may be written. The value written to K1 is simultaneously written to K3, auto-enabling the DEU for 112-bit Triple DES if the key size register indicates 2 key 3DES is to be performed (key size = 16 bytes). To operate in 168-bit Triple DES, K1 must be written first, followed by the write of K2, then K3.

Reading any of these memory locations generates an address error interrupt.

17.7.4.11 DEU FIFOs

DEU uses an input FIFO/output FIFO pair to hold data before and after the encryption process. Normally, the channels control all access to these FIFOs. For host-controlled operation, a write to anywhere in the

DEU FIFO address space enqueues data to the DEU input FIFO, and a read from anywhere in the DEU FIFO address space dequeues data from the DEU output FIFO.

Writes to the input FIFO go first to a staging register which can be written by byte, word (4 bytes), or dword (8 bytes). When all 8 bytes of the staging register have been written, the entire dword is automatically enqueued into the FIFO. If any byte is written twice between enqueues, it causes an error interrupt of type AE from the EU. Since the DEU data length should always be a multiple of 8 bytes, the last write should complete a dword. However, if there is any partial dword in the staging register when the DEU end of message register is written, the partial dword is automatically padded with zeros to a full 8 bytes and enqueued to the input FIFO.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword have been read, that dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between dequeues, it causes an error interrupt of type AE from the EU.

Overflows and underflows caused by reading or writing the DEU FIFOs are reflected in the DEU interrupt status register.

17.7.5 Kasumi Execution Unit (KEU)

This section contains details about the Kasumi execution unit (KEU), including modes of operation, status and control registers, and FIFOs. The KEU has been designed to support the f8 confidentiality function of the 3GPP, GSM A5/3, EDGE A5/3, and GPRS GEA3 algorithms. The KEU also supports the 3GPP f9 integrity function.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purpose. In typical operation, the KEU is used through channel-controlled access, which means that most reads and writes of the KEU registers are directed by the SEC channels. Driver software performs host-controlled register accesses only on a few registers for initial configuration and error handling.

This execution unit (EU) includes an ICVchecking feature, which means it can generate an ICV and compare it to another supplied ICV. The pass/fail result of this ICV check can be returned to the host either through interrupt or by using a writeback of EU status fields into the host memory, but not using both methods at the same time.

To signal the ICV checking result by status writeback, turn on either the IWSE bit or AWSE bit in the channel configuration register (for more information, see Section 17.4.4.1, “Channel Configuration Register (CCR)”), and mask the ICE bit in the interrupt mask register (Section 17.7.5.7, “KEU Interrupt Mask Register (KEUIMR)”). In this case the normal DONE signal (by interrupt or writeback) is undisturbed.

To signal the ICV checking result by interrupt, unmask the ICE bit in the interrupt mask register and turn off the IWSE and AWSE bits in the channel configuration register. If there is no ICV mismatch, the normal DONE signal (by interrupt or writeback) occurs. When there is an ICV mismatch, there is an ERROR interrupt signal to the host, but no DONE interrupt signal or writeback.

17.7.5.1 KEU Mode Register (KEUMR)

The KEU mode register, shown in Figure 17-62, contains several bits which are used to program the KEU. The mode register is cleared when the KEU is reset or re-initialized. Setting a reserved mode bit generates a data error. Setting both the GSM and EDGE bits to one generates a data error. If the KEU mode register is modified during processing, a context error is generated.

Offset 0x3_E000 Access: Read/Write

	0–55	56	57	58	59	60	61	62–63
R/W	—	GSM	CICV	EDGE	PE	INT	—	ALG
Reset	All zeros							

Figure 17-62. KEU Mode Register

Table 17-54 describes the KEU mode register fields.

Table 17-54. KEU Mode Register Field Descriptions

Bits	Name	Description
0–55	—	Reserved
56	GSM	Select GSM A5/3 blocks 0 GSM A5/3 blocks not selected 1 GSM A5/3 blocks selected **Note 1:** For GSM A5/3, Two 114-bit blocks are required to be produced each 4.615mS slot. If GSM = 1, the first read of the output FIFO retrieves the first 64 bits of block 1. The second read of the output FIFO retrieves the next 50 bits of block 1 (the remaining bits of this 64-bit word are cleared to zero). The third read of the output FIFO retrieves the first 64 bits of block 2, while a fourth read of the output FIFO retrieves the next 50 bits of block 2 (the remaining bits of this 64-bit word are cleared to zero). **Note 2:** If GSM = 0, 228 contiguous bits may be read with successive reads of the output FIFO. In this case the host (application) is responsible for handling the A5/3 block formatting. **Note 3:** If GSM is set to 1, while EDGE = 1, an interrupt/error is generated.
57	CICV	Compare integrity check values. 0 Normal operation; no ICV comparison. 1 After the ICV is computed, compare it to the data in the KEU's ICV_In register. If the ICVs do not match, send an error interrupt to the channel. Only applicable when the ALG field is set to a function that uses f9.
58	EDGE	Select EDGE A5/3 blocks 0 EDGE A5/3 blocks not selected 1 EDGE A5/3 blocks selected **Note 1:** For EDGE A5/3, two 348-bit blocks are required to be produced each 4.615mS slot. If EDGE = 1, the first five reads of the output FIFO retrieve the first 320 bits of block 1. The sixth read of the output FIFO retrieves the final 28 bits of block 1 (the remaining bits of the sixth 64-bit word are cleared to zero). The next five reads of the output FIFO retrieve the first 320 bits of block 2. The following read of the output FIFO retrieves the final 28 bits of block 2 (the remaining bits of this 64-bit word are cleared to zero). **Note 2:** If EDGE = 0, 696 contiguous bits may be read with successive reads of the output FIFO. In this case the host (application) is responsible for handling the A5/3 block formatting. **Note 3:** If EDGE is set to 1, whilst GSM = 1, an interrupt/error is generated.
59	PE	Process end of message. Enables final processing of last message block (f9 only). 0 Prevent final block processing (message incomplete) 1 Enable final block processing (message complete) **Note:** PE is closely connected with the KEU data size register, see Section 17.7.5.3, "KEU Data Size Register (KEUDSR)" for more details.

Table 17-54. KEU Mode Register Field Descriptions (continued)

Bits	Name	Description
60	INT	Initialization. Enables initialization for a new message. 0 Prevent initialization 1 Enable initialization **Note:** For f8 or f9 operations, if the 3G frame (or message) is being processed through a single descriptor, the Initialization bit should be set. If the frame is split across multiple descriptors, this bit should only be set in the descriptor that processes the first block of the message.
61	—	Reserved
62–63	ALG	Algorithm selection. Specifies the functions to perform. 00 Perform f8 function only 01 Reserved 10 Perform f9 function only 11 Reserved

17.7.5.2 KEU Key Size Register (KEUKSR)

The KEU key size register, shown in Figure 17-63, stores the number of bytes in the key. It should be set to 16 bytes. This register is cleared when the KEU is reset or re-initialized. If a key size is specified that does not match the selected algorithm(s), an illegal key size error is generated.

	0–51	52–63
Field	—	Key Size (Bytes)
Reset	0	
R/W	R/W	
Addr	KEU 0x3_E008	

Figure 17-63. KEU Key Size Register

17.7.5.3 KEU Data Size Register (KEUDSR)

The KEU data size register (shown in Figure 17-64) stores the number of bits to process in the final message word. As Kasumi allows for bit level granularity for encryption/decryption, there are no illegal data sizes. The proper bit length of the message must be written to notify the KEU of any padding performed by the host. This register is cleared when the KEU is reset or re-initialized.

Writing to this register signals the KEU to start processing data from the input FIFO as soon as it is available. If the value of data size is modified during processing, a context error is generated.

Kasumi processing is determined by both the data size and the setting of the process end of message (PE) bit in the KEU mode register. The PE bit determines how the final block of message data is processed. In typical descriptor-based operations, the data size register is loaded with values which are an integral number of bytes. For descriptor based f8 operations, the software is responsible for padding the data to the next byte boundary, and for removing this padding from the KEU's output. The output of the KEU is an integral number of bytes, as specified in the descriptor, automatically truncating any internal padding required to process the final 64 bits message block. As the KEU can infer when it has reached the final

64 bits message block from the length fields in the descriptor, setting the PE bit through the descriptor header is not required. While performing f8 operations, the KEU's output is the same irrespective of the setting of the PE bit.

For the descriptor-based f9 operations, the PE bit must be set through the descriptor header whenever the descriptor is being used to process the final message block. This causes the KEU to automatically pad the final block before calculating the f9 MAC.

Offset 0xE010 Access: Read/Write

	0–51	52–63
R	—	Data Size (bits)
W		
Reset	All zeros	

Figure 17-64. KEU Data Size Register

The details of data size register and the PE interaction are more relevant when operating the KEU in direct access (slave) mode, rather than using descriptors. Note that operating the KEU in direct access mode is not recommended other than for debug test cases, and the information provided here regarding the use of data size and PE in direct access mode is to explain the behaviors that might be encountered in direct access debug operations.

PE has the following effects for direct access mode f8 operations, using the example of a 64-bit f8 keystream '0x1234567890abcdef' and the data size register containing '0x0a' (10 bits = 1 byte + 2 bits):

- PE = 0: The final ten message bits are XORed with the entire last 64-bit block of keystream, which produces 54 additional non-zero bits after the end of the real message. These additional 54 bits must be removed by the software.
- PE = 1: The final ten message bits are XORed with ten bits of keystream '0x120', and no additional bits of the false message are produced.

For direct access mode f9 operations, assertion of PE enables f9 algorithm-specified padding per the value in the data size register. If this direct access mode operation includes the final message block, then PE must be set in the KEU mode register as follows:

- PE = 0: The KEU does not perform padding. For the KEU to perform the correct computation, the last block of the message provided must be a complete 64-bit block, and the value written into the KEU data size register must indicate a multiple of 64 bits.
- PE = 1: The KEU pads the final 64 bits of the message as specified in the 3GPP f9 algorithm. The communication direction (CD) bit (see Table 17-59 for KEU IV_1 Register Fields Description) and '1' is appended to the end of the message. The result is zero-padded to 64 bits.

  For example, if the last block is xF81A_0000_0000_0000 (big endian) and data size contains 0x0F (15 bits = 1 byte + 7 bits), the word (0xF81A = 1111_1000_0001_1010), the most-significant 15 bits (underlined) are padded left to right as 1111_1000_0001_101$_1000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_0000_000 0, where $ is the value of the CD bit in the mode register.

17.7.5.4 KEU Reset Control Register (KEURCR)

The KEU reset control register, shown in Figure 17-65, allows three levels of reset of the KEU, as defined by the three self-clearing bits.

	0–60	61	62	63
Field	—	CLI	RI	SR
Reset	0			
R/W	R/W			
Addr	KEU 0xE018			

Figure 17-65. KEU Reset Control Register

Table 17-55 describes the KEU reset control register fields.

Table 17-55. KEU Reset Control Register Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	CLI	Clear Interrupts. Setting this bit causes the KEU interrupt signals—DONE and ERROR—to be reset. It further resets the state of the KEU interrupt status register. 0 Normal operation 1 Clear interrupts and the KEU interrupt status register
62	RI	Re- Initialization. It is same as software reset (SR), except that the interrupt mask register remains unchanged. Completion of re-initialization is indicated by the RESET_DONE bit in the KEU status register. 0 Normal operation 1 Re-initialize the KEU
63	SR	Software reset. Functionally equivalent to hardware reset (the $\overline{\text{RESET}}$ signal), but only for the KEU. All registers and internal state are returned to their defined reset state. Upon negation of the SR bit, the KEU enters a routine to perform proper initialization of the parameter memories. The reset done (RD) bit in the KEU status register indicates when this initialization is complete 0 Normal operation 1 Full KEU reset

17.7.5.5 KEU Status Register (KEUSR)

The KEU status register, shown in Figure 17-66, is a read-only register that reflects the state of six status outputs. While writing to this location, an address error is reflected in the KEU interrupt status register.

	0–39	40–47	48–55	56–57	58	59–60	61	62	63
Field	—	OFL	IFL	—	HALT	ICCR	EI	DI	RD
Reset	0								
R/W	R								
Addr	KEU 0x3_E028								

Figure 17-66. KEU Status Register

Table 17-56 describes the KEU status register fields.

Table 17-56. KEU Status Register Fields Description

Bits	Name	Description
0–39	—	Reserved
40–47	OFL	Output FIFO level. The number of dwords currently in the output FIFO.
48–55	IFL	Input FIFO level. The number of dwords currently in the input FIFO.
56–57	—	Reserved
58	HALT	Indicates when the KEU core has halted due to an error. 0 KEU not halted 1 KEU core halted (must be reset/re-initialized) **Note:** As the error causing the KEU to stop operating may be masked to the interrupt status register, the status register is used to provide a second source of information regarding errors preventing normal operation.
59-60	ICCR	Integrity check comparison result. 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved **Note:** A passed or failed result is generated only if ICV checking is enabled and the algorithm selected is f9.
61	EI	Error interrupt. Reflects the state of the ERROR interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 KEU is not signaling error 1 KEU is signaling error
62	DI	Done interrupt. Reflects the state of the DONE interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 Processing not done 1 All bytes processed
63	RD	Reset done. Indicates when the KEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel. 0 Reset in progress 1 Reset done

17.7.5.6 KEU Interrupt Status Register (KEUISR)

The KEU interrupt status register tracks the state of possible errors, provided those errors are not masked, through the KEU interrupt control register.

The KEU interrupt status register indicates the unmasked errors that have occurred and have generated the ERROR interrupt signals to the channel. Each bit in this register can only be set if the corresponding bit of the KEU interrupt mask register is zero (see Section 17.7.5.7, "KEU Interrupt Mask Register (KEUIMR)").

If the KEU interrupt status register is non-zero, the KEU halts and the KEU ERROR interrupt signal is asserted to the controller (see Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). In addition, if the KEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which the EU is assigned. The EU error then appears in the bit 55 of the channel pointer status register (for more information, see Table 17-15 on page 17-43) and generates a channel error interrupt to the controller.

This register can be cleared by setting the RI bit of the KEU reset control register. If a KEU error is reported by the channel while operating in descriptor mode, the user can rely on the channel to clear the KEU interrupt by writing the Continue bit in the channel configuration register (for more information, see Section 17.4.4.1, "Channel Configuration Register (CCR)"). Setting any error bit in this register causes the KEU to signal the corresponding error, unless the associated error has been masked in the KEU interrupt mask register.

The definition of each bit in the KEU interrupt status register is shown in Figure 17-67.

	0–48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Field	—	ICE	—	IE	ERE	CE	KSE	DSE	DE	AE	OFE	IFE	—	IFO	OFU	—
Reset	0															
R/W	R/W															
Addr	KEU 0x3_E030															

Figure 17-67. KEU Interrupt Status Register

Table 17-57 describes the KEU interrupt status register signals.

Table 17-57. KEU Interrupt Status Register Signals Description

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity check error. 0 No error detected 1 Integrity check error detected. An ICV check was performed on an f9 result and the supplied ICV did not match the one computed by the KEU.
50	—	Reserved

Table 17-57. KEU Interrupt Status Register Signals Description (continued)

Bits	Name	Description
51	IE	Internal error. An internal processing error was detected while the KEU was processing. 0 No error detected 1 Internal error This bit is set any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the interrupt mask register or by resetting the KEU.
52	ERE	Early read error. A KEU context or IV register was read while the KEU was processing. 0 No error detected 1 Early read error
53	CE	Context error. A KEU key register, the key size register, the data size register, the mode register, or IV register was modified while the KEU was processing. 0 No error detected 1 Context error
54	KSE	Key size error. An inappropriate value (not 16 or 32 bytes) was written to the KEU key size register. 0 No error detected 1 Key size error
55	DSE	Data size error. A value was written to the KEU data size register that is greater than 64 bits. 0 No error detected 1 Data size error
56	DE	Data error. Invalid data was written to a register or a reserved mode bit was set. 0 Valid data 1 Reserved or invalid mode selected
57	AE	Address error. An illegal read or write address was detected within the KEU address space. 0 No error detected 1 Address error
58	OFE	Output FIFO error. The KEU output FIFO was non-empty upon write of the KEU data size register. 0 No error detected 1 Output FIFO non-empty error
59	IFE	Input FIFO error. The KEU input FIFO was non-empty upon generation of the done interrupt. 0 No error detected 1 Input FIFO non-empty error
60	—	Reserved
61	IFO	Input FIFO overflow. The KEU input FIFO has been pushed while full. 0 No error detected 1 Input FIFO has overflowed
62	OFU	Output FIFO underflow. The KEU output FIFO was read while empty. 0 No error detected 1 Output FIFO has underflow error
63	—	Reserved

17.7.5.7 KEU Interrupt Mask Register (KEUIMR)

The KEUIMR controls the setting of bits in the KEU interrupt status register (KEUISR), as described in Section 17.7.5.6, "KEU Interrupt Status Register (KEUISR)". If a KEUIMR bit is set, then the corresponding KEUISR bit is always zero.

Masking an error bit allows for a hardware error condition to go potentially undetected. Therefore, extreme care should be taken when masking errors, as invalid results may be produced. It is recommended that errors only be masked during debug operation. This register may be reset by resetting the KEU.

The KEUIMR fields are shown in Figure 17-68. The fields are defined in Table 17-58.

	0–48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Field	—	ICE	—	IE	ERE	CE	KSE	DSE	DE	AE	OFE	IFE	—	IFO	OFU	—
Reset	0	0	0	1	0											
R/W	R/W															
Addr	KEU 0x3_E038															

Figure 17-68. KEU Interrupt Mask Register

Table 17-58 describes the KEU interrupt mask register fields.

Table 17-58. KEU Interrupt Mask Register Fields Description

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity check error. 0 ICV check error enabled. WARNING: Do not enable this EU status writeback (see bits IWSE and AWSE in Section 17.4.4.1, "Channel Configuration Register (CCR)" is used. 1 ICV check error disabled
50	—	Reserved
51	IE	Internal error. An internal processing error was detected while performing encryption. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early read error. A KEU context or IV register was read while the KEU was performing encryption. 0 Early read error enabled 1 Early read error disabled
53	CE	Context error. A KEU key register, the key size register, data size register, mode register, or IV register was modified while the KEU was performing encryption. 0 Context error enabled 1 Context error disabled
54	KSE	Key size error. An inappropriate value (not 16 or 32 bytes) was written to the KEU key size register. 0 Key size error enabled 1 Key size error disabled
55	DSE	Data size error. Indicates that the number of bits to process is out of range. 0 Data size error enabled 1 Data size error disabled

Table 17-58. KEU Interrupt Mask Register Fields Description (continued)

Bits	Name	Description
56	DE	Data error. Indicates that invalid data was written to a register or a reserved mode bit was set. 0 Data error enabled 1 Data error disabled
57	AE	Address error. An illegal read or write address was detected within the KEU address space. 0 Address error enabled 1 Address error disabled
58	OFE	Output FIFO error. The KEU output FIFO was detected non-empty upon write of the KEU data size register. 0 Output FIFO non-empty error enabled 1 Output FIFO non-empty error disabled
59	IFE	Input FIFO error. The KEU input FIFO was detected non-empty upon generation of done interrupt. 0 Input FIFO non-empty error enabled 1 Input FIFO non-empty error disabled
60	—	Reserved
61	IFO	Input FIFO overflow. The KEU input FIFO was pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled
62	OFU	Output FIFO underflow. The KEU output FIFO was read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	—	Reserved

17.7.5.8 KEU Data Out Register (f9 MAC) (KEUDOR)

Following a done interrupt, the read-only KEU data out register holds the f9 message authentication code. A 64-bit value is returned. This value may be truncated to 32 bits for some applications. Writing to this location results in an address error reflected in the KEU interrupt status register.

	0 … 63
Field	KEU Data Out Register (f9 MAC)
Reset	0x0000_0000_0000_0000
R/W	R
Addr	KEU 0x3_E048

Figure 17-69. KEU Data Out Register (f9 MAC)

NOTE

According to the ETSI/SAGE 3GPP specification for f9 (version 1.2), only 32 bits of the final MAC are used. This corresponds to the lower 4 bytes of the KEU data out register.

17.7.5.9 KEU End of Message Register (KEUEMR)

The KEU end of message register, shown in Figure 17-70, is used to signal to the KEU that the final message block has been written to the input FIFO (in channel-driven access, this signaling is done automatically). The KEU will not process the last block of data in its input FIFO until this register is written. Once the end of message register is written, the KEU processes any remaining data in the input FIFO and generates the done interrupt.

The value written to this register does not matter. A read of this register always returns a zero value.

	0 — 63
Field	—
Reset	0
R/W	R/W
Addr	KEU 0x3_E050

Figure 17-70. KEU End of Message Register

17.7.5.10 KEU IV_1 Register (KEUIV1)

The KEU IV_1 register is a general purpose IV register, shown in Figure 17-71, is used during the initialization phase of the f8 algorithms for 3GPP, GSM A5/3, EDGE A5/3, GPRS GEA3, and f9 algorithm for 3GPP. The appropriate value as defined by the standards for each algorithm must be written before a new message is started.

After the initialization phase has been completed, the KEU IV_1 register is no longer used for the remainder of f8 processing. However, if 3GPP f9 is selected because the KEU IV_1 register contains the direction bit as defined by the 3GPP standard, the KEU IV_1 register must be written back during context switches to complete the generation of the 3GPP MAC.

	0–31	32–36	37	38–39	40–47	48–63
Field	CC	CB	CD	00	CA	CE
Reset	0					
R/W	R/W					
Addr	KEU 0x3_E100					

Figure 17-71. KEU IV_1 Register

Table 17-59 describes the KEU IV_1 register fields.

Table 17-59. KEU IV_1 Register Fields Description

Bits	Field	3GPP Definition	GSM A5/3 Definition	EDGE A5/3 Definition	GPRS GEA3 Definition
0–31	CC	Count	Count	0000000000 I I Count	Frame dependent input value (32-bits)
32–36	CB	Bearer	00000	00000	00000

Table 17-59. KEU IV_1 Register Fields Description

Bits	Field	3GPP Definition	GSM A5/3 Definition	EDGE A5/3 Definition	GPRS GEA3 Definition
37	CD	Direction bit	0	0	0
38–39	0	00	00	00	00
40–47	CA	00000000	00001111	11110000	11111111
48–63	CE	000000000000000	0000000000000000	0000000000000000	0000000000000000

The following figure shows how the KEU IV_1 register can be differentiated for different applications.

	0 … 31	32 … 36	37	38 … 39	40 … 47	48 … 63
3GPP (f8)	Count	Bearer	Dir.	00	00000000	000000000000000
GSM (A5/3)	Count	00000	0	00	00001111	000000000000000
EDGE (A5/3)	0000000000 ǁ Count	00000	0	00	11110000	000000000000000
GPRS (GEA3)	32 bit Frame Dependent Input Value	00000	0	00	11111111	000000000000000

NOTE

It is the responsibility of the user to ensure that fields of the KEU IV_1 register are programmed correctly in accordance with the algorithm selected.

17.7.5.11 KEU ICV_In Register (KEUICV)

If ICV checking is required, then the value to be compared with the computed f9 MAC value must be written to the KEU ICV_In register before data size is written. As the KEU ICV_In register is in between IV_1 and IV_2, any descriptor operation that loads IV_2 must also load ICV_In. If CICV = 0, the ICV_In register should be loaded with 0x0000_0000_0000_0000.

17.7.5.12 KEU IV_2 Register (FRESH) (KEUIV2)

The KEU IV_2 register, shown in Figure 17-72, holds the f9 value, FRESH, which is used during the initialization phase of the 3GPP f9 algorithm. This value is ignored when the f8 algorithm is selected. The FRESH value must be written to bits 0:31 of the KEU IV_2 register before a new message to be processed with 3GPP f9 is started. After the initialization phase has been completed, the KEU IV_2 register is no longer used during message processing. The KEU IV_2 register need not be written during context switches.

	0 — 31	32 — 63
Field	FRESH	00000000
Reset	0	
R/W	R/W	
Addr	KEU 0x3_E110	

Figure 17-72. KEU IV_2 Register (FRESH)

17.7.5.13 KEU Context Data Registers (KEUC*n*)

The KEU includes six 64-bit KEU context data registers that store the running context used to process a message. The KEU context data registers must be read when changing context and are restored to their original values to resume processing of a partial message. For f8 and 3GPP f9 modes, all 64-bit KEU context data registers must be read to retrieve context, and all six registers must be written back to restore context. The context must be written prior to the key data. If the any of the KEU context data registers are written during message processing, a context error is generated. All KEU context data registers are cleared when a hard/soft reset or initialization is performed.

NOTE

For descriptor operation, if the entire context is unloaded for later reuse, the context data size must be 72 bytes, and the output consists of KEU IV_1, KEU ICV_In, KEU IV_2, and six KEU context data registers. For operations performing processing of partial messages, if the context is unloaded, the PE bit in the KEU mode register must not be set. Also, for partial message processing, if the context is reloaded, the INT bit in the KEU mode register must not be set.

17.7.5.14 KEU Key Data Registers_1 and _2 (Confidentiality Key) (KEUKD*n*)

The first two KEU key data registers, shown in Figure 17-74 and Figure 17-76, together hold one 128-bit key that is used for f8 encryption/decryption. The KEU key data register_1, (CK-high), holds the first 8 bytes (1–8). The KEU key data register_2, (CK-low), holds the second 8 bytes (9–16). The KEU key data registers must be written before message processing begins and cannot be written while the block is processing data, or else, a context error occurs. Reading from either of these registers causes an address error, which is reflected in the KEU interrupt status register.

Address KEU 0x3_E400 — Access: Write-only

	0 — 63
R	
W	KEU Key Data Register _1 (CK-high)
Reset	All zeros

Figure 17-73. KEU Key Data Register_1 (CK-high)

Figure 17-74. KEU Key Data Register_1 (CK-high)

Address KEU 0x3_E408 Access: Write-only

	0 … 63
R	
W	KEU Key Data Register _2 (CK-low)
Reset	All zeros

Figure 17-75. KEU Key Data Register_2 (CK-Low)

Figure 17-76. KEU Key Data Register_2 (CK-Low)

17.7.5.15 KEU Key Data Registers _3 and _4 (Integrity Key) (KEUKD*n*)

The third and fourth KEU key data registers, shown in Figure 17-78 and Figure 17-80, together hold one 128-bit key that is used for f9 message authentication. The KEU key data register_3, (IK-high), holds the first 8 bytes (1–8). The KEU key data register_4, (IK-low), holds the second 8 bytes (9–16). The KEU key data registers must be written before message processing begins and cannot be written while the block is processing data, or else, a context error occurs.

If f9 only mode is set in the KEU mode register, the integrity key data may be optionally written to the KEU key data registers_1 and KEU key data registers_2. This eliminates the need for the host to offset from the base key address to write to the KEU key data registers_3 and KEU key data registers_4 while using the KEU exclusively for the f9 integrity function.

Reading from either of these registers causes an address error, which is reflected in the KEU interrupt status register.

Address KEU 0x3_E410 Access: Write-only

	0 … 63
R	
W	KEU Key Data Register _3 (IK-high)
Reset	All zeros

Figure 17-77. KEU Key Data Register_3 (IK-high)

Figure 17-78. KEU Key Data Register_3 (IK-high)

Address KEU 0x3_E418 Access: Read/write

	0 … 63
R	
W	KEU Key Data Register _4 (IK-low)
Reset	All zeros

Figure 17-79. KEU Key Data Register_4 (IK-low)

Figure 17-80. KEU Key Data Register_4 (IK-low)

17.7.5.16 KEU FIFOs

KEU uses an input FIFO/output FIFO pair to hold data before and after the encryption process. Normally, the channels control all access to these FIFOs. For host-controlled operation, a write to anywhere in the

KEU FIFO address space enqueues data to the KEU input FIFO, and a read from anywhere in the KEU FIFO address space de-queues data from the KEU output FIFO.

A write to the input FIFO go first to a staging register, which can be written by byte, word (4 bytes), or dword (8 bytes). When all 8 bytes of the staging register have been written, the entire dword is automatically enqueued into the FIFO. If any byte is written twice between enqueues, it causes an error interrupt of type AE from the EU. When writing the last portion of data, it is not necessary to write all 8 bytes. The last bytes remaining in the staging register are automatically padded with zeros and forced into the input FIFO when the KEU end of message register is written.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword are read, the dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between de-queues, it causes an error interrupt of type AE from the EU.

Overflows and underflows caused by reading or writing the KEU FIFOs are reflected in the KEU interrupt status register.

The KEU fetches data 64 bits at a time from the KEU Input FIFO. During f8 processing, the input data is XORed with the generated keystream and the results are placed in the KEU output FIFO. During f9 processing, the input data is hashed with the integrity key and the resulting MAC is placed in the KEU data out register. The output size is the same as the input size.

17.7.6 Message Digest Execution Unit (MDEU)

This section contains details about the Message Digest Execution Unit (MDEU), including modes of operation, status and control registers, and FIFO.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the MDEU is used through channel-controlled access, which means that most reads and writes of MDEU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

17.7.6.1 ICV Checking in MDEU

This EU includes an ICV checking feature, that is, it can generate an ICV and compare it to another supplied ICV. The pass/fail result of this ICV check can be returned to the host either by interrupt by a writeback of EU status fields into host memory, but not by both methods at once.

To signal the ICV checking result by status writeback, turn on either the IWSE bit or AWSE bit in the Channel Configuration Register (see Section 17.4.4.1, “Channel Configuration Register (CCR)”), and mask the ICE bit in the interrupt mask register (Section 17.7.6.9, “MDEU Interrupt Mask Register”). In this case the normal done signaling (by interrupt or writeback) is undisturbed.

To signal the ICV checking result by interrupt, unmask the ICE bit in the interrupt mask register and turn off the IWSE and AWSE bits in the Channel Configuration Register. If there is no ICV mismatch, then the normal done signaling (by interrupt or writeback) occurs. When there is an ICV mismatch, there is an error interrupt to the host, but no channel done interrupt or writeback.

17.7.6.2 MDEU Mode Register

The MDEU Mode Register is used to program the function of the MDEU. In channel-driven access, bits 56-63 of this register are specified by the user through the MODE0 or MODE1 field of the descriptor header. The remaining two bits are supplied by the channel and thus are not under direct user control.

The two bits supplied by the channel are bits that control the meanings of other mode register fields. They are the MDEU_B bit, and the NEW bit.

The MDEU_B bit determines which of two sets of algorithms is available through the ALG bits. The two sets of algorithms are referred to as the MDEU-A set (MD5, SHA-1, SHA-224, and SHA-256) and the MDEU-B set (SHA-224, SHA-256, SHA-384, and SHA-512). MDEU_B = 0 selects the MDEU-A set, and MDEU_B = 1 selects the MDEU-B set. In channel-driven operation, the MDEU_B mode bit is supplied by the channel, based on the EU_SEL field of the descriptor header, where the user can choose MDEU-A or MDEU-B (see Table 17-6).

The NEW bit determines the configuration of other mode register fields as shown in Figure 17-81 and Figure 17-82. The "new" configuration (NEW=1) is used only by TLS/SSL descriptor types (1000_1, 1001_1). The old configuration (NEW=0) is used by all other descriptor types. The old configuration is the same as the one used in SEC 2.0, except for the CICV and SMAC bits. When MDEU is configured by the Polychannel, the value of NEW is determined by the descriptor type field of the descriptor header.

The mode register is cleared when the MDEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error is generated.

	0–50	51	52–53	54	55	56	57	58	59	60	61	62–63
Field	—	MDEU_B	—	NEW=0	—	CONT	CICV	SMAC	INIT	HMAC	PD	ALG
Reset	0											
R/W	R/W											
Addr	MDEU 0x3_6000											

Figure 17-81. MDEU Mode Register in Old Configuration

Table 17-60 describes MDEU Mode Register fields in old configuration.

Table 17-60. MDEU Mode Register in Old Configuration

Bits	Name	Description
The following bits are described for information only. They are not under direct user control.		
0–50	—	Reserved
51	MDEU_B	Selects which algorithms are enabled by the ALG bits. 0 MDEU-A enables selection between SHA-1, SHA-256, MD5, and SHA-224 1 MDEU-B enables selection between SHA-384, SHA-256, SHA-512, and SHA-224.
52-53	—	Reserved, must be cleared.
54	NEW (=0)	Determines the configuration of the MDEU Mode Register. This table shows the configuration for NEW=0.
55	—	Reserved, must be cleared.

Table 17-60. MDEU Mode Register in Old Configuration (continued)

Bits	Name	Description
The following bits are controlled through the MODE0 or MODE1 fields of the descriptor header.		
56	CONT	Continue: Most operations require this bit to be cleared. It is set only when the data to be hashed is spread across multiple descriptors. The value programmed in PD must be opposite to the value in this bit. 0 Do autopadding and complete the message digest. Used when the entire hash is performed with one descriptor, or on the last of a sequence of descriptors. 1 This hash is continued in a subsequent descriptor. Do not autopad and do not complete the message digest.
57	CICV	Compare Integrity Check Values: 0 Normal operation; no ICV checking. 1 After the message digest (ICV) is computed, compare it to the data in the MDEU's input FIFO. If the ICVs do not match, send an error interrupt to the channel. The number of bytes to be compared is given by the ICV size register. Only applicable to descriptor types that provide for reading an ICV in value.
58	SMAC	Specifies whether to perform an SSL-MAC operation: 0 Normal operation 1 Perform an SSL3.0 MAC operation. This requires a key and key length. If this is set then the HMAC bit should be 0.
59	INIT	Initialization Bit: Most operations require this bit to be set. Cleared only for operations that load context from a known intermediate hash value. 0 Do not initialize digest registers. In this case the registers must be loaded from a hash context pointer in the descriptor. When the data to be hashed is spread across multiple descriptors, this bit must be 0 on all but the first descriptor. 1 Do an algorithm-specific initialization of the digest registers.
60	HMAC	Specifies whether to perform an HMAC operation: 0 Normal operation 1 Perform an HMAC operation. This requires a key and key length. If this is set then the SMAC bit should be 0.
61	PD	This bit must be programmed opposite to the CONT bit.
62-63	ALG	Message Digest algorithm selection 00 if MDEU-B, then SHA-384. If MDEU-A, then SHA-160 algorithm (full name for SHA-1) 01 SHA-256 algorithm 10 if MDEU-B, then SHA-512. If MDEU-A, then MD5 algorithm 11 SHA-224 algorithm

	0–50	51	52	53	54	55	56	57	58	59	60	61	62–63
Field	—	MDEU_B	—	STIB	NEW=1	—	CONT	CICV	SMAC	INIT	HMAC	EALG	ALG
Reset	0												
R/W	R/W												
Addr	MDEU 0x3_6000												

Figure 17-82. MDEU Mode Register in New Configuration

Table 17-61 describes MDEU Mode Register fields in new configuration.

Table 17-61. MDEU Mode Register in New Configuration

Bits	Name	Description
The following bits are described for information only. They are not under direct user control.		
0–50	—	Reserved
51	MDEU_B	Selects which algorithms are enabled by the ALG bits. 0 MDEU-A enables selection between SHA-1, SHA-256, MD5, and SHA-224 1 MDEU-B enables selection between SHA-384, SHA-256, SHA-512, and SHA-224.
52	—	Reserved, must be cleared.
53	STIB	SSL/TLS inbound, block cipher: 0 Normal operation. 1 Special operation only for SSL/TLS inbound, block cipher. Upon receiving end of message notification, the MDEU performs a calculation involving the last valid byte of data written into its input FIFO (which is Pad Length) to compute a final data size. The MDEU then processes the amount of data specified by this data size, and completes the message digest.
54	NEW (=1)	Determines the configuration of the MDEU Mode Register. This table shows the configuration for NEW=1.
55	—	Reserved, must be cleared.
The following bits are controlled through the MODE0 or MODE1 fields of the descriptor header.		
56	CONT	Continue: Most operations require this bit to be cleared. Set only when the data to be hashed is spread across multiple descriptors. 0 Do autopadding and complete the message digest. Used when the entire hash is performed with one descriptor, or on the last of a sequence of descriptors. 1 This hash is continued in a subsequent descriptor. Do not autopad and do not complete the message digest.
57	CICV	Compare Integrity Check Values: 0 Normal operation; no ICV checking. 1 After the message digest (ICV) is computed, compare it to the data in the MDEU's input FIFO. If the ICVs do not match, send an error interrupt to the channel. The number of bytes to be compared is given by the ICV size register.
58	SMAC	Specifies whether to perform an SSL-MAC operation: 0 Normal operation 1 Perform an SSL3.0 MAC operation. This requires a key and key length. If this is set then the HMAC bit should be 0.
59	INIT	Initialization Bit: Most operations require this bit to be set. Cleared only for operations that load context from a known intermediate hash value. 0 Do not initialize digest registers. In this case the registers must be loaded from a hash context pointer in the descriptor. When the data to be hashed is spread across multiple descriptors, this bit is set on all but the first descriptor. 1 Do an algorithm-specific initialization of the digest registers.
60	HMAC	Specifies whether to perform an HMAC operation: 0 Normal operation 1 Perform an HMAC operation. This requires a key and key length. If this is set then the SMAC bit should be 0.

Table 17-61. MDEU Mode Register in New Configuration (continued)

Bits	Name	Description
61	EALG	The EALG (Extended Algorithm bit) and ALG (Algorithm) bits together specify the message digest algorithm, as follows: 000 if MDEU-B, then SHA-384. If MDEU-A, then SHA-160 algorithm (full name for SHA-1) 001 SHA-256 algorithm 010 if MDEU-B, then SHA-512. If MDEU-A, then MD5 algorithm 011 SHA-224 algorithm others: Reserved
62-63	ALG	

17.7.6.3 Recommended Settings for MDEU Mode Register

The most common task likely to be executed by the MDEU is HMAC generation. HMACs are used to provide message integrity within a number of security protocols, including IPsec, and TLS. The SSL 3.0 protocol uses a slightly different SSL-MAC. If an HMAC or SSL-MAC is to be performed using a single descriptor (with the MDEU acting as sole or secondary EU), the following mode register bit settings should be used:

Table 17-62. Mode Register—HMAC or SSL-MAC Generated by Single Descriptor

Bits	Field	Value	
		for HMAC	for SSL-MAC
56	CONT	0 (off)	0 (off)
58	SMAC	0(on)	1(on)
59	INIT	1(on)	1(on)
60	HMAC	1(on)	0(on)

To generate an HMAC for a message that is spread across a sequence of descriptors, the following mode register bit settings should be used:

Table 17-63. Mode Register—HMAC Generated Across a Sequence of Descriptors

Bits	Field	Value		
		First Descriptor	Middle Descriptor(s)	Final Descriptor
56	CONT	1 (on)	1 (on)	0 (off)
59	INIT	1 (on)	0 (off)	0 (off)
60	HMAC	1 (on)	0 (off)	1 (on)

All descriptors other than the final descriptor must output the intermediate message digest for the next descriptor to reload as MDEU context.

SSL-MAC operations cannot be spread across a sequence of descriptors.

Additional information on descriptors can be found in Section 17.3, “Descriptors.”

17.7.6.4 MDEU Key Size Register

Displayed in Figure 17-83, this value indicates the number of bytes of key memory that should be used in HMAC generation. MDEU supports at most one block of key. MDEU generates a key size error if the value written to this register exceeds 64 bytes for MD5, SHA-1, SHA-224, or SHA-256. If algorithms SHA-384 or SHA-512 are selected, then MDEU generates a key size error if the value written to this register exceeds 128 bytes.

	0 – 55	56 – 63
Field	—	Key Size
Reset	0	
R/W	R/W	
Addr	MDEU 0x3_6008	

Figure 17-83. MDEU Key Size Register

17.7.6.5 MDEU Data Size Register

The MDEU Data Size Register, shown in Figure 17-84, specifies the number of bits of data to be processed.

The Data Size field is a 21-bit signed number. Values written to this register are added to the current register value. Multiple writes are allowed. The MDEU processes data when there is a positive value in this register and there is data available in the MDEU input FIFO. (Negative values can arise in inbound processing, when it is necessary to hold back data from the MDEU until the pad length has been decrypted.)

Since the MDEU does not support bit offsets, bits 61–63 must be written as 0 and are always read as zero. Furthermore, when the CONT bit of the MDEU mode register is set, the data size must be a multiple of the block size (512 bits for MD5, SHA-1, SHA-224 and SHA-256; 1024 bits for SHA-384 and SHA-512). Violating either of these conditions causes a data size error (DSE in the MDEU interrupt status register).

This register is cleared when the MDEU is reset or re-initialized. At the end of processing, its contents has been decremented down to zero (unless there is an error interrupt).

NOTE

Writing to the data size register allows the MDEU to enter auto-start mode. Therefore, the required context registers must be written prior to writing the data size.

	0–42	43–63
Field	—	Data Size
Reset	0	
R/W	R/W	
Addr	MDEU 0x3_6010	

Figure 17-84. MDEU Data Size Register

17.7.6.6 MDEU Reset Control Register

This register, shown in Figure 17-85, allows three levels reset of just the MDEU, as defined by the three self-clearing bits.

Offset 0x3_6018 Access: Read/Write

	0–60	61	62	63
R/W	—	RI	MI	SR
Reset	All zeros			

Figure 17-85. MDEU Reset Control Register

Table 17-64 describes MDEU reset control register fields.

Table 17-64. MDEU Reset Control Register Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	RI	Reset Interrupt. Writing this bit active high causes MDEU interrupts signaling done and error to be reset. It further resets the state of the MDEU interrupt status register. 0 No reset 1 Reset interrupt logic
62	MI	Module initialization is nearly the same as software reset, except that the MDEU Interrupt mask register remains unchanged. 0 No reset 1 Reset most of MDEU
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the MDEU. All registers and internal state are returned to their defined reset state. 0 No reset 1 Full MDEU reset

17.7.6.7 MDEU Status Register

This status register, as seen in Figure 17-86, reflects the state of the MDEU internal signals. The majority of these internal signals reflect the state of low-level MDEU functions, such as data padding, key padding, etc., and are not important to the user, however the user should be aware that reads of this register especially during processing are likely to return non-zero values for many bits between 0:57. The 4 signals shown are those which are most likely to be of interest to the user.

The MDEU status register is read only. Writing to this location results in address error being reflected in the MDEU interrupt status register.

Offset 0x3_6028 Access: Read only

	0–57	58	59–60	61	62	63
R	—	HALT	ICCR	EI	DI	RD
W						
Reset	All zeros					

Figure 17-86. MDEU Status Register

Table 17-65 describes MDEU status register fields.

Table 17-65. MDEU Status Register Field Descriptions

Bits	Name	Description
0–57	—	Reserved
58	HALT	Halt. Indicates that the MDEU has halted due to an error. 0 MDEU not halted 1 MDEU halted **Note:** Because the error causing the MDEU to stop operating may be masked before reaching the interrupt status register, the MDEU interrupt status register is used to provide a second source of information regarding errors preventing normal operation.
59-60	ICCR	Integrity Check Comparison Result 00 No integrity check comparison was performed. 01 The integrity check comparison passed. 10 The integrity check comparison failed. 11 Reserved **Note:** A “passed” or “failed” result is generated only if ICV checking is enabled.
61	EI	Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 MDEU is not signaling error 1 MDEU is signaling error
62	DI	Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the Controller interrupt status register (Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). 0 MDEU is not signaling done 1 MDEU is signaling done
63	RD	Reset Done. This status bit, when high, indicates that MDEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel. 0 Reset in progress 1 Reset done **Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation.

17.7.6.8 MDEU Interrupt Status Register

The interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the MDEU interrupt mask register is zero (see Section 17.7.6.9, “MDEU Interrupt Mask Register”).

If the MDEU interrupt status register is non-zero, the MDEU halts and the MDEU error interrupt signal is asserted to the controller (see Section 17.5.4.2.2, “Interrupt Status Register (ISR)”). In addition, if the MDEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the Channel Status Register (see Table 17-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the MDEU reset control register.

The MDEU interrupt status register fields are shown in Figure 17-87, and described in Table 17-66.

Offset 0x3_6030 — Access: Read/Write

Bits	0–48	49	50	51	52	53	54	55	56	57	58–60	61	62–63
R/W	—	ICE	—	IE	ERE	CE	KSE	DSE	ME	AE	—	IFO	—
Reset	All zeros												

Figure 17-87. MDEU Interrupt Status Register

Table 17-66 describes MDEU interrupt status register fields.

Table 17-66. MDEU Interrupt Status Register Field Descriptions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity Check Error: 0 No error detected 1 Integrity check error detected. An ICV check was performed and the supplied ICV did not match the one computed by the MDEU.
50	—	Reserved
51	IE	Internal Error. Indicates the MDEU has been locked up and requires a reset before use. 0 No internal error detected 1 Internal error detected **Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by resetting the MDEU.
52	ERE	Early Read Error. The MDEU context was read before the MDEU completed the hashing operation. 0 No error detected 1 Early read error
53	CE	Context Error. The MDEU key register, key size register, or data size register was modified while MDEU was hashing. 0 No error detected 1 Context error

Table 17-66. MDEU Interrupt Status Register Field Descriptions (continued)

Bits	Name	Description
54	KSE	Key Size Error. Two possible causes: • A value greater than permitted was written to the MDEU key size register (128 bytes for SHA-384 and SHA-512; 64 bytes otherwise) • In either a HMAC or SMAC operation, key size was not written prior to writing data size or receiving an end of message command. 0 No error detected 1 Key size error
55	DSE	Data Size Error. A value not a multiple of 512 bits (1024 bits for SHA-384 and SHA-512) while the MDEU mode register CONT bit is high. 0 No error detected 1 Data size error
56	ME	Mode Error. Bit is set if any of these error conditions is detected: • any reserved bit of the Mode register is set • the ALG field of the Mode register contains an illegal value 0 No error detected 1 Mode error
57	AE	Address Error. An illegal read or write address was detected within the MDEU address space. 0 No error detected 1 Address Error
58–60	—	Reserved
61	IFO	Input FIFO Overflow. The MDEU Input FIFO was pushed while full. 0 No overflow detected 1 Input FIFO has overflowed **Note:** When operated through channel-controlled access, the SEC implements flow control, and FIFO size is not a limit to data input size. When operated through host-controlled access, the MDEU cannot accept FIFO inputs larger than 256 bytes without overflowing.
62–63	—	Reserved

17.7.6.9 MDEU Interrupt Mask Register

The MDEU interrupt mask register, shown in Figure 17-88, controls the result of detected errors. For a given error (as defined in Section 17.7.6.8, “MDEU Interrupt Status Register”), if the corresponding bit in this register is set, then the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

Offset 0x3_6038 Access: Read/Write

Bits	0–48	49	50	51	52	53	54	55	56	57	58–60	61	62–63
R/W	—	ICE	—	IE	ERE	CE	KSE	DSE	ME	AE	—	IFO	—

Figure 17-88. MDEU Interrupt Mask Register

Table 17-66 describes MDEU interrupt status register fields.

Table 17-67. MDEU Interrupt Mask Register Field Descriptions

Bits	Name	Description
0–48	—	Reserved
49	ICE	Integrity Check Error. The supplied ICV did not match the one computed by the MDEU. 0 Integrity check error enabled. WARNING: Do not enable this if using EU status writeback (see bits IWSE and AWSE in Section 17.4.4.1, “Channel Configuration Register (CCR)”). 1 Integrity check error disabled
50	—	Reserved
51	IE	Internal Error. An internal processing error was detected while performing hashing. 0 Internal error enabled 1 Internal error disabled
52	ERE	Early Read Error. The MDEU register was read while the MDEU was performing hashing. 0 Early read error enabled 1 Early read error disabled
53	CE	Context Error. The MDEU key register, the key size register, the data size register, or the mode register, was modified while the MDEU was performing hashing. 0 Context error enabled 1 Context error disabled
54	KSE	Key Size Error. A value outside the bounds was written to the MDEU key size register 0 Key size error enabled 1 Key size error disabled
55	DSE	Data Size Error. An inconsistent value was written to the MDEU data size register: 0 Data size error enabled 1 Data size error disabled
56	ME	Mode Error. An illegal value was detected in the mode register. 0 Mode error enabled 1 Mode error disabled
57	AE	Address Error. An illegal read or write address was detected within the MDEU address space. 0 Address error enabled 1 Address error disabled
58–60	—	Reserved
61	IFO	Input FIFO Overflow. The MDEU input FIFO was pushed while full. 0 Input FIFO overflow error enabled 1 Input FIFO overflow error disabled
62–63	—	Reserved

17.7.6.10 MDEU ICV Size Register

The MDEU ICV Size Register, shown in Figure 17-89, specifies the number of bytes of the ICV result to be compared if the MDEU performs ICV checking (see Section 17.7.6.2, “MDEU Mode Register”). The data to be compared to the MDEU result is supplied to the MDEU through its input FIFO.

This register is cleared when the MDEU is reset or re-initialized.

Offset 0x3_6040 — Access: Read/Write

	0–56	57–63
R/W	—	ICV_Size
Reset	All zeros	

Figure 17-89. MDEU ICV Size Register

17.7.6.11 MDEU End of Message Register

The MDEU end of message register, shown in Figure 17-90, is used to signal to the MDEU that the final message block has been written to the input FIFO (in channel-driven access, this signaling is done automatically). The MDEU will not process the last block of data in its input FIFO until this register is written.

The value written to this register does not matter: ordinarily, zero is written. A read of this register always returns a zero value.

Offset 0x3_6050 — Access: Write only

	0–63
R/W	—
Reset	All zeros

Figure 17-90. MDEU End of Message Register

17.7.6.12 MDEU Context Registers

For MDEU, context consists of the hash plus the message length count. Write access to this register block allows continuation of a previous hash. Reading these registers provide the resulting message digest or HMAC, along with an aggregate bit count.

NOTE

All SHA algorithms are big endian. MD5 is little endian. The MDEU module internally reverses the byte order of the five registers A, B, C, D, and E upon writing to or reading from the MDEU context if the MDEU mode register indicates MD5 is the hash of choice. Most other endian considerations are performed as 8-byte swaps. In this case, 4-byte endianness swapping is performed within the A, B, C, D, and E fields as individual registers. Reading this memory location while the module is not done generates an error interrupt.

After a power on reset, all the MDEU context register values are cleared to 0. Figure 17-91 shows how the MDEU context registers are initialized if the INIT bit is set in the MDEU mode register. All registers are initialized, regardless of mode selected, however only the appropriate context register values are used in hash generation per the mode selected. The user typically doesn't care about the MDEU context register initialization values; they are documented for completeness in the event the user reads these registers using host-controlled access. MDEU reset through the MDEU reset control register (Figure 17-85) or SEC global software reset (Figure 17-20) does not clear these registers.

Algorithm	0–31	32–63	Register
Algorithm			Context offset 0x3_6100
MD-5	A = 0x01234567	B = 0x89ABCDEF	
SHA-1	A = 0x67452301	B = 0xEFCDAB89	
SHA-224	A = 0xC1059ED8	B = 0x367CD507	
SHA-256	A = 0x6A09E667	B = 0xBB67AE85	
SHA-384	A = 0xcbbb9d5dc1059ed8		
SHA-512	A = 0x6a09e667f3bcc908		
Algorithm			Context offset 0x3_6108
MD-5	C = 0xFEDCBA98	D = 0x76543210	
SHA-1	C = 0x98BADCFE	D = 0x10325476	
SHA-224	C = 0x3070DD17	D = 0xF70E5939	
SHA-256	C = 0x3C6EF372	D = 0xA54FF53A	
SHA-384	B = 0x629a292a367cd507		
SHA-512	B = 0xbb67ae8584caa73b		
Algorithm			Context offset 0x3_6110
MD-5	E = 0xF0E1D2C3	F = 0x8C68059B	
SHA-1	E = 0xC3D2E1F0	F = 0x9B05688C	
SHA-224	E = 0xFFC00B31	F = 0x68581511	
SHA-256	E = 0x510E527F	F = 0x9B05688C	
SHA-384	C = 0x9159015a3070dd17		
SHA-512	C = 0x3c6ef372fe94f82b		

Figure 17-91. MDEU Context Registers

Algorithm			
MD-5	G = 0xABD9831F	H = 0x19CDE05B	Context offset 0x3_6118
SHA-1	G = 0x1F83D9AB	H = 0x5BE0CD19	
SHA-224	G = 0x64F98FA7	H = 0xBEFA4FA4	
SHA-256	G = 0x1F83D9AB	H = 0x5BE0CD19	
SHA-384	D = 0xh152fecd8hf70e5939		
SHA-512	D = 0xha54ff53ah5f1d36f1		
Algorithm			Context offset 0x3_6120
MD5, SHA1, SHA-224, SHA-256	Message Length Count = 0		
SHA-384	E = 0x67332667ffc00b31		
SHA-512	E = 0x510e527fade682d1		
Algorithm			Context offset 0x3_6128
MD5, SHA1, SHA-224, SHA-256	reserved		
SHA-384	F = 0x8eb44a8768581511		
SHA-512	F = 0x9b05688c2b3e6c1f		
Algorithm			Context offset 0x3_6130
MD5, SHA1, SHA-224, SHA-256	reserved		
SHA-384	G = 0xdb0c2e0d64f98fa7		
SHA-512	G = 0x1f83d9abfb41bd6b		
Algorithm			Context offset 0x3_6138
MD5, SHA1, SHA-224, SHA-256	reserved		
SHA-384	H = 0x47b5481dbefa4fa4		
SHA-512	H = 0x5be0cd19137e2179		
Algorithm			Context offset 0x3_6140
MD5, SHA1, SHA-224, SHA-256	reserved		
SHA-384, SHA-512	Message Length Count = 0		

Figure 17-91. MDEU Context Registers (continued)

If SHA-384 or SHA-512 are selected, then each of the registers A, B, C, D, E, F, G, H are 64-bits (instead of 32 bits for other hash algorithms). As a result, the base address for each context register is shifted to adjust.

17.7.6.13 MDEU Key Registers

The MDEU maintains sixteen 64-bit registers for writing an HMAC key; only the first eight are used for MD5, SHA-1, SHA-224, or SHA-256. The IPAD and OPAD operations are performed automatically on the key data when required.

NOTE

All SHA algorithms are big endian. MD5 is little endian. The MDEU module internally reverses the endianness of the key upon writing to or reading from the MDEU key registers if the MDEU mode register indicates MD5 is the hash of choice.

17.7.6.14 MDEU FIFOs

MDEU uses an input FIFO to hold data to be hashed (followed in some case by an ICV value for ICV checking). Normally, the channels control all access to this FIFO. For host-controlled operation, a write to anywhere in the MDEU FIFO address space enqueues data to the MDEU input FIFO, and a read from anywhere in this address space returns all zeros.

When the host writes to the MDEU FIFO (using host-controlled access), it can write to any FIFO address by byte, word (4 bytes), or dword (8 bytes). The MDEU assembles these bytes from left to right, so that the first bytes written are placed in the most significant bit-positions. Whenever the MDEU accumulates 8 bytes, this dword is automatically enqueued into the FIFO, and any remaining bytes are left-justified in preparation for assembling the next dword. It is not necessary to fill all bytes of the final dword. Any last bytes remaining in the staging register are automatically padded with zeros and forced into the input FIFO when the MDEU end of message register is written.

Overflows caused by writing the MDEU FIFO are reflected in the MDEU interrupt status register.

17.7.7 Public Key Execution Units (PKEU)

This section contains details about the public key execution unit (PKEU), including modes of operation, status and control registers, and parameter RAMs.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the PKEU is used through channel-controlled access, which means that most reads and writes of PKEU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

17.7.7.1 PKEU Mode Register

This register specifies the internal PKEU routine to be executed. The mode register is cleared when the PKEU is reset or re-initialized. Setting a reserved mode bit generates a data error. If the mode register is modified during processing, a context error is generated.

Figure 17-92 shows the PKEU Mode Register, and Table 17-68 lists the possible values for the ROUTINE field.

Offset 0x3_C000 Access: Read/Write

	0–55	56–63
R / W	—	ROUTINE
Reset	All zeros	

Figure 17-92. PKEU Mode Register

For channel-controlled access to the PKEU, the descriptor type is determined by the ROUTINE to be used. The descriptor type used with each ROUTINE is listed in Table 17-68.

17.7.7.2 PKEU Key Size Register

The key size register specifies the number of significant bytes to be used from PKEU Parameter Memory E in performing modular exponentiation or elliptic curve point multiplication. The range of values for this register, when performing either modular exponentiation or elliptic curve point multiplication, is from 1 to 512. Specifying a key size outside of this range causes a key size error (KSE) in the PKEU interrupt status register.

	0–51	52–63
Field	—	Key Size
Reset	0	
R/W	R/W	
Addr	PKEU 0x3_C008	

Figure 17-93. PKEU Key Size Register

Table 17-68. ROUTINE Field Description

Mode [56-63]	Routine Name	Routine Description	Descriptor Type
0x00	RESERVED	Reserved	NA
0x01	CLEARMEMORY	Clear memory	pkeu_mm
0x02	MOD_EXP	FP: Exponentiate mod N and deconvert from Montgomery format	pkeu_mm
0x03	MOD_R2MODN	FP: Compute Montgomery converter (R^2 mod N)	pkeu_mm
0x04	MOD_RRMODP	FP: Compute Montgomery converter for Chinese Remainder Theorem (R_nR_p mod N)	pkeu_mm
0x05	EC_FP_AFF_PTMULT	FP EC: Multiply scalar times point in affine coordinates	pkeu_ptmul

Table 17-68. ROUTINE Field Description (continued)

Mode [56-63]	Routine Name	Routine Description	Descriptor Type
0x06	EC_F2M_AFF_PTMULT	F2m EC: Multiply scalar times point in affine coordinates	pkeu_ptmul
0x07	EC_FP_PROJ_PTMULT	FP EC: Multiply scalar times point in projective coordinates	pkeu_ptmul
0x08	EC_F2M_PROJ_PTMULT	F2m EC: Multiply scalar times point in projective coordinates	pkeu_ptmul
0x09	EC_FP_ADD	FP EC: Add two points in projective coordinates	pkeu_ptadd_dbl
0x0A	EC_FP_DOUBLE	FP EC: Double a point in projective coordinates	pkeu_ptadd_dbl
0x0B	EC_F2M_ADD	F2m EC: Add two points in projective coordinates	pkeu_ptadd_dbl
0x0C	EC_F2M_DOUBLE	F2m EC: Double a point in projective coordinates	pkeu_ptadd_dbl
0x0D	F2M_R2	F2m: Compute Montgomery converter (R^2 mod N)	pkeu_mm
0x0E	F2M_INV	F2m: Invert mod N	pkeu_mm
0x0F	MOD_INV	FP: Invert mod N	pkeu_mm
0x10	MOD_ADD	FP: Add mod N	pkeu_mm
0x20	MOD_SUB	FP: Subtract mod N	pkeu_mm
0x30	MOD_MULT1_MONT	FP: Multiply mod N in Montgomery format	pkeu_mm
0x40	MOD_MULT2_DECONV	FP: Multiply mod N and deconvert from Montgomery format	pkeu_mm
0x50	F2M_ADD	F2m: Add mod N	pkeu_mm
0x60	F2M_MULT1_MONT	F2m: Multiply mod N in Montgomery format	pkeu_mm
0x70	F2M_MULT2_DECONV	F2m: Multiply mod N and deconvert from Montgomery format	pkeu_mm
0x80	RSA_SSTEP	FP: Exponentiate mod N (combines MOD_R2MODN, POLY_F2M_MULT1_MONT, and MOD_EXP)	pkeu_mm
0x1d	MOD_EXP_TEQ	FP: Exponentiate mod N and deconvert from Montgomery format with timing equalization	pkeu_mm
0x1e	RSA_SSTEP_TEQ	FP: Exponentiate mod N with timing equalization (combines MOD_R2MODN, EC_F2M_MULT1_MONT, and MOD_EXP_TEQ)	pkeu_mm
0xFF	SPK_BUILD	Build PK data structure (data structure used by all elliptic curve routines)	pkeu_build

17.7.7.3 PKEU AB Size Register

The AB size register (Figure 17-94) represents the size of each operand written into parameter memory A and parameter memory B in bits. An exact size in bits must be provided since a big- to little-endian re-alignment is performed based on this value. No error checking is performed as to whether the operand sizes are greater than the prime modulus or the field size written in N-ram. In other words, it is assumed that operands are modular reduced before being written into the PKEU module. This register must be written to before each write to parameter memory A or parameter memory B and must be written before each read of parameter memory A and parameter memory B if the amount of data being taken out is different than the amount of data put in A or B. The value written to the AB size register must also adhere to the constraints on parameters A and B, set by the chosen routine (see Table 17-68). The AB size register should not be read or written to while the PKEU is processing as this causes a ‘data modify during processing error.’ This register is cleared when the PKEU is reset. The maximum size acceptable is 4096 bits.

Offset 0x3_C040 Access: Read/Write

	0–51	52–63
R/W	—	AB Size
Reset	All zeros	

Figure 17-94. PKEU AB Size Register

17.7.7.4 PKEU Data Size Register

The PKEU Data Size Register, Figure 17-95, specifies the size (in bits) of the significant portion of the modulus or irreducible polynomial. The minimum size valid for all routines to operate properly is 33 bits. The maximum size to operate properly is 4096 bits. A value in bits larger than 4096 results in a data size error (see the DSE bit in Table 17-71).

Offset 0x3_C010 Access: Read/Write

	0–51	52–63
R/W	—	Data Size
Reset	All zeros	

Figure 17-95. PKEU Data Size Register

17.7.7.5 PKEU Reset Control Register

This register, Figure 17-96, contains three reset options specific to the PKEU.

	0–60	61	62	63
Field	—	RI	MI	SR
Reset	0			
R/W	R/W			
Addr	PKEU 0x3_C018			

Figure 17-96. PKEU Reset Control Register

Table 17-69 describes the PKEU reset control register’s fields.

Table 17-69. PKEU Reset Control Register Field Descriptions

Bits	Name	Description
0–60	—	Reserved
61	RI	Reset interrupt. Writing this bit active high causes PKEU interrupts signaling done and error to be reset. It further resets the state of the PKEU interrupt status register. 0 Do not reset 1 Reset interrupt logic

Table 17-69. PKEU Reset Control Register Field Descriptions (continued)

Bits	Name	Description
62	MI	Module initialization. Module initialization is nearly the same as Software Reset, except that the interrupt mask register remains unchanged. This module initialization includes execution of an initialization routine, completion of which is indicated by the RESET_DONE bit in the PKEU status register (Section 17.7.7.6, “PKEU Status Register”). 0 Do not reset 1 Reset most of PKEU
63	SR	SW reset. Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the PKEU. All registers and internal state are returned to their defined reset state. Upon negation of SW_RESET, the PKEU enters a routine to perform proper initialization of the parameter memories. The RESET_DONE bit in the PKEU status register indicates when this initialization routine is complete (Section 17.7.7.6, “PKEU Status Register”). 0 Do not reset 1 Full PKEU reset
8–63	—	Reserved

17.7.7.6 PKEU Status Register

This status register contains six fields which reflect the state of PKEU internal fields.

The PKEU status register is read only. Writing to this location results in address error being reflected in the PKEU interrupt status register.

Offset 0x3_C028 Access: Read only

	0–55	56	57	58	59–60	61	62	63
R	ID	I	Z	HALT	—	EI	DI	RD
W								
Reset	All zeros							

Figure 17-97. PKEU Status Register

Table 17-70 describes the PKEU status register’s fields.

Table 17-70. PKEU Status Register Field Descriptions

Bits	Name	Description
0–55	—	Reserved
56	I	Infinity. This bit reflects the state of the PKEU infinity detect bit when last sampled. Only particular instructions within routines cause infinity to be modified, so this bit should be used with great care.
57	Z	Zero. This bit reflects the state of the PKEU zero detect bit when last sampled. Only particular instructions within routines cause zero to be modified, so this bit should be used with great care.

Table 17-70. PKEU Status Register Field Descriptions (continued)

Bits	Name	Description
58	HALT	Halt indicates that the PKEU has halted due to an error. 0 PKEU not halted 1 PKEU halted **Note:** Because the error causing the PKEU to stop operating may be masked before reaching the interrupt status register, the PKEU interrupt status register is used to provide a second source of information regarding errors preventing normal operation.
59-60	—	Reserved
61	EI	Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). 0 PKEU is not signaling error 1 PKEU is signaling error
62	DI	Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the Controller interrupt status register (Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). 0 PKEU is not signaling done 1 PKEU is signaling done
63	RD	Reset Done. This status bit, when high, indicates that PKEU has completed its reset sequence, as reflected in the signal sampled by the appropriate channel. 0 Reset in progress 1 Reset done **Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation.

17.7.7.7 PKEU Interrupt Status Register

The interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the PKEU interrupt mask register is zero (see Section 17.7.7.8, "PKEU Interrupt Mask Register").

If the PKEU interrupt status register is non-zero, the PKEU halts and the PKEU error interrupt signal is asserted to the controller (see Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). In addition, if the PKEU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the channel status register (see Table 17-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the PKEU reset control register.

The fields of the PKEU interrupt status register are shown in Figure 17-98, and described in Table 17-71.

Offset 0x3_C030 Access: Read/Write

	0–48	49	50	51	52	53	54	55	56	57	58–63
R/W	ID	EVM	INV	EI	BE	CE	KSE	DSE	ME	AE	—
Reset	All zeros										

Figure 17-98. PKEU Interrupt Status Register

Table 17-71 describes PKEU interrupt status register signals.

Table 17-71. PKEU Interrupt Status Register Field Descriptions

Bits	Name	Description
0-48	—	Reserved
49	EVM	Even modulus error. Indicates that an even modulus was supplied for a PK operation that requires an odd modulus. 0 No even modulus error detected 1 Even modulus error detected
50	INV	Inversion error. Indicates that the inversion routine has a zero operand. 0 No inversion error detected 1 Inversion error detected
51	IE	Internal error. An internal processing error was detected while the PKEU was operating. 0 No error detected 1 Internal error **Note:** This bit is asserted any time an enabled error condition occurs and can only be cleared by setting the corresponding bit in the interrupt mask register or by resetting the PKEU.
52	BE	Boot Error. Indicates that either at boot (reset) sequence, RAM locations are not reset correctly. 0 No boot error detected 1 Boot error
53	CE	Context error. A PKEU key register, the key size register, the data size register, or mode register was modified while the PKEU was operating. 0 No error detected 1 Context error
54	KSE	Key size error. Value outside the bounds of 1–256 bytes was written to the PKEU key size register 0 No error detected 1 Key size error detected
55	DSE	Data size error. Value outside the bounds 97– 4096 bits was written to the PKEU data size register 0 No error detected 1 Data size error detected
56	ME	Mode error. An illegal value was detected in the mode register. 0 No error detected 1 Mode error **Note:** Writing to reserved bits in a mode register is a likely source of error.

Table 17-71. PKEU Interrupt Status Register Field Descriptions (continued)

Bits	Name	Description
57	AE	Address error. Illegal read or write address was detected within the PKEU address space. 0 No error detected 1 Address error
58-63	—	Reserved

17.7.7.8 PKEU Interrupt Mask Register

The PKEU interrupt mask register (shown in Figure 17-99) controls the result of detected errors. For a given error (as defined in Section 17.7.7.7, "PKEU Interrupt Status Register"), if the corresponding bit in this register is set, then the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the PKEU interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

Offset 0x3_C038 — Access: Read/Write

	0–48	49	50	51	52	53	54	55	56	57	58–63
R	ID	EVM	INV	EI	BE	CE	KSE	DSE	ME	AE	—
W											

Figure 17-99. PKEU Interrupt Mask Register

Table 17-72 describes the PKEU interrupt mask register fields.

Table 17-72. PKEU Interrupt Mask Register Field Descriptions

Bits	Name	Description
0-48	—	Reserved
49	EVM	Even modulus error 0 Even modulus error enabled 1 Even modulus error disabled
50	INV	Inversion error 0 Inversion error enabled 1 Inversion error disabled
51	IE	Internal error 0 Internal error enabled 1 Internal error disabled
52	BE	Boot error 0 Boot error enabled 1 Boot error disabled
53	CE	Context error 0 Context error enabled 1 Context error disabled

Table 17-72. PKEU Interrupt Mask Register Field Descriptions (continued)

Bits	Name	Description
54	KSE	Key size error 0 Key size error enabled 1 Key size error disabled
55	DSE	Data size error 0 Data size error enabled 1 Data size error disabled
56	ME	Mode error 0 Mode error enabled 1 Mode error disabled
57	AE	Address error 0 Address error enabled 1 Address error disabled
58-63	—	Reserved

17.7.7.9 PKEU End of Message Register

The PKEU end of message register in the PKEU is used to indicate the start of a new computation. Writing to this register causes the PKEU to execute the function requested by the ROUTINE field, according to the contents of the parameter memories described in Section 17.7.7.10, "PKEU Parameter Memories".

The value written to this register does not matter: ordinarily, all zeros are written. A read of this register always returns a zero value.

Offset 0x3_C050 Access: Write only

	0–63
R	—
W	
Reset	All zeros

Figure 17-100. PKEU End of Message Register

17.7.7.10 PKEU Parameter Memories

The PKEU uses four 4096-bit memories to receive and store operands for the arithmetic operations the PKEU is asked to perform. In addition, results are stored in one particular parameter memory.

Data addressing within these memories is big-endian, that is, the most significant byte is stored in the lowest address.

17.7.7.10.1 PKEU Parameter Memory A

This 4096 bit memory is used typically as an input parameter memory space. For modular arithmetic routines, this memory operates as one of the operands of the desired function. For elliptic curve routines, this memory is segmented into four 1024 bit memories, and is used to specify particular curve parameters and input values.

17.7.7.10.2 PKEU Parameter Memory B

This 4096-bit memory is used typically as an input parameter memory space, as well as the result memory space. For modular arithmetic routines, this memory operates as one of the operands of the desired function, as well as the result memory space. For elliptic curve routines, this memory is segmented in to four 1024 bit memories, and is used to specify particular curve parameters and input values, as well as to store result values.

17.7.7.10.3 PKEU Parameter Memory E

This 4096-bit memory is non-segmentable, and specifies the exponent for modular exponentiation, or the multiplier k for elliptic curve point multiplication. This memory space is write only; a read of this memory space causes address error to be reflected in the PKEU interrupt status register.

17.7.7.10.4 PKEU Parameter Memory N

This 4096-bit memory is non-segmentable, and specifies the modulus for modular arithmetic and F_p elliptic curve routines. For F_2m elliptic curve routines, this memory specifies the irreducible polynomial.

17.7.8 Random Number Generator Unit (RNGU)

This section contains details about the random number generator unit, including modes of operation, status and control registers, and FIFO.

The RNGU is an execution unit capable of generating 64-bit random numbers. It contains a True Random Number Generator (TRNG).The RNGU is designed to comply with the FIPS-140 standard for randomness and non-determinism.

The RNGU consists of five major functional blocks:

- 64-bit internal bus interface, registers, and FIFO
- True Random Number Generator (ring oscillator, LFSRs, Statistical Checker)
- Xseed Generator
- Pseudo-Random Number Generator (XKEY, SHA-1, FSM)
- Simultaneous Reseed LFSR

The states of the LFSRs in the TRNG are advanced at an unknown frequency determined by the ring oscillator clock. The entropy generated by this structure is then added into the XKEY structure of the PRNG during seed generation. Seed generation takes approximately 2,000,000 cycles as 20,000 bits of entropy are sampled from the output of the LFSRs of the TRNG.

After the initial seeding, the RNGU turns off the TRNG and uses solely the PRNG to generate random data. After 1,000,000 times through the algorithm the RNGU is once again seeded. This second seed occurs the next time through the algorithm by using data from the Simultaneous Reseed LFSR to modify the algorithm. The data in the simultaneous reseed LFSR comes directly from the TRNG as well and was being generated during the first 20,000 times through the PRNG algorithm after the initial seed was completed.

Most of the registers described here would not normally be accessed by the host. They are documented here mainly for debug purposes. In typical operation, the RNGU is used through channel-controlled access, which means that most reads and writes of RNGU registers are directed by the SEC channels. Driver software would perform host-controlled register accesses only on a few registers for initial configuration and error handling.

17.7.8.1 RNGU Mode Register

The RNGU Mode Register is a writable location but all mode bits are currently reserved. It is documented for the sake of consistency with the other EUs. The RNGU mode register is shown in Figure 17-101.

Offset 0x3_A000 Access: Read/Write

	0–63
R	—
W	
Reset	All zeros

Figure 17-101. RNGU Mode Register

17.7.8.2 RNGU Data Size Register

The RNGU data size register is used to tell the RNGU to begin generating random data. The actual contents of the data size register does not affect the operation of the RNGU. After a reset and prior to the first write of data size, the RNGU builds entropy without pushing data onto the FIFO. Once the data size register is written, the RNGU begins pushing data onto the FIFO. One dword (64 bits) of data is pushed onto the FIFO every 112 cycles until the FIFO is full. The RNGU then attempts to keep the FIFO full.

Offset 0x3_A010 Access: Read/Write

	0–63
R	—
W	
Reset	All zeros

Figure 17-102. RNGU Data Size Register

17.7.8.3 RNGU Reset Control Register

This register, shown in Figure 17-103, contains three reset options specific to the RNGU.

Offset 0x3_A018 Access: Read/Write

	0–60	61	62	63
R	—	RI	MI	SR
W				
Reset	All zeros			

Figure 17-103. RNGU Reset Control Register

Table 17-73 describes RNGU reset control register fields.

Table 17-73. RNGU Reset Control Register Field Descriptions

Bits	Name	Description
0-60	—	Reserved
61	RI	Reset Interrupt. Writing this bit active high causes RNGU interrupts signaling done and error to be reset. It further resets the state of the RNGU interrupt status register. 0 No reset 1 Reset interrupt logic
62	MI	Module Initialization. This reset value performs enough of a reset to prepare the RNGU for another request, without forcing the internal control machines and the output FIFO to be reset, thereby invalidating stored random numbers or requiring reinvocation of a warm-up period. Module initialization is nearly the same as software reset, except that the interrupt mask register remains unchanged. 0 No reset 1 Reset most of RNGU
63	SR	Software reset is functionally equivalent to hardware reset (the RESET# pin), but only for the RNGU. All registers and internal state are returned to their defined reset state. 0 No reset 1 Full RNGU reset
8–63	—	Reserved

17.7.8.4 RNGU Status Register

This RNGU status register, Figure 17-104, contains 6 fields that reflect the state of the RNGU internal signals.

The RNGU status register is read only. Writing to this location results in an address error being reflected in the RNGU interrupt status register.

Offset 0x3_A028 Access: Read only

	0–39	40–47	48–57	58	59–60	61	62	63
R	—	OFL	—	HALT	—	EI	DI	RD
W								
Reset	All zeros							

Figure 17-104. RNGU Status Register

Table 17-74 describes RNGU status register fields.

Table 17-74. RNGU Status Register Field Descriptions

Bits	Name	Description
0–39	—	Reserved
40–47	OFL	The number of dwords currently in the output FIFO
48–57	—	Reserved

Table 17-74. RNGU Status Register Field Descriptions (continued)

Bits	Name	Description
58	HALT	Halt. Indicates that the RNGU has halted due to an error. 0 RNGU not halted 1 RNGU halted Note: Because the error causing the RNGU to stop operating may be masked before reaching the interrupt status register, the RNGU interrupt status register is used to provide a second source of information regarding errors preventing normal operation.
59–60	—	Reserved
61	EI	Error interrupt: This status bit reflects the state of the error interrupt signal, as sampled by the Controller interrupt status register (Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). 0 RNGU is not signaling error 1 RNGU is signaling error
62	DI	Done interrupt: This status bit reflects the state of the done interrupt signal, as sampled by the controller interrupt status register (Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). 0 RNGU is not signaling done 1 RNGU is signaling done
63	RD	Reset Done. This status bit, when high, indicates that the RNGU has completed its reset sequence. 0 Reset in progress 1 Reset done **Note:** Reset Done resets to 0, but has typically switched to 1 by the time a user checks the register, indicating the EU is ready for operation.

17.7.8.5 RNGU Interrupt Status Register

The RNGU interrupt status register indicates which unmasked errors have occurred and have generated error interrupts to the channel. Each bit in this register can only be set if the corresponding bit of the RNGU interrupt mask register is zero (see Section 17.7.8.6, "RNGU Interrupt Mask Register").

If the RNGU interrupt status register is non-zero, the RNGU halts and the RNGU error interrupt signal is asserted to the controller (see Section 17.5.4.2.2, "Interrupt Status Register (ISR)"). In addition, if the RNGU is being operated through channel-controlled access, then an interrupt signal is generated to the channel to which this EU is assigned. The EU error then appears in bit 55 of the Channel Pointer Status Register (see Table 17-15) and generates a channel error interrupt to the controller.

If the interrupt status register is written from the host, 1s in the value written are recorded in the interrupt status register if the corresponding bit is unmasked in the interrupt mask register. All other bits are cleared. This register can also be cleared by setting the RI bit of the RNGU Reset Control Register.

The bit fields of the RNGU interrupt status register are shown in Figure 17-105.

Offset 0x3_A030 Access: Read/Write

	0–50	51	52–55	56	57	58–61	62	63
R/W	—	IE	—	ME	AE	—	OFU	—
Reset	All zeros							

Figure 17-105. RNGU Interrupt Status Register

Table 17-75 describes the RNGU interrupt status register fields.

Table 17-75. RNGU Interrupt Status Register Field Descriptions

Bits	Name	Description
0–50	—	Reserved
51	IE	Internal Error 0 No internal error detected 1 Internal error
52–55	—	Reserved
56	ME	Mode Error. Indicates that the host has attempted to write an illegal value to the mode register 0 Valid data 1 Invalid data error
57	AE	Address Error. An illegal read or write address was detected within the RNGU address space. 0 No error detected 1 Address error
58–61	—	Reserved
62	OFU	Output FIFO Underflow. The RNGU Output FIFO was read while empty. 0 No overflow detected 1 Output FIFO has underflowed
63	—	Reserved

17.7.8.6 RNGU Interrupt Mask Register

The RNGU interrupt mask register controls the result of detected errors. For a given error (as defined in Section 17.7.8.5, “RNGU Interrupt Status Register”), if the corresponding bit in this register is set, then the error is disabled; no error interrupt occurs and the interrupt status register is not updated to reflect the error. If the corresponding bit is not set, then upon detection of an error, the interrupt status register is updated to reflect the error, causing assertion of the error interrupt signal, and causing the module to halt processing.

The bit fields of the RNGU interrupt mask register are shown in Figure 17-106.

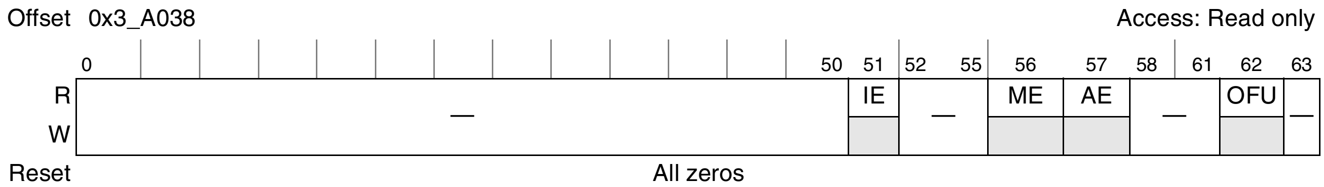


Figure 17-106. RNGU Interrupt Mask Register

Table 17-76 describes RNGU interrupt status register fields.

Table 17-76. RNGU Interrupt Mask Register Field Descriptions

Bits	Name	Description
0–50	—	Reserved
51	IE	Internal Error. An internal processing error was detected while generating random numbers. This error is no longer maskable and can only be cleared by setting one of the reset bits in the Reset Control Register 0 Internal error enabled 1 Internal error disabled
52–55	—	Reserved
56	ME	Mode Error. An illegal value was detected in the mode register. 0 Mode error enabled 1 Mode error disabled
57	AE	Address Error. An illegal read or write address was detected within the RNGU address space. 0 Address error enabled 1 Address error disabled
58–61	—	Reserved
62	OFU	Output FIFO Underflow. RNGU Output FIFO was read while empty. 0 Output FIFO underflow error enabled 1 Output FIFO underflow error disabled
63	—	Reserved

17.7.8.7 RNGU End of Message Register

The RNGU end of message register (shown in Figure 17-107) is a write-only register that can be used to start the RNGU. A write of any value to this register causes the RNGU to begin to produce random numbers in the FIFO.

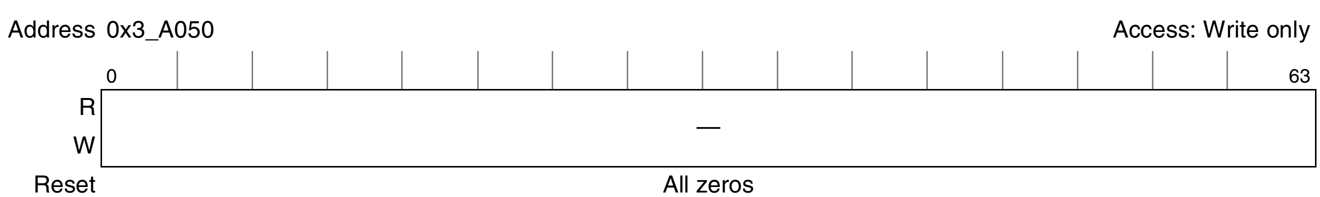


Figure 17-107. RNGU End of Message Register

17.7.8.8 RNGU Entropy Registers

RNGU allows the user to input entropy bits into the PRNG algorithm to modify the randomness of the RNGU. This group of registers are write-only, and all writes to these registers are ignored when the RNGU is busy. However when the RNGU is idle (FIFO is full or RNGU has not yet been started), all data written to these registers is used to modify the internal XKEY structure. These 64-bit registers cannot be written back-to-back—there must be a clock cycle in between writes, since the RNGU only processes 32 bits per cycle.

17.7.8.9 RNGU FIFO

RNGU uses an output FIFO to collect periodically sampled random 64-bit-words, with the intent that random data always be available for reading. Normally, the channels control all access to this FIFO. For host-controlled operation, a read from anywhere in the RNGU FIFO address space dequeues data from the RNGU output FIFO.

The output FIFO is readable by byte, word, or dword. When all 8 bytes of the head dword have been read, that dword is automatically dequeued from the FIFO so that the next dword (if any) becomes available for reading. If any byte is read twice between dequeues, it causes an error interrupt of type AE from the EU.

Underflows caused by reading or writing the RNGU output FIFO are reflected in the RNGU interrupt status register. Also, a write to the RNGU output FIFO space is reflected as an addressing error in the RNGU interrupt status register.

NOTE

Host reads of the RNGB FIFO should be performed on an 8-byte basis, regardless of how many bits of random number is actually required. Partial host reads can leave the RNGB FIFO in a state that results in a channel error.

Chapter 18
Enhanced Three-Speed Ethernet Controllers

18.1 Overview

The enhanced three-speed Ethernet controllers (eTSECs) of the device interface to 10 Mbps, 100 Mbps, and 1 Gbps Ethernet/IEEE 802.3 networks. For Ethernet, an external PHY or SerDes device is required to complete the interface to the media. Each eTSEC supports multiple standard media-independent interfaces. Multiple eTSECs are available, providing flexible options for connectivity and control access at different speeds.

The eTSEC provides the flexibility to accelerate the identification and retrieval of standard and non-standard protocols carried over Ethernet, including both IP versions 4 and 6 and TCP/UDP. CPU-intensive parsing and checksum operations can be optionally off-loaded to an eTSEC to accelerate existing TCP/IP stacks. On transmission, varying fractions of link bandwidth can be allocated to each of multiple transmit queues through a modified weighted round-robin scheduler. On receive, an arbitrary set of queue selection rules can be programmed into each eTSEC to implement flexible quality of service or firewall strategies based on high-level protocol identification. Without enabling these advanced features, each eTSEC emulates a PowerQUICC II Pro TSEC, allowing existing driver software to be re-used with minimal change. Each eTSEC is organized as shown in Figure 18-1.

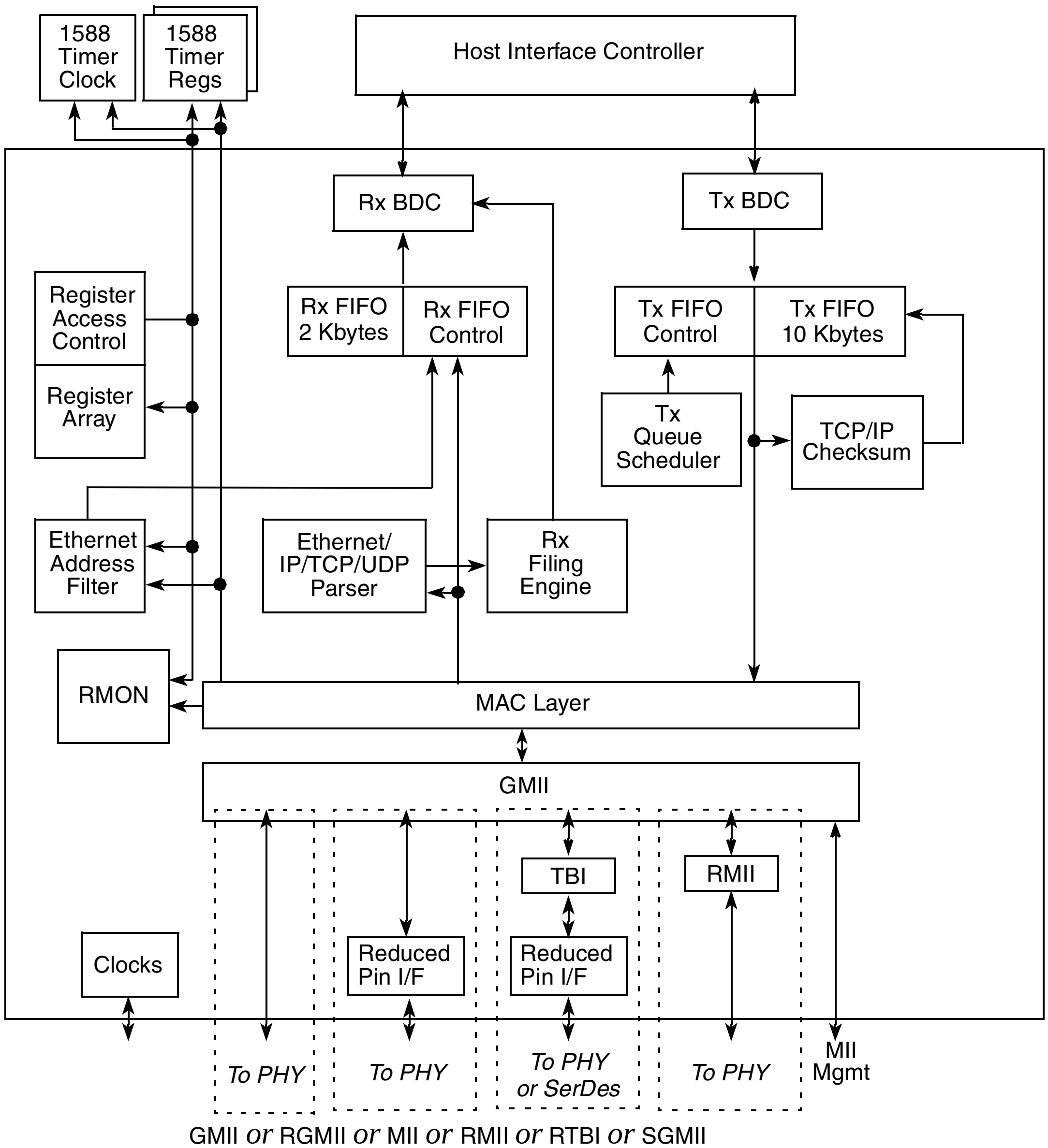


Figure 18-1. eTSEC Block Diagram

18.2 Features

The eTSECs of the device include these distinctive features:

- IEEE 802.3, 802.3u, 820.3x, 802.3z, 802.3ac, 802.3ab compatible
- Support for different Ethernet physical interfaces:
  — 10/100 Mbps IEEE 802.3 MII and RMII

  - 10/100 Mbps RGMII
  - 1000 Mbps full-duplex RGMII and RTBI
  - 10/100 Mbps SGMII
  - 1000 Mbps full-duplex SGMII
- TCP/IP off-load
  - IP v4 and IP v6 header recognition on receive
  - IP v4 header checksum verification and generation
  - TCP and UDP checksum verification and generation
  - Per-packet configurable off-load
  - Recognition of VLAN, stacked-VLAN, 802.2, PPPoE session, MPLS stacks, ARP, and ESP/AH IP-Security headers
- Quality of service (QoS) support
  - Transmission from up to eight queues
    - Priority-based queue selection
    - Modified weighted round-robin queue selection with fair bandwidth allocation
  - Reception to up to eight physical queues
    - 64 virtual receive queues overlaid on 8 physical buffer descriptor rings
    - Table-oriented queue filing strategy based on 16 header fields or flags
    - Frame rejection support for filtering applications
    - Filing based on Ethernet, IP, and TCP/UDP properties, including VLAN fields, Ether-type, IP protocol type, IP TOS or differentiated services, IP source and destination addresses, TCP/UDP port numbers
- Interrupt coalescing
  - Packet-count-based thresholds for both receive and transmit
  - Timer-based thresholds
- Full- and half-duplex Ethernet support (1000 Mbps supports only full duplex):
  - IEEE 802.3 full-duplex flow control (automatic PAUSE frame generation or software programmed PAUSE frame generation and recognition)
  - Programmable maximum frame length supports jumbo frames (up to 9.6 Kbytes) and IEEE 802.1 virtual local area network (VLAN) tags and priority
  - VLAN insertion and deletion
    - Per-frame VLAN control word or default VLAN for each eTSEC
    - Extracted VLAN control word passed to software separately
    - Programmable VLAN tag to support metropolitan bridging
  - Retransmission following a collision
  - Support for CRC generation and verification of inbound/outbound packets
  - Programmable Ethernet preamble insertion and extraction of up to 7 bytes
- MAC address recognition:

  - Exact match on primary and virtual 48-bit unicast addresses
    - VRRP and HSRP support for seamless router fail-over
    - In addition to primary station address, up to fifteen additional exact-match MAC addresses supported
  - Broadcast address (accept/reject)
  - Hash table match on up to 256 unicast/multicast or 512 multicast-only addresses
  - Promiscuous mode
- Remote network monitoring (RMON) statistics support
  - 32-bit byte counters
  - Carry/Overflow of counter interrupts
- Backward compatibility with MPC8349E (PowerQUICC II Pro) TSEC
  - PowerQUICC II Pro buffer descriptor (BD) format and rings supported
  - Common register memory map, with specific exceptions:
    - Out-of-sequence transmit BD not supported
    - Internal DMA BD pointers and data counts not visible
    - MINFLR register not supported
  - Reset state of eTSEC defaults to common PowerQUICC II Pro TSEC subset
  - TSEC_ID register permits TSEC versus enhanced TSEC differentiation
- Hardware assist for 1588 compliant timestamping (1588 not supported in conjunction with SGMII 10/100)
  - Per packet timestamp tag for Receive
  - Programmable timestamp capture for Transmit
  - Recognition of PTP packet
  - Periodic Pulse Generation
  - Self-correcting precision timer with nano-second resolution
  - Phase aligned adjustable (divide by N) clock output
  - Two 64-bit alarm (future time) registers for future time comparison

18.3 Modes of Operation

The eTSEC's primary operational modes are the following:

- Full- and half-duplex operation

  This is determined by the MACCFG2 register's full-duplex bit (MACCFG2[Full Duplex]). Full-duplex mode is intended for use on point-to-point links between switches or end node to switch. Half-duplex mode is used in connections between an end node and a repeater or between repeaters.

  If configured in half-duplex mode (10- and 100-Mbps operation; MACCFG2[Full Duplex] is cleared), the MAC complies with the IEEE CSMA/CD access method.

If configured in full-duplex mode (10/100/1000 Mbps operation; MACCFG2[Full Duplex] is set), the MAC supports flow control. If flow control is enabled, it allows the MAC to receive or send PAUSE frames.

- 10- and 100-Mbps MII interface operation

  The MAC–PHY interface operates in MII mode by setting MACCFG2[I/F Mode] = 01. The MII is the media-independent interface defined by the 802.3 standard for 10/100 Mbps operation. The speed of operation is determined by the TSEC*n*_TX_CLK and TSEC*n*_RX_CLK signals, which are driven by the transceiver. The transceiver either auto-negotiates the speed, or it may be controlled by software using the serial management interface (MDC/MDIO signals) to the transceiver.

  Clause 22.2.4 of the IEEE 802.3 specification describes the MII management interface.

- 10- and 100-Mbps RMII interface operation

  The RMII is the reduced media-independent interface defined by the RMII Consortium (March 1998) for 10/100 Mbps operation. The speed of operation is determined by the TSEC*n*_TX_CLK signal, which is driven by the transceiver.

- MAC address recognition options

  The options supported are promiscuous, broadcast, exact unicast address match, exact unicast virtual address match to support router redundancy, and multicast hash match. For detailed descriptions refer to Section 18.6.2.7, “Frame Recognition.”

  eTSEC supports automatic LAN-initiated wake-up during power management through the AMD Magic Packet™ protocol, as described in Section 18.6.2.8, “Magic Packet Mode.”

- Receive frame parsing options

  Frame parsing options are to disable parsing (no TCP/IP off-load), IP header parsing, and TCP or UDP parsing. Parsing must be enabled to make use of receive queue filing algorithms. The options are detailed in Section 18.6.3, “TCP/IP Off-Load.”

- Receive queue selection options

  Received frames are by default sent to a single buffer descriptor ring. If multiple receive queues are enabled, a receive queue filer can be programmed with selection criteria to differentiate received frames and file them to different buffer descriptor rings. See Section 18.6.4, “Quality of Service (QoS) Provision,” for detailed descriptions.

- TCP/IP transmit options

  Frames for transmission may be sent as-is, with IP header processing, or TCP header processing. The transmit buffer descriptors, described in Section 18.6.6.2, “Transmit Data Buffer Descriptors (TxBD),” enable these options and operate with parameters prepended to frame buffers, as described in Section 18.6.3, “TCP/IP Off-Load.”

- Transmit queue selection options

  The options supported are single transmit queue, priority-based queue selection, and modified weighted round-robin queueing. These options are described further in Section 18.5.3.2.1, “Transmit Control Register (TCTRL).”

- RMON support

  Standard Ethernet interface management information base (MIBs) can be generated through the RMON MIB counters.
- Internal loop back supported for all interfaces except when configured for half-duplex operation

  Internal loop back mode is selected through the loop back bit in the MACCFG1 register. See Section 18.7.1, “Interface Mode Configuration,” for details.

18.4 External Signals Description

This section defines the eTSEC interface signals. The buses are described using the bus convention used in IEEE Std. 802.3 standard because the PHY follows this same convention. (That is, TxD[3:0] means 0 is the lsb.) Note that except for external physical interfaces the buses and registers follow a big-endian format, where 0 denotes the msb.

Each eTSEC network interface supports multiple options:

- The MII option requires 18 I/O signals (including the MDIO and MDC MII management interface) and supports both a data and a management interface to the PHY (transceiver) device. The MII option supports both 10- and 100-Mbps Ethernet rates.
- The RGMII, RTBI, and RMII options are reduced-pin implementations of the GMII, TBI, and MII interfaces, respectively.
- SGMII interfaces are offered via the SerDes1 interface signals.
- 1588 timer signals

Table 18-1 lists the network interface signals.

Table 18-1. eTSEC*n* Network Interface Signal Properties

Signal Name	Function	Reset State
TSEC*n*_COL	MII—collision, input	—
TSEC*n*_CRS	MII—carrier sense, input	—
TSEC*n*_GTX_CLK	RTBI, RGMII—inverted transmit clock feedback, output MII, RMII—transmit clock feedback when transmission is enabled, zero otherwise, output	0
EC_GTX_CLK125	Oscillator source for RGMII, RTBI transmit clock, input, shared by all eTSECs	—
EC_MDC	Management clock, output.	0
EC_MDIO	Management data, bidirectional.	Hi-Z (input)
TSEC*n*_RX_CLK	MII, RGMII—receive clock, input	—
TSEC*n*_RX_DV	MII—receive data valid, input RGMII (RX_CLK rising)—receive data valid, input RGMII (RX_CLK falling)—receive error, input RTBI (RX_CLK rising)—receive code group (RCG) bit 4, input RTBI (RX_CLK falling)—receive code group (RCG) bit 9, input RMII—CRS_DV carrier sense/data valid, input	—

Table 18-1. eTSEC*n* Network Interface Signal Properties (continued)

Signal Name	Function	Reset State
TSEC*n*_RXD[3:0]	MII—Receive data bits 3:0, input RGMII (RX_CLK rising) —Receive data bits 3:0, input RGMII (RX_CLK falling)—Receive data bits 7:4, input RTBI (RX_CLK rising)—RCG bits 3:0, input RTBI (RX_CLK falling)—RCG bits 8:5, input RMII—RXD[1:0] receive data bits, input RMII—RXD[3:2] are unused	—
TSEC*n*_RX_ER	MII, RMII—Receive error, input RGMII, RTBI—Unused	—
TSEC*n*_TX_CLK	MII—transmit clock, input RMII—reference transmit and receive clock, input RGMII, RTBI—unused	—
TSEC*n*_TXD[3:0]	MII—Transmit data bits 3:0, output RGMII (TX_CLK rising)—Transmit data bits 3:0, output RGMII (TX_CLK falling)—Transmit data bits 7:4, output RTBI (TX_CLK rising)—TCG bits 3:0, output RTBI (TX_CLK falling)—TCG bits 8:5, output RMII—TXD[1:0] transmit data bits, output RMII—TXD[3:2] unused, output driven zero	0000
TSEC*n*_TX_ER	MII—transmit error, output RGMII, RTBI, RMII—unused, output driven zero	0
TSEC*n*_TX_EN	MII, RMII—Transmit data valid, output RGMII (TX_CLK rising)—Transmit data enabled, output RGMII (TX_CLK falling)—Transmit error, output RTBI (TX_CLK rising)—TCG bit 4, output RTBI (TX_CLK falling)—TCG bit 9, output	0
TSEC_TMR_CLK	1588—Clock input External high precision timer reference clock input (chip external input pin).	—
TSEC_TMR_GCLK	1588—Clock output Phase aligned timer clock divider output (chip external output pin).	0
TSEC_TMR_TRIG1	1588—Trigger in 1 External timer trigger input 1. This is an asynchronous general purpose input (chip external input pin).	—
TSEC_TMR_TRIG2	1588—Trigger in 2 External timer trigger input 2. This is an asynchronous general purpose input (chip external input pin).	—
TSEC_TMR_PP1	1588—Pulse out 1 Timer pulse per period 1. It is phase aligned with 1588 timer clock (chip external output pin).	0
TSEC_TMR_PP2	1588—Pulse out 2 Timer pulse per period 2. It is phase aligned with 1588 timer clock (chip external output pin).	0
TSEC_TMR_PP3	1588—Pulse out 3 Timer pulse per period 3. It is phase aligned with 1588 timer clock (chip external output pin).	0
TSEC_TMR_ALARM1	1588—Timer alarm 1 Timer current time is equal to or greater than alarm time comparator register. User reprograms the TSEC_TMR_ALARM*n*_H/L register to deactivate this output (chip external output pin).	0

Table 18-1. eTSEC*n* Network Interface Signal Properties (continued)

Signal Name	Function	Reset State
TSEC_TMR_ALARM2	1588—Timer alarm 2 Timer current time is equal to or greater than alarm time comparator register. User reprograms the TSEC_TMR_ALARM*n*_H/L register to deactivate this output (chip external output pin).	0
L1_SD_TXA/E, L1_SD_TXA/E	SGMII transmit data (and complement)	—
L1_SD_RXA/E, L1_SD_RXA/E	SGMII receive data (and complement)	—
L1_SD_REF_CLK, L1_SD_REF_CLK	SGMII SerDes1 PLL reference clock (and complement)	—

18.4.1 Detailed Signal Descriptions

Table 18-2 is a description of the eTSEC interface signals. For RGMII mode details please refer to the Hewlett-Packard reduced gigabit media-independent interface (RGMII) specification version 1.2a, dated 9/22/2000. RMII mode details follow the RMII Consortium Specification, dated 3/20/1998. All other modes follow the IEEE 802.3 standard, 2000 Edition. Input signals not used are internally disabled. Except for TSEC*n*_GTX_CLK, output signals not used are driven low.

Table 18-2. eTSEC Signals—Detailed Signal Descriptions

Signal	I/O	Description	
TSEC*n*_COL	I	Collision input. The behavior of this signal is not specified while in full-duplex mode.	
		State Meaning	Asserted/Negated—In MII mode, this signal is asserted upon detection of a collision, and must remain asserted while the collision persists. This signal is not used in the following modes: • RMII • RTBI • RGMII
		Timing	Asserted/Negated—This signal is not required to transition synchronously with TSEC*n*_TX_CLK or TSEC*n*_RX_CLK.
TSEC*n*_CRS	I	Carrier sense input. In RTBI mode, this signal is used as SDET (signal detect). In RTBI mode SDET is tied high internally. This signal is not used in the following modes: • RMII • RGMII	
		State Meaning	Asserted/Negated—In MII mode, TSEC*n*_CRS is asserted while the transmit or receive medium is not idle. In the event of a collision, TSEC*n*_CRS must remain asserted for the duration of the collision.
		Timing	Asserted/Negated—This signal is not required to transition synchronously with TSEC*n*_TX_CLK or TSEC*n*_RX_CLK.

Table 18-2. eTSEC Signals—Detailed Signal Descriptions (continued)

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td>TSECn_GTX_CLK</td><td>O</td><td colspan="2">Gigabit transmit clock. This signal is an output from the eTSEC into the PHY. TSECn_GTX_CLK is a 125-MHz clock that provides a timing reference for TX_EN, TXD, and TX_ER in the following modes:
• RTBI
In RGMII mode, TSECn_GTX_CLK becomes the transmit clock and provides timing reference during 1000Base-T (125 MHz), 100Base-T (25 MHz) and 10Base-T (2.5 MHz) transmissions. This signal feeds back the uninverted transmit clock in MII mode, but feeds back an inverted transmit clock in RTBI or RGMII modes.
This signal is driven low unless transmission is enabled.</td></tr>
<tr><td>EC_GTX_CLK125</td><td>I</td><td colspan="2">Gigabit transmit 125-MHz source. This signal must be generated externally with a crystal or oscillator, or is sometimes provided by the PHY. EC_GTX_CLK125 is a 125-MHz input into the eTSEC and is used to generate all 125-MHz related signals and clocks in the following modes:
• RTBI
• RGMII
This input is not used in these modes:
• RMII
• SGMII
• MII</td></tr>
<tr><td>EC_MDC</td><td>O</td><td colspan="2">Management data clock.
This signal is a clock (typically 2.5 MHz) supplied by the MAC
(IEEE set minimum period of 400 ns or a frequency of 2.5 MHz, but the device may be configured up to 12.5 MHz if supported by the PHY at that speed.) The frequency can be modified by writing to MIIMCFG[28:31] of the eTSEC1 controller.</td></tr>
<tr><td rowspan="3">EC_MDIO</td><td rowspan="3">I/O</td><td colspan="2">Management data input/output.</td></tr>
<tr><td>State Meaning</td><td>Asserted/Negated—EC_MDIO is a bidirectional signal to input PHY-supplied status during management read cycles and output control during MII management write cycles. Addressed using eTSEC1 memory-mapped registers.</td></tr>
<tr><td>Timing</td><td>Asserted/Negated—This signal is required to be synchronous with the EC_MDC signal.</td></tr>
<tr><td>TSECn_RX_CLK</td><td>I</td><td colspan="2">Receive clock. In MII or RGMII mode, the receive clock TSECn_RX_CLK is a continuous clock (2.5, 25, or 125 MHz) that provides a timing reference for TSECn_RX_DV, TSECn_RXD, and TSECn_RX_ER.
In RTBI mode it is a 125-MHz receive clock.
In RMII mode this clock is not used for the receive clock, as RMII uses a shared reference clock.</td></tr>
<tr><td>TSECn_RX_DV</td><td>I</td><td colspan="2">Receive data valid. In MII mode, if TSECn_RX_DV is asserted, the PHY is indicating that valid data is present on the MII interface.
In RGMII mode, TSECn_RX_DV becomes RX_CTL. The RX_DV and RX_ERR are received on this signal on the rising and falling edges of TSECn_RX_CLK.
In RTBI mode, TSECn_RX_DV represents receive code group (RCG) bit 4 and 9. On the positive edge of the TSECn_RX_CLK, RCG[4] and RCG[3:0] represent the first half of the 10-bit encoded symbol. On the negative edge of the TSECn_RX_CLK, RCG[9] and RCG[8:5] represent the second half of the 10-bit encoded symbol.
In RMII mode the PHY asserts TSECn_RX_DV (CRS_DV) when the receive medium is non-idle. This signal asserts asynchronously with respect to the RMII reference clock, but negates synchronously to indicate loss of carrier.</td></tr>
</table>

Table 18-2. eTSEC Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
TSECn_RXD[3:0]	I	Receive data in. In MII mode, TSECn_RXD[3:0] represents a nibble of data to be transferred from the PHY to the MAC when TSECn_RX_DV is asserted. A completely-formed SFD must be passed across the MII. While TSECn_RX_DV is not asserted, TSECn_RXD has no meaning. In RGMII mode, data bits 3:0 are received on the rising edge of TSECn_RX_CLK and data bits 7:4 are received on the falling edge of TSECn_RX_CLK. In RTBI mode, TSECn_RXD[3:0] represents RCG[3:0] on the rising edge of TSECn_RX_CLK and RCG[8:5] are received on the falling edge of TSECn_RX_CLK. In RMII mode, TSECn_RXD[1:0] represents RXD[1:0], which is considered valid when TSECn_RX_DV (CRS_DV) is asserted, or invalid otherwise.
TSECn_RX_ER	I	Receive error **State Meaning**: Asserted/Negated—In MII, or RMII mode, if TSECn_RX_ER and TSECn_RX_DV are asserted, the PHY has detected an error in the current frame. This signal is not used in the RTBI or RGMII modes.
TSECn_TX_CLK	I	Transmit clock in. In MII mode, TSECn_TX_CLK is a continuous clock (2.5 or 25 MHz) that provides a timing reference for the TSECn_TX_EN, TSECn_TXD, and TSECn_TX_ER signals. _In RMII mode this signal is the reference clock shared between transmit and receive, and is supplied by the PHY. This signal is not used in the eTSEC RTBI or RGMII modes.
TSECn_TXD[3:0]	O	Transmit data out. DVIn MII mode, TSECn_TXD[3:0] represent a nibble of data to be sent from the MAC to the PHY when TSECn_TX_EN is asserted and have no meaning while TSECn_TX_EN is negated. In RGMII or RTBI mode, data bits 3:0 are transmitted on the rising edge of TSECn_GTX_CLK, and data bits 7:4 are transmitted on the falling edge of TSECn_GTX_CLK. In RMII mode TSECn_TXD[1:0] represents TXD[1:0], which is valid data sent to the PHY when TSECn_TX_EN is asserted, or undefined otherwise. Note that some of these signals are also used during reset to configure the eTSEC interface mode.
TSECn_TX_EN	O	Transmit data valid. In MII, or RMII mode, if TSECn_TX_EN is asserted, the MAC is indicating that valid data is present on the MII's TSECn_TXD signals. In RGMII mode, TSECn_TX_EN becomes TX_CTL. TX_EN and TX_ERR are asserted on this signal on rising and falling edges of the TSECn_GTX_CLK, respectively. In RTBI mode, TSECn_TX_EN represents TCG[4] on the rising edge and TCG[9] on the falling edge of TSECn_GTX_CLK, respectively. Together with TCG[3:0] and TCG[8:5], they represent the 10-bit encoded symbol.
TSECn_TX_ER	O	Transmit error. In MII mode, assertion of TSECn_TX_ER for one or more clock cycles while TSECn_TX_EN is asserted causes the PHY to transmit one or more illegal symbols. Asserting TSECn_TX_ER has no effect while operating at 10 Mbps or while TSECn_TX_EN is negated. This signal transitions synchronously with respect to TSECn_TX_CLK. This signal is not used in the eTSEC RMII, RTBI, or RGMII modes and is driven low.
TSEC_TMR_CLK	I	1588 clock in. External high precision timer reference clock input (chip external input pin).
TSEC_TMR_GCLK	O	1588 clock out. Phase aligned timer clock divider output (chip external output pin).
TSEC_TMR_TRIG1	I	1588 trigger in 1. External timer trigger input 1.This is an asynchronous general purpose input (chip external input pin).
TSEC_TMR_TRIG2	i	1588 trigger in 2. External timer trigger input 2.This is an asynchronous general purpose input (chip external input pin).

Table 18-2. eTSEC Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
TSEC_TMR_PP1	O	1588 pulse out 1. Timer pulse per period 1. It is phase aligned with 1588 timer clock (chip external output pin)
TSEC_TMR_PP2	O	1588 pulse out 2. Timer pulse per period 2. It is phase aligned with 1588 timer clock (chip external output pin)
TSEC_TMR_PP3	O	1588 pulse out 3. Timer pulse per period 3. It is phase aligned with 1588 timer clock (chip external output pin)
TSEC_TMR_ALARM1	O	1588 timer alarm 1. Timer current time is equal to or greater than alarm time comparator register. User reprograms the TSEC_TMR_ALARM*n*_H/L register to deactivate this output (chip external output pin)
TSEC_TMR_ALARM2	O	1588 timer alarm 2. Timer current time is equal to or greater than alarm time comparator register. User reprograms the TSEC_TMR_ALARM*n*_H/L register to deactivate this output (chip external output pin)
L1_SD_TXA/E, $\overline{\text{L1\_SD\_TXA/E}}$	O	SGMII transmit data (and complement) When in SGMII interface mode: • eTSEC1 utilizes L1_SD_TXA and $\overline{\text{L1\_SD\_TXA}}$ • eTSEC2 utilizes L1_SD_TXE and $\overline{\text{L1\_SD\_TXE}}$
L1_SD_RXA/E, $\overline{\text{L1\_SD\_RXA/E}}$	I	SGMII receive data (and complement) When in SGMII interface mode: • eTSEC1 utilizes L1_SD_RXA and $\overline{\text{L1\_SD\_RXA}}$ • eTSEC2 utilizes L1_SD_RXE and $\overline{\text{L1\_SD\_RXE}}$
L1_SD_REF_CLK, $\overline{\text{L1\_SD\_REF\_CLK}}$	I	SGMII SerDes1 PLL reference clock (and complement)

18.5 Memory Map/Register Definition

The eTSECs use a software model that is a superset of the PowerQUICC II Pro TSEC functionality and is similar to that employed by the Fast Ethernet function supported on the Freescale MPC8260 CPM FCC and in the FEC of the MPC860T.

The eTSEC device is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs are used for mode control, interrupts, and to extract status information. The descriptors are used to pass data buffers and related buffer status or frame information between the hardware and software.

All accesses to and from the registers must be made as 32-bit accesses. There is no support for accesses of sizes other than 32 bits. Writes to reserved register bits must always store 0, as writing 1 to reserved bits may have unintended side-effects. Reads from unmapped register addresses return zero. Unless otherwise specified, the read value of reserved bits in mapped registers is not defined, and must not be assumed to be 0.

This section of the document defines the memory map and describes the registers in detail. The buffer descriptor is described in Section 18.6.6, “Buffer Descriptors.”

18.5.1 Top-Level Module Memory Map

Each of the eTSECs is allocated 4 Kbytes of memory-mapped space. The space for each eTSEC is divided as indicated in Table 18-3.

Table 18-3. Module Memory Map Summary

Address Offset	Function
000–0FF	eTSEC general control/status registers
100–2FF	eTSEC transmit control/status registers
300–4FF	eTSEC receive control/status registers
500–5FF	MAC registers
600–7FF	RMON MIB registers
800–8FF	Hash table registers
900–9FF	—
A00–AFF	FIFO control/status registers
B00–BFF	DMA system registers
C00–DFF	—
E00–EFF	1588 Hardware Assist

18.5.2 Detailed Memory Map

The eTSEC memory mapped registers are accessed by reading and writing to an address comprised of the base address (specified in IMMRBAR as defined in Chapter 2, “Memory Map.”) plus the block base address, plus the offset of the specific register to be accessed. Note that all memory-mapped registers must only be accessed as 32-bit quantities.

Table 18-4 lists the offset, name, and a cross-reference to the complete description of each register. The offsets to the memory map table are applicable to each eTSEC. Block base addresses are as follows:

- eTSEC1 starts at 0x2_4000 address offset
- eTSEC2 starts at 0x2_5000 address offset

In this table and in the register figures and field descriptions, the following access definitions apply:

- Reserved fields are always ignored for the purposes of determining access type.
- R/W, R, and W (read/write, read only, and write only) indicate that all the non-reserved fields in a register have the same access type.
- w1c indicates that all of the non-reserved fields in a register are cleared by writing ones to them.
- Mixed indicates a combination of access types.
- Special is used when no other category applies. In this case the register figure and field description table should be read carefully.

Table 18-4. Module Memory Map

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
eTSEC General Control and Status Registers				
0x2_4000	TSEC_ID*—Controller ID register	R	0x0124_0005	18.5.3.1.1/18-22
0x2_4004	TSEC_ID2*—Controller ID register	R	0x00EC_00F0	18.5.3.1.2/18-22
0x2_4008–0x2_400C	Reserved	—	—	—
0x2_4010	IEVENT—Interrupt event register	w1c	0x0000_0000	18.5.3.1.3/18-23
0x2_4014	IMASK—Interrupt mask register	R/W	0x0000_0000	18.5.3.1.4/18-27
0x2_4018	EDIS—Error disabled register	R/W	0x0000_0000	18.5.3.1.5/18-29
0x2_401C	Reserved	—	—	—
0x2_4020	ECNTRL—Ethernet control register	R/W	0x0000_RR00[3]	18.5.3.1.6/18-31
0x2_4024	Reserved	—	—	—
0x2_4028	PTV—Pause time value register	R/W	0x0000_0000	18.5.3.1.7/18-33
0x2_402C	DMACTRL—DMA control register	R/W	0x0000_0000	18.5.3.1.8/18-34
0x2_4030	TBIPA—TBI PHY address register	R/W	0x0000_0000	18.5.3.1.9/18-35
0x2_4034–0x2_40FC	Reserved	—	—	—
eTSEC Transmit Control and Status Registers				
0x2_4100	TCTRL—Transmit control register	R/W	0x0000_0000	18.5.3.2.1/18-36
0x2_4104	TSTAT—Transmit status register	w1c	0x0000_0000	18.5.3.2.2/18-38
0x2_4108	DFVLAN*—Default VLAN control word	R/W	0x8100_0000	18.5.3.2.3/18-42
0x2_410C	Reserved	—	—	—
0x2_4110	TXIC—Transmit interrupt coalescing register	R/W	0x0000_0000	18.5.3.2.4/18-43
0x2_4114	TQUEUE*—Transmit queue control register	R/W	0x0000_8000	18.5.3.2.5/18-44
0x2_4118–0x2_413C	Reserved	—	—	—
0x2_4140	TR03WT*—TxBD Rings 0–3 round-robin weightings	R/W	0x0000_0000	18.5.3.2.6/18-44
0x2_4144	TR47WT*—TxBD Rings 4–7 round-robin weightings	R/W	0x0000_0000	18.5.3.2.7/18-45
0x2_4148–0x2_4180	Reserved	—	—	—
0x2_4184	TBPTR0—TxBD pointer for ring 0	R/W	0x0000_0000	18.5.3.2.8/18-46
0x2_4188	Reserved	—	—	—
0x2_418C	TBPTR1*—TxBD pointer for ring 1	R/W	0x0000_0000	18.5.3.2.8/18-46
0x2_4190	Reserved	—	—	—
0x2_4194	TBPTR2*—TxBD pointer for ring 2	R/W	0x0000_0000	18.5.3.2.8/18-46
0x2_4198	Reserved	—	—	—

Table 18-4. Module Memory Map (continued)

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
0x2_419C	TBPTR3*—TxBD pointer for ring 3	R/W	0x0000_0000	18.5.3.2.8/18-46
0x2_41A0	Reserved	—	—	—
0x2_41A4	TBPTR4*—TxBD pointer for ring 4	R/W	0x0000_0000	18.5.3.2.8/18-46
0x2_41A8	Reserved	—	—	—
0x2_41AC	TBPTR5*—TxBD pointer for ring 5	R/W	0x0000_0000	18.5.3.2.8/18-46
0x2_41B0	Reserved	—	—	—
0x2_41B4	TBPTR6*—TxBD pointer for ring 6	R/W	0x0000_0000	18.5.3.2.8/18-46
0x2_41B8	Reserved	—	—	—
0x2_41BC	TBPTR7*—TxBD pointer for ring 7	R/W	0x0000_0000	18.5.3.2.8/18-46
0x2_41C0–0x2_4200	Reserved	—	—	—
0x2_4204	TBASE0—TxBD base address of ring 0	R/W	0x0000_0000	18.5.3.2.9/18-47
0x2_4208	Reserved	—	—	—
0x2_420C	TBASE1*—TxBD base address of ring 1	R/W	0x0000_0000	18.5.3.2.9/18-47
0x2_4210	Reserved	—	—	—
0x2_4214	TBASE2*—TxBD base address of ring 2	R/W	0x0000_0000	18.5.3.2.9/18-47
0x2_4218	Reserved	—	—	—
0x2_421C	TBASE3*—TxBD base address of ring 3	R/W	0x0000_0000	18.5.3.2.9/18-47
0x2_4220	Reserved	—	—	—
0x2_4224	TBASE4*—TxBD base address of ring 4	R/W	0x0000_0000	18.5.3.2.9/18-47
0x2_4228	Reserved	—	—	—
0x2_422C	TBASE5*—TxBD base address of ring 5	R/W	0x0000_0000	18.5.3.2.9/18-47
0x2_4230	Reserved	—	—	—
0x2_4234	TBASE6*—TxBD base address of ring 6	R/W	0x0000_0000	18.5.3.2.9/18-47
0x2_4238	Reserved	—	—	—
0x2_423C	TBASE7*—TxBD base address of ring 7	R/W	0x0000_0000	18.5.3.2.9/18-47
0x2_4240–0x2_427C	Reserved	—	—	—
0x2_4280	TMR_TXTS1_ID*—Tx time stamp identification tag (set 1)	R/W	0x0000_0000	18.5.3.2.10/18-47
0x2_4284	TMR_TXTS2_ID*—Tx time stamp identification tag (set 2)	R/W	0x0000_0000	18.5.3.2.10/18-47
0x2_4288–0x2_42BC	Reserved	—	—	—
0x2_42C0	TMR_TXTS1_H*—Tx time stamp high (set 1)	R/W	0x0000_0000	18.5.3.2.11/18-48
0x2_42C4	TMR_TXTS1_L*—Tx time stamp high (set 1)	R/W	0x0000_0000	18.5.3.2.11/18-48
0x2_42C8	TMR_TXTS2_H*—Tx time stamp high (set 2)	R/W	0x0000_0000	18.5.3.2.11/18-48

Table 18-4. Module Memory Map (continued)

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
0x2_42CC	TMR_TXTS2_L*—Tx time stamp high (set 2)	R/W	0x0000_0000	18.5.3.2.11/18-48
0x2_42D0–0x2_42FC	Reserved	—	—	—
eTSEC Receive Control and Status Registers				
0x2_4300	RCTRL—Receive control register	R/W	0x0000_0000	18.5.3.3.1/18-48
0x2_4304	RSTAT—Receive status register	w1c	0x0000_0000	18.5.3.3.2/18-50
0x2_4308–0x2_430C	Reserved	—	—	—
0x2_4310	RXIC—Receive interrupt coalescing register	R/W	0x0000_0000	18.5.3.3.3/18-52
0x2_4314	RQUEUE*—Receive queue control register.	R/W	0x0080_0080	18.5.3.3.4/18-53
0x2_4318–0x2_432C	Reserved	—	—	—
0x2_4330	RBIFX*—Receive bit field extract control register	R/W	0x0000_0000	18.5.3.3.5/18-54
0x2_4334	RQFAR*—Receive queue filing table address register	R/W	0x0000_0000	18.5.3.3.6/18-56
0x2_4338	RQFCR*—Receive queue filing table control register	R/W	0x*nnnn_nnnn*	18.5.3.3.7/18-56
0x2_433C	RQFPR*—Receive queue filing table property register	R/W	0x*nnnn_nnnn*	18.5.3.3.8/18-57
0x2_4340	MRBLR—Maximum receive buffer length register	R/W	0x0000_0000	18.5.3.3.9/18-61
0x2_4344–0x2_4380	Reserved	—	—	—
0x2_4384	RBPTR0—RxBD pointer for ring 0	R/W	0x0000_0000	18.5.3.3.10/18-62
0x2_4388	Reserved	—	—	—
0x2_438C	RBPTR1*—RxBD pointer for ring 1	R/W	0x0000_0000	18.5.3.3.10/18-62
0x2_4390	Reserved	—	—	—
0x2_4394	RBPTR2*—RxBD pointer for ring 2	R/W	0x0000_0000	18.5.3.3.10/18-62
0x2_4398	Reserved	—	—	—
0x2_439C	RBPTR3*—RxBD pointer for ring 3	R/W	0x0000_0000	18.5.3.3.10/18-62
0x2_43A0	Reserved	—	—	—
0x2_43A4	RBPTR4*—RxBD pointer for ring 4	R/W	0x0000_0000	18.5.3.3.10/18-62
0x2_43A8	Reserved	—	—	—
0x2_43AC	RBPTR5*—RxBD pointer for ring 5	R/W	0x0000_0000	18.5.3.3.10/18-62
0x2_43B0	Reserved	—	—	—
0x2_43B4	RBPTR6*—RxBD pointer for ring 6	R/W	0x0000_0000	18.5.3.3.10/18-62
0x2_43B8	Reserved	—	—	—
0x2_43BC	RBPTR7*—RxBD pointer for ring 7	R/W	0x0000_0000	18.5.3.3.10/18-62
0x2_43C0–0x2_44400	Reserved	—	—	—

Table 18-4. Module Memory Map (continued)

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
0x2_4404	RBASE0—RxBD base address of ring 0	R/W	0x0000_0000	18.5.3.3.11/18-62
0x2_4408	Reserved	—	—	—
0x2_440C	RBASE1*—RxBD base address of ring 1	R/W	0x0000_0000	18.5.3.3.11/18-62
0x2_4410	Reserved	—	—	—
0x2_4414	RBASE2*—RxBD base address of ring 2	R/W	0x0000_0000	18.5.3.3.11/18-62
0x2_4418	Reserved	—	—	—
0x2_441C	RBASE3*—RxBD base address of ring 3	R/W	0x0000_0000	18.5.3.3.11/18-62
0x2_4420	Reserved	—	—	—
0x2_4424	RBASE4*—RxBD base address of ring 4	R/W	0x0000_0000	18.5.3.3.11/18-62
0x2_4428	Reserved	—	—	—
0x2_442C	RBASE5*—RxBD base address of ring 5	R/W	0x0000_0000	18.5.3.3.11/18-62
0x2_4430	Reserved	—	—	—
0x2_4434	RBASE6*—RxBD base address of ring 6	R/W	0x0000_0000	18.5.3.3.11/18-62
0x2_4438	Reserved	—	—	—
0x2_443C	RBASE7*—RxBD base address of ring 7	R/W	0x0000_0000	18.5.3.3.11/18-62
0x2_4440–0x2_44BC	Reserved	—	—	—
0x2_44C0	TMR_RXTS_H*—Rx timer time stamp register high	R/W	0x0000_0000	18.5.3.3.12/18-63
0x2_44C4	TMR_RXTS_L*—Rx timer time stamp register low	R/W	0x0000_0000	18.5.3.3.12/18-63
0x2_44C8–0x2_44FC	Reserved	—	—	—
eTSEC MAC Registers				
0x2_4500	MACCFG1—MAC configuration register 1	R/W	0x0000_0000	18.5.3.5.1/18-66
0x2_4504	MACCFG2—MAC configuration register 2	R/W	0x0000_7000	18.5.3.5.2/18-68
0x2_4508	IPGIFG—Inter-packet/inter-frame gap register	R/W	0x4060_5060	18.5.3.5.3/18-70
0x2_450C	HAFDUP—Half-duplex control	R/W	0x00A1_F037	18.5.3.5.4/18-71
0x2_4510	MAXFRM—Maximum frame length	R/W	0x0000_0600	18.5.3.5.5/18-72
0x2_4514–0x2_451C	Reserved	—	—	—
0x2_4520	MIIMCFG—MII management configuration	R/W	0x0000_0007	18.5.3.5.6/18-72
0x2_4524	MIIMCOM—MII management command	R/W	0x0000_0000	18.5.3.5.7/18-73
0x2_4528	MIIMADD—MII management address	R/W	0x0000_0000	18.5.3.5.8/18-74
0x2_452C	MIIMCON—MII management control	WO	0x0000_0000	18.5.3.5.9/18-74
0x2_4530	MIIMSTAT—MII management status	R	0x0000_0000	18.5.3.5.10/18-75
0x2_4534	MIIMIND—MII management indicator	R	0x0000_0000	18.5.3.5.11/18-75

Table 18-4. Module Memory Map (continued)

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
0x2_4538	Reserved	—	—	—
0x2_453C	IFSTAT—Interface status	R	0x0000_0000	18.5.3.5.12/18-76
0x2_4540	MACSTNADDR1—MAC station address register 1	R/W	0x0000_0000	18.5.3.5.13/18-76
0x2_4544	MACSTNADDR2—MAC station address register 2	R/W	0x0000_0000	18.5.3.5.14/18-77
0x2_4548	MAC01ADDR1*—MAC exact match address 1, part 1	R/W	0x0000_0000	18.5.3.5.15/18-78 18.5.3.5.16/18-78
0x2_454C	MAC01ADDR2*—MAC exact match address 1, part 2	R/W	0x0000_0000	
0x2_4550	MAC02ADDR1*—MAC exact match address 2, part 1	R/W	0x0000_0000	
0x2_4554	MAC02ADDR2*—MAC exact match address 2, part 2	R/W	0x0000_0000	
0x2_4558	MAC03ADDR1*—MAC exact match address 3, part 1	R/W	0x0000_0000	
0x2_455C	MAC03ADDR2*—MAC exact match address 3, part 2	R/W	0x0000_0000	
0x2_4560	MAC04ADDR1*—MAC exact match address 4, part 1	R/W	0x0000_0000	
0x2_4564	MAC04ADDR2*—MAC exact match address 4, part 2	R/W	0x0000_0000	
0x2_4568	MAC05ADDR1*—MAC exact match address 5, part 1	R/W	0x0000_0000	
0x2_456C	MAC05ADDR2*—MAC exact match address 5, part 2	R/W	0x0000_0000	
0x2_4570	MAC06ADDR1*—MAC exact match address 6, part 1	R/W	0x0000_0000	18.5.3.5.15/18-78 18.5.3.5.16/18-78
0x2_4574	MAC06ADDR2*—MAC exact match address 6, part 2	R/W	0x0000_0000	
0x2_4578	MAC07ADDR1*—MAC exact match address 7, part 1	R/W	0x0000_0000	
0x2_457C	MAC07ADDR2*—MAC exact match address 7, part 2	R/W	0x0000_0000	
0x2_4580	MAC08ADDR1*—MAC exact match address 8, part 1	R/W	0x0000_0000	
0x2_4584	MAC08ADDR2*—MAC exact match address 8, part 2	R/W	0x0000_0000	
0x2_4588	MAC09ADDR1*—MAC exact match address 9, part 1	R/W	0x0000_0000	
0x2_458C	MAC09ADDR2*—MAC exact match address 9, part 2	R/W	0x0000_0000	
0x2_4590	MAC10ADDR1*—MAC exact match address 10, part 1	R/W	0x0000_0000	
0x2_4594	MAC10ADDR2*—MAC exact match address 10, part 2	R/W	0x0000_0000	
0x2_4598	MAC11ADDR1*—MAC exact match address 11, part 1	R/W	0x0000_0000	
0x2_459C	MAC11ADDR2*—MAC exact match address 11, part 2	R/W	0x0000_0000	
0x2_45A0	MAC12ADDR1*—MAC exact match address 12, part 1	R/W	0x0000_0000	
0x2_45A4	MAC12ADDR2*—MAC exact match address 12, part 2	R/W	0x0000_0000	
0x2_45A8	MAC13ADDR1*—MAC exact match address 13, part 1	R/W	0x0000_0000	
0x2_45AC	MAC13ADDR2*—MAC exact match address 13, part 2	R/W	0x0000_0000	
0x2_45B0	MAC14ADDR1*—MAC exact match address 14, part 1	R/W	0x0000_0000	
0x2_45B4	MAC14ADDR2*—MAC exact match address 14, part 2	R/W	0x0000_0000	
0x2_45B8	MAC15ADDR1*—MAC exact match address 15, part 1	R/W	0x0000_0000	
0x2_45BC	MAC15ADDR2*—MAC exact match address 15, part 2	R/W	0x0000_0000	
0x2_45C0–0x2_467C	Reserved	—	—	—

Table 18-4. Module Memory Map (continued)

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
eTSEC Transmit and Receive Counters				
0x2_4680	TR64—Transmit and receive 64-byte frame counter	R/W	0x0000_0000	18.5.3.6.1/18-79
0x2_4684	TR127—Transmit and receive 65- to 127-byte frame counter	R/W	0x0000_0000	18.5.3.6.2/18-80
0x2_4688	TR255—Transmit and receive 128- to 255-byte frame counter	R/W	0x0000_0000	18.5.3.6.3/18-80
0x2_468C	TR511—Transmit and receive 256- to 511-byte frame counter	R/W	0x0000_0000	18.5.3.6.4/18-81
0x2_4690	TR1K—Transmit and receive 512- to 1023-byte frame counter	R/W	0x0000_0000	18.5.3.6.5/18-81
0x2_4694	TRMAX—Transmit and receive 1024- to 1518-byte frame counter	R/W	0x0000_0000	18.5.3.6.6/18-82
0x2_4698	TRMGV—Transmit and receive 1519- to 1522-byte good VLAN frame count	R/W	0x0000_0000	18.5.3.6.7/18-82
eTSEC Receive Counters				
0x2_469C	RBYT—Receive byte counter	R/W	0x0000_0000	18.5.3.6.8/18-83
0x2_46A0	RPKT—Receive packet counter	R/W	0x0000_0000	18.5.3.6.9/18-83
0x2_46A4	RFCS—Receive FCS error counter	R/W	0x0000_0000	18.5.3.6.10/18-83
0x2_46A8	RMCA—Receive multicast packet counter	R/W	0x0000_0000	18.5.3.6.11/18-84
0x2_46AC	RBCA—Receive broadcast packet counter	R/W	0x0000_0000	18.5.3.6.12/18-84
0x2_46B0	RXCF—Receive control frame packet counter	R/W	0x0000_0000	18.5.3.6.13/18-85
0x2_46B4	RXPF—Receive PAUSE frame packet counter	R/W	0x0000_0000	18.5.3.6.14/18-85
0x2_46B8	RXUO—Receive unknown OP code counter	R/W	0x0000_0000	18.5.3.6.15/18-86
0x2_46BC	RALN—Receive alignment error counter	R/W	0x0000_0000	18.5.3.6.16/18-86
0x2_46C0	RFLR—Receive frame length error counter	R/W	0x0000_0000	18.5.3.6.17/18-87
0x2_46C4	RCDE—Receive code error counter	R/W	0x0000_0000	18.5.3.6.18/18-87
0x2_46C8	RCSE—Receive carrier sense error counter	R/W	0x0000_0000	18.5.3.6.19/18-88
0x2_46CC	RUND—Receive undersize packet counter	R/W	0x0000_0000	18.5.3.6.20/18-88
0x2_46D0	ROVR—Receive oversize packet counter	R/W	0x0000_0000	18.5.3.6.21/18-89
0x2_46D4	RFRG—Receive fragments counter	R/W	0x0000_0000	18.5.3.6.22/18-89
0x2_46D8	RJBR—Receive jabber counter	R/W	0x0000_0000	18.5.3.6.23/18-90
0x2_46DC	RDRP—Receive drop counter	R/W	0x0000_0000	18.5.3.6.24/18-90
eTSEC Transmit Counters				
0x2_46E0	TBYT—Transmit byte counter	R/W	0x0000_0000	18.5.3.6.25/18-91
0x2_46E4	TPKT—Transmit packet counter	R/W	0x0000_0000	18.5.3.6.26/18-91
0x2_46E8	TMCA—Transmit multicast packet counter	R/W	0x0000_0000	18.5.3.6.27/18-92
0x2_46EC	TBCA—Transmit broadcast packet counter	R/W	0x0000_0000	18.5.3.6.28/18-92
0x2_46F0	TXPF—Transmit PAUSE control frame counter	R/W	0x0000_0000	18.5.3.6.29/18-93
0x2_46F4	TDFR—Transmit deferral packet counter	R/W	0x0000_0000	18.5.3.6.30/18-93

Table 18-4. Module Memory Map (continued)

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
0x2_46F8	TEDF—Transmit excessive deferral packet counter	R/W	0x0000_0000	18.5.3.6.31/18-94
0x2_46FC	TSCL—Transmit single collision packet counter	R/W	0x0000_0000	18.5.3.6.32/18-94
0x2_4700	TMCL—Transmit multiple collision packet counter	R/W	0x0000_0000	18.5.3.6.33/18-95
0x2_4704	TLCL—Transmit late collision packet counter	R/W	0x0000_0000	18.5.3.6.34/18-95
0x2_4708	TXCL—Transmit excessive collision packet counter	R/W	0x0000_0000	18.5.3.6.35/18-96
0x2_470C	TNCL—Transmit total collision counter	R/W	0x0000_0000	18.5.3.6.36/18-96
0x2_4710	Reserved	—	—	—
0x2_4714	TDRP—Transmit drop frame counter	R/W	0x0000_0000	18.5.3.6.37/18-97
0x2_4718	TJBR—Transmit jabber frame counter	R/W	0x0000_0000	18.5.3.6.38/18-97
0x2_471C	TFCS—Transmit FCS error counter	R/W	0x0000_0000	18.5.3.6.39/18-98
0x2_4720	TXCF—Transmit control frame counter	R/W	0x0000_0000	18.5.3.6.40/18-98
0x2_4724	TOVR—Transmit oversize frame counter	R/W	0x0000_0000	18.5.3.6.41/18-99
0x2_4728	TUND—Transmit undersize frame counter	R/W	0x0000_0000	18.5.3.6.42/18-99
0x2_472C	TFRG—Transmit fragments frame counter	R/W	0x0000_0000	18.5.3.6.43/18-100
eTSEC Counter Control and TOE Statistics Registers				
0x2_4730	CAR1—Carry register one register[4]	R	0x0000_0000	18.5.3.6.44/18-100
0x2_4734	CAR2—Carry register two register [4]	R	0x0000_0000	18.5.3.6.45/18-101
0x2_4738	CAM1—Carry register one mask register	R/W	0xFE03_FFFF	18.5.3.6.46/18-103
0x2_473C	CAM2—Carry register two mask register	R/W	0x000F_FFFD	18.5.3.6.47/18-104
0x2_4740	RREJ*—Receive filer rejected packet counter	R/W	0x0000_0000	18.5.3.6.48/18-105
0x2_4744–0x2_47FC	Reserved	—	—	—
Hash Function Registers				
0x2_4800	IGADDR0—Individual/group address register 0	R/W	0x0000_0000	18.5.3.7.1/18-106
0x2_4804	IGADDR1—Individual/group address register 1	R/W	0x0000_0000	
0x2_4808	IGADDR2—Individual/group address register 2	R/W	0x0000_0000	
0x2_480C	IGADDR3—Individual/group address register 3	R/W	0x0000_0000	
0x2_4810	IGADDR4—Individual/group address register 4	R/W	0x0000_0000	
0x2_4814	IGADDR5—Individual/group address register 5	R/W	0x0000_0000	
0x2_4818	IGADDR6—Individual/group address register 6	R/W	0x0000_0000	
0x2_481C	IGADDR7—Individual/group address register 7	R/W	0x0000_0000	
0x2_4820–0x2_487C	Reserved	—	—	—

Table 18-4. Module Memory Map (continued)

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
0x2_4880	GADDR0—Group address register 0	R/W	0x0000_0000	18.5.3.7.2/18-106
0x2_4884	GADDR1—Group address register 1	R/W	0x0000_0000	
0x2_4888	GADDR2—Group address register 2	R/W	0x0000_0000	
0x2_488C	GADDR3—Group address register 3	R/W	0x0000_0000	
0x2_4890	GADDR4—Group address register 4	R/W	0x0000_0000	
0x2_4894	GADDR5—Group address register 5	R/W	0x0000_0000	
0x2_4898	GADDR6—Group address register 6	R/W	0x0000_0000	
0x2_489C	GADDR7—Group address register 7	R/W	0x0000_0000	
0x2_48A0–0x2_4AFC	Reserved	—	—	—
eTSEC DMA Attribute Registers				
0x2_4B00–0x2_4BF4	Reserved	—	—	—
0x2_4BF8	ATTR—Attribute register	R/W	0x0000_0000	18.5.3.8.1/18-107
eTSEC Future Expansion Space				
–0x2_4D94	Reserved	—	—	—
eTSEC IEEE 1588 Registers				
0x2_4E00	TMR_CTRL*—Timer control register	R/W	0x0001_0001	18.5.3.9.1/18-108
0x2_4E04	TMR_TEVENT*—time stamp event register	W1C	0x0000_0000	18.5.3.9.2/18-110
0x2_4E08	TMR_TEMASK*—Timer event mask register	R/W	0x0000_0000	18.5.3.9.3/18-112
0x2_4E0C	TMR_PEVENT*—time stamp event register	R/W	0x0000_0000	18.5.3.9.4/18-112
0x2_4E10	TMR_PEMASK*—Timer event mask register	R/W	0x0000_0000	18.5.3.9.5/18-113
0x2_4E14	TMR_STAT*—time stamp status register	R/W	0x0000_0000	18.5.3.9.6/18-114
0x2_4E18	TMR_CNT_H*—timer counter high register	R/W	0x0000_0000	18.5.3.9.7/18-114
0x2_4E1C	TMR_CNT_L*—timer counter low register	R/W	0x0000_0000	18.5.3.9.7/18-114
0x2_4E20	TMR_ADD*—Timer drift compensation addend register	R/W	0x0000_0000	18.5.3.9.8/18-115
0x2_4E24	TMR_ACC*—Timer accumulator register	R/W	0x0000_0000	18.5.3.9.9/18-116
0x2_4E28	TMR_PRSC* -Timer prescale	R/W	0x0000_0002	18.5.3.9.10/18-116
0x2_4E2C	Reserved	—	—	—
0x2_4E30	TMROFF_H*—Timer offset high	R/W	0x0000_0000	18.5.3.9.11/18-117
0x2_4E34	TMROFF_L*—Timer offset low	R/W	0x0000_0000	18.5.3.9.11/18-117

Table 18-4. Module Memory Map (continued)

eTSEC1 Offset	Name[1]	Access [2]	Reset	Section/Page
0x2_4E40	TMR_ALARM1_H*—Timer alarm 1 high register	R/W	0xFFFF_FFFF	18.5.3.9.12/18-117
0x2_4E44	TMR_ALARM1_L*—Timer alarm 1 high register	R/W	0xFFFF_FFFF	
0x2_4E48	TMR_ALARM2_H*—Timer alarm 2 high register	R/W	0xFFFF_FFFF	
0x2_4E4C	TMR_ALARM2_L*—Timer alarm 2 high register	R/W	0xFFFF_FFFF	
0x2_4E50–0x2_4E7C	Reserved	—	—	—
0x2_4E80	TMR_FIPER1*—Timer fixed period interval	R/W	0xFFFF_FFFF	18.5.3.9.13/18-118
0x2_4E84	TMR_FIPER2*—Timer fixed period interval	R/W	0xFFFF_FFFF	
0x2_4E88	TMR_FIPER*3—Timer fixed period interval	R/W	0xFFFF_FFFF	
0x2_4EA0	TMR_ETTS1_H*—Time stamp of general purpose external trigger	R/W	0x0000_0000	18.5.3.9.14/18-119
0x2_4EA4	TMR_ETTS1_L*—Time stamp of general purpose external trigger	R/W	0x0000_0000	
0x2_4EA8	TMR_ETTS2_H*—Time stamp of general purpose external trigger	R/W	0x0000_0000	
0x2_4EAC	TMR_ETTS2_L*—Time stamp of general purpose external trigger	R/W	0x0000_0000	
0x2_4EB0 – 0x2_4FFF	Reserved	—	—	
Other eTSECs				
0x2_5000–0x2_5FFF	eTSEC2 REGISTERS[5]			

[1] Registers denoted * are new to the enhanced TSEC and not supported by PowerQUICC II Pro TSECs.

[2] Key: R = read only, WO = write only, R/W = read and write, LH = latches high, SC = self-clearing.

[3] Reset value of ENCTRL is configured from the value of RCWH[TSEC*n*M] which is loaded during reset (See Section 4.3.2.2, "Reset Configuration Word High Register (RCWHR)")

[4] Cleared on read.

[5] eTSEC2 has the same memory-mapped registers that are described for eTSEC1 from 0x 2_4000 to 0x2_4FFF, except the offsets are from 0x 2_5000 to 0x2_5FFF.

18.5.3 Memory-Mapped Register Descriptions

This section provides a detailed description of all the eTSEC registers. Because all of the eTSEC registers are 32 bits wide, only 32-bit register accesses are supported.

18.5.3.1 eTSEC General Control and Status Registers

This section describes general control and status registers used for both transmitting and receiving Ethernet frames. All of the registers are 32 bits wide.

18.5.3.1.1 Controller ID Register (TSEC_ID)

The controller ID register (TSEC_ID) is a read-only register. The TSEC_ID register is used to identify the eTSEC block and revision.

Offset eTSEC1:0x2_4000; eTSEC2:0x2_5000 — Access: Read only

	0–15	16–23	24–31
R	TSEC_ID	TSEC_REV_MJ	TSEC_REV_MN
W			
Reset	0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0	0 0 0 0 0 0 0 1	0 0 0 0 0 1 1 0

Figure 18-2. TSEC_ID Register

Table 18-10 describes the fields of the TSEC_ID register.

Table 18-5. TSEC_ID Field Descriptions

Bits	Name	Description
0–15	TSEC_ID	Value identifying the eTSEC (10/100/1000 Ethernet MAC). 0124 Unique identifier for eTSEC with 8 Rx and 8 Tx BD rings..
16–23	TSEC_REV_MJ	Value identifies the major revision of the eTSEC. 00 Initial revision (silicon revision 1.0) 01 Silicon revision 2.0 and 2.1
24–31	TSEC_REV_MN	Value identifies the minor revision of the eTSEC. 05 Initial revision (silicon revision 1.0) 06 Silicon revision 2.0 and 2.1

18.5.3.1.2 Controller ID Register (TSEC_ID2)

The controller ID register (TSEC_ID2) is a read-only register. The TSEC_ID2 register is used to identify the eTSEC block configuration.

Offset eTSEC1:0x2_4004; eTSEC2:0x2_5004 — Access: Read only

	0–7	8–15	16–23	24–31
R	—	TSEC_INT	—	TSEC_CFG
W				
Reset	0 0 0 0 0 0 0 0	1 1 1 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 1 1 0 0 0 0

Figure 18-3. TSEC_ID2 Register

Table 18-6 describes the fields of the TSEC_ID2 register.

Table 18-6. TSEC_ID2 Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	TSEC_INT	Interface mode support. See Table 18-7 for settings.
16–23	—	Reserved
24–31	TSEC_CFG	Value identifies configuration options of the eTSEC. 00 eTSEC multiple ring, Rx TOE, Filer and Tx TOE supports are off F0 eTSEC multiple ring, Rx TOE, Filer and Tx TOE supports are on 30 eTSEC multiple ring support is OFF and Rx TOE, Filer and Tx TOE supports are on 50 eTSEC multiple ring and filer supports are OFF and Rx TOE and Tx TOE supports are on

Table 18-7 describes the field settings for TSEC_ID2[TSEC_INT].

Table 18-7. TSEC_ID2[TSEC_INT] Field Settings

Bit	Mode
8	0 1588 protocol not supported 1 1588 protocol supported
9	0 SGMII not supported 1 SGMII supported
10	0 Ethernet mode not supported 1 Ethernet mode supported
11–13	Reserved
14	0 Can be configured to run in Ethernet normal/full mode 1 Ethernet normal/full mode off
15	0 Can be configured to run in Ethernet reduced mode 1 Ethernet reduced mode off

18.5.3.1.3 Interrupt Event Register (IEVENT)

Interrupt events cause bits in the IEVENT register to be set. Software may poll this register at any time to check for pending interrupts. If an event occurs and its corresponding enable bit is set in the interrupt mask register (IMASK), the event also causes a hardware interrupt at the PIC. A bit in the interrupt event register is cleared by writing a 1 to that bit position. A write of 0 has no effect.

Each eTSEC can issue three kinds of hardware interrupt to the PIC:

1. Transmit data frame interrupts—Issued whenever bits TXB or TXF of IEVENT are set to 1 and either transmit interrupt coalescing is disabled or the interrupt coalescing thresholds have been met for TXF. To negate this hardware interrupt, software must clear both TXB and TXF bits.
2. Receive data frame interrupts—Issued whenever bits RXB or RXF of IEVENT are set to 1 and either receive interrupt coalescing is disabled or the interrupt coalescing thresholds have been met for RXF. To negate this hardware interrupt, software must clear both RXB and RXF bits.

3. Error, diagnostic, and special interrupts—Issued whenever bits MAG, GTSC, GRSC, TXC, RXC, BABR, BABT, LC, CRL, FGPI, FIR, FIQ, DPE, PERR, EBERR, TXE, XFUN, BSY, MSRO, MMRD, or MMRW of IEVENT are set to 1. Software must clear all of these bits to negate an error/diagnostic/special hardware interrupt.
   - Magic Packet reception event is: MAG
   - Operational diagnostics are events on: GTSC, GRSC, TXC, and RXC
   - Interrupts resulting from errors/problems detected in the network or transceiver are: BABR, BABT, LC, and CRL
   - Interrupts resulting from internal or combination errors are: FIR, FIQ, DPE, PERR, EBERR, TXE, XFUN, and BSY
   - Special function interrupts are: FGPI, MSRO, MMRD, and MMRW

Some of the error interrupts are independently counted in the MIB block counters. Software may choose to mask off these interrupts because these errors are visible to network management through the MIB counters.

Figure 18-4 describes the definition for the IEVENT register.

Offset eTSEC1:0x2_4010; eTSEC2: 0x2_5010 Access: w1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BABR	RXC	BSY	EBERR	—	MSRO	GTSC	BABT	TXC	TXE	TXB	TXF	—	LC	CRL	XFUN
W	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c	w1c	w1c
Reset	All zeros															

	16	17–19	20	21	22	23	24	25–26	27	28	29	30	31
R	RXB	—	MAG	MMRD	MMWR	GRSC	RXF	—	FGPI	FIR	FIQ	DPE	PERR
W	w1c		w1c	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c	w1c
Reset	All zeros												

Figure 18-4. IEVENT Register Definition

Table 18-8 describes the fields of the IEVENT register.

Table 18-8. IEVENT Field Descriptions

Bits	Name	Description
0	BABR	Babbling receive error. This bit indicates that a frame was received with length in excess of the MAC's maximum frame length register while MACCFG2[Huge Frame] is set. 0 Excessive frame not received. 1 Excessive frame received.
1	RXC	Receive control interrupt. A control frame was received while MACCFG1[Rx_Flow] is set. As soon as the transmitter finishes sending the current frame, a pause operation is performed. 0 Control frame not received. 1 Control frame received.
2	BSY	Busy condition interrupt. Indicates that a frame was received and discarded due to a lack of buffers. 0 No frame received and discarded. 1 Frame received and discarded.

Table 18-8. IEVENT Field Descriptions (continued)

Bits	Name	Description
3	EBERR	Internal bus error. This bit indicates that a system bus error occurred while a DMA transaction was underway. As a result, transferred data is expected to be partially or completely invalid. 0 No system bus error occurred. 1 System bus error occurred.
4	—	Reserved
5	MSRO	MIB counter overflow. This interrupt is asserted if the count for one of the MIB counters has exceeded the size of its register. 0 MIB count not exceeding its register size. 1 MIB count exceeds its register size.
6	GTSC	Graceful transmit stop complete. This interrupt is asserted for one of two reasons. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted. • A graceful stop, which was initiated by setting DMACTRL[GTS], is now complete. • A transmission of a flow control PAUSE frame, which was initiated by setting TCTRL[TFC_PAUSE], is now complete. 0 No graceful stop interrupt. 1 Graceful stop requested.
7	BABT	Babbling transmit error. This bit indicates that the transmitted frame length has exceeded the value in the MAC's maximum frame length register and MACCFG2[Huge Frame] is cleared. Frame truncation occurs when this condition occurs. 0 Transmitted frame length not exceeding maximum frame length. 1 Transmitted frame length exceeding maximum frame length when MACCFG2[Huge Frame] = 0.
8	TXC	Transmit control interrupt. This bit indicates that a control frame was transmitted. 0 Control frame not transmitted. 1 Control frame transmitted.
9	TXE	Transmit error. This bit indicates that an error occurred on the transmitted channel that has caused TSTAT[THLT] to be set by the eTSEC. This bit is set whenever any transmit error occurs that causes the transmitter to halt (EBERR, LC, CRL, XFUN). 0 No transmit channel error occurred. 1 Transmit channel error occurred.
10	TXB	Transmit buffer. This bit indicates that a transmit buffer descriptor was updated whose I (interrupt) bit was set in its status word and was not the last buffer descriptor of the frame. 0 No transmit buffer descriptor updated. 1 Transmit buffer descriptor updated.
11	TXF	Transmit frame interrupt. This bit indicates that a frame was transmitted and that the last corresponding transmit buffer descriptor (TxBD) was updated. This only occurs if the I (interrupt) bit in the status word of the buffer descriptor is set. The specific transmit queue that was updated has its TXF bit set in TSTAT. 0 No frame transmitted/TxBD not updated. 1 Frame transmitted/TxBD updated.
12	—	Reserved
13	LC	Late collision. This bit indicates that a collision occurred beyond the collision window (slot time) in half-duplex mode. The frame is truncated with a bad CRC and the remainder of the frame is discarded. 0 No late collision occurred. 1 Late collision occurred.

Table 18-8. IEVENT Field Descriptions (continued)

Bits	Name	Description
14	CRL	Collision retry limit. This bit indicates that the number of successive transmission collisions has exceeded the MAC's half-duplex register's retransmission maximum count (HAFDUP[Retransmission Maximum]). The frame is discarded without being transmitted and transmission of the next frame commences. This only occurs while in half-duplex mode. 0 Successive transmission collisions do not exceed maximum. 1 Successive transmission collisions exceed maximum.
15	XFUN	Transmit FIFO underrun. This bit indicates that the transmit FIFO became empty before the complete frame was transmitted. 0 Transmit FIFO not underrun. 1 Transmit FIFO underrun.
16	RXB	Receive buffer. This bit indicates that a receive buffer descriptor was updated which had the I (Interrupt) bit set in its status word and was not the last buffer descriptor of the frame. 0 Receive buffer descriptor not updated. 1 Receiver buffer descriptor updated.
17–19	—	Reserved
20	MAG	Magic Packet detected when the eTSEC is in Magic Packet detection mode (MACCFG2[MPEN] = 1). 0 No Magic Packet received, or Magic Packet mode was not enabled. 1 A Magic Packet was received while in Magic Packet mode. MACCFG2[MPEN] is also cleared upon receiving the Magic Packet.
21	MMRD	MII management read completion 0 MII management read not issued or in process. 1 MII management read completed that was initiated by a user through the MII Scan or Read cycle command.
22	MMWR	MII management write completion 0 MII management write not issued or in process. 1 MII management write completed that was initiated by a user write to the MIIMCON register.
23	GRSC	Graceful receive stop complete. This interrupt is asserted if a graceful receive stop is completed. It allows the user to know if the system has completed the stop and it is safe to write to receive registers (status, control or configuration registers) that are used by the system during normal operation. 0 Graceful stop not completed. 1 Graceful stop completed.
24	RXF	Receive frame interrupt. This bit indicates that a frame was received and the last receive buffer descriptor (RxBD) in that frame was updated. This occurs either if the I (interrupt) bit in the buffer descriptor status word is set, or an overrun error occurs. The specific receive queue that was updated has its RXF bit set in RSTAT. 0 Frame not received. 1 Frame received.
25–26	—	Reserved
27	FGPI	Filer generated general purpose interrupt on a set of filer rule match. This bit will be set upon reception of a frame that matches a GPI rule sequence that is specified in the filer. It is synchronized with the setting of RXF. 0 No filer generated interrupt has occurred. 1 The filer has accepted a frame via a matching rule that the RQFCR[GPI] bit set.

Table 18-8. IEVENT Field Descriptions (continued)

Bits	Name	Description
28	FIR	The receive queue filer result is invalid, either because not enough time between frames was available to find a matching rule, or no entry in the filer table could be matched. 0 Receive queue filer reached a definite result; however, bit FIQ may still be set if a frame was filed to a disabled RxBD ring. 1 Receive queue filer was unable to reach a definite result. In this case, bit FIQ is also set if no entry in the filer table could provide a rule match.
29	FIQ	Filed frame to invalid receive queue. This bit indicates that either the receive queue filer chose to DMA a received frame to a disabled RxBD ring, or that no rule in the filer table could be matched. 0 Received frames filed to valid queues or rejected. Note that a frame may be rejected if the filer has insufficient time to reach a conclusive result between frames, in which case bit FIR is set. 1 Received frames filed to RxBD rings that are not enabled. The frame is discarded. If bit FIR is also set this indicates that the filer exhausted all of its table entries without a rule match.
30	DPE	Internal data parity error. This bit indicates that the eTSEC has detected a parity error on its stored data, which is likely to compromise the validity of recently transferred frames. 0 No parity errors detected. 1 Data held in the FIFO or filer arrays is expected to be corrupted due to a parity error.
31	PERR	Receive frame parse error for TCP/IP off-load. This bit indicates that a received frame could not be parsed unambiguously, due to encapsulated header type fields contradicting each other. 0 Received frame parsed successfully. 1 Received frame parse revealed header inconsistencies.

18.5.3.1.4 Interrupt Mask Register (IMASK)

The interrupt mask register provides control over which possible interrupt events in the IEVENT register are permitted to participate in generating hardware interrupts to the PIC. All implemented bits in this register are R/W and cleared upon a hardware reset. If the corresponding bits in both the IEVENT and IMASK registers are set, the PIC receives an interrupt (for each eTSEC these are grouped into transmit, receive, and error/diagnostic interrupts). The interrupt signal remains asserted until either the IEVENT bit is cleared, by writing a 1 to it, or by writing a 0 to the corresponding IMASK bit.

Figure 18-5 describes the IMASK register.

Offset eTSEC1:0x2_4014; eTSEC2:0x2_5014 Access: Read/Write

	0	1	2	3	4	5	6	7
R W	BREN	RXCEN	BSYEN	EBERREN	—	MSROEN	GTSCEN	BTEN
Reset	All zeros							

	8	9	10	11	12	13	14	15
R W	TXCEN	TXEEN	TXBEN	TXFEN	—	LCEN	CRLEN	XFUNEN
Reset	All zeros							

	16	17–19	20	21	22	23
R W	RXBEN	—	MAGEN	MMRDEN	MMWREN	GRSCEN
Reset	All zeros					

	24	25–26	27	28	29	30	31
R W	RXFEN	—	FGPIEN	FIREN	FIQEN	DPEEN	PERREN
Reset	All zeros						

Figure 18-5. IMASK Register Definition

Table 18-9 describes the fields of the IMASK register.

Table 18-9. IMASK Field Descriptions

Bits	Name	Description
0	BREN	Babbling receiver interrupt enable
1	RXCEN	Receive control interrupt enable
2	BSYEN	Busy interrupt enable
3	EBERREN	Ethernet controller bus error enable
4	—	Reserved
5	MSROEN	MIB counter overflow interrupt enable
6	GTSCEN	Graceful transmit stop complete interrupt enable
7	BTEN	Babbling transmitter interrupt enable
8	TXCEN	Transmit control interrupt enable
9	TXEEN	Transmit error interrupt enable
10	TXBEN	Transmit buffer interrupt enable
11	TXFEN	Transmit frame interrupt enable
12	—	Reserved

Table 18-9. IMASK Field Descriptions (continued)

Bits	Name	Description
13	LCEN	Late collision enable
14	CRLEN	Collision retry limit enable
15	XFUNEN	Transmit FIFO underrun enable
16	RXBEN	Receive buffer interrupt enable
17–19	—	Reserved
20	MAGEN	Magic packet received interrupt enable
21	MMRDEN	MII management read completion interrupt enable
22	MMWREN	MII management write completion interrupt enable
23	GRSCEN	Graceful receive stop complete interrupt enable
24	RXFEN	Receive frame interrupt enable
25–26	—	Reserved
27	FGPIEN	Filer general purpose interrupt enable
28	FIREN	Filer invalid result interrupt enable
29	FIQEN	Filed frame to invalid queue interrupt enable
30	DPEEN	Data parity error interrupt enable
31	PERREN	Receive frame parse error enable

18.5.3.1.5 Error Disabled Register (EDIS)

Figure 18-6 describes the definition for the EDIS register. The error disabled register allows the user to disable an error interruption, possibly to avoid spurious error indications external to the eTSECs.

Offset eTSEC1:0x2_4018; eTSEC2:0x2_5018 Access: Read/Write

	0–1	2	3	4–6	7	8	9	10–12	13	14	15
R/W	—	BSYDIS	EBERRDIS	—	BABTDIS	—	TXEDIS	—	LCDIS	CRLDIS	XFUNDIS
Reset	All zeros										

	16–27	28	29	30	31
R/W	—	FIRDIS	FIQDIS	DPEDIS	PERRDIS
Reset	All zeros				

Figure 18-6. EDIS Register Definition

Table 18-10 describes the fields of the EDIS register.

Table 18-10. EDIS Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2	BSYDIS	Busy disable. 0 Allow eTSEC to report IEVENT[BSY] status and halt buffer descriptor queue if BSY condition occurs. 1 Do not set IEVENT[BSY] and do not halt buffer descriptor queue if BSY condition occurs.
3	EBERRDIS	Ethernet controller bus error disable. 0 Allow eTSEC to report IEVENT[EBERR] status and halt buffer descriptor queue if EBERR condition occurs. 1 Do not set IEVENT[EBERR] and do not halt buffer descriptor queue if EBERR condition occurs.
4–6	—	Reserved
7	BABTDIS	Babbling transmit error disable. 0 Allow eTSEC to report IEVENT[BABT] status and set the buffer descriptor TR field. 1 Do not set IEVENT[BABT] nor the buffer descriptor TR field.
8	—	Reserved
9	TXEDIS	Transmit error disable. 0 Allow eTSEC to report IEVENT[TXE] status. 1 Do not set IEVENT[TXE] if TXE condition occurs.
10–12	—	Reserved
13	LCDIS	Late collision disable. 0 Allow eTSEC to report IEVENT[LC] status, set the buffer descriptor LC field, and halt buffer descriptor queue if LC condition occurs. 1 Do not set IEVENT[LC] nor the buffer descriptor LC field, and do not halt buffer descriptor queue if LC condition occurs.
14	CRLDIS	Collision retry limit disable. 0 Allow eTSEC to report IEVENT[CRL] status, set the buffer descriptor RL field, and halt buffer descriptor queue if CRL condition occurs. 1 Do not set IEVENT[CRL] nor the buffer descriptor RL field, and do not halt buffer descriptor queue if CRL condition occurs.
15	XFUNDIS	Transmit FIFO underrun disable. 0 Allow eTSEC to report IEVENT[XFUN] status, set the buffer descriptor UN field, and halt buffer descriptor queue if XFUN condition occurs. 1 Do not set IEVENT[XFUN] nor the buffer descriptor UN field, and do not halt buffer descriptor queue if XFUN condition occurs.
16–27	—	Reserved
28	FIRDIS	Filer invalid result error disable. 0 Allow eTSEC to report IEVENT[FIR] status. 1 Do not set IEVENT[FIR] if eTSEC fails to reach a definite filer result when attempting to file a received frame, but discard the frame silently.
29	FIQDIS	Filed frame to invalid queue error disable. 0 Allow eTSEC to report IEVENT[FIQ] status. 1 Do not set IEVENT[FIQ] if eTSEC attempts to file a received frame to an invalid (disabled) RxBD ring, but discard the frame silently.

Table 18-10. EDIS Field Descriptions (continued)

Bits	Name	Description
30	DPEDIS	Data parity error disable. 0 Allow eTSEC to report IEVENT[DPE] status. 1 Do not set IEVENT[DPE] if a parity error occurs in eTSEC's FIFO or filer arrays.
31	PERRDIS	Receive frame parse error disable. 0 Allow eTSEC to report IEVENT[PERR] status. 1 Do not set IEVENT[PERR] if a parse error occurs on a received frame.

18.5.3.1.6 Ethernet Control Register (ECNTRL)

ECNTRL is a register writable by the user to reset, configure, and initialize the eTSEC. Note that the FIFM, GMIIM, RPM, and RMM fields are read-only, having been set after sampling signals at power-on-reset. (Refer to the TSEC mode inSection 4.3.2.2, "Reset Configuration Word High Register (RCWHR)."). The reset value is configured from the value of RCWH[TSECnM] which is loaded during reset.

Figure 18-7 describes the definition for the ECNTRL register.

Offset eTSEC1:0x2_4020; eTSEC2:0x2_5020 Access: Mixed

	0–15
R/W	—
Reset	All zeros

	16	17	18	19	20–24	25	26	27	28	29	30	31
R	FIFM	CLRCNT	AUTOZ	STEN	—	GMIIM	TBIM	RPM	R100M	RMM	SGMIIM	—
W		CLRCNT	AUTOZ	STEN	—				R100M			
Reset	0	0	0	0	0 0 0 0 0	R	R	R	0	R	R	0

Figure 18-7. ECNTRL Register Definition

Table 18-11 describes the fields of the ECNTRL register.

Table 18-11. ECNTRL Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	FIFM	FIFO mode. Not supported.
17	CLRCNT	Clear all statistics counters 0 Allow MIB counters to continue to increment. 1 Reset all MIB counters. This bit is self-resetting.
18	AUTOZ	Automatically zero MIB counter values. 0 The user must write the addressed counter zero after a host read. 1 The addressed counter value is automatically cleared to zero after a host read. This is a steady state signal and must be set prior to enabling the Ethernet controller and must not be changed without proper care.

Table 18-11. ECNTRL Field Descriptions (continued)

Bits	Name	Description
19	STEN	MIB counter statistics enabled. 0 Statistics not enabled 1 Enables internal counters to update This is a steady state signal and must be set prior to enabling the Ethernet controller and must not be changed without proper care.
20–24	—	Reserved
25	GMIIM	GMII interface mode. If this bit is set, a PHY with a RGMII interface is expected to be connected. If cleared, a PHY with an MII or RMII interface is expected. The user should then set MACCFG2[I/F Mode] accordingly. The state of this status bit is defined during power-on reset. 0 MII or RMII mode interface expected 1 RGMII mode interface expected
26	TBIM	(Reduced) ten-bit interface mode. If this bit is set, reduced ten-bit interface (RTBI) mode is enabled. This bit can be pin-configured at reset to set or clear. 0 MII or RMII mode interface 1 RTBI mode interface
27	RPM	Reduced-pin mode for Gigabit interfaces. If this bit is set, a reduced-pin interface is expected on Ethernet interfaces. RPM and RMM are never set together. This register can be pin-configured at reset to 0 or 1. 0 SGMII or MII in non-reduced-pin mode configuration 1 RGMII or RTBI reduced-pin mode
28	R100M	RGMII/RMII 100 mode. This bit is ignored unless SGMIIM, RPM or RMM are set and MACCFG2[I/F Mode] is assigned to 10/100 (01). 0 RGMII is in 10 Mbps mode; RMII is in 10 Mbps mode, and every 10th RMII Reference clock is used to transfer data SGMII is in 10 Mbps mode, and every 100th SGMII Reference clock is used to transfer data 1 RGMII is in 100 Mbps mode; RMII is in 100 Mbps mode, and data is transferred on every Reference clock SGMII is in 100 Mbps mode, and every 10th SGMII Reference clock is used to transfer data This bit must be cleared for 1-Gbps SGMII operation.
29	RMM	Reduced-pin mode for 10/100 interfaces. If this bit is set, an RMII pin interface is expected. RMM must be 0 if RPM = 1. This register can be pin-configured at reset to 0 or 1. 0 Non-RMII interface mode 1 RMII interface mode
30	SGMIIM	Serial GMII mode. If this bit is set, a SGMII pin interface is expected to be connected via an on chip SerDes. This register can be pin-configured at reset to 0 or 1. 0 SGMII mode disabled. eTSEC connected via a parallel interface. 1 SGMII mode enabled.
31	—	Reserved

The different interface configurations indicated by registers ECNTRL and MACCFG2 are summarized in Table 18-12.

Table 18-12. eTSEC Interface Configurations

Interface Mode	ECNTRL Field							MACCFG2 Field
	FIFM[1]	GMIIM	TBIM	RPM	R100M	RMM	SGMIIM	I/F Mode
RTBI 1Gbps	0	0	1	1	0	0	0	10
RGMII 1Gbps	0	1	0	1	0	0	0	10
RGMII 100 Mbps	0	1	0	1	1	0	0	01
RGMII 10 Mbps	0	1	0	1	0	0	0	01
MII 10/100 Mbps	0	0	0	0	0	0	0	01
RMII 100 Mbps	0	0	0	0	1	1	0	01
RMII 10 Mbps	0	0	0	0	0	1	0	01
SGMII 1 Gbps	0	0	1	0	0	0	1	10
SGMII 100 Mbps	0	0	1	0	1	0	1	01
SGMII 10 Mbps	0	0	1	0	0	0	1	01

[1] FIFO mode not supported.

18.5.3.1.7 Pause Time Value Register (PTV)

PTV is a 32-bit register written by the user to store the pause duration used when the eTSEC initiates an IEEE 802.3 PAUSE control frame through TCTRL[TFC_PAUSE]. The low-order 16 bits (PT) represent the pause time and the high-order 16 bits (PTE) represent the extended pause control parameter. The pause time is measured in units of *pause_quanta*, equal to 512 bit times. The pause time can range from 0 to 65,535 *pause_quanta*, or 0 to 33,553,920 bit times. See Section 18.6.2.9, "Flow Control," for additional details. Figure 18-8 describes the definition for the PTV register.

Offset eTSEC1:0x2_4028; eTSEC2:0x2_5028 Access: Read/Write

	0–15	16–31
R/W	PTE	PT
Reset	All zeros	

Figure 18-8. PTV Register Definition

Table 18-13 describes the fields of the PTV register.

Table 18-13. PTV Field Descriptions

Bits	Name	Description
0–15	PTE	Extended pause control. This field allows software to add a 16-bit additional control parameter into the PAUSE frame to be sent when TCTRL[TFC_PAUSE] is set. Note that current IEEE 802.3 PAUSE frame format requires this parameter to be cleared.
16–31	PT	Pause time value. Represents the 16-bit pause quanta (that is, 512 bit times). This pause value is used as part of the PAUSE frame to be sent when TCTRL[TFC_PAUSE] is set. See Section 18.6.2.9, "Flow Control," on page 18-149 for more information.

18.5.3.1.8 DMA Control Register (DMACTRL)

DMACTRL is writable by the user to configure the DMA block. Figure 18-9 describes the definition for the DMACTRL register.

Offset eTSEC1:0x2_402C; eTSEC2:0x2_502C — Access: Read/Write

	0–15	16	17–23	24	25	26	27	28	29	30	31
R/W	—	LE	—	TDSEN	TBDSEN	—	GRS	GTS	TOD	WWR	WOP
Reset	All zeros										

Figure 18-9. DMACTRL Register

Table 18-14 describes the fields of the DMACTRL register.

Table 18-14. DMACTRL Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	LE	Little-endian descriptor mode enable. This bit controls both the reading and writing of descriptors; data buffers are always transferred in network byte order. 0 RxBDs and TxBDs are interpreted with big-endian byte ordering, as shown in Section 18.6.6.1, "Data Buffer Descriptors." 1 RxBDs and TxBDs are interpreted with little-endian byte ordering. That is, the 16 bits of flags are considered a complete half-word unit, the buffer length is considered another complete half-word unit, and the buffer pointer is considered a complete word unit.
17–23	—	Reserved
24	TDSEN	Tx Data snoop enable. 0 Disables snooping of all transmit frames from memory. 1 Enables snooping of all transmit frames from memory.
25	TBDSEN	TxBD snoop enable. 0 Disables snooping of all transmit BD memory accesses. 1 Enables snooping of all transmit BD memory accesses.
26	—	Reserved

Table 18-14. DMACTRL Field Descriptions (continued)

Bits	Name	Description
27	GRS	Graceful receive stop. If this bit is set, the Ethernet controller stops receiving frames following completion of the frame currently being received. (That is, after a valid end of frame was received). The contents of the Rx FIFO are then written to memory, and the IEVENT[GRSC] is set to indicate that all current receive buffers have been closed. Because the receive enable bit of the MAC may still be set, the MAC may continue to receive but the eTSEC ignores the receive data until GRS is cleared. If this bit is cleared, the eTSEC scans the input data stream for the start of a new frame (preamble sequence and start of frame delimiter) and the first valid frame received uses the next RxBD. If GRS is set, the user must monitor the graceful receive stop complete (GRSC) bit in the IEVENT register to insure that the graceful receive stop was completed. The user can then clear IEVENT[GRSC] and can write to receive registers that are accessible to both user and the eTSEC hardware without fear of conflict. 0 eTSEC scans input data stream for valid frame. 1 eTSEC stops receiving frames following completion of current frame.
28	GTS	Graceful transmit stop. If this bit is set, the Ethernet controller stops transmission after all frames that are currently in the Tx FIFO or scheduled have been transmitted, and the GTSC interrupt in the IEVENT register is asserted. A frame that has started reading buffer descriptors or data from memory is read to completion and transmitted before the GTSC interrupt occurs. However, if no frame has been scheduled for transmission and the Tx FIFO is empty, the GTSC interrupt is asserted immediately. Once transmission has completed, clearing GTS “restart” transmit. 0 Controller continues. 1 Controller stops transmission after completion of current frame.
29	TOD	Transmit on demand for TxBD ring 0. This bit is applicable only to the transmitter, and requires both TCTRL[TXSCHED] = 00 and DMACTRL[WOP] = 0. If 1 is written to this bit, the eTSEC immediately begins fetching the next TxBD from ring 0, avoiding waiting the normal polling time to check the TxBD’s R bit. This bit is always read as 0. 0 eTSEC continues waiting for the TxBD ring 0 poll timer to expire. 1 eTSEC immediately fetches a new TxBD from ring 0, and resets the poll timer.
30	WWR	Write with response. This bit gives the user the assurance that a BD was updated in memory before it receives an interrupt concerning a transmit or receive frame. 0 Do not wait for acknowledgement from system for BD writes before setting IEVENT bits. 1 Before setting IEVENT bits TXB, TXF, TXE, XFUN, LC, CRL, RXB, RXF, the eTSEC waits for acknowledgement from system that the transmit or receive BD being updated was stored in memory.
31	WOP	Wait or poll for TxBD ring 0. This bit, which is applicable only to the transmitter and when TCTRL[TXSCHED] = 00, provides the user the option for the eTSEC to periodically poll TxBDs or to wait for software to tell eTSEC to fetch a buffer descriptor. While operating in the “Wait” mode, the eTSEC allows two additional reads of a descriptor which is not ready before entering a halt state. No interrupt is driven. To resume transmission, software must clear TSTAT[THLT]. 0 Poll TxBD on ring 0 every 512 serial clocks. 1 Do not poll, but wait for TSTAT[THLT] to be cleared by the user.

18.5.3.1.9 TBI Physical Address Register (TBIPA)

The TBIPA, shown in Figure 18-10, is writable by the user to assign a physical address to the TBI (or RTBI) for MII management configuration. The TBI registers are accessed at the offset of TBIPA. For detailed descriptions of the TBI registers (the MII register set for the ten-bit interface) please refer to Section 18.5.4, “Ten-Bit Interface (TBI).”

Offset eTSEC1:0x2_4030; eTSEC2:0x2_5030 Access: Read/Write

	0–26	27–31
R W	—	TBIPA
Reset	All zeros	

Figure 18-10. TBIPA Register Definition

Table 18-15 describes the fields of the TBIPA register.

Table 18-15. TBIPA Field Descriptions

Bits	Name	Description
0–26	—	Reserved
27–31	TBIPA	This field is used to program the PHY address of the ten-bit interface's MII management bus. To access the TBI register the user must write the TBIPA value to the MIIMADD [PHY Address] register located in the MAC register section. PHY Address 0 is reserved. Refer to Section 18.5.3.5.8, "MII Management Address Register (MIIMADD)."

18.5.3.2 eTSEC Transmit Control and Status Registers

This section describes the control and status registers that are used specifically for transmitting Ethernet frames. All of the registers are 32 bits wide.

18.5.3.2.1 Transmit Control Register (TCTRL)

This register is writable by the user to configure the transmit block. Figure 18-11 describes the TCTRL register.

Offset eTSEC1:0x2_4100; eTSEC2:0x2_5100 Access: Mixed

	0–16	17	18	19	20	21–26	27	28	29–30	31
R W	—	IPCSEN	TUCSEN	VLINS	THDF	—	RFC_PAUSE	TFC_PAUSE	TXSCHED	—
Reset	All zeros									

Figure 18-11. TCTRL Register Definition

Table 18-16 describes the fields of the TCTRL register.

Table 18-16. TCTRL Field Descriptions

Bits	Name	Description
0–16	—	Reserved
17	IPCSEN	IP header checksum generation enable. When set, the eTSEC offloads IPv4 header checksum generation. See Section 18.6.3.2, "Transmit Path Off-Load and Tx PTP Packet Parsing," on page 18-156. 0 IP header checksum generation is disabled even if enabled in a transmit frame control block. 1 IP header checksum generation is performed for IPv4 headers as determined by the settings in the current transmit frame control block.

Table 18-16. TCTRL Field Descriptions (continued)

Bits	Name	Description
18	TUCSEN	TCP/UDP header checksum generation enable. When set, the eTSEC offloads TCP or UDP header checksum generation. See Section 18.6.3.2, “Transmit Path Off-Load and Tx PTP Packet Parsing,” on page 18-156. 0 TCP or UDP header checksum generation is disabled even if enabled in a transmit frame control block. 1 TCP or UDP header checksum generation is performed as determined by the settings in the current transmit frame control block.
19	VLINS	VLAN (IEEE Std. 802.1Q) tag insertion enable. Applicable only for transmission through the Ethernet MAC. 0 Do not insert a VLAN tag into the frame. 1 Insert a VLAN tag into the frame. If the frame FCB has a valid VLAN field, use the FCB to source the VLAN control word, otherwise take the default VLAN control word from register DFVLAN.
20	THDF	Transmit half-duplex flow control under software control for 10-/100-Mbps half-duplex media. This bit is not self-resetting. 0 Disable back pressure 1 Back pressure is applied to media by raising carrier
21–26	—	Reserved
27	RFC_PAUSE	Receive flow control pause frame (written by the eTSEC). This read-only status bit is set if a flow control pause frame was received and the transmitter is paused for the duration defined in the received pause frame. This bit automatically clears after the pause duration is complete. 0 Pause duration complete. 1 Flow control pause frame received.
28	TFC_PAUSE	Transmit flow control pause frame. Set this bit to transmit a PAUSE frame. If this bit is set, the MAC stops transmission of data frames after the currently transmitting frame completes. Next, the MAC transmits a pause control frame with the duration value obtained from the PTV register. The TXC event occurs after sending the pause control frame. Finally, the controller clears TFC_PAUSE and resumes transmitting data frames as before. Note that pause control frames can still be transmitted if the Tx controller is stopped due to user assertion of DMACTRL[GTS] or reception of a PAUSE frame. 0 No request for Tx PAUSE frame pending or transmission complete. 1 Software request for Tx PAUSE frame pending.

Table 18-16. TCTRL Field Descriptions (continued)

Bits	Name	Description
29–30	TXSCHED	Transmit ring scheduling algorithm. This field determines which scheme the transmit scheduler uses to arbitrate between the enabled TxBD rings. The scheme chosen also controls how the DMACTRL and TQUEUE bits are interpreted. Ring polling is supported only by mode 00; the other modes require software to restart rings with the TSTAT register. TCP/IP offload can be enabled with any scheduling mode. 00 Single polled ring mode. TxBD ring 0 is the only ring serviced, even if other rings are enabled and ready. In this scheduler mode, the DMACTRL[WOP] and DMACTRL[TOD] bits control polling and retry behavior. This mode supports ring polling, and allows fetching of a non-ready TxBD to be retried twice. 01 Priority scheduling mode. All enabled TxBD rings are serviced in ascending ring index order. Once a non-ready TxBD has been fetched from the lowest-numbered ring, the eTSEC attempts to fetch TxBDs from the next enabled ring having a higher index, until transmission stops for lack of data. TSTAT records whenever a TxBD ring is exhausted. 10 Modified weighted round-robin scheduling mode. Each TxBD ring is polled in sequence for frames that are ready for transmission. If a non-ready TxBD is fetched from a ring, that ring is removed from the scheduling pool until software re-enables it. Ready frames are repeatedly transmitted from a chosen ring until its transmission quota is exhausted. The transmission quota for TxBD ring *n* is set to WT*n* × 64 bytes, where WT*n* is a weight from the TR03WT/TR47WT registers. If a ring transmits more data than its quota allows, the excess is deducted from its quota on the next transmission opportunity, thereby preventing large frames from monopolizing the eTSEC bandwidth. 11 Reserved
31	—	Reserved

18.5.3.2.2 Transmit Status Register (TSTAT)

This register is read/write-one-to-clear and is written by the eTSEC to convey DMA status information for each TxBD ring. The halt bit only has meaning for enabled rings. After processing transmit-related interrupts, software should use TSTAT to restart transmission from rings that may have been affected by the interrupt condition. In particular, an error condition that prevents eTSEC from continuing transmission halts DMA from all rings, including the ring that gave rise to the error. Figure 18-12 describes the TSTAT register.

Offset eTSEC1:0x2_4104; eTSEC2:0x2_5104 — Access: w1c

	0	1	2	3	4	5	6	7	8–15
R	THLT0	THLT1	THLT2	THLT3	THLT4	THLT5	THLT6	THLT7	—
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	
Reset	All zeros								

	16	17	18	19	20	21	22	23	24–31
R	TXF0	TXF1	TXF2	TXF3	TXF4	TXF5	TXF6	TXF7	—
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	
Reset	All zeros								

Figure 18-12. TSTAT Register Definition

Table 18-17 describes the fields of the TSTAT register.

Table 18-17. TSTAT Field Descriptions

Bits	Name	Description
0	THLT0	Transmit halt of ring 0. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN0], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again. Repeatable error conditions which cause halt include: Bus error: • Invalid BD or data address • Uncorrectable error on BD or data read TxBD programming errors: • Ready=1 and length=0
1	THLT1	Transmit halt of ring 1. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN1], or if no ready TxBDs can be fetched.DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again. Repeatable error conditions which cause halt include: Bus error: • Invalid BD or data address • Uncorrectable error on BD or data read TxBD programming errors: • Ready=1 and length=0
2	THLT2	Transmit halt of ring 2. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN2], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again. Repeatable error conditions which cause halt include: Bus error: • Invalid BD or data address • Uncorrectable error on BD or data read TxBD programming errors: • Ready=1 and length=0

Table 18-17. TSTAT Field Descriptions (continued)

Bits	Name	Description
3	THLT3	Transmit halt of ring 3. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN3], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again. Repeatable error conditions which cause halt include: Bus error: • Invalid BD or data address • Uncorrectable error on BD or data read TxBD programming errors: • Ready=1 and length=0
4	THLT4	Transmit halt of ring 4. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN4], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again. Repeatable error conditions which cause halt include: Bus error: • Invalid BD or data address • Uncorrectable error on BD or data read TxBD programming errors: • Ready=1 and length=0
5	THLT5	Transmit halt of ring 5. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN5], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again. Repeatable error conditions which cause halt include: Bus error: • Invalid BD or data address • Uncorrectable error on BD or data read TxBD programming errors: • Ready=1 and length=0

Table 18-17. TSTAT Field Descriptions (continued)

Bits	Name	Description
6	THLT6	Transmit halt of ring 6. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN6], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again. Repeatable error conditions which cause halt include: Bus error: • Invalid BD or data address • Uncorrectable error on BD or data read TxBD programming errors: • Ready=1 and length=0
7	THLT7	Transmit halt of ring 7. Set by the eTSEC if is no longer processing transmit frames from this TxBD ring, and DMA from this ring is disabled. To re-start transmission from this TxBD ring, this bit must be cleared by writing 1 to it. This bit is set only on a general error condition (as in IEVENT[TXE]), regardless of TQUEUE[EN7], or if no ready TxBDs can be fetched. DMACTRL[GTS] being set by the user does not cause this bit to be set. Software should examine the halted queue's buffer descriptors for repeatable error conditions before taking it out of the halt state. Failure to do so may cause an effective livelock, in which the error condition recurs and halts all queues again. Repeatable error conditions which cause halt include: Bus error: • Invalid BD or data address • Uncorrectable error on BD or data read TxBD programming errors: • Ready=1 and length=0
8–15	—	Reserved
16	TXF0	Transmit frame event occurred on ring 0. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring.
17	TXF1	Transmit frame event occurred on ring 1. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring.
18	TXF2	Transmit frame event occurred on ring 2. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring.
19	TXF3	Transmit frame event occurred on ring 3. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring.
20	TXF4	Transmit frame event occurred on ring 4. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring.
21	TXF5	Transmit frame event occurred on ring 5. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring.
22	TXF6	Transmit frame event occurred on ring 6. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring.

Table 18-17. TSTAT Field Descriptions (continued)

Bits	Name	Description
23	TXF7	Transmit frame event occurred on ring 7. Set by the eTSEC if IEVENT[TXF] was set in relation to transmitting a frame from this ring.
24–31	—	Reserved

18.5.3.2.3 Default VLAN Control Word Register (DFVLAN)

This register defines the default value for the VLAN Ethertype and control word when VLAN tags are automatically inserted by the eTSEC, and no per-frame VLAN data is supplied by software. On receive, this register defines a customizable VLAN Ethertype for automatic deletion. Note that an Ethertype of 0x8808 (Control Word) is not permitted as a custom VLAN tag. Frames with an Ethertype of 0x8808 are dropped by the receiver. In the case of frames containing stacked VLAN tags, this register defines the tag associated with the outer or metropolitan area VLAN. Figure 18-13 describes the DFVLAN register.

Offset eTSEC1:0x2_4108; eTSEC2:0x2_5108 Access: Read/Write

	0–15	16–18	19	20–31
R/W	TAG	PRI	CFI	VID
Reset	1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	0 0 0	0	0 0 0 0 0 0 0 0 0 0 0 0

Figure 18-13. DFVLAN Register Definition

Table 18-18 describes the fields of the DFVLAN register.

Table 18-18. DFVLAN Field Descriptions

Bits	Name	Description
0–15	TAG	This is the default Ethertype used to tag VLAN frames. On transmit, this tag is inserted ahead of the VLAN control word; TAG should be set to 0x8100 for IEEE 802.1Q VLAN. On receive, an Ethertype matching TAG or an Ethertype of 0x8100 marks a VLAN-tagged frame. Note that if using DFVLAN to set a custom ethertype (that is, using a value other than 0x8100), packets received with a custom tag are not counted by any of the RMON counters. Affected counters include TRMGV, RMCA, RBCA, RXCF, RXPF, RXUO, RALN, RFLR, ROVR, RJBR, TMCA, TBCA, TXPF, TXCF.
16–18	PRI	This is the default value used for the IEEE Std. 802.1p frame priority.
19	CFI	This is the default value used for the IEEE Std. 802.1Q canonical format indicator.
20–31	VID	This is the default value used for the virtual-LAN identifier in VLAN-tagged frames. A value of zero is defined as the null VLAN, however field PRI may be still set independently.

18.5.3.2.4 Transmit Interrupt Coalescing Register (TXIC)

The TXIC register enables and configures the operational parameters for interrupt coalescing associated with transmitted frames. Figure 18-14 describes the definition for the TXIC register.

Offset eTSEC1:0x2_4110; eTSEC2:0x2_5110 Access: Read/Write

	0	1	2	3–10	11–15	16–31
R/W	ICEN	ICCS	—	ICFT	—	ICTT
Reset	All zeros					

Figure 18-14. TXIC Register Definition

Table 18-19 describes the fields of the TXIC register.

Table 18-19. TXIC Field Descriptions

Bits	Name	Description
0	ICEN	Interrupt coalescing enable 0 Interrupt coalescing is disabled. Interrupts are raised as they are received. 1 Interrupt coalescing is enabled. If the eTSEC transmit frame interrupt is enabled (IMASK[TXFEN] is set), an interrupt is raised when the threshold number of frames is reached (defined by TXIC[ICFT]) or when the threshold timer expires (determined by TXIC[ICTT]).
1	ICCS	Interrupt coalescing timer clock source. 0 The coalescing timer advances count every 64 eTSEC Tx interface clocks (TSEC*n*_GTX_CLK). 1 The coalescing timer advances count every 64 system clocks. This mode is recommended for FIFO operation.
2	—	Reserved
3–10	ICFT	Interrupt coalescing frame count threshold. While interrupt coalescing is enabled (TXIC[ICEN] is set), this value determines how many frames are transmitted before raising an interrupt. The eTSEC threshold counter is reset to ICFT following an interrupt. The value of ICFT must be greater than zero to avoid unpredictable behavior.
11–15	—	Reserved
16–31	ICTT	Interrupt coalescing timer threshold. While interrupt coalescing is enabled (TXIC[ICEN] is set), this value determines the maximum amount of time after transmitting a frame before raising an interrupt. If frames have been transmitted but the frame count threshold has not been met, an interrupt is raised when the threshold timer reaches zero. The threshold timer is reset to the value in this field and begins counting down upon transmission of the first frame having its TxBD[I] bit set. The threshold value is represented in units of 64 clock periods as specified by the timer clock source (TXIC[ICCS[). The value of ICTT must be greater than zero to avoid unpredictable behavior.

18.5.3.2.5 Transmit Queue Control Register (TQUEUE)

The TQUEUE register, shown in Figure 18-15, selectively enables each of the TxBD rings 0–7. By default, TxBD ring 0 is enabled.

Offset eTSEC1:0x2_4114; eTSEC2:0x2_5114 — Access: Read/Write

Bits	0–15	16	17	18	19	20	21	22	23	24–31
R/W	—	EN0	EN1	EN2	EN3	EN4	EN5	EN6	EN7	—
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0

Figure 18-15. TQUEUE Register Definition

Table 18-20 describes the TQUEUE register.

Table 18-20. TQUEUE Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	EN0	Transmit queue 0 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
17	EN1	Transmit queue 1 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
18	EN2	Transmit queue 2 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
19	EN3	Transmit queue 3 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
20	EN4	Transmit queue 4 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
21	EN5	Transmit queue 5 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
22	EN6	Transmit queue 6 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
23	EN7	Transmit queue 7 enable. 0 TxBD ring is not queried for transmission. In effect the transmit queue is disabled. 1 TxBD ring is queried for transmission.
24–31	—	Reserved

18.5.3.2.6 TxBD Ring 0–3 Weighting Register (TR03WT)

When modified weighted round-robin Tx scheduling is enabled (TCTRL[TXSCHED] = 10), this register determines the weighting applied to each transmit queue for queues 0 to 3. For priority-based scheduling, TR03WT has no effect. A description of how queue weights affect eTSEC's round-robin algorithm appears in Section 18.6.4.3.2, "Modified Weighted Round-Robin Queuing (MWRR)." Figure 18-16 describes the TR03WT register.

Offset eTSEC1:0x2_4140; eTSEC2:0x2_5140 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	WT0	WT1	WT2	WT3
Reset	All zeros			

Figure 18-16. TR03WT Register Definition

Table 18-21 describes the fields of the TR03WT register.

Table 18-21. TR03WT Field Descriptions

Bits	Name	Description
0–7	WT0	Weighting value for TxBD ring 0 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT0 × 64 bytes of data are scheduled for transmission from TxBD ring 0. Clearing this field prevents transmission.
8–15	WT1	Weighting value for TxBD ring 1 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT1 × 64 bytes of data are scheduled for transmission from TxBD ring 1. Clearing this field prevents transmission.
16–23	WT2	Weighting value for TxBD ring 2 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT2 × 64 bytes of data are scheduled for transmission from TxBD ring 2. Clearing this field prevents transmission.
24–31	WT3	Weighting value for TxBD ring 3 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT3 × 64 bytes of data are scheduled for transmission from TxBD ring 3. Clearing this field prevents transmission.

18.5.3.2.7 TxBD Ring 4–7 Weighting Register (TR47WT)

When modified weighted round-robin Tx scheduling is enabled (TCTRL[TXSCHED] = 10), this register determines the weighting applied to each enabled transmit queue for queues 4 to 7. For priority-based scheduling, TR47WT has no effect. A description of how queue weights affect eTSEC's modified weighted round-robin algorithm appears in Section 18.6.4.3.2, "Modified Weighted Round-Robin Queuing (MWRR)." Figure 18-17 describes the definition for the TR47WT register.

Offset eTSEC1:0x2_4144; eTSEC2:0x2_5144 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	WT4	WT5	WT6	WT7
Reset	All zeros			

Figure 18-17. TR47WT Register Definition

Table 18-22 describes the fields of the TR47WT register.

Table 18-22. TR47WT Field Descriptions

Bits	Name	Description
0–7	WT4	Weighting value for TxBD ring 4 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT4 × 64 bytes of data are scheduled for transmission from TxBD ring 4. Clearing this field prevents transmission.
8–15	WT5	Weighting value for TxBD ring 5 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT5 × 64 bytes of data are scheduled for transmission from TxBD ring 5. Clearing this field prevents transmission.
16–23	WT6	Weighting value for TxBD ring 6 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT6 × 64 bytes of data are scheduled for transmission from TxBD ring 6. Clearing this field prevents transmission.
24–31	WT7	Weighting value for TxBD ring 7 when TCTRL[TXSCHED] = 10. On each round of the Tx scheduler, a minimum of WT7 × 64 bytes of data are scheduled for transmission from TxBD ring 7. Clearing this field prevents transmission.

18.5.3.2.8 Transmit Buffer Descriptor Pointers 0–7 (TBPTR0–TBPTR7)

TBPTR0–TBPTR7 each contains the low-order 32 bits of the next transmit buffer descriptor address for their respective TxBD ring. Figure 18-18 describes the TBPTR registers. These registers takes on the value of their ring's associated TBASE when the TBASE register is written by software. Software must not write TBPTR0–TBPTR7 while eTSEC is actively transmitting frames. However, TBPTR0– TBPTR7 can be modified when the transmitter is disabled or when no Tx buffer is in use (after a GRACEFUL STOP TRANSMIT command is issued and the frame completes its transmission) in order to change the next TxBD eTSEC transmits.

Offset eTSEC1:0x2_4184+8×*n*; eTSEC2:0x2_5184+8×*n* Access: Read/Write

	0–28	29–31
R/W	TBPTR*n*	—
Reset	All zeros	

Figure 18-18. TBPTR0–TBPTR7 Register Definition

Table 18-23 describes the fields of the TBPTR*n* register.

Table 18-23. TBPTR*n* Field Descriptions

Bits	Name	Description
0–28	TBPTR*n*	Current TxBD pointer for TxBD ring *n*. Points to the current BD being processed or to the next BD the transmitter uses when it is idling. When the end of the TxBD ring is reached, eTSEC initializes TBPTR*n* to the value in the corresponding TBASE*n*. The TBPTR register is internally written by the eTSEC's DMA controller during transmission. The pointer increments by eight (bytes) each time a descriptor is closed successfully by the eTSEC. Note that the three least significant bits of this register are read-only and zero. After an error condition, the eTSEC returns TBPTR*n* to point to the first BD of the frame partially transmitted.
29–31	—	Reserved

18.5.3.2.9 Transmit Descriptor Base Address Registers (TBASE0–TBASE7)

The TBASE*n* registers are written by the user with the base address of each TxBD ring *n*. Each such value must be divisible by eight, since the three least significant bits always write as 000. Figure 18-19 describes the definition for the TBASE*n* registers.

Offset eTSEC1:0x2_4204+8×*n*; eTSEC2:0x2_5204+8×*n* Access: Read/Write

	0–28	29–31
R / W	TBASE*n*	—
Reset	All zeros	

Figure 18-19. TBASE Register Definition

Table 18-24 describes the fields of the TBASE*n* registers.

Table 18-24. TBASE0–TBASE7 Field Descriptions

Bits	Name	Description
0–28	TBASE*n*	Transmit base for ring *n*. TBASE defines the starting location in the memory map for the eTSEC TxBDs. This field must be 8-byte aligned. Together with setting the W (wrap) bit in the last BD, the user can select how many BDs to allocate for the transmit packets. The user must initialize TBASE before enabling the eTSEC transmit function on the associated ring.
29–31	—	Reserved

18.5.3.2.10 Transmit Time Stamp Identification Register (TMR_TXTS1–2_ID)

Transmit time stamp identification register (TMR_TXTS*n*_ID). This register holds the identification number of the transmitted frame corresponding to the timestamp captured in TMR_TXTS*n*_H/L. Each time the eTSEC is instructed to capture the timestamp of an outgoing frame via TxFCB[PTP] the associated field in TxFCB[PTP_ID] is stored in this register, overwriting the previous value.

This register is read only in normal operation. Figure 18-20 describes the definition for the TMR_TXTS*n*_ID register.

Offset eTSEC1:0x2_4280+4×n; eTSEC2:0x2_5280+4×n Access: Read only

	0–15	16–31
R	—	TXTS_ID
W		
Reset	All zeros	

Figure 18-20. TMR_TXTS*n*_ID Register Definition

Table 18-25 describes the fields of the TMR_TXTS*n*_ID register.

Table 18-25. TMR_TXTS*n*_ID Register Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	TXTS_ID	Tx time stamp identification field

18.5.3.2.11 Transmit Time Stamp Register (TMR_TXTS1–2_H/L)

Transmit stamp register (TMR_TXTS*n*_H/L). This register holds the value of the TMR_CNT_H/L when a frame tagged for timestamp capture (via Tx FCB[PTP]) is transmitted. Upon transmission of the start of frame symbol of such a frame, the value in TMR_CNT_H/L is copied into TMR_TXTS*n*_H/L.

This register is read only in normal operation. Figure 18-21 depicts TMR_TXTS*n*_H/L.

Offset eTSEC1:0x2_42C0+8×n; eTSEC2:0x2_52C0+8×n — Access: Read only

	0–31	32–63
R	TXTS_H	TXTS_L
W		
Reset	All zeros	

Figure 18-21. TMR_TXTS*n*_H/L Register Definition

Table 18-26 describes the fields of the TMR_TXTS*n*_H/L register.

Table 18-26. TMR_TXTS*n*_H/L Register Field Descriptions

Bits	Name	Description
0–63	TXTS_H/L	Time stamp field of the transmitted PTP packet's start of frame detection.

18.5.3.3 eTSEC Receive Control and Status Registers

This section describes the control and status registers that are used specifically for receiving Ethernet frames. All of the registers are 32 bits wide.

18.5.3.3.1 Receive Control Register (RCTRL)

The RCTRL register is programmed by the user and controls the operational mode of the receiver. It must be written only after a system reset (at initialization) or after a graceful receive stop has completed. Figure 18-22 describes the RCTRL register.

Offset eTSEC1:0x2_4300; eTSEC2:0x2_5300 — Access: Read/Write

	0–7	7	8–10	11–15
R/W	L2OFF	TS	—	PAL
Reset	All zeros			

	16	17	18	19	20	21	22	23	24–25	26	27	28	29	30	31
R/W	—	—	VLEX	FILREN	FSQEN	GHTX	IPCSEN	TUCSEN	PRSDEP	—	BC_REJ	PROM	RSF	EMEN	—
Reset	All zeros														

Figure 18-22. RCTRL Register Definition

Table 18-27 describes the fields of the RCTRL register.

Table 18-27. RCTRL Field Descriptions

Bits	Name	Description
0–6	L2OFF	Layer 2 offset. The number of octet pairs from the start of the frame that the parser should expect to see before the first byte of the Ethernet DA. For frames received over Ethernet, the start of frame is regarded as the SFD symbol. For packets received through the FIFO packet interface the start of frame is regarded as the first octet of receive data. The user may think of this value as representing the length - in multiples of two bytes - of a ‘shim’ header that is inserted between the SFD and DA. By writing to RCTRL with a mask of 0xFE00_0000, the even byte length restriction is guaranteed. For normal frames, this field should be left as 0.
7	TS	Time stamp incoming packets as padding bytes. PAL field is set to 8 if the PAL field is programmed to less than 8. Must be set to zero if TMR_CTRL[TE]=0.
8–10	—	Reserved
11–15	PAL	Packet alignment padding length. If not zero, PAL (1–31) bytes of zero padding are inserted before the start of each received frame, but following the RxFCB if TOE is enabled. For Ethernet where optional preamble extraction is enabled, the padding appears before the preamble, otherwise the padding precedes the layer 2 header. The value of PAL can be set so that the start of the IP header in the receive data buffer is aligned to a 32-bit boundary. Normally, setting PAL = 2 provides minimal padding to ensure such alignment of the IP header. Note that the minimum zero padding value for this field should be PAL–8 if the TS field is set and 0 when PAL is < 8.
16–17	—	Reserved
18	VLEX	Enable automatic VLAN tag extraction and deletion from Ethernet frames. Note that VLEX must be cleared if L2OFF is non-zero. 0 Do not delete VLAN tags from received Ethernet frames. 1 If a VLAN tag is seen after the Ethernet source address, and PRSDEP is non-zero, delete the VLAN tag and return the VLAN control word in the frame control block returned with this frame. Note that if PRSDEP is cleared, VLEX must be cleared as well. (VLAN tag extraction is only supported when the parser is enabled.)
19	FILREN	Filer enable. When set, the receive frame filer is enabled. This file accepted frames to a particular RxBD ring according to rules defined in the filer table. In this case, PRSDEP must not be cleared. 0 Do not search the receive queue filer table for received frames. All received frames are sent to RxBD ring 0 by default. 1 Search the receive queue filer table for received frames, and let the filer determine the index of the RxBD ring for each frame. Note that if PRSDEP is cleared, FILREN must be cleared as well.
20	FSQEN	Enable single-queue mode for the receive frame filer. This bit is ignored unless FILREN is also set. 0 The filer chooses the RxBD ring using the least significant bits of the virtual queue ID as a ring index. 1 The filer always attempts to file received frames to ring 0, regardless of virtual queue ID. This mode is intended for operating the filer as a packet classification engine.

Table 18-27. RCTRL Field Descriptions (continued)

Bits	Name	Description
21	GHTX	Group address hash table extend. By default, the group address hash table is 256 entries (as defined by registers GADDR0–GADDR7); registers IGADDR0–IGADDR7 are then used to define the individual address hash table. When this bit is set, the hash table is extended to a total of 512 entries (IGADDR0–IGADDR7 are then the first 256 entries of the extended 512-entry group address hash table). 0 Both the individual and group hash functions are the 8 MSBs of the CRC-32 of the Ethernet destination address. 1 The group hash function is the 9 MSBs of the CRC-32 of the Ethernet destination address. The individual address hash function is unavailable.
22	IPCSEN	IP Checksum verification enable. See Section 18.6.3.3, “Receive Path Off-Load.” 0 IPv4 header checksums are not verified by the eTSEC—even if layer 3 parsing is enabled. 1 Perform IPv4 header checksum verification if PRSDEP > 01.
23	TUCSEN	TCP or UDP Checksum verification enable. See Section 18.6.3.3, “Receive Path Off-Load.” 0 TCP or UDP checksums are not verified by the eTSEC—even if layer 4 parsing is enabled. 1 Perform TCP or UDP checksum verification if PRSDEP = 11.
24–25	PRSDEP	Parser control. The level of parser layer recognition is determined as follows: 00 Parser disabled. Receive frame filer must also be disabled by clearing RCTRL[FILREN]. 01 Only L2 (Ethernet) protocols are recognized. 10 L2 and L3 (IP) protocols are recognized. 11 L2, L3, and L4 (TCP/UDP) protocols are recognized. If this field is non-zero, a TOE frame control block is prepended to the received frame, and the first RxBD points to the FCB. Note that if PRSDEP is cleared, VLEX must be cleared as well. (VLAN tag extraction is only supported when the parser is enabled.) Also, if PRSDEP is cleared, FILREN must also be cleared.
26	—	Reserved
27	BC_REJ	Broadcast frame reject. If this bit is set, frames with DA (destination address) = FFFF_FFFF_FFFF are rejected unless RCTRL[PROM] is set. If both BC_REJ and RCTRL[PROM] are set, then frames with broadcast DA are accepted and the M (MISS) bit is set in the receive BD.
28	PROM	Promiscuous mode. All Ethernet frames, regardless of destination address, are accepted.
29	RSF	Receive short frame mode. When set, enables the reception of frames shorter than 64 bytes. 0 Ethernet frames less than 64B in length are silently dropped. 1) Frames more than 16B and less than 64B in length are accepted upon a DA match. Note that frames less than or equal to 16B in length are always silently dropped.
30	EMEN	Exact match MAC address enable. If this bit is set, the MAC01ADDR1–MAC15ADDR1 and MAC01ADDR2–MAC15ADDR2 registers are recognized as containing MAC addresses aliasing the MAC’s station address. Setting this bit therefore allows eTSEC to receive Ethernet frames having a destination address matching one of these 15 addresses.
31	—	Reserved

18.5.3.3.2 Receive Status Register (RSTAT)

The eTSEC writes to this register under the following conditions:

- A frame interrupt event occurred on one or more RxBD rings
- The receiver runs out of descriptors due to a busy condition on a RxBD ring
- The receiver was halted because an error condition was encountered while receiving a frame

Writing 1 to any bit of this register clears it. Software should clear the QHLT bit to take eTSEC's receiver function out of halt state for the associated queue. Figure 18-23 describes the definition for the RSTAT register.

Offset eTSEC1:0x2_4304; eTSEC2:0x2_5304 Access: w1c

	0–7	8	9	10	11	12	13	14	15
R	—	QHLT0	QHLT1	QHLT2	QHLT3	QHLT4	QHLT5	QHLT6	QHLT7
W		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros								

	16–23	24	25	26	27	28	29	30	31
R	—	RXF0	RXF1	RXF2	RXF3	RXF4	RXF5	RXF6	RXF7
W		w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros								

Figure 18-23. RSTAT Register Definition

Table 18-28 describes the fields of the RSTAT register.

Table 18-28. RSTAT Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8	QHLT0	RxBD queue 0 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT0 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.
9	QHLT1	RxBD queue 1 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT1 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.
10	QHLT2	RxBD queue 2 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT2 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.
11	QHLT3	RxBD queue 3 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT3 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.
12	QHLT4	RxBD queue 4 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT4 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.

Table 18-28. RSTAT Field Descriptions (continued)

Bits	Name	Description
13	QHLT5	RxBD queue 5 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT5 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.
14	QHLT6	RxBD queue 6 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT6 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.
15	QHLT7	RxBD queue 7 is halted. It is a hardware-initiated stop indication. (DMACTRL[GRS] being set by the user does not cause a QHLT7 to be set.). The current frame and all other frames directed to a halted queue are discarded. A write with a value of 1 re-enables the queue for receiving. 0 This queue is enabled for reception. (That is, it is not halted) 1 All controller receive activity to this queue is halted.
16–23	—	Reserved
24	RXF0	Receive frame event occurred on ring 0. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring.
25	RXF1	Receive frame event occurred on ring 1. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring.
26	RXF2	Receive frame event occurred on ring 2. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring.
27	RXF3	Receive frame event occurred on ring 3. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring.
28	RXF4	Receive frame event occurred on ring 4. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring.
29	RXF5	Receive frame event occurred on ring 5. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring.
30	RXF6	Receive frame event occurred on ring 6. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring.
31	RXF7	Receive frame event occurred on ring 7. Set by the eTSEC if IEVENT[RXF] was set in relation to receiving a frame to this ring.

18.5.3.3.3 Receive Interrupt Coalescing Register (RXIC)

The RXIC register enables and configures the operational parameters for interrupt coalescing associated with received frames. Figure 18-24 describes the RXIC register.

Offset eTSEC1:0x2_4310; eTSEC2:0x2_5310 Access: Read/Write

	0	1	2	3–10	11–15	16–31
R / W	ICEN	ICCS	—	ICFT	—	ICTT
Reset	All zeros					

Figure 18-24. RXIC Register Definition

Table 18-29 describes the fields of the RXIC register.

Table 18-29. RXIC Field Descriptions

Bits	Name	Description
0	ICEN	Interrupt coalescing enable 0 Interrupt coalescing is disabled. Interrupts are raised as they are received. 1 Interrupt coalescing is enabled. If the eTSEC receive frame interrupt is enabled (IMASK[RXFEN] is set), an interrupt is raised when the threshold number of frames is reached (defined by RXIC[ICFT]) or when the threshold timer expires (determined by RXIC[ICTT]).
1	ICCS	Interrupt coalescing timer clock source. 0 The coalescing timer advances count every 64 eTSEC Rx interface clocks (TSECn_GTX_CLK). 1 The coalescing timer advances count every 64 system clocks. This mode is recommended for FIFO operation.
2	—	Reserved
3–10	ICFT	Interrupt coalescing frame count threshold. While interrupt coalescing is enabled (RXIC[ICE] is set), this value determines how many frames are received before raising an interrupt. The eTSEC threshold counter is reset to ICFT following an interrupt. The value of ICFT must be greater than zero avoid unpredictable behavior.
11–15	—	Reserved
16–31	ICTT	Interrupt coalescing timer threshold. While interrupt coalescing is enabled (RXIC[ICE] is set), this value determines the maximum amount of time after receiving a frame before raising an interrupt. If frames have been received but the frame count threshold has not been met, an interrupt is raised when the threshold timer reaches zero. The threshold timer is reset to the value in this field and begins counting down upon receiving the first frame having its RxBD[I] bit set. The threshold value is represented in units equal to 64 periods of the clock specified by RXIC[ICCS]. ICTT must be greater than zero to avoid unpredictable behavior.

18.5.3.3.4 Receive Queue Control Register (RQUEUE)

The RQUEUE register enables each of the RxBD rings 0–7. By default, RxBD ring 0 is enabled. Figure 18-25 describes the definition for the RQUEUE register.

Offset eTSEC1:0x2_4314; eTSEC2:0x2_5314 Access: Read/Write

	0–23	24	25	26	27	28	29	30	31
R/W	—	EN0	EN1	EN2	EN3	EN4	EN5	EN6	EN7
Reset	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	0	0	0	0	0	0

Figure 18-25. RQUEUE Register Definition

Table 18-30 describes the RQUEUE register.

Table 18-30. RQUEUE Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24	EN0	Receive queue 0 enable. 0 RxBD ring is not queried for reception. In effect the receive queue is disabled. 1 RxBD ring is queried for reception.

Table 18-30. RQUEUE Field Descriptions (continued)

Bits	Name	Description
25	EN1	Receive queue 1 enable. 0 RxBD ring is not queried for reception. In effect the receive queue is disabled. 1 RxBD ring is queried for reception.
26	EN2	Receive queue 2 enable. 0 RxBD ring is not queried for reception. In effect the receive queue is disabled. 1 RxBD ring is queried for reception.
27	EN3	Receive queue 3 enable. 0 RxBD ring is not queried for reception. In effect the receive queue is disabled. 1 RxBD ring is queried for reception.
28	EN4	Receive queue 4 enable. 0 RxBD ring is not queried for reception. In effect the receive queue is disabled. 1 RxBD ring is queried for reception.
29	EN5	Receive queue 5 enable. 0 RxBD ring is not queried for reception. In effect the receive queue is disabled. 1 RxBD ring is queried for reception.
30	EN6	Receive queue 6 enable. 0 RxBD ring is not queried for reception. In effect the receive queue is disabled. 1 RxBD ring is queried for reception.
31	EN7	Receive queue 7 enable. 0 RxBD ring is not queried for reception. In effect the receive queue is disabled. 1 RxBD ring is queried for reception.

18.5.3.3.5 Receive Bit Field Extract Control Register (RBIFX)

The RBIFX register provides a set of four 6-bit offsets for locating up to four octets in a received frame and passing them to the receive queue filer as the user-defined ARB property. Through RBIFX a custom ARB filer property can be constructed from arbitrary bytes, which allows frame filing on the basis of bitfields not ordinarily provided to the filer, such as bits from the Ethernet preamble or TCP flags. The value of property ARB is the concatenation of {B0, B1, B2, B3} to 32-bits, where B0–B3 are the bytes as defined by RBIFX.

Figure 18-26 describes the definition for the RBIFX register.

Offset eTSEC1:0x2_4330; eTSEC2:0x2_5330 Access: Read/Write

	0–1	2–7	8–9	10–15	16–17	18–23	24–25	26–31
R W	B0CTL	B0OFFSET	B1CTL	B1OFFSET	B2CTL	B2OFFSET	B3CTL	B3OFFSET
Reset	All zeros							

Figure 18-26. RBIFX Register Definition

Table 18-31 describes the RBIFX register.

Table 18-31. RBIFX Field Descriptions

Bits	Name	Description
0–1	B0CTL	Location of byte 0 of property ARB. 00 Byte 0 is not extracted, and appears as zero in property ARB. 01 Byte 0 is located in the received frame at offset (B0OFFSET – 8) bytes from the first byte of the Ethernet DA. In non-FIFO modes, a negative effective offset points to bytes of the standard Ethernet preamble. Values of B0OFFSET less than 8 are reserved in FIFO modes. 10 Byte 0 is located in the received frame at offset B0OFFSET bytes from the byte after the last byte of the layer 2 header. 11 Byte 0 is located in the received frame at offset B0OFFSET bytes from the byte after the last byte of the layer 3 header.
2–7	B0OFFSET	Offset relative to the header defined by B0CTL that locates byte 0 of property ARB. An effective offset of zero points to the first byte of the specified header.
8–9	B1CTL	Location of byte 1 of property ARB. 00 Byte 1 is not extracted, and appears as zero in property ARB. 01 Byte 1 is located in the received frame at offset (B1OFFSET – 8) bytes from the first byte of the Ethernet DA. In non-FIFO modes, a negative effective offset points to bytes of the standard Ethernet preamble. Values of B1OFFSET less than 8 are reserved in FIFO modes. 10 Byte 0 is located in the received frame at offset B1OFFSET bytes from the byte after the last byte of the layer 2 header. 11 Byte 0 is located in the received frame at offset B1OFFSET bytes from the byte after the last byte of the layer 3 header.
10–15	B1OFFSET	Offset relative to the header defined by B1CTL that locates byte 1 of property ARB. An effective offset of zero points to the first byte of the specified header.
16–17	B2CTL	Location of byte 2 of property ARB. 00 Byte 2 is not extracted, and appears as zero in property ARB. 01 Byte 2 is located in the received frame at offset (B2OFFSET – 8) bytes from the first byte of the Ethernet DA. In non-FIFO modes, a negative effective offset points to bytes of the standard Ethernet preamble. Values of B2OFFSET less than 8 are reserved in FIFO modes. 10 Byte 0 is located in the received frame at offset B2OFFSET bytes from the byte after the last byte of the layer 2 header. 11 Byte 0 is located in the received frame at offset B2OFFSET bytes from the byte after the last byte of the layer 3 header.
18–23	B2OFFSET	Offset relative to the header defined by B2CTL that locates byte 2 of property ARB. An effective offset of zero points to the first byte of the specified header.
24–25	B3CTL	Location of byte 3 of property ARB. 00 Byte 3 is not extracted, and appears as zero in property ARB. 01 Byte 3 is located in the received frame at offset (B3OFFSET – 8) bytes from the first byte of the Ethernet DA. In non-FIFO modes, a negative effective offset points to bytes of the standard Ethernet preamble. Values of B3OFFSET less than 8 are reserved in FIFO modes. 10 Byte 0 is located in the received frame at offset B3OFFSET bytes from the byte after the last byte of the layer 2 header. 11 Byte 0 is located in the received frame at offset B3OFFSET bytes from the byte after the last byte of the layer 3 header.
26–31	B3OFFSET	Offset relative to the header defined by B3CTL that locates byte 3 of property ARB. An effective offset of zero points to the first byte of the specified header.

18.5.3.3.6 Receive Queue Filer Table Address Register (RQFAR)

RQFAR, shown in Figure 18-27, contains the index of the current, indirectly accessible entry of the received queue filer table. Each table entry occupies a pair of 32-bit words, denoted RQCTRL and RQPROP. To access the RQCTRL and RQPROP words of entry *n*, write *n* to RQFAR. Then read or write the indexed RQCTRL and RQPROP words by reading or writing the RQFCR and RQFPR registers, respectively.

Offset eTSEC1:0x2_4334; eTSEC2:0x2_5334 Access: Read/Write

	0–23	24–31
R/W	—	RQFAR
Reset	All zeros	

Figure 18-27. Receive Queue Filer Table Address Register Definition

Table 18-32 describes the fields of the RQFAR register.

Table 18-32. RQFAR Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24–31	RQFAR	Current index of receive queue filer table, which spans a total of 256 entries.

18.5.3.3.7 Receive Queue Filer Table Control Register (RQFCR)

RQFCR is accessed to read or write the RQCTRL words in entries of the receive queue filer table. The table entries are described in greater detail in Section 18.6.4.2, "Receive Queue Filer." The word accessed through RQFCR is defined by the current value of RQFAR.

Figure 18-28 describes the definition for the RQFCR register.

Offset eTSEC1:0x2_4338; eTSEC2:0x2_5338 Access: Read/Write

	0	1–15	16–21	22	23	24	25–26	27	28–31
R/W	GPI	—	Q	CLE	REJ	AND	CMP	—	PID
Reset	(undefined)								

Figure 18-28. Receive Queue Filer Table Control Register Definition

Table 18-33 describes the fields of the RQFCR register.

Table 18-33. RQFCR Field Descriptions

Bit	Name	Description
0	GPI	General purpose interrupt. When a property matches the value in the RQPROP entry at this index, and REJ = 0 and AND = 0, the filer will instruct the Rx descriptor controller to set IEVENT[FGPI] when the corresponding receive frame is written to memory. If the timer is enabled (TMR_CTRL[TE] = 1), then TMR_PEVENT[RXP] will also be set.
1–15	—	Reserved, should be written with zero.

Table 18-33. RQFCR Field Descriptions (continued)

Bit	Name	Description
16–21	Q	Receive queue index, from 0 to 63, inclusive, written into the Rx frame control block associated with the received frame. When a property matches the value in the RQPROP entry at this index, and REJ = 0 and AND = 0, the frame is sent to either RxBD ring 0 (if RCTRL[FSQEN] = 1) or the RxBD ring with index (Q mod 8) and the filing table search is terminated. In the case where RCTRL[FSQEN] = 0, 8 virtual receive queues are overlaid on every RxBD ring, and software needs to consult the RQ field of the Rx frame control block to determine which virtual receive queue was chosen.
22	CLE	Cluster entry/exit (used in combination with AND bit). This bit brackets clusters, marking the start and end entries of a cluster. Clusters cannot be nested. 0 Regular RQCTRL entry. 1 If entry matches and AND = 1, treat subsequent entries as belonging to a nested cluster and enter the cluster; otherwise skip all entries up to and including the next cluster exit. If AND = 0, exit current cluster.
23	REJ	Reject frame. This bit and its specified action are ignored if AND = 1. 0 If entry matches, accept frame and file it to RxBD ring Q. 1 If entry matches, reject frame and discard it, ignoring Q.
24	AND	Match this entry and the next entry as a pair. 0 Match property[PID] against RQPROP, independent of the next entry. 1 Match property[PID] against RQPROP. If matched and CLE = 0, attempt to match next entry, otherwise, skip all entries up to and including the entry with AND = 0. If matched and CLE = 1, enter cluster of entries, otherwise, skip all entries up to and including the entry with CLE = 1 (cluster exit).
25–26	CMP	Comparison operation to perform on the RQPROP entry at this index when PID > 0. The property value extracted by the frame parser is masked by the 32-bit *mask_register* prior to comparison against RQPROP. However, the property value is not permanently altered by the value in *mask_register*. By default, *mask_register* is initialized to 0xFFFF_FFFF before each frame is processed. In the case where PID = 0, CMP is interpreted as follows: 00/01 Filer *mask_register* is set to all 32 bits of RQPROP, and this entry always *matches*. 10/11 Filer *mask_register* is set to all 32 bits of RQPROP, and this entry always *fails to match*. In the case where PID > 0, CMP is interpreted as follows (& is bit-wise AND operator): 00 *property*[PID] & *mask_register* = RQPROP 01 *property*[PID] & *mask_register* >= RQPROP 10 *property*[PID] & *mask_register* != RQPROP 11 *property*[PID] & *mask_register* < RQPROP
27	—	Reserved, should be written with zero.
28–31	PID	Property identifier. The value in the RQPROP entry at this index is interpreted according to PID (see Table 18-34).

18.5.3.3.8 Receive Queue Filer Table Property Register (RQFPR)

RQFPR (see Figure 18-29) is accessed to read or write the RQPROP words in entries of the receive queue filer table. The table entries are described in greater detail in Section 18.6.4.2, “Receive Queue Filer.” The word accessed through RQFPR is defined by the current value of RQFAR. Figure 18-29 and Figure 18-30 describe the fields of the RQFPR register according to property ID.

Offset eTSEC1:0x2_433C; eTSEC2:0x2_533C; Access: Read/Write

	0–31
R/W	RQPROP
Reset	*(undefined)*

Figure 18-29. Receive Queue Filer Table Property IDs 0, 2–15 Register Definition

Offset eTSEC1:0x2_433C; eTSEC2:0x2_533C Access: Read/Write

	0–15
R/W	—
Reset	All zeros

	16	17	18	19	20	21	22	23	24	25	26	27	28–29	30	31
R/W	EBC	VLN	CFI	JUM	IPF	FIF	IP4	IP6	ICC	ICV	TCP	UDP	—	PER	EER
Reset	All zeros														

Figure 18-30. Receive Queue Filer Table Property ID1 Register Definition

Table 18-34 describes the fields of the RQFPR register.

Table 18-34. RQFPR Field Descriptions

PID[1]	Bit	Name	Description
0000	0–31	MASK	Mask bits to be written to Filer *mask_register* for masking of property values. The rule match/fail status for this PID is determined by RQCTRL[CMP]. Since *mask_register* is bit-wise ANDed with properties, every bit of MASK that is cleared also results in the corresponding property bit being cleared in comparisons. Therefore setting MASK to 0xFFFF_FFFF ensures that all property bits participate in rule matches.

Table 18-34. RQFPR Field Descriptions (continued)

PID[1]	Bit	Name	Description
0001	0–13	—	Reserved
	14	AR	Set if an ARP response packet is seen.
	15	ARQ	Set if an ARP request packet is seen.
	16	EBC	Set if the destination Ethernet address is to the broadcast address.
	17	VLN	Set if a VLAN tag (Ethertype DFVLAN[TAG] or 0x8100) was seen in the frame.
	18	CFI	Set to the value of the Canonical Format Indicator in the VLAN control tag if VLAN is set, zero otherwise.
	19	JUM	Set if a jumbo Ethernet frame was parsed.
	20	IPF	Set if a fragmented IPv4 or IPv6 header was encountered. See the descriptions of receive FCB fields IP and PRO in Section 18.6.3.3, “Receive Path Off-Load,” for more information on determining the status of received packets for which IPF is set.
	21	—	Reserved
	22	IP4	Set if an IPv4 header was parsed.
	23	IP6	Set if an IPv6 header was parsed.
	24	ICC	Set if the IPv4 header checksum was checked.
	25	ICV	Set if the IPv4 header checksum was verified correct.
	26	TCP	Set if a TCP header was parsed.
	27	UDP	Set if a UDP header was parsed.
	28–29	—	Reserved.
	30	PER	Set on a parse error, such as header inconsistency.
	31	EER	Set on an Ethernet framing error that prevents parsing.
0010	0–7	ARB	User-defined arbitrary bit field property: byte 0 extracted. Defaults to 0x00.
	8–15		User-defined arbitrary bit field property: byte 1 extracted. Defaults to 0x00.
	16–23		User-defined arbitrary bit field property: byte 2 extracted. Defaults to 0x00.
	24–31		User-defined arbitrary bit field property: byte 3 extracted. Defaults to 0x00.
0011	0–7	—	Reserved, should be written with zero.
	8–31	DAH	Destination MAC address, most significant 24 bits. Defaults to 0x000000.
0100	0–7	—	Reserved, should be written with zero.
	8–31	DAL	Destination MAC address, least significant 24 bits. Defaults to 0x000000.
0101	0–7	—	Reserved, should be written with zero.
	8–31	SAH	Source MAC address, most significant 24 bits. Defaults to 0x000000.
0110	0–7	—	Reserved, should be written with zero.
	8–31	SAL	Source MAC address, least significant 24 bits. Defaults to 0x000000.

Table 18-34. RQFPR Field Descriptions (continued)

PID[1]	Bit	Name	Description
0111	0–15	—	Reserved, should be written with zero.
	16–31	ETY	Ethertype of next layer protocol, that is, last ethertype if layer 2 headers nest. Defaults to 0xFFFF. Using the filer to match ETY does not work in the case of PPPoE packets, because the PPPoE ethertype in the original packet, 0x8864, is always overwritten with the PPP protocol field. Thus, matches on ETY == 0x8864 always fail. Instead, software should use PID=1 fields IP4 (ETY = 0x0021) and IP6 (ETY = 0x0057) to distinguish PPPoE session packets carrying IPv4 and IPv6 datagrams. Other PPP protocols are encoded in the ETY field, but many of them overlap with real ethertype definitions. Consult IANA and IEEE for possible ambiguities. A value in the length/type field greater than 1500 and less than 1536 is treated as a type encoding by the parser. Since no recognized types exist in this range, the controller will not parse beyond the length/type field of any such frame. Note that the eTSEC filer gets multiple packet attributes as a result of parsing the packet. The behavior of the eTSEC is that it pulls the innermost ethertype found in the packet; this means that in many supported protocols that have inner ethertypes, in order to file based on the outer ethertype, arbitrary extraction should be used instead of the ETY PID. There are four cases that need to be highlighted. 1. The jumbo ethertype (0x8870)—In this case, the eTSEC assumes that the following header is LLC/SNAP. LLC/SNAP has an associated Ethertype, and the ETY field is populated with that ethertype. This makes it impossible to file on jumbo frames. In this case, one can use arbitrary extracted bytes to pull the outermost Ethertype. 2. The PPPoE ethertype described above. 3. The VLAN tag ethertype (0x8100)—In this case, one can use the PID=1 VLN bit to indicate that the packet had a VLAN tag. 4. The MPLS tagged packets. In this case, one can use arbitrary extraction bytes to compare to the actual ethertype if a filer rule is intending to file based on an MPLS label existence.NOTE Users of the eTSEC parser/filer should be aware of a difference in behavior between rev 1 and rev 2 silicon in cases where the Ethernet type/length field contains a value between 1500 and 1536. In rev 2 silicon, values between 1500 and 1536 are interpreted as a type. Since there are currently no valid types in this range publicly defined by IANA, the controller will not parse beyond the length/type field of any such frame. If the same packet is encountered with rev 1 silicon, parser/filer behavior is different. With rev 1 silicon, such packets are treated as payload length. S/W must confirm the parser and filer results by checking the type/length field after the packet has been written to memory to see if it falls in this range.
1000	0–19	—	Reserved, should be written with zero.
	20–31	VID	VLAN network identifier (as per IEEE Std 802.1Q). This value defaults to 0x000 if no VLAN tag was found, or the VLAN tag contained only priority information.
1001	0–28	—	Reserved, should be written with zero.
	29–31	PRI	VLAN user priority (as per IEEE Std 802.1p). This value defaults to 000 (best effort priority) if no VLAN tag was found.
1010	0–23	—	Reserved, should be written with zero.
	24–31	TOS	IPv4 header Type Of Service field or IPv6 Traffic Class field. This value defaults to 0x00 (default RFC 2474 best-effort behavior) if no IP header appeared. Note that for IPv6 the Traffic Class field is extracted using the IP header definition in RFC 2460. IPv6 headers formed using the earlier RFC 1883 have a different format and must be handled with software.

Table 18-34. RQFPR Field Descriptions (continued)

PID[1]	Bit	Name	Description
1011	0–23	—	Reserved, should be written with zero.
	24–31	L4P	Layer 4 protocol identifier as per published IANA specification. This is the last recognized protocol type recognized in the case of IPv6 extension headers. This value defaults to 0xFF to indicate that no layer 4 header was recognized (possibly due to absence of an IP header).
1100	0–31	DIA	Destination IP address. If an IPv4 header was found, this is the entire destination address. If an IPv6 header was found, this is the 32 most significant bits of the 128-bit destination address. This value defaults to 0x0000_0000 if no IP header appeared.
1101	0–31	SIA	Source IP address. If an IPv4 header was found, this is the entire source address. If an IPv6 header was found, this is the 32 most significant bits of the 128-bit source address. This value defaults to 0x0000_0000 if no IP header appeared.
1110	0–15	—	Reserved, should be written with zero.
	16–31	DPT	Destination port number for TCP or UDP headers. This value defaults to 0x0000 if no TCP or UDP headers were recognized.
1111	0–15	—	Reserved, should be written with zero.
	16–31	SPT	Source port number for TCP or UDP headers. This value defaults to 0x0000 if no TCP or UDP headers were recognized.

[1] PID is the property identifier field of the filer table control entry (see RQFCR[PID]) at the same index.

18.5.3.3.9 Maximum Receive Buffer Length Register (MRBLR)

The MRBLR register is written by the user. It informs the eTSEC how much space is in the receive buffer pointed to by the RxBD. Figure 18-31 describes the definition for the MRBLR.

Offset eTSEC1:0x2_4340; eTSEC2:0x2_5340 Access: Read/Write

	0–15	16–25	26–31
R W	—	MRBL	—
Reset	All zeros		

Figure 18-31. MRBLR Register Definition

Table 18-35. MRBLR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–25	MRBL	Maximum receive buffer length. MRBL is the number of bytes that the eTSEC receiver writes to the receive buffer. The MRBL register is written by the user with a multiple of 64 for all modes. The eTSEC can write fewer bytes to the buffer than the value set in MRBL if a condition such as an error or end-of-frame occurs, but it never exceeds the MRBL value; therefore, user-supplied buffers must be at least as large as the MRBL. MRBL must be set, together with the number of buffer descriptors, to ensure adequate space for received frames. See Section 18.5.3.5.5, “Maximum Frame Length Register (MAXFRM),” for further discussion.
26–31	—	To ensure that MRBL is a multiple of 64, these bits are reserved and should be cleared.

18.5.3.3.10 Receive Buffer Descriptor Pointers 0–7 (RBPTR0–RBPTR7)

RBPTR0–RBPTR7 each contains the low-order 32 bits of the next receive buffer descriptor address for their respective RxBD ring. Figure 18-32 describes the RBPTR registers. These registers takes on the value of their ring's associated RBASE when the RBASE register is written by software. Software must not write RBPTR*n* while eTSEC is actively receiving frames. However, RBPTR*n* can be modified when the receiver is disabled or when no Rx buffer is in use (after a GRACEFUL STOP RECEIVE command is issued and the frame completes its reception) in order to change the next RxBD eTSEC receives.

Offset eTSEC1:0x2_4384+8×*n*; eTSEC2:0x2_5384+8×*n* Access: Read/Write

	0–28	29–31
R / W	RBPTR*n*	—
Reset	All zeros	

Figure 18-32. RBPTR0–RBPTR7 Register Definition

Table 18-23 describes the fields of the RBPTR*n* register.

Table 18-36. RBPTR*n* Field Descriptions

Bits	Name	Description
0–28	RBPTR*n*	Current RxBD pointer for RxBD ring *n*. Points to the current BD being processed or to the next BD the receiver uses when it is idling. After reset or when the end of the RxBD ring is reached, eTSEC initializes RBPTR*n* to the value in the corresponding RBASE*n*. The RBPTR register is internally written by the eTSEC's DMA controller during reception. The pointer increments by 8 (bytes) each time a descriptor is closed successfully by the eTSEC. Note that the 3 least-significant bits of this register are read only and zero.
29–31	—	Reserved

18.5.3.3.11 Receive Descriptor Base Address Registers (RBASE0–RBASE7)

The RBASE*n* registers are written by the user with the base address of each RxBD ring *n*. Each such value must be divisible by eight, since the 3 least-significant bits always write as 000. Figure 18-33 describes the RBASE*n* registers.

Offset eTSEC1:0x2_4404+8×*n*; eTSEC2:0x2_5404+8×*n* Access: Read/Write

	0–28	29–31
R / W	RBASE*n*	—
Reset	All zeros	

Figure 18-33. RBASE Register Definition

Table 18-24 describes the fields of the RBASE*n* registers.

Table 18-37. RBASE0–RBASE7 Field Descriptions

Bits	Name	Description
0–28	RBASE*n*	Receive base for ring *n*. RBASE defines the starting location in the memory map for the eTSEC RxBDs. This field must be 8-byte aligned. Together with setting the W (wrap) bit in the last BD, the user can select how many BDs to allocate for the receive packets. The user must initialize RBASE before enabling the eTSEC receive function on the associated ring.
29–31	—	Reserved

18.5.3.3.12 Receive Stamp Register (TMR_RXTS_H/L)

Receive time stamp register (RXTS_H/L). This register holds the value present in TMR_CNT_H/L when the eTSEC detects a new incoming Ethernet frame. This register is only updated when the precision time stamp logic is enable via TMR_CTRL[TE]. This register is read only in normal operation. Figure 18-34 describes the definition for the RXTS_H/L register.

Offset eTSEC1:0x2_44C4; eTSEC2:0x2_54C4 Access: Read/Write

	0–31	32–63
R W	TMR_RXTS_H	TMR_RXTS_L
Reset	All zeros	

Figure 18-34. TMR_RXTS_H/L Register Definition

Table 18-38 describes the fields of the TMR_RXTS_H/L register.

Table 18-38. TMR_RXTS_H/L Register Field Descriptions

Bits	Name	Description
0–63	TMR_RXTS_H/L	Value of the eTSEC precision timer upon detection of a start of frame symbol for the received frame.

18.5.3.4 MAC Functionality

This section describes the MAC registers and provides a brief overview of the functionality that can be exercised through the use of these registers, particularly those that provide functionality not explicitly required by the IEEE 802.3 standard. All of the MAC registers are 32 bits wide.

18.5.3.4.1 Configuring the MAC

The MAC configuration registers 1 and 2 provide for configuring the MAC in multiple ways:

- Adjusting the preamble length—The length of the preamble can be adjusted from the nominal seven bytes to some other (non-zero) value. Should custom preamble insertion/extraction be configured, then this register must by left at its default value.
- Varying pad/CRC combinations—Three different pad/CRC combinations are provided to handle a variety of system requirements. Simplest are frames that already have a valid frame check sequence (FCS) field. The other two options include appending a valid CRC or padding and then appending a valid CRC, resulting in a minimum frame of 64 octets. In addition to the

programmable register set, the pad/CRC behavior can be dynamically adjusted on a per-packet basis.

18.5.3.4.2 Controlling CSMA/CD

The half-duplex register (HAFDUP) allows control over the carrier-sense multiple access/collision detection (CSMA/CD) logic of the eTSEC. Half-duplex mode is only supported for 10- and 100-Mbps operation. Following the completion of the packet transmission the part begins timing the inter packet gap (IPG) as programmed in the back-to-back IPG configuration register. The system is now free to begin another frame transfer.

In full-duplex mode both the carrier sense (CRS) and collision (COL) indications from the PHY are ignored, but in half-duplex mode the eTSEC defers to CRS, and following a carrier event, times the IPG using the non-back-to-back IPG configuration values that include support for the optional two-thirds/one-third CRS deferral process. This optional IPG mechanism enhances system robustness and ensures fair access to the medium. During the first two-thirds of the IPG, the IPG timer is cleared if CRS is sensed. During the final one-third of the IPG, CRS is ignored and the transmission begins once IPG is timed. The two-thirds/one-third ratio is the recommended value.

18.5.3.4.3 Handling Packet Collisions

While transmitting a packet in half-duplex mode, the eTSEC is sensitive to COL. If a collision occurs, it aborts the packet and outputs the 32-bit jam sequence. The jam sequence is comprised of several bits of the CRC, inverted to guarantee an invalid CRC upon reception. A signal is sent to the system indicating that a collision occurred and that the start of the frame is needed for retransmission. The eTSEC then backs off of the medium for a time determined by the truncated binary exponential back off (BEB) algorithm. Following this back-off time, the packet is retried. The back-off time can be skipped if configured through the half-duplex register. However, this is non-standard behavior and its use must be carefully applied. Should any one packet experience excessive collisions, the packet is aborted. The system should flush the frame and move to the next one in line. If the system requests to send a packet while the eTSEC is deferring to a carrier, the eTSEC simply waits until the end of the carrier event and the timing of IPG before it honors the request.

If packet transmission attempts experience collisions, the eTSEC outputs the jam sequence and waits some amount of time before retrying the packet. This amount of time is determined by a controlled randomization process called truncated binary exponential back-off. The amount of time is an integer number of slot times. The number of slot times to delay before the *n*th retransmission attempt is chosen as a uniformly-distributed random integer *r* in the range:

$$0 \le r \le 2^{k}, \text{ where } k = \min(n,10).$$

So after the first collision, the eTSEC backs off either 0 or 1 slot times. After the fifth collision, the eTSEC backs off between 0 and 32 slot times. After the tenth collision, the maximum number of slot times to back off is 1024. This can be adjusted through the half-duplex register. An alternate truncation point, such as 7 for instance, can be programmed. On average, the MAC is more aggressive after seven collisions than other stations on the network.

18.5.3.4.4 Controlling Packet Flow

Packet flow can be dealt with in a number of ways within eTSEC. A default retransmit attempt limit of 15 can be reduced using the half-duplex register. The slot time or collision window can be used to gate the retry window and possibly reduce the amount of transmit buffering within the system. The slot time for 10/100 Mbps is 512 bit times. Because the slot time begins at the beginning of the packet (including preamble), the end occurs around the 56th byte of the frame data. Slot time in 1000-Mbps mode is not supported.

Full-duplex flow control is provided for in IEEE 802.3x. Currently the standard does not address flow control in half-duplex environments. Common in the industry, however, is the concept of back pressure. The eTSEC implements the optional back pressure mechanism using the raise carrier method. If the system receive logic wishes to stop the reception of packets in a network-friendly way, transmit half-duplex flow control (THDF) is set (TCTRL[THDF]). If the medium is idle, the eTSEC raises carrier by transmitting preamble. Other stations on the half-duplex network then defer to the carrier.

In the event the preamble transmission happens to cause a collision, the eTSEC ensures the minimum 96-bit presence on the wire, then drops preamble and waits a back-off time depending on the value of the back-pressure-no-back-off configuration bit HAFDUP[BP No BackOff]. These transmitting-preamble-for-back pressure collisions are not counted. If HAFDUP[BP No BackOff] is set, the eTSEC waits an inter-packet gap before resuming the transmission of preamble following the collision and does not defer. If HAFDUP[BP No BackOff] is cleared, the eTSEC adheres to the truncated BEB algorithm that allows the possibility of packets being received. This also can be detrimental in that packets can now experience excessive collisions, causing them to be dropped in the stations from which they originate. To reduce the likelihood of lost packets and packets leaking through the back pressure mechanism, HAFDUP[BP No BackOff] must be set.

The eTSEC drops carrier (cease transmitting preamble) periodically to avoid excessive defer conditions in other stations on the shared network. If, while applying back pressure, the eTSEC is requested to send a packet, it stops sending preamble, and waits one IPG before sending the packet. HAFDUP[BP No BackOff] applies for any collision that occurs during the sending of this packet. Collisions for packets while half duplex back pressure is asserted are counted. The eTSEC does not defer while attempting to send packets while in back pressure. Again, back pressure is non-standard, yet it can be effective in reducing the flow of receive packets.

18.5.3.4.5 Controlling PHY Links

Control and status to and from the PHY is provided through the two-wire MII management interface described in IEEE 802.3u. The MII management registers (MII management configuration, command, address, control, status, and indicator registers) are used to exercise this interface between a host processor and one or more PHY devices.

The eTSEC MII's registers provide the ability to perform continuous read cycles (called a scan cycle); although, scan cycles are not explicitly defined in the standard. If requested (by setting MIIMCOM[Scan Cycle]), the part performs repetitive read cycles of the PHY status register, for example. In this way, link characteristics may be monitored more efficiently. The different fields in the MII management indicator register (scan, not valid and busy) are used to indicate availability of each read of the scan cycle to the host from MIIMSTAT[PHY scan].

Yet another parameter that can be modified through the MII registers is the length of the MII management interface preamble. After establishing that a PHY supports preamble suppression, the host may so configure the eTSEC. While enabled, the length of MII management frames are reduced from 64 clocks to 32 clocks. This effectively doubles the efficiency of the interface.

18.5.3.5 MAC Registers

This section describes the MAC registers.

18.5.3.5.1 MAC Configuration 1 Register (MACCFG1)

MACCFG1 is written by the user. Figure 18-35 describes the definition for the MACCFG1 register.

Offset eTSEC1:0x2_4500; eTSEC2:0x2_5500 Access: Mixed

	0	1–11	12	13	14	15
R/W	Soft_Reset	—	Reset Rx MC	Reset Tx MC	Reset Rx Fun	Reset Tx Fun
Reset	All zeros					

	16–22	23	24–25	26	27	28	29	30	31
R	—	Loop Back	—	Rx_Flow	Tx_Flow	Sync'd Rx EN	Rx_EN	Sync'd Tx EN	Tx_EN
W									
Reset	All zeros								

Figure 18-35. MACCFG1 Register Definition

Table 18-39 describes the fields of the MACCFG1 register.

Table 18-39. MACCFG1 Field Descriptions

Bits	Name	Description
0	Soft_Reset	Soft reset. This bit is cleared by default. See Section 18.6.2.2, "Soft Reset and Reconfiguring Procedure," for more information on setting this bit. 0 Normal operation. 1 Place the entire MAC in reset except for the host interface.
1–11	—	Reserved
12	Reset Rx MC	Reset receive MAC control block. This bit is cleared by default. 0 Normal operation. 1 Place the receive part of the MAC in reset. This block detects control frames and contains the pause timers.
13	Reset Tx MC	Reset transmit MAC control block. This bit is cleared by default. 0 Normal operation. 1 Place the transmit part of the MAC in reset. This block multiplexes data and control frame transfers. It also responds to XOFF PAUSE control frames.
14	Reset Rx Fun	Reset receive function block. This bit is cleared by default. 0 Normal operation. 1 Place the receive function in reset. This block performs the receive frame protocol.

Table 18-39. MACCFG1 Field Descriptions (continued)

Bits	Name	Description
15	Reset Tx Fun	Reset transmit function block. This bit is cleared by default. 0 Normal operation. 1 Place the transmit function in reset. This block performs the frame transmission protocol.
16–22	—	Reserved
23	Loop Back	Loop back. This bit is cleared by default. 0 Normal operation. 1 Loop back the MAC transmit outputs to the MAC receive inputs.
24–25	—	Reserved
26	Rx_Flow	Receive flow. This bit is cleared by default. Must be 0 if MACCFG2[Full Duplex] = 0. 0 The receive MAC control ignores PAUSE flow control frames. 1 The receive MAC control detects and acts on PAUSE flow control frames.
27	Tx_Flow	Transmit flow. This bit is cleared by default. Must be 0 if MACCFG2[Full Duplex] = 0. 0 The transmit MAC control may not send PAUSE flow control frames if requested by the system. 1 The transmit MAC control may send PAUSE flow control frames if requested by the system.
28	Sync'd Rx EN	Receive enable synchronized to the receive stream. (Read-only) 0 Frame reception is not enabled. 1 Frame reception is enabled.
29	Rx_EN	Receive enable. This bit is cleared by default. If set, prior to clearing this bit, set DMACTRL[GRS] then confirm subsequent occurrence of the graceful receive stop interrupt (IEVENT[GRSC] is set). 0 The MAC may not receive frames from the PHY. 1 The MAC may receive frames from the PHY.
30	Sync'd Tx EN	Transmit enable synchronized to the transmit stream. (Read-only) 0 Frame transmission is not enabled. 1 Frame transmission is enabled.
31	Tx_EN	Transmit enable. This bit is cleared by default. If set, prior to clearing this bit, set DMACTRL[GTS] then confirm subsequent occurrence of the graceful receive stop interrupt (IEVENT[GTSC] is set). 0 The MAC may not transmit frames from the system. 1 The MAC may transmit frames from the system.

18.5.3.5.2 MAC Configuration 2 Register (MACCFG2)

The MACCFG2 register is written by the user. Figure 18-36 describes the definition for the MACCFG2 register.

Offset eTSEC1:0x2_4504; eTSEC2:0x2_5504 Access: Read/Write

	0–15
R	—
W	
Reset	All zeros

	16–19	20–21	22–23	24	25	26	27	28	29	30	31
R/W	Preamble Length	—	I/F Mode	PreAmRxEN	PreAmTxEN	Huge Frame	Length check	MPEN	PAD/CRC	CRC EN	Full Duplex
Reset	0 1 1 1	0 0	0 0	0	0	0	0	0	0	0	0

Figure 18-36. MACCFG2 Register Definition

Table 18-40 describes the fields of the MACCFG2 register.

Table 18-40. MACCFG2 Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–19	Preamble Length	This field determines the length in bytes of the preamble field preceding each Ethernet start-of-frame delimiter byte. Values from 0x3 to 0xF are supported by the controller. The default value of 0x7 should not be altered in order to guarantee reliable operation with IEEE 802.3 compliant hardware.
20–21	—	Reserved
22–23	I/F Mode	This field determines the type of interface to which the MAC is connected. Its default is 00. 00 Reserved bit mode (not supported) (10 Mbps GENDEC/GPSI) 01 Nibble mode (MII) (10/100 Mbps MII/RMII) 10 Byte mode () (1000 Mbps). Reserved if neither GMII or TBI are supported. 11 Reserved
24	PreAM RxEN	User defined preamble enable for received frames. This bit is cleared by default. 0 The MAC skips the Ethernet preamble without returning it. 1 The MAC recovers the received Ethernet preamble and passes it to the driver at the start of each received frame. If the preamble is less than 7 bytes, 0's are prepended to pad it to 7 bytes. Not applicable to or RMII 10/100 modes.
25	PreAM TxEN	User defined preamble enable for transmitted frames. This bit is cleared by default. 0 The MAC generates a standard Ethernet preamble. 1 If a user-defined preamble has been passed to the MAC it is transmitted instead of the standard preamble. Otherwise the standard Ethernet preamble is generated. The Preamble Length field should be left at its default setting if a user-defined preamble is transmitted. Not applicable to or RMII 10/100 modes.

Table 18-40. MACCFG2 Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>26</td><td>Huge Frame</td><td>Huge frame enable. This bit is cleared by default.
0 Limit the length of frames received to less than or equal to the maximum frame length value (MAXFRM[Maximum Frame]) and limit the length of frames transmitted to less than the maximum frame length.
See Section 18.6.6, “Buffer Descriptors,” for further details of buffer descriptor bit updating.
<table>
<tr><th>Frame type</th><th>Frame length</th><th>Packet truncation</th><th>Buffer descriptor updated</th></tr>
<tr><td>Receive or transmit</td><td>> maximum frame length</td><td>yes</td><td>yes</td></tr>
<tr><td>Receive</td><td>= maximum frame length</td><td>no</td><td>yes</td></tr>
<tr><td>Transmit</td><td>= maximum frame length</td><td>no</td><td>no</td></tr>
<tr><td>Receive or transmit</td><td>< maximum frame length</td><td>no</td><td>no</td></tr>
</table>
1 Frames are transmitted and received regardless of their relationship to the maximum frame length.
Note that if Huge Frame is cleared, the user must ensure that adequate buffer space is allocated for received frames. See Section 18.5.3.5.5, “Maximum Frame Length Register (MAXFRM),” for further information.</td></tr>
<tr><td>27</td><td>Length check</td><td>Length check. This bit is cleared by default.
0 No length field checking is performed.
1 The MAC checks the frame’s length field on receive to ensure it matches the actual data field length. Transmitted frames are not checked.</td></tr>
<tr><td>28</td><td>MPEN</td><td>Magic packet enable for Ethernet modes. This bit is cleared by default. MPEN should be enabled only after GRACEFUL RECEIVE STOP and GRACEFUL TRANSMIT STOP are completed successfully (in other words, transmission and reception have stopped).
0 Normal receive behavior on receive, or Magic Packet mode has exited with reception of a valid Magic Packet.
1 Commence Magic Packet detection by the MAC provided that frame reception is enabled in MACCFG1. In this mode the MAC ignores all received frames until the specific Magic Packet frame is received, at which point this bit is cleared by the eTSEC, and a maskable interrupt through IEVENT[MAG] occurs.</td></tr>
<tr><td>29</td><td>PAD/CRC</td><td>Pad and append CRC. This bit is cleared by default.This bit must be set when in half-duplex mode (MACCFG2[Full Duplex] is cleared).
0 Frames presented to the MAC have a valid length and contain a CRC.
1 The MAC pads all transmitted short frames and appends a CRC to every frame regardless of padding requirement.</td></tr>
<tr><td>30</td><td>CRC EN</td><td>CRC enable. If the configuration bit PAD/CRC ENABLE or the per-packet PAD/CRC ENABLE is set, CRC ENABLE is ignored. This bit is cleared by default.
0 Frames presented to the MAC have a valid length and contain a valid CRC.
1 The MAC appends a CRC on all frames. Clear this bit if frames presented to the MAC have a valid length and contain a valid CRC.</td></tr>
<tr><td>31</td><td>Full Duplex</td><td>Full duplex configure. This bit is cleared by default.
0 The MAC operates in half-duplex mode only.
1 The MAC operates in full-duplex mode.</td></tr>
</table>

18.5.3.5.3 Inter-Packet Gap/Inter-Frame Gap Register (IPGIFG)

The IPGIFG register is written by the user. Figure 18-37 describes the definition for IPGIFG.

Offset eTSEC1:0x2_4508; eTSEC2:0x2_5508 — Access: Read/Write

	0	1–7	8	9–15	16–23	24	25–31
R/W	—	Non-Back-to-Back Inter-Packet-Gap, Part 1	—	Non-Back-to-Back Inter-Packet-Gap, Part 2	Minimum IFG Enforcement	—	Back-to-Back Inter-Packet-Gap
Reset	0	1 0 0 0 0 0 0	0	1 1 0 0 0 0 0	0 1 0 1 0 0 0 0	0	1 1 0 0 0 0 0

Figure 18-37. IPGIFG Register Definition

Table 18-41 describes the fields of the IPGIFG register.

Table 18-41. IPGIFG Field Descriptions

Bits	Name	Description
0	—	Reserved
1–7	Non-Back-to-Back Inter-Packet-Gap, Part 1	This is a programmable field representing the optional carrier sense window referenced in IEEE 802.3/4.2.3.2.1 ‘carrier deference’. If carrier is detected during the timing of IPGR1, the MAC defers to carrier. If, however, carrier becomes active after IPGR1, the MAC continues timing IPGR2 and transmits, knowingly causing a collision, thus ensuring fair access to medium. Its range of values is 0x00 to IPGR2. Its default is 0x40 (64d) which follows the two-thirds/one-third guideline.
8	—	Reserved
9–15	Non-Back-to-Back Inter-Packet-Gap, Part 2	This is a programmable field representing the non-back-to-back inter-packet-gap in bits. Its default is 0x60 (96d), which represents the minimum IPG of 96 bits.
16–23	Minimum IFG Enforcement	This is a programmable field representing the minimum number of bits of IFG to enforce between frames. A frame is dropped whose IFG is less than that programmed. The default setting of 0x50 (80d) represents half of the nominal minimum IFG which is 160 bits.
24	—	Reserved
25–31	Back-to-Back Inter-Packet-Gap	This is a programmable field representing the IPG between back-to-back packets. This is the IPG parameter used exclusively in full-duplex mode and in half-duplex mode if two transmit packets are sent back-to-back. Set this field to the number of bits of IPG desired. The default setting of 0x60 (96d) represents the minimum IPG of 96 bits.

18.5.3.5.4 Half-Duplex Register (HAFDUP)

The HAFDUP register is written by the user. Figure 18-38 describes the HAFDUP register.

Offset eTSEC1:0x2_450C; eTSEC2:0x2_550C — Access: Read/Write

Bits	0–7	8–11	12	13	14	15
R/W	—	Alternate BEB Truncation	Alt BEB	BP No BackOff	No BackOff	Excess Defer
Reset	0 0 0 0 0 0 0 0	1 0 1 0	0	0	0	1

Bits	16–19	20–25	26–31
R/W	Retransmission Maximum	—	Collision Window
Reset	1 1 1 1	0 0 0 0 0 0	1 1 0 1 1 1

Figure 18-38. Half-Duplex Register Definition

Table 18-42 describes the fields of the HAFDUP register.

Table 18-42. HAFDUP Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–11	Alternate BEB Truncation	This field is used while ALTERNATE BINARY EXPONENTIAL BACKOFF ENABLE is set. The value programmed is substituted for the Ethernet standard value of ten. Its default is 0xA.
12	Alt BEB	Alternate binary exponential backoff. This bit is cleared by default. 0 The Tx MAC follows the standard binary exponential back off rule. 1 The Tx MAC uses the ALTERNATE BINARY EXPONENTIAL BACKOFF TRUNCATION setting instead of the 802.3 standard tenth collision. The standard specifies that any collision after the tenth uses one less than 210 as the maximum backoff time.
13	BP No BackOff	Back pressure no backoff. This bit is cleared by default. 0 The Tx MAC follows the binary exponential back off rule. 1 The Tx MAC immediately re-transmits, following a collision, during back pressure operation.
14	No BackOff	No backoff. This bit is cleared by default. 0 The Tx MAC follows the binary exponential back off rule. 1 The Tx MAC immediately re-transmits following a collision.
15	Excess Defer	Excessively deferred. This bit is set by default. 0 The Tx MAC aborts the transmission of a packet that is excessively deferred. 1 The Tx MAC allows the transmission of a packet that is excessively deferred.
16–19	Retransmission Maximum	This is a programmable field specifying the number of retransmission attempts following a collision before aborting the packet due to excessive collisions. The standard specifies the attempt limit to be 0xF (15d). Its default value is 0xF.
20–25	—	Reserved
26–31	Collision Window	This is a programmable field representing the slot time or collision window during which collisions occur in properly configured networks. Because the collision window starts at the beginning of transmission, the preamble and SFD are included. Its default of 0x37 (55d) corresponds to the count of frame bytes at the end of the window.

18.5.3.5.5 Maximum Frame Length Register (MAXFRM)

The MAXFRM register is written by the user. Figure 18-39 shows the MAXFRM register.

Offset eTSEC1:0x2_4510; eTSEC2:0x2_5510 Access: Read/Write

	0–15	16–31
R/W	—	Maximum Frame
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0

Figure 18-39. Maximum Frame Length Register Definition

Table 18-43 describes the fields of the MAXFRM register.

Table 18-43. MAXFRM Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	Maximum Frame	By default this field is set to 0x0600 (1536 bytes). It sets the maximum Ethernet frame size in both the transmit and receive directions. (Refer to MACCFG2[Huge Frame].) Note that if MACCFG2[Huge Frame] = 0, the value of this field must be less than or equal to MRBLR[MRBL] × (minimum number of RxBDs per ring). See Section 18.5.3.5.2, "MAC Configuration 2 Register (MACCFG2)," Section 18.5.3.3.9, "Maximum Receive Buffer Length Register (MRBLR)," and Section 18.6.6.3, "Receive Buffer Descriptors (RxBD)."

18.5.3.5.6 MII Management Configuration Register (MIIMCFG)

The MIIMCFG register is written by the user to configure all MII management operations. Note that MII management hardware is shared by all eTSECs. Thus, only through the MIIM registers of eTSEC1 can external PHYs be accessed and configured. Note: when an eTSEC is configured to use RTBI, configuration of the RTBI (described in Section 18.5.4, "Ten-Bit Interface (TBI)") is done through the MIIM registers for that eTSEC. For example, if a RTBI interface is required on eTSEC2, then the MIIM registers starting at offset 0x2_5520 are used to configure it.

Figure 18-40 describes the definition for the MIIMCFG register.

Offset eTSEC1:0x2_4520 Access: Read/Write

	0	1–26	27	28	29–31
R/W	Reset Mgmt	—	No Pre	—	MgmtClk
Reset	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	1 1 1

Figure 18-40. MII Management Configuration Register Definition

Table 18-44 describes the fields of the MIIMCFG register.

Table 18-44. MIIMCFG Field Descriptions

Bits	Name	Description
0	Reset Mgmt	Reset management. This bit is cleared by default. 0 Allow the MII MGMT to perform mgmt read/write cycles if requested through the host interface. 1 Reset the MII MGMT.
1–26	—	Reserved
27	No Pre	Preamble suppress. This bit is cleared by default. 0 The MII MGMT performs Mgmt read/write cycles with 32 clocks of preamble. 1 The MII MGMT suppresses preamble generation and reduces the Mgmt cycle from 64 clocks to 32 clocks. This is in accordance with IEEE 802.3/22.2.4.4.2.
28	—	Reserved
29–31	MgmtClk	This field determines the clock frequency of the MII management clock (EC_MDC). Its default value is 111. **Note:** The eTSEC system clock is derived from csb_clk/SCCR[TSEC*n*CM]. **Note:** The eTSEC system clock is derived from (CCB Clock)/2. 000 1/4 of the eTSEC system clock divided by 8 001 1/4 of the eTSEC system clock divided by 8 010 1/6 of the eTSEC system clock divided by 8 011 1/8 of the eTSEC system clock divided by 8 100 1/10 of the eTSEC system clock divided by 8 101 1/14 of the eTSEC system clock divided by 8 110 1/20 of the eTSEC system clock divided by 8 111 1/28 of the eTSEC system clock divided by 8

18.5.3.5.7 MII Management Command Register (MIIMCOM)

The MIIMCOM register is written by the user. Figure 18-41 describes the definition for MIIMCOM.

Offset eTSEC1:0x2_4524 Access: Read/Write

	0–29	30	31
R W	—	Scan Cycle	Read Cycle
Reset	All zeros		

Figure 18-41. MIIMCOM Register Definition

Table 18-45 describes the fields of the MIIMCOM register.

Table 18-45. MIIMCOM Descriptions

Bits	Name	Description
0–29	—	Reserved

Table 18-45. MIIMCOM Descriptions (continued)

Bits	Name	Description
30	Scan Cycle	Scan cycle. This bit is cleared by default. 0 Normal operation. 1 The MII management continuously performs read cycles. This is useful for monitoring link fail, for example.
31	Read Cycle	Read cycle. This bit is cleared by default but is not self-clearing once set. 0 Normal operation. 1 The MII management performs a single read cycle upon the transition of this bit from 0 to 1 using the PHY address (at MIIMADD[PHY Address]) and the register address (at MIIMADD[Register Address]). The 0-to-1 transition of this bit also causes the MIIMIND[Busy] bit to be set. The read is complete when the MIIMIND[Busy] bit clears. Data is returned in register MIIMSTAT[PHY Status].

18.5.3.5.8 MII Management Address Register (MIIMADD)

The MIIMADD register is written by the user. Figure 18-42 shows the MIIMADD register.

Offset eTSEC1:0x2_4528 Access: Read/Write

	0–18	19–23	24–26	27–31
R W	—	PHY Address	—	Register Address
Reset	All zeros			

Figure 18-42. MIIMADD Register Definition

Table 18-46 describes the fields of the MIIMADD register.

Table 18-46. MIIMADD Field Descriptions

Bits	Name	Description
0–18	—	Reserved
19–23	PHY Address	This field represents the 5-bit PHY address field of Mgmt cycles. Up to 31 PHYs can be addressed (0 is reserved). Its default value is 0x00.
24–26	—	Reserved
27–31	Register Address	This field represents the 5-bit register address field of Mgmt cycles. Up to 32 registers can be accessed. Its default value is 0x00.

18.5.3.5.9 MII Management Control Register (MIICON)

MIIMCON, shown in Figure 18-43, is written by the user.

Offset eTSEC1:0x2_452C Access: WO

	0–15	16–31
R	—	
W		PHY Control
Reset	All zeros	

Figure 18-43. MII Mgmt Control Register Definition

Table 18-47 describes the fields of the MIIMCON register.

Table 18-47. MIIMCON Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	PHY Control	If written, an MII Mgmt write cycle is performed using this 16-bit data, the pre-configured PHY address (at MIIMADD[PHY Address]) and the register address (at MIIMADD[Register Address]). Its default value is 0x0000.

18.5.3.5.10 MII Management Status Register (MIIMSTAT)

The MIIMSTAT register is read only by the user. Figure 18-44 describes the definition for the MIIMSTAT register.

Offset eTSEC1:0x2_4530 Access: Read only

	0–15	16–31
R	—	PHY Status
W		
Reset	All zeros	

Figure 18-44. MIIMSTAT Register Definition

Table 18-48 describes the fields of the MIIMSTAT register.

Table 18-48. MIIMSTAT Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	PHY Status	Following an MII Mgmt read cycle, the 16-bit data can be read from this location. Its default value is 0x0000.

18.5.3.5.11 MII Management Indicator Register (MIIMIND)

The MIIMIND register is read-only by the user. Figure 18-45 describes the definition for the MIIMIND register.

Offset eTSEC1:0x2_4534 Access: Read only

	0–28	29	30	31
R	—	Not Valid	Scan	Busy
W				
Reset	All zeros			

Figure 18-45. MII Mgmt Indicator Register Definition

Table 18-49. MIIMIND Field Descriptions

Bits	Name	Description
0–28	—	Reserved
29	Not Valid	Not valid. 0 MII Mgmt read cycle has completed and the read data is valid. 1 MII Mgmt read cycle has not completed and the read data is not yet valid.
30	Scan	Scan in progress. 0 A scan operation (continuous MII Mgmt read cycles) is not in progress. 1 A scan operation (continuous MII Mgmt read cycles) is in progress.
31	Busy	Busy. 0 MII Mgmt block is not currently performing an MII Mgmt read or write cycle. 1 MII Mgmt block is currently performing an MII Mgmt read or write cycle.

18.5.3.5.12 Interface Status Register (IFSTAT)

Figure 18-46 shows the IFSTAT register.

Offset eTSEC1:0x2_453C; eTSEC2:0x2_553C — Access: Read only

	0–21	22	23–31
R	—	Excess Defer	—
W			
Reset	All zeros		

Figure 18-46. Interface Status Register Definition

Table 18-50 describes the fields of the FSTAT register.

Table 18-50. IFSTAT Field Descriptions

Bits	Name	Description
0–21	—	Reserved
22	Excess Defer	Excessive transmission defer. This bit latches high and is cleared when read. This bit is cleared by default. 0 Normal operation. 1 The MAC excessively defers a transmission.
23–31	—	Reserved

18.5.3.5.13 MAC Station Address Part 1 Register (MACSTNADDR1)

The MACSTNADDR1 register is written by the user. The value of the station address written into MACSTNADDR1 and MACSTNADDR2 is byte reversed from how it would appear in the DA field of a frame in memory. For example, for a station address of 0x12345678ABCD, MACSTNADDR1 is set to 0xCDAB7856 and MACSTNADDR2 is set to 0x34120000.

Figure 18-47 shows the MACSTNADDR1 register.

Offset eTSEC1:0x2_4540; eTSEC2:0x2_5540 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	Station Address, 6th Octet	Station Address, 5th Octet	Station Address, 4th Octet	Station Address, 3rd Octet
Reset	All zeros			

Figure 18-47. MAC Station Address Part 1 Register Definition

Table 18-51 describes the fields of the MACSTNADDR1 register.

Table 18-51. MACSTNADDR1 Field Descriptions

Bit	Name	Description
0–7	Station Address, 6th Octet	This field holds the sixth octet of the station address. The sixth octet (station address bits 40–47) defaults to a value of 0x0.
8–15	Station Address, 5th Octet	This field holds the fifth octet of the station address. The fifth octet (station address bits 32–39) defaults to a value of 0x0.
16–23	Station Address, 4th Octet	This field holds the fourth octet of the station address. The fourth octet (station address bits 24–31) defaults to a value of 0x0.
24–31	Station Address, 3rd Octet	This field holds the third octet of the station address. The third octet (station address bits 16–23) defaults to a value of 0x0.

18.5.3.5.14 MAC Station Address Part 2 Register (MACSTNADDR2)

The MACSTNADDR2 register is written by the user. Figure 18-48 describes the definition for the MACSTNADDR2 register.

Offset eTSEC1:0x2_4544; eTSEC2:0x2_5544 Access: Read/Write

	0–7	8–15	16–31
R W	Station Address, 2nd Octet	Station Address, 1st Octet	—
Reset	All zeros		

Figure 18-48. MAC Station Address Part 2 Register Definition

Table 18-52 describes the fields of the MACSTNADDR2 register.

Table 18-52. MACSTNADDR2 Field Descriptions

Bit	Name	Description
0–7	Station Address, 2nd Octet	This field holds the second octet of the station address. The second octet (station address bits 8–15) defaults to a value of 0x0.
8–15	Station Address, 1st Octet	This field holds the first octet of the station address. The first octet (station address bits 0–7) defaults to a value of 0x0.
16–31	—	Reserved

18.5.3.5.15 MAC Exact Match Address 1–15 Part 1 Registers (MAC01ADDR1–MAC15ADDR1)

The MAC01ADDR1–MAC15ADDR1 registers are written by the user with the unicast or multicast addresses aliasing the MAC. Figure 18-49 describes the definition for all of the fifteen MAC*n*ADDR1 registers. The value of the address written into MAC*x*ADDR1 and MAC*n*ADDR2 is byte reversed from how it would appear in the DA field of a frame in memory. For example, for a MAC address of 0x12345678ABCD, MACnADDR1 is set to 0xCDAB7856 and MACnADDR2 is set to 0x34120000. For any valid, non-zero MAC address received, exact match registers can be excluded individually by clearing them to all zero bytes.

Offset eTSEC1:0x2_4548+8×*n*; eTSEC2:0x2_5548+8×*n* Access: Read/Write

	0–7	8–15	16–23	24–31
R W	Exact Match Address, 6th Octet	Exact Match Address, 5th Octet	Exact Match Address, 4th Octet	Exact Match Address, 3rd Octet
Reset	All zeros			

Figure 18-49. MAC Exact Match Address *n* Part 1 Register Definition

Table 18-51 describes the fields of a MAC*n*ADDR1 register.

Table 18-53. MAC*n*ADDR1 Field Descriptions

Bit	Name	Description
0–7	Exact Match Address, 6th Octet	Holds the sixth octet of the exact match address. The sixth octet (destination address bits 40–47) defaults to a value of 0x0.
8–15	Exact Match Address, 5th Octet	Holds the fifth octet of the exact match address. The fifth octet (destination address bits 32–39) defaults to a value of 0x0.
16–23	Exact Match Address, 4th Octet	Holds the fourth octet of the exact match address. The fourth octet (destination address bits 24–31) defaults to a value of 0x0.
24–31	Exact Match Address, 3rd Octet	Holds the third octet of the exact match address. The third octet (destination address bits 16–23) defaults to a value of 0x0.

18.5.3.5.16 MAC Exact Match Address 1–15 Part 2 Registers (MAC01ADDR2–MAC15ADDR2)

The MAC01ADDR2–MAC15ADDR2 registers are written by the user with the unicast or multicast addresses aliasing the MAC. Figure 18-50 describes the definition for all of the fifteen MACxADDR2 registers.

Offset eTSEC1:0x2_454C+8×*n*; eTSEC2:0x2_554C+8×*n* Access: Read/Write

	0–7	8–15	16–31
R W	Exact Match Address, 2nd Octet	Exact Match Address, 1st Octet	—
Reset	All zeros		

Figure 18-50. MAC Exact Match Address *x* Part 2 Register Definition

Table 18-52 describes the fields of a MAC*x*ADDR2 register.

Table 18-54. MAC01ADDR2–MAC15ADDR2 Field Descriptions

Bit	Name	Description
0–7	Exact Match Address, 2nd Octet	This field holds the second octet of the exact match address. The second octet (destination address bits 8–15) defaults to a value of 0x0.
8–15	Exact Match Address, 1st Octet	This field holds the first octet of the exact match address. The first octet (destination address bits 0–7) defaults to a value of 0x0.
16–31	—	Reserved

18.5.3.6 MIB Registers

This section describes the MIB registers. The eTSEC RMON module has 37 separate statistics counters, which simply count or accumulate statistical events that occur as packets transmitted and received. These counters support RMON MIB group 1, RMON MIB group 2 if table counters, RMON MIB group 3, RMON MIB group 9, RMON MIB 2, and the IEEE 802.3 Ethernet MIB.

An interrupt can be generated upon any one counter's rollover condition through a carry interrupt output from the RMON. Each counter's rollover condition can be discretely masked from causing an interrupt by internal masking registers. In addition, each individual counter value may be reset on read access, or all counters may be simultaneously reset by setting ECNTRL[CLRCNT].

The majority of MIB counters are Ethernet-specific.

NOTE

RMON counters do not comprehend custom VLAN tagged frames. Affected counters include TRMGV, RMCA, RBCA, RXCF, RXPF, RXUO, RALN, RFLR, ROVR, RJBR, TMCA, TBCA, TXPF, TXCF. Specifically, custom VLAN tagged frames are not afforded the ability to be greater than 1518, as compared to the IEEE standard tagged frames.

18.5.3.6.1 Transmit and Receive 64-Byte Frame Counter (TR64)

Figure 18-51 describes the definition for the TR64 register.

Offset eTSEC1:0x2_4680; eTSEC2:0x2_5680 Access: Read/Write

	0–9	10–31
R W	—	TR64
Reset	All zeros	

Figure 18-51. Transmit and Receive 64-Byte Frame Register Definition

Table 18-55 describes the fields of the TR64 register.

Table 18-55. TR64 Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TR64	Transmit and receive 64-byte frame counter—Increment for each good or bad frame transmitted and received which is 64 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes).

18.5.3.6.2 Transmit and Receive 65- to 127-Byte Frame Counter (TR127)

Figure 18-52 describes the definition for the TR127 register.

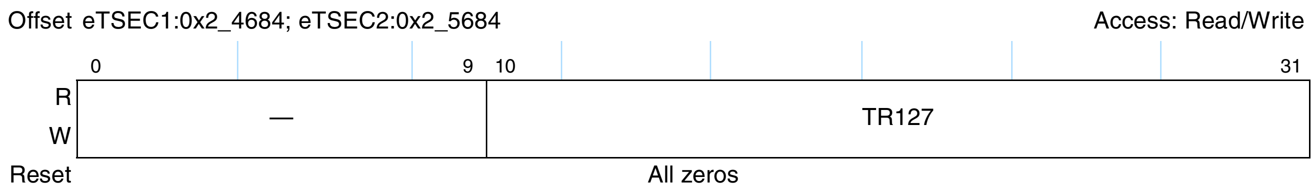


Figure 18-52. Transmit and Receive 65- to 127-Byte Frame Register Definition

Table 18-56 describes the fields of the TR127 register.

Table 18-56. TR127 Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TR127	Transmit and receive 65- to 127-byte frame counter—Increments for each good or bad frame transmitted and received which is 65–127 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes).

18.5.3.6.3 Transmit and Receive 128- to 255-Byte Frame Counter (TR255)

Figure 18-53 describes the definition for the TR255 register.

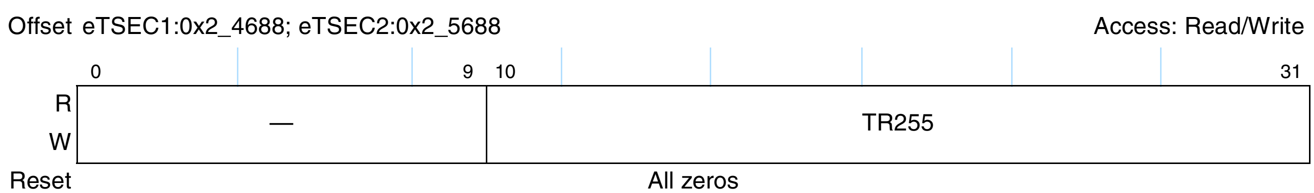


Figure 18-53. Transmit and Received 128- to 255-Byte Frame Register Definition

Table 18-57 describes the fields of the TR255 register.

Table 18-57. TR255 Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TR255	Transmit and receive 128- to 255-byte frame counter—Increments for each good or bad frame transmitted and received which is 128–255 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes).

18.5.3.6.4 Transmit and Receive 256- to 511-Byte Frame Counter (TR511)

Figure 18-54 describes the definition for the TR511 register.

Offset eTSEC1:0x2_468C; eTSEC2:0x2_568C — Access: Read/Write

	0–9	10–31
R/W	—	TR511
Reset	All zeros	

Figure 18-54. Transmit and Received 256- to 511-Byte Frame Register Definition

Table 18-58 describes the fields of the TR511 register.

Table 18-58. TR511 Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TR511	Increments for each good or bad frame transmitted and received which is 256–511 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes).

18.5.3.6.5 Transmit and Receive 512- to 1023-Byte Frame Counter (TR1K)

Figure 18-55 shows the TR1K register.

Offset eTSEC1:0x2_4690; eTSEC2:0x2_5690 — Access: Read/Write

	0–9	10–31
R/W	—	TR1K
Reset	All zeros	

Figure 18-55. Transmit and Received 512- to 1023-Byte Frame Register Definition

Table 18-59 describes the fields of the TR1K register.

Table 18-59. TR1K Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TR1K	Increments for each good or bad frame transmitted and received which is 512–1023 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes).

18.5.3.6.6 Transmit and Receive 1024- to 1518-Byte Frame Counter (TRMAX)

Figure 18-56 describes the definition for the TRMAX register.

Offset eTSEC1:0x2_4694; eTSEC2:0x2_5694 Access: Read/Write

	0–9	10–31
R W	—	TRMAX
Reset	All zeros	

Figure 18-56. Transmit and Received 1024- to 1518-Byte Frame Register Definition

Table 18-60 describes the fields of the TRMAX register.

Table 18-60. TRMAX Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TRMAX	Increments for each good or bad frame transmitted and received which is 1024–1518 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes).

18.5.3.6.7 Transmit and Receive 1519- to 1522-Byte VLAN Frame Counter (TRMGV)

Figure 18-57 describes the definition for the TRMGV register.

Offset eTSEC1:0x2_4698; eTSEC2:0x2_5698 Access: Read/Write

	0–9	10–31
R W	—	TRMGV
Reset	All zeros	

Figure 18-57. Transmit and Received 1519- to 1522-Byte VLAN Frame Register Definition

Table 18-61 describes the fields of the TRMGV register.

Table 18-61. TRMGV Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TRMGV	Increments for each good or bad frame transmitted and received which is 1519–1522 bytes in length, inclusive (excluding preamble and SFD but including FCS bytes).

18.5.3.6.8 Receive Byte Counter (RBYT)

Figure 18-58 shows the RBYT register.

Offset eTSEC1:0x2_469C; eTSEC2:0x2_569C — Access: Read/Write

	0–31
R/W	RBYT
Reset	All zeros

Figure 18-58. Receive Byte Counter Register Definition

Table 18-62 describes the fields of the RBYT register.

Table 18-62. RBYT Field Descriptions

Bits	Name	Description
0–31	RBYT	Receive byte counter. The statistic counter register increments by the byte count of frames received, including those in bad packets, excluding preamble and SFD but including FCS bytes.

18.5.3.6.9 Receive Packet Counter (RPKT)

Figure 18-59 describes the definition for the RPKT register.

Offset eTSEC1:0x2_46A0; eTSEC2:0x2_56A0 — Access: Read/Write

	0–9	10–31
R/W	—	RPKT
Reset	All zeros	

Figure 18-59. Receive Packet Counter Register Definition

Table 18-63 describes the fields of the RPKT register.

Table 18-63. RPKT Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10-31	RPKT	Receive packet counter. Increments for each frame received packet (including bad packets, all unicast, broadcast, and multicast packets).

18.5.3.6.10 Receive FCS Error Counter (RFCS)

Figure 18-60 describes the definition for the RFCS register.

Offset eTSEC1:0x2_46A4; eTSEC2:0x2_56A4 — Access: Read/Write

	0–15	16–31
R/W	—	RFCS
Reset	All zeros	

Figure 18-60. Receive FCS Error Counter Register Definition

Table 18-64 describes the fields of the RFCS register.

Table 18-64. RFCS Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RFCS	Receive FCS error counter. In Ethernet mode, increments for each frame received that has an integral 64–1518 length and contains a frame check sequence error.

18.5.3.6.11 Receive Multicast Packet Counter (RMCA)

Figure 18-61 describes the definition for the RMCA register.

Offset eTSEC1:0x2_46A8; eTSEC2:0x2_56A8 Access: Read/Write

	0–9	10–31
R W	—	RMCA
Reset	All zeros	

Figure 18-61. Receive Multicast Packet Counter Register Definition

Table 18-65 describes the fields of the RMCA register.

Table 18-65. RMCA Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	RMCA	Receive multicast packet counter. Increments for each multicast frame with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN), excluding broadcast frames. This count does not include range/length errors.

18.5.3.6.12 Receive Broadcast Packet Counter (RBCA)

Figure 18-62 describes the definition for the RBCA register.

Offset eTSEC1:0x2_46AC; eTSEC2:0x2_56AC Access: Read/Write

	0–9	10–31
R W	—	RBCA
Reset	All zeros	

Figure 18-62. Receive Broadcast Packet Counter Register Definition

Table 18-66 describes the fields of the RBCA register.

Table 18-66. RBCA Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	RBCA	Receive broadcast packet counter. Increments for each broadcast frame with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN), excluding multicast frames. Does not include range/length errors.

18.5.3.6.13 Receive Control Frame Packet Counter (RXCF)

Figure 18-63 describes the definition for the RXCF register.

Offset eTSEC1:0x2_46B0; eTSEC2:0x2_56B0 Access: Read/Write

	0–15	16–31
R	—	RXCF
W		
Reset	All zeros	

Figure 18-63. Receive Control Frame Packet Counter Register Definition

Table 18-67 describes the fields of the RXCF register.

Table 18-67. RXCF Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RXCF	Receive control frame packet counter. Increments for each MAC control frame received (PAUSE and unsupported) with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN).

18.5.3.6.14 Receive Pause Frame Packet Counter (RXPF)

Figure 18-64 describes the definition for the RXPF register.

Offset eTSEC1:0x2_46B4; eTSEC2:0x2_56B4 Access: Read/Write

	0–15	16–31
R	—	RXPF
W		
Reset	All zeros	

Figure 18-64. Receive Pause Frame Packet Counter Register Definition

Table 18-68 describes the fields of the RXPF register.

Table 18-68. RXPF Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RXPF	Receive PAUSE frame packet counter. Increments each time a PAUSE MAC control frame is received with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN).

18.5.3.6.15 Receive Unknown Opcode Packet Counter (RXUO)

Figure 18-65 describes the definition for the RXUO register.

Offset eTSEC1:0x2_46B8; eTSEC2:0x2_56B8 Access: Read/Write

	0–15	16–31
R W	—	RXUO
Reset	All zeros	

Figure 18-65. Receive Unknown OPCode Packet Counter Register Definition

Table 18-69 describes the fields of the RXUO register.

Table 18-69. RXUO Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RXUO	Receive unknown opcode counter. Increments each time a MAC control frame is received which contains an opcode other than PAUSE, but the frame has valid CRC and length 64 to 1518 (non VLAN) or 1522 (VLAN).

18.5.3.6.16 Receive Alignment Error Counter (RALN)

Figure 18-66 describes the definition for the RALN register.

Offset eTSEC1:0x2_46BC; eTSEC2:0x2_56BC Access: Read/Write

	0–15	16–31
R W	—	RALN
Reset	All zeros	

Figure 18-66. Receive Alignment Error Counter Register Definition

Table 18-70 describes the fields of the RALN register.

Table 18-70. RALN Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RALN	Receive alignment error counter. Increments for each received frame from 64 to 1518 (non VLAN) or 1522 (VLAN) which contains an invalid FCS and is not an integral number of bytes.

18.5.3.6.17 Receive Frame Length Error Counter (RFLR)

Figure 18-67 describes the definition for the RFLR register.

Offset eTSEC1:0x2_46C0; eTSEC2:0x2_56C0 — Access: Read/Write

	0–15	16–31
R/W	—	RFLR
Reset	All zeros	

Figure 18-67. Receive Frame Length Error Counter Register Definition

Table 18-71 describes the fields of the RFLR register.

Table 18-71. RFLR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RFLR	Receive frame length error counter. Increments for each frame received in which the 802.3 length field did not match the number of data bytes actually received (46–1500 bytes). The counter does not increment if the length field is not a valid 802.3 length, such as an Ethertype value.

18.5.3.6.18 Receive Code Error Counter (RCDE)

Figure 18-68 describes the definition for the RCDE register.

Offset eTSEC1:0x2_46C4; eTSEC2:0x2_56C4 — Access: Read/Write

	0–15	16–31
R/W	—	RCDE
Reset	All zeros	

Figure 18-68. Receive Code Error Counter Register Definition

Table 18-72 describes the fields of the RCDE register.

Table 18-72. RCDE Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RCDE	Receive code error counter. Increments each time a valid carrier is present and at least one invalid data symbol is detected.

18.5.3.6.19 Receive Carrier Sense Error Counter (RCSE)

Figure 18-69 describes the definition for the RCSE register.

Offset eTSEC1:0x2_46C8; eTSEC2:0x2_56C8 Access: Read/Write

	0–15	16–31
R / W	—	RCSE
Reset	All zeros	

Figure 18-69. Receive Carrier Sense Error Counter Register Definition

Table 18-73 describes the fields of the RCSE register.

Table 18-73. RCSE Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RCSE	Receive false carrier counter. Counts the number of times that the carrier sense condition was lost or never asserted when attempting to transmit a frame on a particular interface. The count represented by an instance of this object is incremented at most once per transmission attempt, even if the carrier sense condition fluctuates during a transmission attempt. The event is reported along with the statistics generated on the next received frame, as defined by a 1 on TSEC*n*_RX_ER and an 0xE on TSEC*n*_RXD. Only one false carrier condition can be detected and logged between frames.

18.5.3.6.20 Receive Undersize Packet Counter (RUND)

Figure 18-70 describes the definition for the RUND register.

Offset eTSEC1:0x2_46CC; eTSEC2:0x2_56CC Access: Read/Write

	0–15	16–31
R / W	—	RUND
Reset	All zeros	

Figure 18-70. Receive Undersize Packet Counter Register Definition

Table 18-74 describes the fields of the RUND register.

Table 18-74. RUND Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RUND	Receive undersize packet counter. Increments each time a frame is received which is less than 64 bytes in length and contains a valid FCS and were otherwise well formed. This count does not include range length errors.

18.5.3.6.21 Receive Oversize Packet Counter (ROVR)

Figure 18-71 describes the definition for the ROVR register.

Offset eTSEC1:0x2_46D0; eTSEC2:0x2_56D0 — Access: Read/Write

	0–15	16–31
R W	—	ROVR
Reset	All zeros	

Figure 18-71. Receive Oversize Packet Counter Register Definition

Table 18-75 describes the fields of the ROVR register.

Table 18-75. ROVR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	ROVR	Receive oversize packet counter. Increments each time a frame is received which exceeded 1518 (non VLAN) or 1522 (VLAN) and contains a valid FCS and was otherwise well formed.

18.5.3.6.22 Receive Fragments Counter (RFRG)

Figure 18-72 describes the definition for the RFRG register.

Offset eTSEC1:0x2_46D4; eTSEC2:0x2_56D4 — Access: Read/Write

	0–15	16–31
R W	—	RFRG
Reset	All zeros	

Figure 18-72. Receive Fragments Counter Register Definition

Table 18-76 describes the fields of the RFRG register.

Table 18-76. RFRG Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RFRG	Receive fragments counter. Increments for each frame received which is less than 64 bytes in length and contains an invalid FCS. This includes integral and non-integral lengths.

18.5.3.6.23 Receive Jabber Counter (RJBR)

Figure 18-73 describes the definition for the RJBR register.

Offset eTSEC1:0x2_46D8; eTSEC2:0x2_56D8 — Access: Read/Write

	0–15	16–31
R W	—	RJBR
Reset	All zeros	

Figure 18-73. Receive Jabber Counter Register Definition

Table 18-77 describes the fields of the RJBR register.

Table 18-77. RJBR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RJBR	Receive jabber counter. Increments for frames received which exceed 1518 (non VLAN) or 1522 (VLAN) bytes and contain an invalid FCS. This includes alignment errors.

18.5.3.6.24 Receive Dropped Packet Counter (RDRP)

Figure 18-74 describes the definition for the RDRP register.

Offset eTSEC1:0x2_46DC; eTSEC2:0x2_56DC — Access: Read/Write

	0–15	16–31
R W	—	RDRP
Reset	All zeros	

Figure 18-74. Receive Dropped Packet Counter Register Definition

Table 18-78 describes the fields of the RDRP register.

Table 18-78. RDRP Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	RDRP	Receive dropped packets counter. Increments for frames received which are streamed to system but are later dropped due to lack of system resources.

18.5.3.6.25 Transmit Byte Counter (TBYT)

Figure 18-75 depicts the TBYT register.

Offset eTSEC1:0x2_46E0; eTSEC2:0x2_56E0 — Access: Read/Write

	0–31
R W	TBYT
Reset	All zeros

Figure 18-75. Transmit Byte Counter Register Definition

Table 18-79 describes the fields of the TBYT register.

Table 18-79. TBYT Field Descriptions

Bits	Name	Description
0–31	TBYT	Transmit byte counter. Increments by the number of bytes that were put on the wire including fragments of frames that were involved with collisions. This count does not include preamble/SFD or jam bytes, except for half-duplex flow control (back-pressure triggered by TCTRL[THDF]=1). For THDF, the sum total of 'phantom' preamble bytes transmitted for flow control purposes is included in the TBYT increment value of the next frame to be transmitted, up to 65,535 bytes of frame and phantom preamble. Note that the value of TBYT may be greater than the actual number of bytes transmitted if the frame is truncated because it exceeds MAXFRM.

18.5.3.6.26 Transmit Packet Counter (TPKT)

Figure 18-76 describes the definition for the TPKT register.

Offset eTSEC1:0x2_46E4; eTSEC2:0x2_56E4 — Access: Read/Write

	0–9	10–31
R W	—	TPKT
Reset	All zeros	

Figure 18-76. Transmit Packet Counter Register Definition

Table 18-80 describes the fields of the TPKT register.

Table 18-80. TPKT Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TPKT	Transmit packet counter. Increments for each transmitted packet (including bad packets, excessive deferred packets, excessive collision packets, late collision packets, all unicast, broadcast, and multicast packets).

18.5.3.6.27 Transmit Multicast Packet Counter (TMCA)

Figure 18-77 describes the definition for the TMCA register.

Offset eTSEC1:0x2_46E8; eTSEC2:0x2_56E8 — Access: Read/Write

	0–9	10–31
R	—	TMCA
W		
Reset	All zeros	

Figure 18-77. Transmit Multicast Packet Counter Register Definition

Table 18-81 describes the fields of the TMCA register.

Table 18-81. TMCA Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TMCA	Transmit multicast packet counter. Increments for each multicast valid frame transmitted (excluding broadcast frames) with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN).

18.5.3.6.28 Transmit Broadcast Packet Counter (TBCA)

Figure 18-78 describes the definition for the TBCA register.

Offset eTSEC1:0x2_46EC; eTSEC2:0x2_56EC — Access: Read/Write

	0–9	10–31
R	—	TBCA
W		
Reset	All zeros	

Figure 18-78. Transmit Broadcast Packet Counter Register Definition

Table 18-82 describes the fields of the TBCA register.

Table 18-82. TBCA Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	TBCA	Transmit broadcast packet counter. Increments for each broadcast frame transmitted (excluding multicast frames) with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN).

18.5.3.6.29 Transmit Pause Control Frame Counter (TXPF)

Figure 18-79 describes the definition for the TXPF register.

Offset eTSEC1:0x2_46F0; eTSEC2:0x2_56F0 — Access: Read/Write

	0–15	16–31
R W	—	TXPF
Reset	All zeros	

Figure 18-79. Transmit Pause Control Frame Counter Register Definition

Table 18-83 describes the fields of the TXPF register.

Table 18-83. TXPF Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	TXPF	Transmit PAUSE frame packet counter. Increments each time a valid PAUSE MAC control frame is transmitted with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN).

18.5.3.6.30 Transmit Deferral Packet Counter (TDFR)

Figure 18-80 describes the definition for the TDFR register.

Offset eTSEC1:0x2_46F4; eTSEC2:0x2_56F4 — Access: Read/Write

	0–19	20–31
R W	—	TDFR
Reset	All zeros	

Figure 18-80. Transmit Deferral Packet Counter Register Definition

Table 18-84 describes the fields of the TDFR register.

Table 18-84. TDFR Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TDFR	Transmit deferral packet counter. Increments for each frame, which was deferred on its first transmission attempt. This count does not include frames involved in collisions.

18.5.3.6.31 Transmit Excessive Deferral Packet Counter (TEDF)

Figure 18-81 describes the definition for the TEDF register.

Offset eTSEC1:0x2_46F8; eTSEC2:0x2_56F8 Access: Read/Write

	0–19	20–31
R W	—	TEDF
Reset	All zeros	

Figure 18-81. Transmit Excessive Deferral Packet Counter Register Definition

Table 18-85 describes the fields of the TEDF register.

Table 18-85. TEDF Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TEDF	Transmit excessive deferral packet counter. Increments for frames aborted which were deferred for an excessive period of time (3036 byte times).

18.5.3.6.32 Transmit Single Collision Packet Counter (TSCL)

Figure 18-82 describes the definition for the TSCL register.

Offset eTSEC1:0x2_46FC; eTSEC2:0x2_56FC Access: Read/Write

	0–19	20–31
R W	—	TSCL
Reset	All zeros	

Figure 18-82. Transmit Single Collision Packet Counter Register Definition

Table 18-86 describes the fields of the TSCL register.

Table 18-86. TSCL Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TSCL	Transmit single collision packet counter. Increments for each frame transmitted which experienced exactly one collision during transmission.

18.5.3.6.33 Transmit Multiple Collision Packet Counter (TMCL)

Figure 18-83 describes the definition for the TMCL register.

Offset eTSEC1:0x2_4700; eTSEC2:0x2_5700 Access: Read/Write

	0–19	20–31
R/W	—	TMCL
Reset	All zeros	

Figure 18-83. Transmit Multiple Collision Packet Counter Register Definition

Table 18-87 describes the fields of the TMCL register.

Table 18-87. TMCL Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TMCL	Transmit multiple collision packet counter. Increments for each frame transmitted which experienced 2–15 collisions (including any late collisions) during transmission as defined using the Half_Duplex[RETRANSMISSION MAXIMUM] field.

18.5.3.6.34 Transmit Late Collision Packet Counter (TLCL)

Figure 18-84 describes the definition for the TLCL register.

Offset eTSEC1:0x2_4704; eTSEC2:0x2_5704 Access: Read/Write

	0–19	20–31
R/W	—	TLCL
Reset	All zeros	

Figure 18-84. Transmit Late Collision Packet Counter Register Definition

Table 18-88 describes the fields of the TLCL register.

Table 18-88. TLCL Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TLCL	Transmit late collision packet counter. Increments for each frame transmitted which experienced a late collision during a transmission attempt. Late collisions are defined using the collision window field of the half-duplex [26:31] register.

18.5.3.6.35 Transmit Excessive Collision Packet Counter (TXCL)

Figure 18-85 describes the definition for the TXCL register.

Offset eTSEC1:0x2_4708; eTSEC2:0x2_5708 Access: Read/Write

	0–19	20–31
R/W	—	TXCL
Reset	All zeros	

Figure 18-85. Transmit Excessive Collision Packet Counter Register Definition

Table 18-89 describes the fields of the TXCL register.

Table 18-89. TXCL Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TXCL	Transmit excessive collision packet counter. Increments for each frame that experienced 16 collisions during transmission and was aborted.

18.5.3.6.36 Transmit Total Collision Counter (TNCL)

Figure 18-86 describes the definition for the TNCL register.

Offset eTSEC1:0x2_470C; eTSEC2:0x2_570C Access: Read/Write

	0–19	20–31
R/W	—	TNCL
Reset	All zeros	

Figure 18-86. Transmit Total Collision Counter Register Definition

Table 18-90 describes the fields of the TNCL register.

Table 18-90. TNCL Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TNCL	Transmit total collision counter. Increments by the number of collisions experienced during the transmission of a frame as defined as the simultaneous presence of signals on the DO and RD circuits (That is, transmitting and receiving at the same time). **Note:** This count does not include collisions that result in an excessive collision condition.

18.5.3.6.37 Transmit Drop Frame Counter (TDRP)

Figure 18-87 describes the definition for the TDRP register.

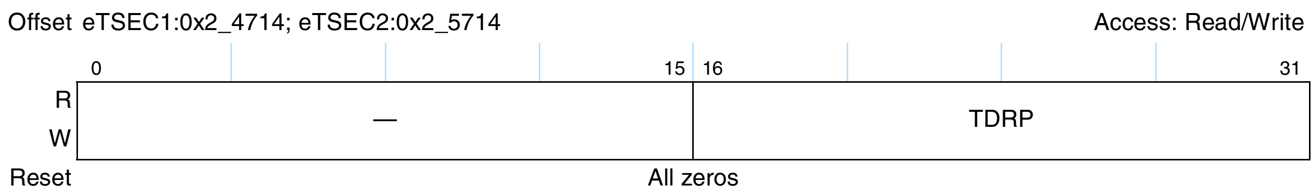


Figure 18-87. Transmit Drop Frame Counter Register Definition

Table 18-91 describes the fields of the TDRP register.

Table 18-91. TDRP Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	TDRP	Transmit drop frame counter. Increments each time a memory error or an underrun has occurred.

18.5.3.6.38 Transmit Jabber Frame Counter (TJBR)

Figure 18-88 describes the definition for the TJBR register.

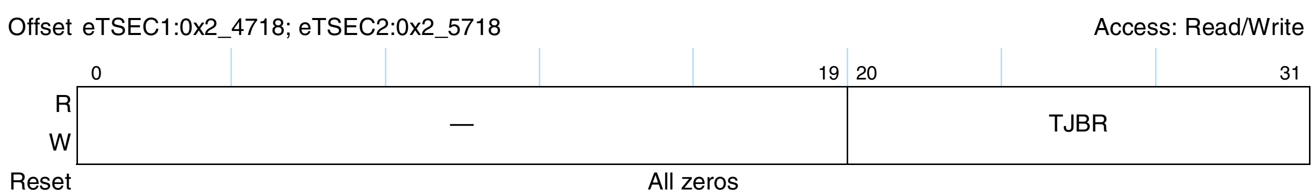


Figure 18-88. Transmit Jabber Frame Counter Register Definition

Table 18-92 describes the fields of the TJBR register.

Table 18-92. TJBR Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TJBR	Transmit jabber frame counter. Increments for each oversized transmitted frame with an incorrect FCS value.

18.5.3.6.39 Transmit FCS Error Counter (TFCS)

Figure 18-89 describes the definition for the TFCS register.

Offset eTSEC1:0x2_471C; eTSEC2:0x2_571C — Access: Read/Write

	0–19	20–31
R	—	TFCS
W		
Reset	All zeros	

Figure 18-89. Transmit FCS Error Counter Register Definition

Table 18-93 describes the fields of the TFCS register.

Table 18-93. TFCS Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TFCS	Transmit FCS error counter. Increments for every valid sized packet with an incorrect FCS value.

18.5.3.6.40 Transmit Control Frame Counter (TXCF)

Figure 18-90 describes the definition for the TXCF register.

Offset eTSEC1:0x2_4720; eTSEC2:0x2_5720 — Access: Read/Write

	0–19	20–31
R	—	TXCF
W		
Reset	All zeros	

Figure 18-90. Transmit Control Frame Counter Register Definition

Table 18-94 describes the fields of the TXCF register.

Table 18-94. TXCF Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TXCF	Transmit control frame counter. Increments for every control frame with valid CRC and of lengths 64 to 1518 (non VLAN) or 1522 (VLAN).

18.5.3.6.41 Transmit Oversize Frame Counter (TOVR)

Figure 18-91 describes the definition for the TOVR register.

Offset eTSEC1:0x2_4724; eTSEC2:0x2_5724 Access: Read/Write

	0–19	20–31
R W	—	TOVR
Reset	All zeros	

Figure 18-91. Transmit Oversized Frame Counter Register Definition

Table 18-95 describes the fields of the TOVR register.

Table 18-95. TOVR Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TOVR	Transmit oversize frame counter. Increments for each oversized transmitted frame with a correct FCS value.

18.5.3.6.42 Transmit Undersize Frame Counter (TUND)

Figure 18-92 describes the definition for the TUND register.

Offset eTSEC1:0x2_4728; eTSEC2:0x2_5728 Access: Read/Write

	0–19	20–31
R W	—	TUND
Reset	All zeros	

Figure 18-92. Transmit Undersize Frame Counter Register Definition

Table 18-96 describes the fields of the TUND register.

Table 18-96. TUND Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TUND	Transmit undersize frame counter. Increments for every frame less then 64 bytes, with a correct FCS value.

18.5.3.6.43 Transmit Fragment Counter (TFRG)

Figure 18-93 describes the definition for the TFRG register.

Offset eTSEC1:0x2_472C; eTSEC2:0x2_572C — Access: Read/Write

	0–19	20–31
R/W	—	TFRG
Reset	All zeros	

Figure 18-93. Transmit Fragment Counter Register Definition

Table 18-97 describes the fields of the TFRG register.

Table 18-97. TFRG Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TFRG	Transmit fragment counter. Increments for every frame less then 64 bytes, with an incorrect FCS value.

18.5.3.6.44 Carry Register 1 (CAR1)

Carry register bits are cleared on carry register writes when the respective bits are set. Figure 18-94 describes the definition for the CAR1 register.

Offset eTSEC1:0x2_4730; eTSEC2:0x2_5730 — Access: Read/Write

	0	1	2	3	4	5	6	7–13	14	15
R/W	C1 64	C1 127	C1 255	C1 511	C1 1K	C1 MAX	C1 MGV	—	C1 REJ	C1 RBY
Reset	All zeros									

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	C1 RPK	C1 RFC	C1 RMC	C1 RBC	C1 RXC	C1 RXP	C1 RXU	C1 RAL	C1 RFL	C1 RCD	C1 RCS	C1 RUN	C1 ROV	C1 RFR	C1 RJB	C1 RDR
Reset	All zeros															

Figure 18-94. Carry Register 1 (CAR1) Register Definition

Table 18-98 describes the fields of the CAR1 register.

Table 18-98. CAR1 Field Descriptions

Bits	Name	Description
0	C164	Carry register 1 TR64 counter carry bit
1	C1127	Carry register 1 TR127 counter carry bit
2	C1255	Carry register 1 TR255 counter carry bit
3	C1511	Carry register 1 TR511 counter carry bit
4	C11K	Carry register 1 TR1K counter carry bit
5	C1MAX	Carry register 1 TRMAX counter carry bit

Table 18-98. CAR1 Field Descriptions (continued)

Bits	Name	Description
6	C1MGV	Carry register 1 TRMGV counter carry bit
7–13	—	Reserved
14	C1REJ	Carry register 1 RREJ counter carry bit
15	C1RBY	Carry register 1 RBYT counter carry bit
16	C1RPK	Carry register 1 RPKT counter carry bit
17	C1RFC	Carry register 1 RFCS counter carry bit
18	C1RMC	Carry register 1 RMCA counter carry bit
19	C1RBC	Carry register 1 RBCA counter carry bit
20	C1RXC	Carry register 1 RXCF counter carry bit
21	C1RXP	Carry register 1 RXPF counter carry bit
22	C1RXU	Carry register 1 RXUO counter carry bit
23	C1RAL	Carry register 1 RALN counter carry bit
24	C1RFL	Carry register 1 RFLR counter carry bit
25	C1RCD	Carry register 1 RCDE counter carry bit
26	C1RCS	Carry register 1 RCSE counter carry bit
27	C1RUN	Carry register 1 RUND counter carry bit
28	C1ROV	Carry register 1 ROVR counter carry bit
29	C1RFR	Carry register 1 RFRG counter carry bit
30	C1RJB	Carry register 1 RJBR counter carry bit
31	C1RDR	Carry register 1 RDRP counter carry bit

18.5.3.6.45 Carry Register 2 (CAR2)

Figure 18-95 describes the definition for the CAR2 register.

Offset eTSEC1:0x2_4734; eTSEC2:0x2_5734 Access: Read/Write

	0–11	12	13	14	15
R/W	—	C2 TJB	C2 TFC	C2 TCF	C2 TOV
Reset	All zeros				

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	C2 TUN	C2 TFG	C2 TBY	C2 TPK	C2 TMC	C2 TBC	C2 TPF	C2 TDF	C2 TED	C2 TSC	C2 TMA	C2 TLC	C2 TXC	C2 TNC	—	C2 TDP
Reset	All zeros															

Figure 18-95. Carry Register 2 (CAR2) Register Definition

Carry register bits are cleared on carry register write when the respective bits are set. Table 18-99 describes the fields of the CAR2 register.

Table 18-99. CAR2 Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12	C2TJB	Carry register 2 TJBR counter carry bit
13	C2TFC	Carry register 2 TFCS counter carry bit
14	C2TCF	Carry register 2 TXCF counter carry bit
15	C2TOV	Carry register 2 TOVR counter carry bit
16	C2TUN	Carry register 2 TUND counter carry bit
17	C2TFG	Carry register 2 TFRG counter carry bit
18	C2TBY	Carry register 2 TBYT counter carry bit
19	C2TPK	Carry register 2 TPKT counter carry bit
20	C2TMC	Carry register 2 TMCA counter carry bit
21	C2TBC	Carry register 2 TBCA counter carry bit
22	C2TPF	Carry register 2 TXPF counter carry bit
23	C2TDF	Carry register 2 TDFR counter carry bit
24	C2TED	Carry register 2 TEDF counter carry bit
25	C2TSC	Carry register 2 TSCL counter carry bit
26	C2TMA	Carry register 2 TMCL counter carry bit
27	C2TLC	Carry register 2 TLCL counter carry bit
28	C2TXC	Carry register 2 TXCL counter carry bit
29	C2TNC	Carry register 2 TNCL counter carry bit
30	—	Reserved, should be cleared
31	C2TDP	Carry register 2 TDRP counter carry bit

18.5.3.6.46 Carry Mask Register 1 (CAM1)

While one of the below mask bits are cleared, the corresponding carry bit in CAR1 is allowed to cause interrupt indications in register IEVENT[MSR0]. These bits all default to a set state. Figure 18-96 describes the definition for the CAM1 register.

Offset eTSEC1:0x2_4738; eTSEC2:0x2_5738 — Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	M1 64	M1 127	M1 255	M1 511	M1 1K	M1 MAX	M1 MGV	—							M1 REJ	M1 RBY
Reset	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	M1 RPK	M1 RFC	M1 RMC	M1 RBC	M1 RXC	M1 RXP	M1 RXU	M1 RAL	M1 RFL	M1 RCD	M1 RCS	M1 RUN	M1 ROV	M1 RFR	M1 RJB	M1 RDR
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 18-96. Carry Mask Register 1 (CAM1) Register Definition

Table 18-100 describes the fields of the CAM1 register.

Table 18-100. CAM1 Field Descriptions

Bits	Name	Description
0	M164	Mask register 1 TR64 counter carry bit mask
1	M1127	Mask register 1 TR127 counter carry bit mask
2	M1255	Mask register 1 TR255 counter carry bit mask
3	M1511	Mask register 1 TR511 counter carry bit mask
4	M11k	Mask register 1 TR1K counter carry bit mask
5	M1MAX	Mask register 1 TRMAX counter carry bit mask
6	M1MGV	Mask register 1 TRMGV counter carry bit mask
7–13	—	Reserved
14	M1REJ	Mask register 1 RREJ counter carry bit mask
15	M1RBY	Mask register 1 RBYT counter carry bit mask
16	M1RPK	Mask register 1 RPKT counter carry bit mask
17	M1RFC	Mask register 1 RFCS counter carry bit mask
18	M1RMC	Mask register 1 RMCA counter carry bit mask
19	M1RBC	Mask register 1 RBCA counter carry bit mask
20	M1RXC	Mask register 1 RXCF counter carry bit mask
21	M1RXP	Mask register 1 RXPF counter carry bit mask
22	M1RXU	Mask register 1 RXUO counter carry bit mask
23	M1RAL	Mask register 1 RALN counter carry bit mask
24	M1RFL	Mask register 1 RFLR counter carry bit mask

Table 18-100. CAM1 Field Descriptions (continued)

Bits	Name	Description
25	M1RCD	Mask register 1 RCDE counter carry bit mask
26	M1RCS	Mask register 1 RCSE counter carry bit mask
27	M1RUN	Mask register 1 RUND counter carry bit mask
28	M1ROV	Mask register 1 ROVR counter carry bit mask
29	M1RFR	Mask register 1 RFRG counter carry bit mask
30	M1RJB	Mask register 1 RJBR counter carry bit mask
31	M1RDR	Mask register 1 RDRP counter carry bit mask

18.5.3.6.47 Carry Mask Register 2 (CAM2)

While one of the below mask bits are cleared, the corresponding carry bit in CAR2 is allowed to cause interrupt indications in register IEVENT[MSR0]. These bits default to a set state. Figure 18-97 describes the definition for the CAM2 register.

Offset eTSEC1:0x2_473C; eTSEC2:0x2_573C — Access: Read/Write

	0–11	12	13	14	15
R/W	—	M2 TJB	M2 TFC	M2 TCF	M2 TOV
Reset	0 0 0 0 0 0 0 0 0 0 0 0	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	M2 TUN	M2 TFG	M2 TBY	M2 TPK	M2 TMC	M2 TBC	M2 TPF	M2 TDF	M2 TED	M2 TSC	M2 TMA	M2 TLC	M2 TXC	M2 TNC	—	M2 TDP
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1

Figure 18-97. Carry Mask Register 2 (CAM2) Register Definition

Table 18-101 describes the fields of the CAM2 register.

Table 18-101. CAM2 Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12	M2TJB	Mask register 2 TJBR counter carry bit mask
13	M2TFC	Mask register 2 TFCS counter carry bit mask
14	M2TCF	Mask register 2 TXCF counter carry bit mask
15	M2TOV	Mask register 2 TOVR counter carry bit mask
16	M2TUN	Mask register 2 TUND counter carry bit mask
17	M2TFG	Mask register 2 TFRG counter carry bit mask
18	M2TBY	Mask register 2 TBYT counter carry bit mask
19	M2TPK	Mask register 2 TPKT counter carry bit mask

Table 18-101. CAM2 Field Descriptions (continued)

Bits	Name	Description
20	M2TMC	Mask register 2 TMCA counter carry bit mask
21	M2TBC	Mask register 2 TBCA counter carry bit mask
22	M2TPF	Mask register 2 TXPF counter carry bit mask
23	M2TDF	Mask register 2 TDFR counter carry bit mask
24	M2TED	Mask register 2 TEDF counter carry bit mask
25	M2TSC	Mask register 2 TSCL counter carry bit mask
26	M2TMA	Mask register 2 TMCL counter carry bit mask
27	M2TLC	Mask register 2 TLCL counter carry bit mask
28	M2TXC	Mask register 2 TXCL counter carry bit mask
29	M2TNC	Mask register 2 TNCL counter carry bit mask
30	—	Reserved
31	M2TDP	Mask register 2 TDRP counter carry bit mask

18.5.3.6.48 Receive Filer Rejected Packet Counter (RREJ)

Figure 18-98 describes the definition for the RREJ register.

Offset eTSEC1:0x2_4740; eTSEC2:0x2_5740 Access: Read/Write

	0–9	10–31
R W	—	RREJ
Reset	All zeros	

Figure 18-98. Receive Filer Rejected Packet Counter Register Definition

Table 18-66 describes the fields of the RREJ register.

Table 18-102. RREJ Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–31	RREJ	Receive filer rejected packet counter. Increments for each frame with valid CRC received, but rejected by the receive queue filer—either due to a matching rule that asserted the REJ flag or due to filing to a RxBD ring that was not enabled (see IEVENT[FIQ] error).

18.5.3.7 Hash Function Registers

This section provides detailed descriptions of the registers used for hash functions. All of the registers are 32 bits wide. The DA field of every received frame is processed through a 32-bit CRC generator (CRC-32 polynomial), and the 8 or 9 most significant bits of the CRC are mapped to a hash table entry. The user can enable a hash entry by setting its bit. A hash entry usually represents a set of addresses. A hash table hit occurs if the DA CRC result points to an enabled hash entry. Software may need to further filter the address in order to eliminate false-positive hits in the hash table.

If RCTRL[GHTX] = 0, the 8 most significant bits of the CRC are used as the hash table index. In this case, registers IGADDR0–IGADDR7 comprise a 256-entry hash table exclusively for individual (unicast) address matching, while registers GADDR0–GADDR7 comprise a 256-entry hash table for group (multicast) address matching. If RCTRL[GHTX] = 1, the group hash table is extended to all 512 entries, and the 9 most significant bits of the CRC are used as the hash table index. In this case, registers IGADDR0–IGADDR7 hold hash table entries 0–255 for group addresses, while registers GADDR0–GADDR7 hold entries 256–511 of the extended group hash table.

See Section 18.6.2.7.2, “Hash Table Algorithm,” for more information on the hash algorithm.

18.5.3.7.1 Individual/Group Address Registers 0–7 (IGADDR*n*)

The IGADDR*n* registers are written by the user. Together these registers represent, depending on RCTRL[GHTX], either the 256 entries of the individual address hash table, or the first 256 entries of the extended group address hash table used in the address recognition process. The user can enable a hash entry by setting the appropriate bit. A hash table hit occurs if the DA CRC-32 result points to an enabled hash entry.

Figure 18-99 describes the definition for the IGADDR*n* register.

Offset eTSEC1:0x2_4800+4×*n*; eTSEC2:0x2_5800+4×*n* — Access: Read/Write

	0 … 31
R W	IGADDR*n*
Reset	All zeros

Figure 18-99. IGADDR*n* Register Definition

Table 18-104 describes the fields of the IGADDR*n* register.

Table 18-103. IGADDR*n* Field Descriptions

Bits	Name	Description
0–31	IGADDR*n*	Represents the 32-bit value associated with the corresponding register. When RCTRL[GHTX] = 0, IGADDR0 contains entries 0–31 of the 256-entry individual hash table and IGADDR7 represents entries 224–255. When RCTRL[GHTX] = 1, IGADDR0 contains entries 0–31 of the 512-entry extended group hash table and IGADDR7 represents entries 224–255.

18.5.3.7.2 Group Address Registers 0–7 (GADDR*n*)

The GADDR*n* registers are written by the user. Together these registers represent, depending on RCTRL[GHTX], either the 256 entries of the group address hash table, or the last 256 entries of the extended group address hash table used in the address recognition process. The user can enable a hash entry by setting the appropriate bit. A hash table hit occurs if the DA CRC result points to an enabled hash entry. Figure 18-100 describes the definition for the GADDR*n* register.

Offset eTSEC1:0x2_4880+4×*n*; eTSEC2:0x2_5880+4×*n* Access: Read/Write

	0–31
R W	GADDR*n*
Reset	All zeros

Figure 18-100. GADDR*n* Register Definition

Table 18-104 describes the fields of the GADDR*n* register.

Table 18-104. GADDR*n* Field Descriptions

Bits	Name	Description
0–31	GADDR*n*	Represents the 32-bit value associated with the corresponding register. When RCTRL[GHTX] = 0, GADDR0 contains entries 0–31 of the 256-entry group hash table and GADDR7 represents entries 224–255. When RCTRL[GHTX] = 1, GADDR0 contains entries 256–287 of the 512-entry extended group hash table and GADDR7 represents entries 480–511.

18.5.3.8 DMA Attribute Registers

This section describes the two eTSEC DMA attribute registers.

18.5.3.8.1 Attribute Register (ATTR)

The attribute register defines memory access attributes and transaction types used to access buffer descriptors, to write receive data, and to read transmit data. Snoop enable attributes may be set for reading buffer descriptors and for reading transmit data.

Figure 18-101 describes the definition for the ATTR register.

Offset eTSEC1:0x2_4BF8; eTSEC2:0x2_5BF8; Access: Read/Write

	0–23	24	25	26–31
R W	—	RDSEN	RBDSEN	—
Reset	All zeros			

Figure 18-101. ATTR Register Definition

Table 18-105 describes the fields of the ATTR register.

Table 18-105. ATTR Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24	RDSEN	Rx data snoop enable. 0 Disables snooping of all receive frames data to memory. 1 Enables snooping of all receive frames data to memory.

Table 18-105. ATTR Field Descriptions (continued)

Bits	Name	Description
25	RBDSEN	RxBD snoop enable. 0 Disables snooping of all receive BD memory accesses. 1 Enables snooping of all receive BD memory accesses.
26–31	—	Reserved

18.5.3.9 Hardware Assist for IEEE1588 Compliant Timestamping

IEEE 1588 compliant timestamping on this device is accomplished using the per-port transmit timestamping registers within each Ethernet controller memory space (See Section 18.5.3.2.10, “Transmit Time Stamp Identification Register (TMR_TXTS1–2_ID),” and Section 18.5.3.2.11, “Transmit Time Stamp Register (TMR_TXTS1–2_H/L).”) in conjunction with the following common registers, which are located within the memory space for eTSEC1. Because the common 1588 timestamping registers exist within the eTSEC1 memory space, the eTSEC1 controller must remain enabled in order to use 1588 timestamping for any Ethernet port.

18.5.3.9.1 Timer Control Register (TMR_CTRL)

This register is used to reset, configure, and initialize the eTSEC precision timer clock. The control of all timer function is performed via programming eTSEC1.The register in eTSEC1 is shared for all eTSECs. Figure 18-7 describes the definition for the TMR_CTRL register.

Register fields not described below are reserved.

Offset eTSEC1:0x2_4E00 Access: Mixed

	0	1	2	3	4	5	6									15
R W	ALM1P	ALM2P	—	FS	PP1L	PP2L	TCLK_PERIOD									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	RTPE	FRD			ESFDP	ESFDE	ETEP2	ETEP1	COPH	CIPH	TMSR		BYP	TE	CKSEL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 18-102. TMR_CTRL Register Definition

Table 18-106 describes the fields of the TMR_CTRL register. Register fields not described below are reserved.

Table 18-106. TMR_CTRL Register Field Descriptions

Bits	Name	Description
0	ALM1P	Alarm1 output polarity 0 active high output 1 active low output
1	ALM2P	Alarm2 output polarity 0 active high output 1 active low output
3	FS	FIPER start indication 0 Fiper is enabled through timer enable 1 Fiper is enabled through timer enable and alarm indication.
4	PP1L	Fiper1 pulse loopback mode enabled. 0 Trigger1 input is based upon normal external trigger input. 1 Fiper1 pulse is looped back into Trigger1 input.
5	PP2L	Fiper2 pulse loopback mode enabled. 0 Trigger2 input is based upon normal external trigger input. 1 Fiper2 pulse is looped back into Trigger2 input.
6–15	TCLK_ PERIOD	1588 timer reference clock period. The timer clock counter will increment by TCLK_PERIOD every time the accumulator register overflows. This clock period must be larger than the clock period of the timer reference clock. For applications where user does not want the clock period to be added, they can program this field to 1 to count the clock ticks. This field defaulted to 1 to count overflow ticks. For nanosecond granularity on 1588 timer counter rate, the TCLK_PERIOD should be calculated using the following equation: TCLK_PERIOD = 10^9/Nominal_Frequency
16	RTPE	Record Tx Time-Stamp to PAL Enable. When set, and FCB[PTP] is set, the 8-byte time-stamp for the packet is written to the PAL located in external memory location at an offset of 16 bytes from the start of the Data Buffer Pointer of the first TxBD. For guidelines on using the RTPE bit, refer to Section 18.6.5.5, "Time-Stamp Insertion on Transmit Packets."
17	FRD	FIPER Realignment Disable 0 Fiper Realignment is enabled. 1 Fiper Realignment is disabled.
20	ESFDP	External Tx/Rx SFD Polarity. 0 Time stamp on rising edge of external SFD indication. 1 Time stamp on falling edge of external SFD indication.
21	ESFDE	External Tx/Rx SFD Enable. 0 Time stamp PTP TX frame based on MAC's SFD indication. 1 Time stamp PTP TX frame based on external SFD indication from PHY.
22	ETEP2	External trigger 2 edge polarity 0 Time stamp on the rising edge of the external trigger 1 Time stamp on the falling edge of the external trigger
23	ETEP1	External trigger 1 edge polarity 0 time stamp on the rising edge of the external trigger 1 time stamp on the falling edge of the external trigger
24	COPH	Generated clock (TSEC_TMR_GCLK) output phase. 0 non-inverted divided clock is output 1 inverted divided clock is output

Table 18-106. TMR_CTRL Register Field Descriptions (continued)

Bits	Name	Description
25	CIPH	External oscillator input clock phase. 0 non-inverted frequency tuned timer input clock 1 inverted frequency tuned timer input clock (NOTE: this setting is reserved if CKSEL=01.)
26	TMSR	Timer soft reset. When enabled, it resets all the timer registers and state machines. 0 normal operation 1 place entire timer in reset except control and config registers NOTE: Prior to initiating timer reset (setting TMSR), must gracefully stop receiver (See MACCFG1[RX_EN] description). User programmable registers are not reset by the soft reset e.g. TMR_CTRL, TMR_TEMASK, TMR_PEMASK, TMR_ADD, TMR_PRSC, TMROFF_H/L, TMR_ALARMn, and TMR_FIPERn.
28	BYP	Bypass drift compensated clock 0 64-bit clock counter is incremented on the accumulator overflow 1 64-bit clock counter is directly driven from the external oscillator ignoring accumulator overflow
29	TE	1588 timer enable. If not enabled, all the timer registers and state machines are disabled. 0 timer not enabled 1 timer enabled and resume normal operation
30–31	CKSEL	1588 Timer reference clock source select. 00 External high precision timer reference clock (TSEC_TMR_CLK) 01 eTSEC system clock 10 eTSEC1 transmit clock 11 RTC clock input Note that the 1588 reference clock must be no slower than 1/7 the Rx_clk frequency. The default clock select is eTSEC system clock, which is always active when eTSEC is enabled. The user must ensure the corresponding clock source is active before changing the 1588 refclk selection to external reference, RTC, or TX clock. Selecting an inactive 1588 reference clock may cause boundedly undefined behavior in the ethernet controller and on accesses to the 1588 registers.

18.5.3.9.2 Timer Event Register (TMR_TEVENT)

The eTSEC precision timer implementation can generate additional interrupts that are independent of the frame based events that controlled via IEVENT. The timer interrupts are not affected by any interrupt coalescing that may be specified in TXIC/RXIC. Software may poll this register at any time to check for pending interrupts. If an event occurs and its corresponding enable bit is set in the event mask register (TEMASK), the event also causes a hardware interrupt at the PIC. A bit in the timer event register is cleared by writing a 1 to that bit position. Figure 18-4 describes the definition for the TMR_TEVENT register.

Offset eTSEC1:0x2_4E04 Access: W1C

	0–5	6	7	8–13	14	15
R W	—	ETS2	ETS1	—	ALM2	ALM1
Reset	All zeros					

	16–23	24	25	26	27–31
R W	—	PP1	PP2	PP3	—
Reset	All zeros				

Figure 18-103. TMR_TEVENT Register Definition

Table 18-107 describes the fields of the TMR_TEVENT register fields for the timer.

Table 18-107. TMR_TEVENT Register Field Descriptions

Bits	Name	Description
0–6	—	Reserved
6	ETS2	External trigger 2 timestamp sampled 0 external trigger timestamp not sampled 1 external trigger timestamp sampled
7	ETS1	External trigger 1 timestamp sampled 0 external trigger timestamp not sampled 1 external trigger timestamp sampled
8–13	—	Reserved
14	ALM2	Current time equaled alarm time register 2 0 alarm time has not be reached yet 1 alarm time has been reached
15	ALM1	Current time equaled alarm time register 1 0 alarm time has not be reached yet 1 alarm time has been reached
16–23	—	Reserved
24	PP1	Indicates that a periodic pulse has been generated based on FIPER1 register. 0 periodic pulse not generated 1 periodic pulse generated
25	PP2	Indicates that a periodic pulse has been generated based on FIPER2 register. 0 periodic pulse not generated 1 periodic pulse generated
26	PP3	Indicates that a periodic pulse has been generated based on FIPER3 register. 0 periodic pulse not generated 1 periodic pulse generated
27–31	—	Reserved

18.5.3.9.3 Timer Event Mask Register (TMR_TEMASK)

Timer event mask register. The event mask register provides control over which possible interrupt events in the TMR_TEVENT register are permitted to participate in generating hardware interrupts to the PIC. All implemented bits in this register are R/W and cleared upon a hardware reset. Figure 18-108 describes the definition for the TMR_TEMASK register.

Offset eTSEC1:0x2_4E08 — Access: Read/Write

	0–5	6	7	8–13	14	15
R/W	—	ETS2EN	ETS1EN	—	ALM2EN	ALM1EN
Reset	All zeros					

	16–23	24	25	26–31
R/W	—	PP1EN	PP2EN	—
Reset	All zeros			

Table 18-108. TMR_TEMASK Register Definition

Table 18-109 describes the fields of the TMR_TEMASK register fields for the timer.

Table 18-109. TMR_TEMASK Register Field Descriptions

Bits	Name	Description
0–5	—	Reserved
6	ETS2EN	External trigger 2 timestamp sample event enable
7	ETS1EN	External trigger 1 timestamp sample event enable
8–13	—	Reserved
14	ALM2EN	Timer ALM1 event enable
15	ALM1EN	Timer ALM2 event enable
16–23	—	Reserved
24	PP1EN	Periodic pulse event 1 enable
25	PP2EN	Periodic pulse event 2 enable
26–31	—	Reserved

18.5.3.9.4 Timer PTP Packet Event Register (TMR_PEVENT)

The eTSEC precision timer logic can generate interrupts upon the capture of a timestamp due to either transmission or reception of a frame. If an event occurs and its corresponding enable bit is set in the event mask register (PEMASK), the event also causes a hardware interrupt at the PIC. A bit in the timer event register is cleared by writing a 1 to that bit position. Figure 18-104 describes the definition for the TMR_PEVENT register.

Offset eTSEC1:0x2_4E0C | Access: Read/Write

	0–15
R	—
W	
Reset	All zeros

	16–21	22	23	24–30	31
R	—	TXP2	TXP1	—	RXP
W					
Reset	All zeros				

Figure 18-104. TMR_PEVENT Register Definition

Table 18-110 describes the fields of the TMR_PEVENT register fields for the timer.

Table 18-110. TMR_PEVENT Register Field Descriptions

Bits	Name	Description
0–21	—	Reserved
22	TXP2	Indicates that a PTP frame has been transmitted and its timestamp is stored in TXTS2 register. 0 PTP packet not transmitted 1 PTP packet has been transmitted
23	TXP1	Indicates that a PTP frame has been transmitted and its timestamp is stored in TXTS1 register. 0 PTP packet not transmitted 1 PTP packet has been transmitted
24–30	—	Reserved
31	RXP	Indicates that a PTP frame has been received 0 PTP packet not received 1 PTP packet has been received

18.5.3.9.5 Timer Event Mask Register (TMR_PEMASK)

Timer event mask register. The event mask register provides control over which possible interrupt events in the TMR_PEVENT register are permitted to participate in generating hardware interrupts to the PIC. All implemented bits in this register are R/W and cleared upon a hardware reset. Figure 18-105 describes the definition for the TMR_PEMASK register.

Offset eTSEC1:0x2_4E10 | Access: Read/Write

	0–15
R	—
W	
Reset	All zeros

	16–21	22	23	24–30	31
R	—	TXP2EN	TXP1EN	—	RXPEN
W					
Reset	All zeros				

Figure 18-105. TMR_PEMASK Register Definition

Table 18-111 describes the fields of the TMR_PEMASK register fields for the timer.

Table 18-111. TMR_PEMASK Register Field Descriptions

Bits	Name	Description
0–21	—	Reserved
22	TXP2EN	Transmit PTP packet event 2 enable
23	TXP1EN	Transmit PTP packet event 1 enable
24–30	—	Reserved
31	RXPEN	Receive PTP packet event enable

18.5.3.9.6 Timer Status Register (TMR_STAT)

This register requires the eTSEC filer to be enabled (via RCTRL[FILREN]). When eTSEC generates an interrupt based on the timestamp event for a received packet, the queue ID which the incoming packet will be sent to is captured in this register. This register update is synchronized with the RXF interrupt of the corresponding received packet. Writing 1 to any bit of this register clears it. Figure 18-112 describes the definition for the TMR_STAT register.

Offset eTSEC1:0x2_4E14 Access: Mixed

	0–25	26–31
R / W	—	STAT_VEC
Reset	All zeros	

Table 18-112. TMR_STAT Register Definition

Table 18-113 describes the fields of the TMR_STAT register.

Table 18-113. TMR_STAT Register Field Descriptions

Bits	Name	Description
0–25	—	Reserved
26–31	STAT_VEC	Timer general purpose status vector. It will store the 6-bit queue number generated by the filer. User to decode this status vector. For example, user can encode received PTP packet message types (Sync, Delay_req, Follow_up, Delay_resp, Management) in the filer virtual queue field.

18.5.3.9.7 Timer Counter Register (TMR_CNT_H/L)

The timer register (TMR_CNT_H/L) represents accurate time in terms clock ticks or in nano-seconds. Writes to these registers will override the previous time. The register in eTSEC1 is shared for all eTSECs. This is a read/write register. Figure 18-106 describes the definition for the TMR_CNT_H/L register.

Offset eTSEC1:0x2_4E18 (H); 0x2_4E1C (L) Access: Read/Write

	0–31	32–63
R W	TMR_CNT_H	TMR_CNT_L
Reset	All zeros	

Figure 18-106. TMR_CNT_H Register Definition

Table 18-114 describes the fields of the TMR_CNT_H/L register.

Table 18-114. TMR_CNT_H/L Register Field Descriptions

Bits	Name	Description
0–63	TMR_CNT_H/L	Value of the current time counter. Current time is calculated by adding TMROFF_H/L with the TMR_CNT_H/L counter. This register can be written through the register writes.Writes to the TMR_CNT_L register copies the written value into the shadow TMR_CNT_L register. Writes to the TMR_CNT_H register copies the values written into the shadow TMR_CNT_H register. Contents of the shadow registers are copied into the TMR_CNT_L and TMR_CNT_H registers following a write into the TMR_CNT_H register. Writes to these registers have precedence over the timer increment. The user must write to TMR_CNT_L register first. Reads from the TMR_CNT_L register copies the entire 64-bit clock time of the read enable into the TMR_CNT_H/L shadow registers. Read instruction from the TMR_CNT_H register reads the value stored in the TMR_CNT_H shadow register. The user must read the TMR_CNT_L register first to get correct 64-bit TMR_CNT_H/L counter values.

18.5.3.9.8 Timer Drift Compensation Addend Register (TMR_ADD)

Timer drift compensation addend register (TMR_ADD) is used to hold timer frequency compensation value (FreqCompensationValue). The nominal frequency of the clock counter is determined by the FreqDivRatio and the clock frequency (FreqClock). This register is programmed with 2^{32}/FreqDivRatio. Frequency division ratio (FreqDivRatio) is the ratio between the frequency of the oscillator (TimerOsc) and the desired clock frequency (NominalFreq). FreqDivRatio is a design constant chosen to be greater than 1.0001. The ADDEND value is added to the 32-bit accumulator register at every rising edge of the oscillator clock (TimerOsc). The clock counter is incremented at every carry pulse of the accumulator. Only one of this register is required for the entire group of eTSECs. Figure 18-107 describes the definition of the TMR_ADD register.

Offset eTSEC1:0x2_4E20 Access: Read/Write

	0–31
R W	ADDEND
Reset	All zeros

Figure 18-107. TMR_ADD Register Definition

Table 18-115 describes the fields of the TMR_ADD register fields for the timer.

Table 18-115. TMR_ADD Register Field Descriptions

Bits	Name	Description
0–31	ADDEND	Timer drift compensation addend register value. It is programmed with a value of 2^32/FreqDivRatio. For example, TimerOsc = 50 MHz NominalFreq = 40 MHz FreqDivRatio = 1.25 ADDEND = ceil(2^32/1.25) = 0xCCCC_CCCD

18.5.3.9.9 Timer Accumulator Register (TMR_ACC)

Timer accumulator register accumulates the value of the addend register into it. An overflow pulse of the accumulator is used to increment the timer clock by TMR_CTRL[TCLK_PERIOD]. This register is read only in normal operation. The register in eTSEC1 is shared for all eTSECs. Figure 18-108 describes the definition of the TMR_ACC register.

Offset eTSEC1:0x2_4E24 Access: Read/Write

	0–31
R W	TMR_ACC
Reset	All zeros

Figure 18-108. TMR_ACC Register Definition

Table 18-116 describes the fields of the TMR_ACC register.

Table 18-116. TMR_ACC Register Field Descriptions

Bits	Name	Description
0–31	TMR_ACC	32-bit timer accumulator register

18.5.3.9.10 Timer Prescale Register (TMR_PRSC)

Timer generated output clock prescale register. It is used to adjust output clock frequency that is put onto the 1588 clock output signal. The register in eTSEC1 is shared for all eTSECs. Figure 18-109 describes the definition for the TMR_PRSC register.

Offset eTSEC1:0x2_4E28 Access: Read/Write

	0–15	16–31
R W	—	PRSC_OCK
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

Figure 18-109. TMR_PRSC Register Definition

Table 18-117 describes the fields of the TMR_PRSC register.

Table 18-117. TMR_PRSC Register Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	PRSC_OCK	Output clock division/prescale factor. Output clock is generated by dividing the timer input clock by this number. Programmed value in this field must be greater than 1. Any value less than 1 is treated as 2.

18.5.3.9.11 Timer Offset Register (TMROFF_H/L)

The timer offset register is used to provide current time by adding its value to the clock counter. Figure 18-110 describes the definition of the TMROFF_H/L register.

NOTE

All TMROFF_H registers in a device should be set to the same value, and all TMROFF_L registers in a device should be set to the same value. Otherwise, the precision time protocol may not work.

Offset eTSEC1:0x2_4E30 (H); 0x2_4E34 (L) Access: Read/Write

	0–31	32–63
R W	TMROFF_H	TMROFF_L
Reset	All zeros	

Figure 18-110. TMROFF_H/L Register Definition

Table 18-118 describes the fields of the TMROFF_H/L register.

Table 18-118. TMROFF_H/L Register Field Descriptions

Bits	Name	Description
0–63	TMROFF_H/L	Offset value of the clock counter. Current time in is calculated by adding TMROFF_H/L with the timer's counter TMR_CNT_H/L register.

18.5.3.9.12 Alarm Time Comparator Register (TMR_ALARM1–2_H/L)

Alarm time comparator register (TMR_ALARM*n*_H/L). This register holds alarm time for comparison with the current time counter. There are two of these registers for eTSEC1 which are shared amongst all eTSECs. Figure 18-111 describes the definition for the TMR_ALARM*n*_H/L register.

Offset eTSEC1:0x2_4E40+8×*n* Access: Read/Write

	0–31	32–63
R W	ALARM_H	ALARM_L
Reset	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 18-111. TMR_ALARM1-2_H/L Register Definition

Table 18-119 describes the fields of the TMR_ALARM*n*_H/L register.

Table 18-119. TMR_ALARM*n*_H/L Register Field Descriptions

Bits	Name	Description
0–63	ALARM_H/L	Alarm time comparator register. The corresponding alarm event in TMR_TEVENT is set when the current time counter becomes equal to or greater than the alarm time compare value in TMR_ALARM*n*_L/H. Writing the TMR_ALARM*n*_L register deactivates the alarm event after it has fired. Writing the TMR_ALARM*n*_L followed by the TMR_ALARM*n*_H register rearms the alarm function with the new compare value. The value programmed in this register must be an integer multiple of TMR_CTRL[TCLK_PERIOD] in order to get correct result. This register is reset to all ones to avoid false alarm after reset. In FS mode the alarm trigger is used as an indication to the fiper start down counting. Only alarm 1 supports this mode. In FS mode, alarm polarity bit should be configured to 0 (rising edge).

18.5.3.9.13 Timer Fixed Interval Period Register (TMR_FIPER1–3)

Timer fixed interval period pulse generator register. It is used to generate periodic pulses. This register is reset with 0xFFFF_FFFF to prevent any false pulse upon initialization. The down count register loads the value programmed in the fixed period interval (FIPER). FIPER register must be programmed before the timer is enabled. At every tick of the timer accumulator overflow, the counter decrements by the value of TMR_CTRL[TCLK_PERIOD]. It generates a pulse when the down counter value reaches zero. It reloads the down counter in the cycle following a pulse.

Should a user wish to use the TMR_FIPER1 register to generate a 1 PPS event, the following setup should be used:

- Program TMR_FIPER1 to a value that will generate a pulse every second,
- Program TMR_ALARM1 to the correct time for the first PPS event
- Enable the timer

The eTSEC will then wait for TMR_ALARM1 to expire before enabling the count down of TMR_FIPER1. The end result will be that TMR_FIPER1 will pulse every second after the original timer ALARM1 expired.

Note:

In the case where the PPS signals are required to be phased aligned to the prescale output clock, the alarm value should be configured to **1 clock period less** than the wanted value.

In order to keep tracking the prescale output clock, each time before enabling the FIPER, the user must reset the FIPER by writing a new value to the register. The ratio between the prescale register value and the FIPER value should be devisable by the clk period.

FIPER_VALUE = (prescale_value × tclk_per × N) – tclk_per

For example:

prescale = 9

clock period = 10

The FIPER can get the following values: 80, 170, 260 ....

The three registers in eTSEC1 are shared for all eTSECs. Figure 18-112 describes the definition for the TMR_FIPER register.

Offset eTSEC1:0x2_4E80+4*n Access: Read/Write

	0 … 31
R	FIPER
W	
Reset	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 18-112. TMR_FIPER*n* Register Definition

Table 18-120 describes the fields of the TMR_FIPER register.

Table 18-120. TMR_FIPER Register Field Descriptions

Bits	Name	Description
0–31	FIPER	Fixed interval pulse period register. This field must be programmed to an integer multiple of TMR_CTRL[TCLK_PERIOD] value to ensure a period pulse being generated correctly.

18.5.3.9.14 External Trigger Stamp Register (TMR_ETTS1–2_H/L)

General purpose external trigger -stamp register (TMR_ETTS*n*_H/L). This register holds time at the programmable edge of the external trigger. The registers in eTSEC1 are shared for all eTSECs. This register is read only in normal operation. Figure 18-113 describes the definition for the TMR_ETTS*n*_H/L register.

Offset eTSEC1:0x2_4EA0+8×n Access: Read/Write

	0 … 31	32 … 63
R	ETTS_H	ETTS_L
W		
Reset	All zeros	

Figure 18-113. TMR_ETTS1-2_H/L Register Definition

Table 18-121 describes the fields of the TMR_ETTS*n*_H/L register.

Table 18-121. TMR_ETTS1-2_H Register Field Descriptions

Bits	Name	Description
0–63	ETTS_H/L	Time stamp field at the programmable edge of the external trigger.

18.5.4 Ten-Bit Interface (TBI)

This section describes the ten-bit interface (TBI), reduced ten-bit interface (RTBI), and the TBI/RTBI MII set of registers. TBI and RTBI operate in the same manner (the only difference is that RTBI has reduced I/O signalling).

18.5.4.1 TBI Transmit Process

The eTSEC's TBI implements the transmit portion of the physical coding sublayer as found in Clause 36 of IEEE 802.3z. In SerDes mode, packets conveyed across the GMII are encapsulated and encoded into 10-bit symbols and output to the SerDes. In GMII mode, the GMII signals are passed through to the attached GMII PHY.

18.5.4.1.1 Packet Encapsulation

If TX_EN is de-asserted the eTSEC outputs an idle stream. If TX_EN is asserted, a Start_of_Packet symbol is output. This symbol replaces the first byte of the preamble field. All other bytes of the packet pass through an 8B10B encoding module. After the last byte of the FCS field is signaled through the GMII, the MAC de-asserts TX_EN. The eTSEC then outputs an End_of_Packet symbol. Then, depending on the position of the End_of_Packet symbols (being in either an odd or even position) the eTSEC outputs one or two Carrier_Extend symbols. Following the last Carrier_Extend symbol, the eTSEC resumes sending idle codes. If, during a packet, the eTSEC wishes to mark a byte invalid, TX_ER is asserted. The eTSEC, upon detection of TX_ER, substitutes the data symbol for an Error_Propagation symbol.

18.5.4.1.2 8B10B Encoding

Every eight-bit data octet has two (not necessarily different) ten-bit symbols associated with it. Depending on the running disparity (the cumulative difference of ones and zeroes) the eTSEC module chooses the appropriate symbol.

Special encapsulation symbols are called ordered_sets. Ordered_sets are comprised of one to four ten-bit symbols. Ordered_sets can be found in clause 36 of the IEEE 802.3z specification.

18.5.4.1.3 Preamble Shortening

Because the idle ordered_set comprises two symbols and begins on an even symbols boundary, packets can only begin on an even boundary. However, the GMII has no such restriction and may signal TX_EN on an odd boundary. If this happens, the eTSEC delays the Start_of_Packet symbol, effectively ignoring the first byte of preamble; thus, a seven octet preamble becomes six octets on the Ten-Bit Interface.

18.5.4.2 TBI Receive Process

The eTSEC's TBI Implements the receive portion of the physical coding sublayer as found in Clause 36 of IEEE 802.3z specification. The Receive portion includes the Synchronization state machine. In SerDes mode, the eTSEC first attempts to acquire synchronization on the link by examining received symbols. Once synchronization is acquired, received packets are decoded and sent across the Receive GMII interface. In GMII mode, the GMII signals are passed through to the MAC.

18.5.4.2.1 Synchronization

The eTSEC examines received symbols looking for the seven bit 'comma' string embedded in some special symbols. Both the idle ordered_set and the Configuration ordered_set contain a symbol which has the comma. Once a certain number of codes with comma are detected, the eTSEC is considered to have acquired synchronization.

18.5.4.2.2 Auto-Negotiation for 1000BASE-X

Once synchronization is acquired, ordered_sets are decoded. If Configuration ordered_sets are received, the eTSEC decodes the two octet data field and the sixteen-bit Configuration data is stored and used to Auto-Negotiate with the link partner. in the Receive Configuration Register (RXCR[15:0]) an internal register used to receive all the link partners informations and used to compare to local ability during negotiation. Not visible to user. If, during Auto-Negotiation an invalid symbol is detected, Auto-Negotiation re-starts. After Auto-Negotiation is completed the TBI MII Status Register SR[AN done] in set. In this mode, packets may be received from the link partner.

18.5.4.3 TBI MII Set Register Descriptions

This section describes the TBI MII registers. All of the TBI registers are 16 bits wide. The TBI registers are accessed at the offset of the TBI physical address. The eTSEC's TBI physical address is stored in the TBIPA register. Writing to the TBI registers is performed in a way similar to writing to an external PHY, using the MII management interface. Using TBIPA in place of the PHY address, in the MIIMADD[PHY Address] field, and setting the MIIMADD[Register Address] to the appropriate address offset that corresponds to the register that one wants to read or write (see Table 18-122), the user can read (set MIIMCOM[read cycle]) or write (writing to MIIMCON[PHY control]) to the TBI block. Refer to the TBI physical address register in Section 18.5.3.1, "eTSEC General Control and Status Registers," and the TBI MII register set in Table 18-122. Notice that jitter diagnostics and TBI control are not IEEE 802.3 required registers and are only used for test and control of the eTSEC TBI block. The TBI's TBI control register (TBI) is for configuring the eTSEC ten-bit interface block. However, because this TBI block has an MII management interface (just like any other PHY), it has an IEEE 802.3 register called the control register (CR).

Table 18-122. TBI MII Register Set

Offset Address	Name	Access	Size	Section/page
TEN-BIT INTERFACE (TBI) REGISTERS				18.5.4/18-119
0x00	Control (CR)	R/W[1]	16 bits	18.5.4.3.1/18-122
0x01	Status (SR)	R, LH, LL	16 bits	18.5.4.3.2/18-123
0x02–0x03	Reserved	R	2 bytes	—
0x04	AN advertisement (ANA)	RW, R	16 bits	18.5.4.3.2/18-123
0x05	AN link partner base page ability (ANLPBPA)	R	16 bits	18.5.4.3.4/18-126
0x06	AN expansion (ANEX)	R, LH	16 bits	18.5.4.3.5/18-127
0x07	AN next page transmit (ANNPT)	R/W, R	16 bits	18.5.4.3.6/18-127
0x08	AN link partner ability next page (ANLPANP)	R	16 bits	18.5.4.3.7/18-128
0x0F	Extended status (EXST)	R	16 bits	18.5.4.3.8/18-129
0x10	Jitter diagnostics (JD)	R/W	16 bits	18.5.4.3.9/18-130
0x11	TBI control (TBICON)	R/W	16 bits	18.5.4.3.10/18-131

[1] R = Read-only, WO = Write Only, R/W = Read and Write, LH = Latches High, LL = Latches Low, SC = Self-clearing,

18.5.4.3.1 Control Register (CR)

Figure 18-114 describes the definition for the CR register.

Offset 0x00 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R / W	PHY Reset	—	Speed[0]	AN Enable	—		Reset AN	Full Duplex	—	Speed[1]	—					
Reset	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0

Figure 18-114. Control Register Definition

Table 18-123 describes the fields of the CR register.

Table 18-123. CR Field Descriptions

Bits	Name	Description
0	PHY Reset	PHY reset. This bit is cleared by default. This bit is self-clearing. 0 Normal operation. 1 The internal state of the TBI is reset. This in turn may change the state of the TBI link partner.
1	—	Reserved
2	Speed[0]	Speed selection. This bit defaults to a cleared state and should always be cleared, which corresponds to 1000 Mbps speed.Setting this field controls the speed at which the TBI operates. The table for Speed[1] provides the appropriate encoding. Its default is bit[2] = ‘0’; bit[9] = ’1’.
3	AN Enable	Auto-negotiation enable. This bit is set by default. 0 The values programmed in bits 2, 7 and 9 determine the operating condition of the link. 1 Auto-negotiation process enabled.
4–5	—	Reserved
6	Reset AN	Reset auto-negotiation. This bit is cleared by default and is self-clearing. 0 Normal operation. 1 The auto-negotiation process restarts. This action is only available if auto-negotiation is enabled.
7	Full Duplex	Duplex mode. This bit is set by default. 0 Reserved. 1 Full-duplex operation.
8	—	Reserved, should be cleared.
9	Speed[1]	Speed selection. This bit defaults to a set state and should always be set, which corresponds to 1000 Mbps speed.Setting this field controls the speed at which the TBI operates. The following table provides the appropriate encoding. Its default is bit[2] = ‘0’; bit[9] = ’1’. Maximum Operating Speed / Bit 2 / Bit 9: Reserved / 0 / 0 Reserved / 1 / 0 1000 Mbps / 0 / 1 Reserved / 1 / 1
10–15	—	Reserved

18.5.4.3.2 Status Register (SR)

Figure 18-115 describes the definition for the SR register.

Offset 0x01 Access: Read only

	0–6	7	8	9	10	11	12	13	14	15
R	—	Extend Status	—	No Pre	AN Done	Remote Fault	AN Ability	Link Status	—	Extend Ability
W										
Reset	0 0 0 0 0 0 0	1	0	1	0	0	1	0	0	1

Figure 18-115. Status Register Definition

Table 18-124 describes the fields of the SR register.

Table 18-124. SR Descriptions

Bits	Name	Description
0–6	—	Reserved, should be cleared.
7	Extend Status	This bit indicates that PHY status information is also contained in the Register 15, Extended Status Register. Returns 1 on read. This bit is read-only.
8	—	Reserved, should be cleared.
9	No Pre	MF preamble suppression enable. This bit indicates whether or not the PHY is capable of handling MII management frames without the 32-bit preamble field. Returns 1, indicating support for suppressed preamble MII management frames. This bit is read-only.
10	AN Done	Auto-negotiation complete. This bit is read-only and is cleared by default. 0 Either the auto-negotiation process is underway or the auto-negotiation function is disabled. 1 The auto-negotiation process has completed.
11	Remote Fault	Remote fault. This bit is read-only and is cleared by default. Each read of the status register clears this bit. 0 Normal operation. 1 A remote fault condition was detected. This bit latches high in order for software to detect the condition.
12	AN Ability	Auto-negotiation ability. While read as set, this bit indicates that the PHY has the ability to perform auto-negotiation. While read as cleared, this bit indicates the PHY lacks the ability to perform auto-negotiation. Returns 1 on read. This bit is read-only.
13	Link Status	Link status. This bit is read-only and is cleared by default. 0 A valid link is not established. This bit latches low allowing for software polling to detect a failure condition. 1 A valid link is established.
14	—	Reserved, should be cleared.
15	Extend Ability	Extended capability. This bit indicates that the PHY contains the extended set of registers (those beyond control and status). Returns 1 on read. This bit is read-only.

18.5.4.3.3 AN Advertisement Register (ANA)

Figure 18-116 describes the definition for the ANA register.

Offset 0x04 — Access: Read/Write

	0	1	2–3	4–6	7	8	9	10	11–15
R/W	Next Page	—	Remote Fault	—	Pause		Half Duplex	Full Duplex	—
Reset	All zeros								

Figure 18-116. AN Advertisement Register Definition

Table 18-125 describes the fields of the ANA register.

Table 18-125. ANA Field Descriptions

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Next Page</td><td>Next page configuration. The local device sets this bit to either request next page transmission or advertise next page exchange capability.
0 The local device wishes not to engage in next page exchange.
1 The local device has no next pages but wishes to allow reception of next pages. If the local device has no next pages and the link partner wishes to send next pages, the local device shall send null message codes and have the message page set to 0b000_0000_0001, as defined in annex 28C.</td></tr>
<tr><td>1</td><td>—</td><td>Reserved. (Ignore on read)</td></tr>
<tr><td>2–3</td><td>Remote Fault</td><td>The local device’s remote fault condition is encoded in bits 2 and 3 of the base page. Values are shown in the following table. The default value is 00. Indicate a fault by setting a non-zero remote fault encoding and re-negotiating.
<table>
<tr><th>RF1 bit[3]</th><th>RF2 bit[2]</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>No error, link OK</td></tr>
<tr><td>0</td><td>1</td><td>Offline</td></tr>
<tr><td>1</td><td>0</td><td>Link_Failure</td></tr>
<tr><td>1</td><td>1</td><td>Auto-Negotiation_Error</td></tr>
</table></td></tr>
<tr><td>4–6</td><td>—</td><td>Reserved, should be cleared.</td></tr>
<tr><td>7–8</td><td>Pause</td><td>The local device’s PAUSE capability is encoded in bits 7 and 8, and the decodes are shown in the following table. For priority resolution information consult Table 18-126.
<table>
<tr><th>PAUSE bit[8]</th><th>ASM_DIR bit[7]</th><th>Capability</th></tr>
<tr><td>0</td><td>0</td><td>No PAUSE</td></tr>
<tr><td>0</td><td>1</td><td>Asymmetric PAUSE toward link partner</td></tr>
<tr><td>1</td><td>0</td><td>Symmetric PAUSE</td></tr>
<tr><td>1</td><td>1</td><td>Both symmetric PAUSE and
Asymmetric PAUSE toward local device</td></tr>
</table></td></tr>
</table>

Table 18-125. ANA Field Descriptions (continued)

Bits	Name	Description
9	Half Duplex	Half-duplex capability. 0 Designates local device as not capable of half-duplex operation. 1 Designates local device as capable of half-duplex operation.
10	Full Duplex	Full-duplex capability. 0 Designates the local device as not capable of full-duplex operation. 1 Designates the local device as capable of full-duplex operation.
11–15	—	Reserved, should be cleared.

Table 18-126 describes the resolution of pause priority.

Table 18-126. PAUSE Priority Resolution

Local Device		Link Partner		Local Resolution	Link Partner Resolution
PAUSE	ASM_DIR	PAUSE	ASM_DIR		
0	0	x	x	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
0	1	0	x	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
0	1	1	0	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
0	1	1	1	Enable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Enable PAUSE receive
1	0	0	x	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
1	0	1	x	Enable PAUSE transmit Enable PAUSE receive	Enable PAUSE transmit Enable PAUSE receive
1	1	0	0	Disable PAUSE transmit Disable PAUSE receive	Disable PAUSE transmit Disable PAUSE receive
1	1	0	1	Disable PAUSE transmit Enable PAUSE receive	Enable PAUSE transmit Disable PAUSE receive
1	1	1	x	Enable PAUSE transmit Enable PAUSE receive	Enable PAUSE transmit Enable PAUSE receive

18.5.4.3.4 AN Link Partner Base Page Ability Register (ANLPBPA)

Figure 18-117 describes the definition for the ANLPBPA register.

Offset 0x05 Access: Read/Write

	0	1	2–3	4–6	7–8	9	10	11–15
R	Next Page	—	Remote Fault	—	Pause	Half Duplex	Full Duplex	—
W								
Reset	All zeros							

Figure 18-117. AN Link Partner Base Page Ability Register Definition

Table 18-127 describes the fields of the ANLPBPA register.

Table 18-127. ANLPBPA Field Descriptions

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Next Page</td><td>Next page. This bit is read-only. The link partner sets or clears this bit.
0 Link partner has no subsequent next pages or is not capable of receiving next pages.
1 Link partner either requesting next page transmission or indicating the capability to receive next pages.</td></tr>
<tr><td>1</td><td>—</td><td>Reserved. (Ignore on read)</td></tr>
<tr><td>2–3</td><td>Remote Fault</td><td>The link partner’s remote fault condition is encoded in bits 2 and 3 of the base page. Values are shown in the remote fault encoding field table below. This bit is read-only.
<table>
<tr><th>RF1 bit[3]</th><th>RF2 bit[2]</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>No error, link OK</td></tr>
<tr><td>0</td><td>1</td><td>Offline</td></tr>
<tr><td>1</td><td>0</td><td>Link_Failure</td></tr>
<tr><td>1</td><td>1</td><td>Auto-Negotiation_Error</td></tr>
</table></td></tr>
<tr><td>4–6</td><td>—</td><td>Reserved, should be cleared.</td></tr>
<tr><td>7–8</td><td>Pause</td><td>Encoding of the link partner’s PAUSE capability is shown in the PAUSE encoding table below. For priority resolution information consult. This bit is read-only
<table>
<tr><th>PAUSE bit[8]</th><th>ASM_DIR bit[7]</th><th>Capability</th></tr>
<tr><td>0</td><td>0</td><td>No PAUSE</td></tr>
<tr><td>0</td><td>1</td><td>Asymmetric PAUSE toward link partner</td></tr>
<tr><td>1</td><td>0</td><td>Symmetric PAUSE</td></tr>
<tr><td>1</td><td>1</td><td>Both symmetric PAUSE and
Asymmetric PAUSE toward local device</td></tr>
</table></td></tr>
<tr><td>9</td><td>Half Duplex</td><td>Half-duplex capability. This bit is read-only.
0 Link partner is not capable of half-duplex mode.
1 Link partner is capable of half-duplex mode.</td></tr>
</table>

Table 18-127. ANLPBPA Field Descriptions (continued)

Bits	Name	Description
10	Full Duplex	Full-duplex capability. This bit is read-only. 0 Link partner is not capable of full-duplex mode. 1 Link partner is capable of full-duplex mode.
11–15	—	Reserved, should be cleared.

18.5.4.3.5 AN Expansion Register (ANEX)

Figure 18-118 describes the definition for the ANEX register.

Offset 0x06 — Access: Read only

	0–12	13	14	15
R	—	NP Able	Page Rx'd	—
W				
Reset	All zeros			

Figure 18-118. AN Expansion Register Definition

Table 18-128 describes the fields of the ANEX register.

Table 18-128. ANEX Field Descriptions

Bits	Name	Description
0–12	—	Reserved, should be cleared.
13	NP Able	Next page able. This bit is read-only and returns 1 on read. While read as set, indicates local device supports next page function.
14	Page Rx'd	Page received. This bit is read-only. The bit clears on a read to the register. 0 Normal operation. 1 A new page was received and stored in the applicable AN link partner ability or AN next page register. This bit latches high in order for software to detect while polling.
15	—	Reserved, should be cleared.

18.5.4.3.6 AN Next Page Transmit Register (ANNPT)

Figure 18-119 describes the definition for the ANNPT register.

Offset 0x07 — Access: Mixed

	0	1	2	3	4	5–15
R	Next Page	—	Msg Page	Ack2	Toggle	Message/Un-formatted Code Field
W						
Reset	All zeros					

Figure 18-119. AN Next Page Transmit Register Definition

Table 18-129 describes the fields of the ANNPT register.

Table 18-129. ANNPT Field Descriptions

Bits	Name	Description
0	Next Page	Next page indication. [Reference MII bit 7.15 in IEEE 802.3, 2000 Edition Clause 28.2.4] 0 Last page. 1 Additional next pages to follow.
1	—	Reserved. (Ignore on read)
2	Msg Page	Message page. [Reference MII bit 7.13] 0 Unformatted page. 1 Message page.
3	Ack2	Acknowledge 2. Used by the next page function to indicate that the device has the ability to comply with the message. [Reference MII bit 7.12] 0 The local device cannot comply with message. 1 The local device complies with message.
4	Toggle	Toggle. Used to ensure synchronization with the link partner during next page exchange. This bit always takes the opposite value of the toggle bit of the previously-exchanged link code word. The initial value in the first next page transmitted is the inverse of bit 11 in the base link code word. [Reference MII bit 7.11] This bit is read-only. 0 Toggle bit of the previously-exchanged link code word was 1. 1 Toggle bit of the previously-exchanged link code word was 0.
5–15	Message/ Un-formatted Code Field	Message pages are formatted pages that carry a pre-defined message code, which is enumerated in IEEE 802.3u/Annex 28C. Unformatted code fields take on an arbitrary value. [Reference MII field 7.10:0]

18.5.4.3.7 AN Link Partner Ability Next Page Register (ANLPANP)

Figure 18-120 describes the definition for the ANLPANP register.

Offset 0x08 Access: Read only

	0	1	2	3	4	5–15
R	Next Page	—	Msg Page	Ack2	Toggle	Message/Un-formatted Code Field
W						
Reset	All zeros					

Figure 18-120. AN Link Partner Ability Next Page Register Definition

Table 18-130 describes the fields of the ANLPANP register.

Table 18-130. ANLPANP Field Descriptions

Bits	Name	Description
0	Next Page	Next page. The link partner sets and clears this bit. 0 Last page from link partner 1 Additional next pages to follow
1	—	Reserved. (Ignore on read)
2	Msg Page	Message page. 0 Unformatted page 1 Message page

Table 18-130. ANLPANP Field Descriptions (continued)

Bits	Name	Description
3	Ack2	Acknowledge 2. Indicates the link partner's ability to comply with the message. 0 Link partner cannot comply with message. 1 Link partner complies with message.
4	Toggle	Toggle. Used to ensure synchronization with the link partner during next page exchange. This bit always takes the opposite value of the toggle bit of the previously-exchanged link code word. The initial value in the first next page transmitted is the inverse of bit 11 in the base link code word. This bit is read-only. 0 Toggle bit of the previously-exchanged link code word was 1. 1 Toggle bit of the previously-exchanged link code word was 0.
5–15	Message/ Un-formatted Code Field	Message pages are formatted pages that carry a pre-defined message code, which is enumerated in IEEE 802.3u/Annex 28C. Unformatted code fields take on an arbitrary value.

18.5.4.3.8 Extended Status Register (EXST)

Figure 18-121 describes the definition for the EXST register.

Offset 0x0F — Access: Read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1000X Full	1000X Half	1000T Full	1000T Half	—											
W																
Reset	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-121. Extended Status Register Definition

Table 18-131 describes the fields of the EXST register.

Table 18-131. EXST Field Descriptions

Bits	Name	Description
0	1000X Full	1000X full-duplex capability. Returns 1 on read. This bit is read-only. 0 PHY cannot operation in 1000BASE-X full-duplex mode. 1 PHY can operate in 1000BASE-X full-duplex mode.
1	1000X Half	1000X half-duplex capability. Returns 0 on read. This bit is read-only. 0 PHY cannot operation in 1000BASE-X half-duplex mode. 1 PHY can operate in 1000BASE-X half-duplex mode.
2	1000T Full	1000T full-duplex capability. Returns 1 on read. This bit is read-only. 0 PHY cannot operation in 1000BASE-T full-duplex mode. 1 PHY can operate in 1000BASE-T full-duplex mode.
3	1000T Half	1000T half-duplex capability. Returns 0 on read. This bit is read-only. 0 PHY cannot operation in 1000BASE-T half-duplex mode. 1 PHY can operate in 1000BASE-T half-duplex mode.
4–15	—	Reserved

18.5.4.3.9 Jitter Diagnostics Register (JD)

Annex 36A in IEEE 802.3z specification describes several jitter test patterns. These can be configured to be sent by writing the jitter diagnostics register. See the register description for more information. In may be wise to auto-negotiate and advertise a remote fault signaling of offline prior to beginning the test patterns. Figure 18-122 describes the definition for the JD register.

Offset 0x10 — Access: Read/Write

	0	1–3	4–5	6–15
R/W	Jitter Enable	Jitter Select	—	Custom Jitter Pattern
Reset	All zeros			

Figure 18-122. Jitter Diagnostics Register Definition

Table 18-132 describes the fields of the JD register.

Table 18-132. JD Field Descriptions

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>0</td><td>Jitter Enable</td><td>Jitter enable. This bit is cleared by default.
0 Normal transmit operation.
1 Enable the TBI to transmit the jitter test patterns defined in IEEE 802.3z 36A specification.</td></tr>
<tr><td>1–3</td><td>Jitter Select</td><td>Selects the jitter pattern to be transmitted in diagnostics mode. Encoding of this field is shown in the following table. Default is 00.
<table>
<tr><th>Jitter Pattern Select</th><th>bit[1]</th><th>bit[2]</th><th>bit[3]</th></tr>
<tr><td>User defined uses custom jitter pattern, bits 6–15</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>High frequency (± D21.5)
1010101010101010101010101010101010101010…</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>Mixed frequency (± K28.5)
1111101011000001010011111010110000010100…</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>Low frequency
1111100000111110000011111000001111100000…</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>Complex pattern (10’h17c,10’h0c9,10’h0e5,10’h2a3, 10’h17c,...)</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>Square Wave (– K28.7)
0011111000001111100000111110000011111000…</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>Reserved</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>Reserved</td><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>4–5</td><td>—</td><td>Reserved</td></tr>
<tr><td>6–15</td><td>Custom Jitter Pattern</td><td>Used in conjunction with jitter (pattern) select and jitter (diagnostic) enable; set this field to the desired custom pattern which is continuously transmitted. Its default is 0x000.</td></tr>
</table>

18.5.4.3.10 TBI Control Register (TBICON)

Figure 18-123 describes the definition for the TBICON register.

Offset 0x11 — Access: Mixed

Bits	0	1	2	3	4–6	7	8–9	10	11	12–15
R/W	Soft_Reset	—	Disable Rx Dis	Disable Tx Dis	—	AN Sense	—	Clock Select	MII Mode (R)	—
Reset	0	0	0	0	0 0 0	0	0 0	0	1	0 0 0 0

Figure 18-123. TBI Control Register Definition

Table 18-133 describes the fields of the TBICON register.

Table 18-133. TBICON Field Descriptions

Bits	Name	Description
0	Soft_Reset	Soft reset. This bit is cleared by default. 0 Normal operation. 1 Resets the functional modules in the TBI.
1	—	Reserved. (Ignore on read)
2	Disable Rx Dis	Disable receive disparity. This bit is cleared by default. 0 Normal operation. 1 Disables the running disparity calculation and checking in the receive direction.
3	Disable Tx Dis	Disable transmit disparity. This bit is cleared by default. 0 Normal operation. 1 Disables the running disparity calculation and checking in the transmit direction.
4–6	—	Reserved
7	AN Sense	Auto-negotiation sense enable. This bit is cleared by default. 0 IEEE 802.3z Clause 37 behavior is desired, which results in the link not completing. 1 Allow the auto-negotiation function to sense either a Gigabit MAC in auto-negotiation bypass mode or an older Gigabit MAC without auto-negotiation capability. If sensed, auto-negotiation complete becomes true; however, the page received is low, indicating no page was exchanged. Management can then act accordingly.
8–9	—	Reserved
10	Clock Select	Clock select. This bit is cleared by default. 0 Allow the TBI to accept dual split-phase 62.5 MHz receive clocks. 1 Configure the TBI to accept a 125 MHz receive clock from the SerDes/PHY. The 125 MHz clock must be physically connected to ‘PMA receive clock 0’ if using a parallel (non-SGMII) Ethernet protocol.
11	MI Mode	This bit describes the configuration mode of the TBI. The user reads a 1 while the TBI is configured in GMII/MII mode (connected to a GMII/MII PHY) and a 0 while configured in TBI mode (connected to a 1000BASE-X SerDes). Its value is the inverse of ECNTRL[TBIM]. 0 TBI mode. 1 GMII mode.
12–15	—	Reserved

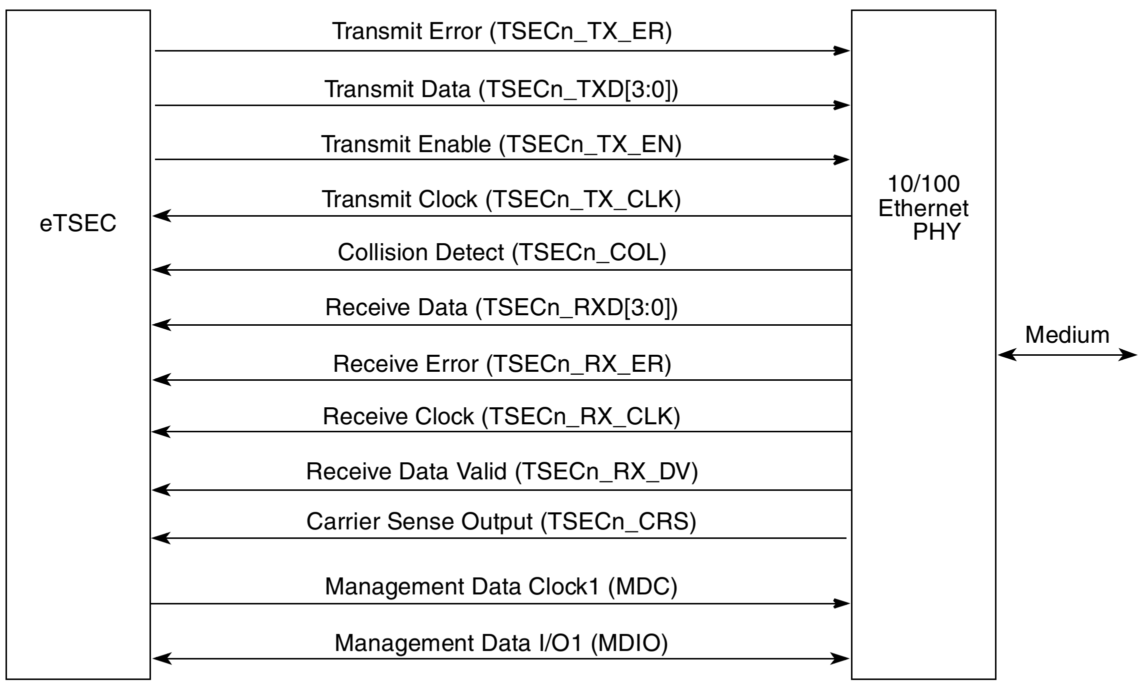
18.6 Functional Description

18.6.1 Connecting to Physical Interfaces on Ethernet

This section describes how to connect the eTSEC to various interfaces: MII, RMII, RGMII, and RTBI. To avoid confusion, all of the buses follow the bus conventions used in the IEEE 802.3 specification because the PHYs follow the same conventions. (For instance, in the bus TSEC*n*_TXD[3:0], bit 3 is the msb and bit 0 is the lsb). If a mode does not use all input signals available to a particular eTSEC, those inputs that are not used must be pulled low on the board.

18.6.1.1 Media-Independent Interface (MII)

This section describes the media-independent interface (MII) intended to be used between the PHYs and the eTSEC. Figure 18-124 depicts the basic components of the MII including the signals required to establish eTSEC module connection with a PHY.



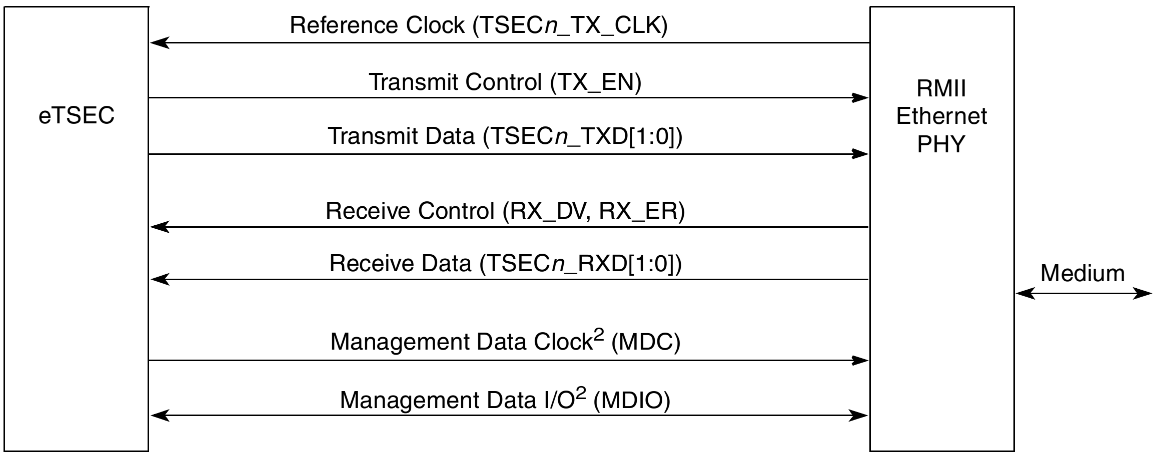
[1] The management signals (MDC and MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

Figure 18-124. eTSEC-MII Connection

An MII interface has 18 signals (including the MDC and MDIO signals), as defined by the IEEE 802.3u standard, for connecting to an Ethernet PHY.

18.6.1.2 Reduced Media-Independent Interface (RMII)

This section describes the reduced media-independent interface (RMII) intended to be used between the PHYs and the GMII MAC. The RMII is a reduced-pin alternative to the IEEE 802.3u MII. The RMII reduces the number of signals required to interconnect the MAC and the PHY from a maximum of 18 signals (MII) to 10 signals. To accomplish this objective, the data paths are halved in width and clocked at twice the MII clock frequency, while clocks, carrier sense and error signals have been partly combined. For 100 Mbps operation, the reference clock operates at 50 MHz, whereas for 10 Mbps operation, the clock remains at 50MHz, but only every 10th cycle is used. Figure 18-125 depicts the basic components of the reduced media-independent interface and the signals required to establish an eTSEC's connection with a PHY. The RMII is implemented as defined by the RMII Specification of the RMII Consortium, as of March 20, 1998.

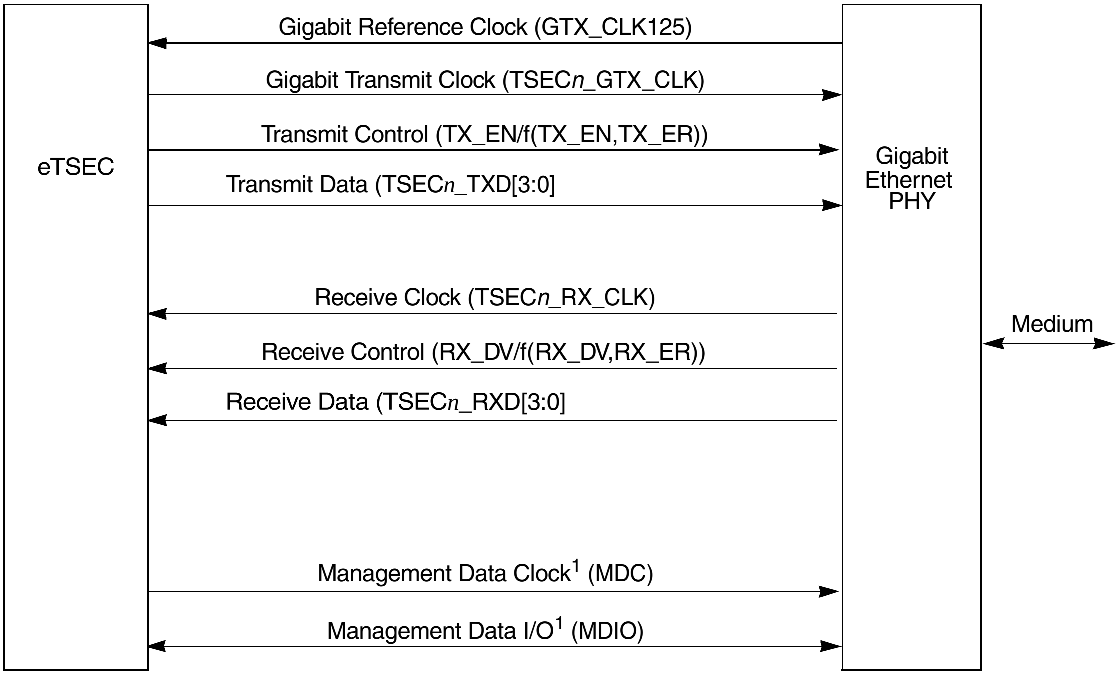


[2] The management signals (MDC and MDIO) are common to all of the Ethernet controllers module connections in the system, assuming that each PHY has a different management address.

Figure 18-125. eTSEC-RMII Connection

18.6.1.3 Reduced Gigabit Media-Independent Interface (RGMII)

This section describes the reduced gigabit media-independent interface (RGMII) intended to be used between the PHYs and the GMII MAC. The RGMII is an alternative to the IEEE 802.3u MII, the IEEE 802.3z GMII. The RGMII reduces the number of signals required to interconnect the MAC and the PHY from a maximum of 28 signals (GMII) to 15 signals (GTX_CLK125 included) in a cost effective and technology independent manner. To accomplish this objective, the data paths and all associated control signals are multiplexed using both edges of the clock. For gigabit operation, the clocks operate at 125MHz, and for 10/100 operation, the clocks operate at 2.5 MHz or 25 MHz, respectively. Note that the GTX_CLK125 input must be provided at 125 MHz for an RGMII interface, regardless of operation speed (1 Gbps, 100 Mbps, or 10 Mbps). Figure 18-126 depicts the basic components of the gigabit reduced media-independent interface and the signals required to establish the gigabit Ethernet controllers' module connection with a PHY. The RGMII is implemented as defined by the RGMII specification Version 1.2a 9/22/00.

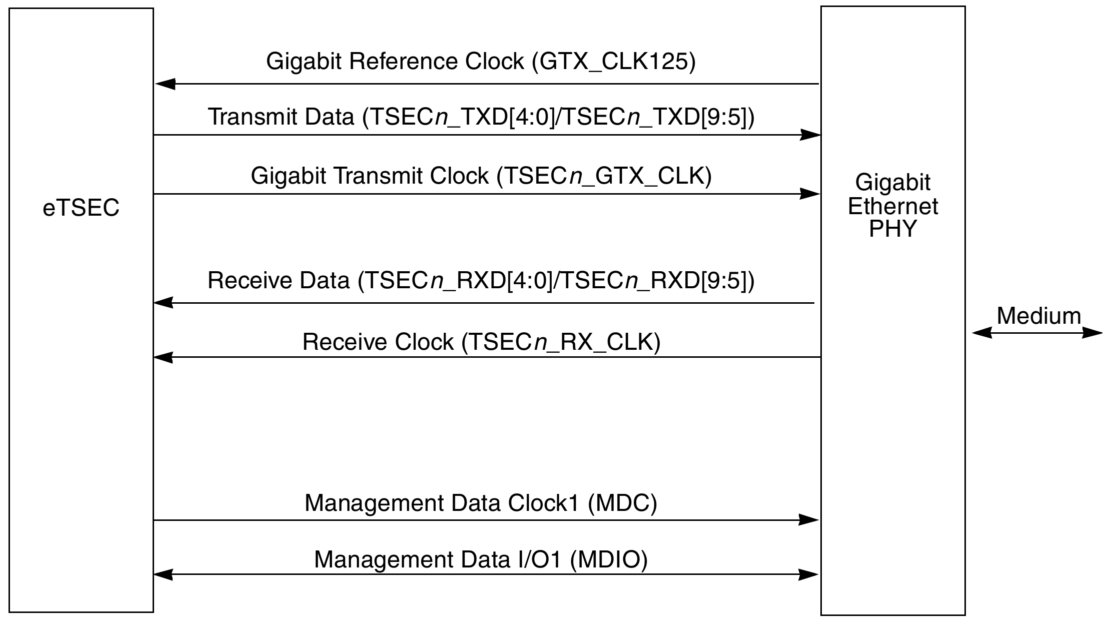


[1] The management signals (MDC and MDIO) are common to all of the gigabit Ethernet controllers module connections in the system, assuming that each PHY has a different management address.

Figure 18-126. eTSEC-RGMII Connection

18.6.1.4 Reduced Ten-Bit Interface (RTBI)

This section describes the reduced ten-bit interface (RTBI) intended to be used between the PHYs and the eTSEC to implement a reduced-pin count version of a SerDes interface for optical-fiber devices in 1000BASE-SX/LX applications. Figure 18-127 depicts the basic components of the RTBI including the signals required to establish eTSEC module connection with a PHY. Note that in RTBI the eTSEC immediately begins auto-negotiation with the SerDes.



[1] The management signals (MDC and MDIO) are common to all of the Ethernet controllers' connections in the system, assuming that each PHY has a different management address.

Figure 18-127. eTSEC-RTBI Connection

A RTBI interface has 15 signals (GE_GTX_CLK125 included), as defined by the RGMII specification Version 1.2a 9/22/00, and is intended to be an alternative to the IEEE 802.3u MII, the IEEE 802.3z GMII and the TBI standard for connecting to an Ethernet PHY.

18.6.1.5 Ethernet Physical Interfaces Signal Summary

Table 18-134 describes the signal multiplexing for MII and RMII interfaces.

Table 18-134. MII and RMII Signals Multiplexing

eTSEC Signals			MII Interface			RMII Interface		
Frequency [MHz] 125			Frequency [MHz] 25			Frequency [MHz] 50		
Voltage[V] 3.3/2.5			Voltage[V] 3.3			Voltage[V] 3.3		
Signals (TSEC*n*_)	I/O	No. of Signals	Signals (TSEC*n*_)	I/O	No. of Signals	Signals (TSEC*n*_)	I/O	No. of Signals
GTX_CLK	O	1						
TX_CLK	I	1	TX_CLK	I	1	REF_CLK	I	1
TxD[0]	O	1	TxD[0]	O	1	TxD[0]	O	1
TxD[1]	O	1	TxD[1]	O	1	TxD[1]	O	1
TxD[2]	O	1	TxD[2]	O	1			
TxD[3]	O	1	TxD[3]	O	1			
TX_EN	O	1	TX_EN	O	1	TX_EN	O	1
TX_ER	O	1	TX_ER	O	1			
RX_CLK	I	1	RX_CLK	I	1			
RxD[0]	I	1	RxD[0]	I	1	RxD[0]	I	1
RxD[1]	I	1	RxD[1]	I	1	RxD[1]	I	1
RxD[2]	I	1	RxD[2]	I	1			
RxD[3]	I	1	RxD[3]	I	1			
RX_DV	I	1	RX_DV	I	1	CRS_DV	I	1
RX_ER	I	1	RX_ER	I	1	RX_ER	I	1
COL	I	1	COL	I	1			
CRS	I	1	CRS	I	1			
Sum		25	**Sum**		16	**Sum**		8

Table 18-135 describes the signal multiplexing for RGMII and RTBI interfaces.

Table 18-135. RGMII and RTBI Signals Multiplexing

eTSEC Signals			RGMII Interface			RTBI Interface		
Frequency [MHz] 125			Frequency [MHz] 125			Frequency [MHz] 62.5		
Voltage[V] 3.3/2.5			Voltage[V] 2.5			Voltage[V] 2.5		
Signals (TSEC*n*_)	I/O	No. of Signals	Signals (TSEC*n*_)	I/O	No. of Signals	Signals (TSEC*n*_)	I/O	No. of Signals
GTX_CLK	O	1	GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1						
TxD[0]	O	1	TxD[0	O	1	TCG[0]/TCG[5]	O	1

Table 18-135. RGMII and RTBI Signals Multiplexing (continued)

eTSEC Signals			RGMII Interface			RTBI Interface		
Frequency [MHz] 125			Frequency [MHz] 125			Frequency [MHz] 62.5		
Voltage[V] 3.3/2.5			Voltage[V] 2.5			Voltage[V] 2.5		
Signals (TSEC*n*_)	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**
TxD[1]	O	1	TxD[1]	O	1	TCG[1]/TCG[6]	O	1
TxD[2]	O	1	TxD[2]	O	1	TCG[2]/TCG[7]	O	1
TxD[3]	O	1	TxD[3]	O	1	TCG[3]/TCG[8]	O	1
TX_EN	O	1	TX_CTL (TX_EN/ TX_ERR)	O	1	TCG[4]/TCG[9]	O	1
TX_ER	O	1						
RX_CLK	I	1	RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RxD[0]/RxD[4]	I	1	RCG[0]/RCG[5]	I	1
RxD[1]	I	1	RxD[1]/RxD[5]	I	1	RCG[1]/RCG[6]	I	1
RxD[2]	I	1	RxD[2]/RxD[6]	I	1	RCG[2]/RCG[7]	I	1
RxD[3]	I	1	RxD[3]/RxD[7]	I	1	RCG[3]/RCG[8]	I	1
RX_DV	I	1	RX_CTL (RX_DV/ RX_ERR)	I	1	RCG[4]/RCG[9]	I	1
RX_ER	I	1						
COL	I	1						
CRS	I	1					I	
Sum		25	**Sum**		12	**Sum**		12

Table 18-136. RGMII Signals Multiplexing

eTSEC Signals			RGMII Interface		
Frequency [MHz] 125			Frequency [MHz] 125		
Voltage[V] 3.3/2.5			Voltage[V] 2.5		
Signals (TSEC*n*_)	**I/O**	**No. of Signals**	**Signals (TSEC*n*_)**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1			
TxD[0]	O	1	TxD[0	O	1
TxD[1]	O	1	TxD[1]	O	1
TxD[2]	O	1	TxD[2]	O	1
TxD[3]	O	1	TxD[3]	O	1

Table 18-136. RGMII Signals Multiplexing (continued)

eTSEC Signals			RGMII Interface		
Frequency [MHz] 125			Frequency [MHz] 125		
Voltage[V] 3.3/2.5			Voltage[V] 2.5		
Signals (TSEC*n*_)	I/O	No. of Signals	Signals (TSEC*n*_)	I/O	No. of Signals
TX_EN	O	1	TX_CTL (TX_EN/TX_ERR)	O	1
TX_ER	O	1			
RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RxD[0]/RxD[4]	I	1
RxD[1]	I	1	RxD[1]/RxD[5]	I	1
RxD[2]	I	1	RxD[2]/RxD[6]	I	1
RxD[3]	I	1	RxD[3]/RxD[7]	I	1
RX_DV	I	1	RX_CTL (RX_DV/RX_ERR)	I	1
RX_ER	I	1			
COL	I	1			
CRS	I	1			
Sum		25	**Sum**		12

Table 18-137 describes the signals shared by all interfaces.

Table 18-137. Shared Signals

Signals	I/O	No. of Signals	Function
MDIO	I/O	1	Management interface I/O
MDC	O	1	Management interface clock
GTX_CLK125	I	1	Reference clock
Sum			—

18.6.1.6 SGMII Interface

SGMII communication using the eTSEC is accomplished through the SerDes interface. See Table 18-1 on page 18-6 for specific signal assignments.

18.6.2 Gigabit Ethernet Controller Channel Operation

This section describes the operation of the eTSEC. First, the software initialization sequence is described. Next, the software (Ethernet driver) interface for transmitting and receiving frames is reviewed. Frame filtering and receive filing algorithm features are also discussed. The section concludes with interrupt handling, inter-packet gap time, and loop back descriptions.

18.6.2.1 Initialization Sequence

This sections describes which registers are reset due to a hard or software reset and what registers the user must initialize prior to enabling the eTSEC.

18.6.2.1.1 Hardware Controlled Initialization

A hard reset occurs when the system powers up. All eTSEC's registers and control logic are reset to their default states after a hard reset has occurred. In this state, each eTSEC behaves like a PowerQUICC II Pro device, except for the absence of out-of-sequence TxBD features. That is, initially TCP/IP off-load is disabled and only single RxBD and TxBD rings are accessible.

18.6.2.1.2 User Initialization

After the system has undergone a hard reset, software must initialize certain basic eTSEC registers. Other registers can also be initialized during this time, but they are optional and must be determined based on the requirements of the system. See Table 18-3 for the register list. Table 18-138 describes the minimum steps for register initialization.

Table 18-138. Steps for Minimum Register Initialization

Description
1. Set and clear MACCFG1 [Soft_Reset]
2. Initialize MACCFG2
3. Initialize MAC station address
4. Set up the PHY using the MII Mgmt Interface
5. Configure the GMII
6. Clear IEVENT
7. Initialize IMASK
8. Initialize RCTRL
9. Initialize DMACTRL

After the initialization of registers is performed, the user must execute the following steps in the order described below to bring the eTSEC into a functional state (out of reset):

1. Write to the MACCFG1 register and set the appropriate bits. These need to include RX_EN and TX_EN. To enable flow control, Rx_Flow and Tx_Flow should also be set.
2. For the transmission of Ethernet frames, TxBDs must first be built in memory, linked together as a ring, and pointed to by the TBASE*n* registers. A minimum of two buffer descriptors per ring is required, unless the ring is disabled. Setting the ring to a size of one causes the same frame to be transmitted twice. If TCP/IP off-load is to be enabled, the TxBD[TOE] bit must be set for each frame.

3. Likewise, for the reception of Ethernet frames, the receive queue (or queues) must be ready, with its RxBD pointed to by the RBASE*n* registers. If TCP/IP off-load is to be enabled, RCTRL[PRSDEP] must be set to the required off-load level. Both transmit and receive can be gracefully stopped after transmission and reception begins.
4. Clearing DMACTRL[GTS] triggers the transmission of frame data if the transmitter had been previously stopped. The DMACTRL[GRS] must be cleared if the receiver had been previously stopped. Refer to the DMACTRL register section, and Section 18.6.6.1, "Data Buffer Descriptors," for more information.

18.6.2.2 Soft Reset and Reconfiguring Procedure

Before issuing a soft-reset to and/or reconfiguring the MAC with new parameters, the user must properly shutdown the DMA and make sure it is in an idle state for the entire duration. User must gracefully stop the DMA by setting both GRS and GTS bits in the DMACTRL register, then wait for both GRSC and GTSC bits to be set in the IEVENT register before resetting the MAC or changing parameters. Both GRS and GTS bits must be cleared before re-enabling the MAC to resume the DMA.

During the MAC configuration, if a new set of Tx buffer descriptors are used, the user must load the pointers into the TBASE registers. Likewise if a new set of Rx buffer descriptors are used, the RBASE registers must be written with new pointers.

Following is a procedure to gracefully reset and reconfigure the MAC:

1. Set GRS/GTS bits in DMACTRL register
2. Poll GRSC/GTSC bits in IEVENT register until both are set
3. Set SOFT_RESET bit in MACCFG1 register (Note that SOFT_RESET must remain set for at least 3 TX clocks before proceeding.)
4. Clear SOFT_RESET bit in MACCFG1 register
5. Load TBASE0–TBASE7 with new Tx BD pointers
6. Load RBASE0–RBASE7 with new Rx BD pointers
7. Setup other MAC registers (MACCFG2, MAXFRM, and so on)
8. Setup group address hash table (GADDR0–GADDR15) if address filtering is required
9. Setup receive frame filer table (through RQFAR, RQFCR, and RQFPR) if filing to multiple RxBD rings is required
10. Setup WWR, WOP, TOD bits in DMACTRL register
11. Enable transmit queues in TQUEUE, and ensure that the transmit scheduling mode is correctly set in TCTRL.
12. Enable receive queues in RQUEUE, and optionally set TOE functionality in RCTRL.
13. Clear THLT and TXF bits in TSTAT register by writing 1 to them
14. Clear QHLT and RXF bits in RSTAT register by writing 1 to them.
15. Clear GRS/GTS bits in DMACTRL (do not change other bits)
16. Enable Tx_EN/Rx_EN in MACCFG1 register

18.6.2.3 Gigabit Ethernet Frame Transmission

The Ethernet transmitter requires little core intervention. After the software driver initializes the system, the eTSEC begins to poll the first transmit buffer descriptor (TxBD) in TxBD ring 0 every 512 transmit clocks. If TxBD[R] is set, and the TxBD ring is scheduled for transmission, the eTSEC begins copying the associated transmit buffer from memory to its Tx FIFO. The transmitter takes data from the Tx FIFO and transmits data to the MAC. The MAC transmits the data through the GMII interface to the physical media. The transmitter, once initialized, runs until the end-of-frame (EOF) condition is detected unless a collision within the collision window occurs (half-duplex mode) or an abort condition is encountered.

If the user has a frame ready to transmit, setting the DMACTRL[TOD] eliminates waiting for the next poll and a DMA transfer of the transmit data buffers can begin immediately. The transmission begins once all data for the frame is loaded into the Tx FIFO or sufficient transmit data (determined by the Tx FIFO threshold register) is in the Tx FIFO. If the line is not busy, the MAC transmit logic asserts TX_EN and sends the 7-octet preamble sequence, 1-octet start of frame delimiter, and frame information in that order. If the line is busy, the controller waits for the carrier sense signal, CRS, to remain inactive for 60 bit times (60 clocks) and transmission begins after an additional 36 bit times (96 bit times after CRS became active). In full-duplex mode, because collisions are ignored, frame transmission maintains only the interframe gap (96 bit times) regardless of CRS.

In half-duplex mode (MACCFG2[Full Duplex] is cleared) the MAC defers transmission if the line is busy (CRS asserted). Before transmitting, the MAC waits for carrier sense to become inactive, at which point it then determines if CRS remains negated for 60 clocks. If so, transmission begins after an additional 36 bit times (96 bit times after CRS originally became negated). If CRS continues to be asserted, the MAC follows a specified back-off procedure and tries to retransmit the frame until the retry limit is reached. Data stored in the Tx FIFO is re-transmitted in case of a collision. This avoids unnecessary memory traffic.

The transmitter also monitors for an abort condition and terminates the current frame if an abort condition is encountered. In full-duplex mode the protocol is independent of network activity, and only the transmit inter-frame gap must be enforced.

The transmitter implements full-duplex flow control. If a flow control frame is received, the MAC does not service the transmitter's request to send data until the pause duration is over. If the MAC is currently sending data after a pause frame has been received and processed, the MAC finishes sending the current frame, then suspends subsequent frames (except a pause frame) until the pause duration is over. In addition, the transmitter supports transmission of flow control frames through TCTRL[TFC_PAUSE]. The transmit pause frame is generated internally based on the PAUSE register that defines the pause value to be sent. Note that it is possible to send a pause frame while the pause timer has not expired.

The MAC automatically appends FCS (32-bit CRC) bytes to the frame if any of the following values are set:

- TxBD[PAD/CRC] is set in first TxBD
- TxBD[TC] is set in first TxBD
- MACCFG2[PAD/CRC] is set
- MACCFG2[CRC] is set

The Tx_EN is negated after the FCS is sent. This notifies the PHY of the need to generate the illegal Manchester encoding that signifies the end of an Ethernet frame. Following the transmission of the FCS, the Ethernet controller writes the frame status bits into the BD and clears TxBD[R]. If the end of the current buffer is reached and TxBD[L] is cleared (a frame is comprised of multiple buffer descriptors), only TxBD[R] is cleared.

For both half- and full-duplex modes, an interrupt can be issued depending on TxBD[I]. The Ethernet controller then proceeds to the next TxBD in the table. In this way, the core can be interrupted after each frame, after each buffer, or after a specific buffer is sent. If TxBD[PAD/CRC] is set, the Ethernet controller pads any frame shorter than 64 bytes with zero bytes to make up the minimum length.

To pause transmission, or rearrange the transmit queue, set DMACTRL[GTS]. This can be useful for transmitting expedited data ahead of previously-linked buffers or for error situations. If this bit is set, the eTSEC transmitter performs a graceful transmit stop. The Ethernet controller stops immediately if no transmission is in progress or continues transmission until all queued frames in the Tx FIFO have been disposed of. The IEVENT[GTSC] interrupt occurs once the graceful transmit stop operation is completed. After the DMACTRL[GTS] is cleared, the eTSEC resumes transmission with the next frame.

While the eTSEC is in 10/100Mbps mode it sends bytes least-significant nibble first and each nibble is sent lsb first. While it is in 1000Mbps mode it sends bytes LSB first.

18.6.2.4 Gigabit Ethernet Frame Reception

The eTSEC Ethernet receiver is designed to work with little core intervention and can perform data extraction, address recognition, CRC checking, short frame checking, and maximum frame-length checking.

After a hardware reset, the software driver clears the RSTAT register and sets MACCFG1[RX_EN]. The Ethernet receiver is enabled and immediately starts processing receive frames. The MAC checks for when TSEC*n*_RX_DV is asserted and as long as TSEC*n*_COL remains negated (full-duplex mode ignores TSEC*n*_COL), the MAC looks for the start of a frame by searching for a valid preamble/SFD (start of frame delimiter) header, which is stripped (unless MACCFG2[PreAM RxEN] is set) and the frame begins to be processed. If a valid header is not found, the frame is ignored.

If the receiver detects the first bytes of a frame, the eTSEC controller begins to perform the frame recognition function through destination address (DA) recognition (see Section 18.6.2.7, "Frame Recognition"). Based on this match the frame can be accepted or rejected. The receiver can filter frames based on individual (unicast), group (multicast), and broadcast addresses. Because Ethernet receive frame data is not written to memory until the internal frame recognition algorithm is complete, system bus usage is not wasted on frames unwanted by this station.

If a frame is accepted, the Ethernet controller fetches the receive buffer descriptor (RxBD) from either queue 0 or the queue determined by the filer. If the RxBD is not being used by software (RxBD[E] is set), the eTSEC starts transferring the incoming frame. RxBD[F] is set for the first RxBD used for any particular receive frame. If the current RxBD is not available for the received frame, a receive busy error condition is raised in IEVENT[BSY].

After the buffer is filled, the eTSEC clears RxBD[E] and, if RxBD[I] is set, generates an interrupt. If the incoming frame is larger than the buffer, the Ethernet controller fetches the next RxBD in the table. If it is empty, the controller continues receiving the rest of the frame. In half-duplex mode, if a collision is detected during the frame, no RxBDs are used; thus, no collision frames are presented to the user except late collisions, which indicate LAN problems.

The RxBD length is determined by the MRBL field in the maximum receive buffer length register (MRBLR). The smallest valid value is 64 bytes, with larger values being be some integral multiple of 64 bytes. During reception, the Ethernet controller checks for frames that are too short or too long. After the frame ends (CRS is negated), the receive CRC field is checked and written to the data buffer. The data length written to the last RxBD in the Ethernet frame is the length of the entire frame, which enables the software to recognize an oversized frame condition.

Receive frames are not truncated when they exceed maximum frame bytes in the MAC's maximum frame register if MACCFG2[Huge Frame] is set, yet the babbling receiver error interrupt occurs (IEVENT[BABR] is set) and RxBD[LG] is set.

After the receive frame is complete, the Ethernet controller sets RxBD[L], updates the frame status bits in the RxBD, and clears RxBD[E]. If RxBD[I] is set, the Ethernet controller next generates an interrupt (that can be masked) indicating that a frame was received and is in memory. The Ethernet controller then waits for a new frame.

To interrupt reception or rearrange the receive queue, DMACTRL[GRS] must be set. If this bit is set, the eTSEC receiver performs a graceful receive stop. The Ethernet controller stops immediately if no frames are being received or continues receiving until the current frame either finishes or an error condition occurs. The IEVENT[GRSC] interrupt event is signaled after the graceful receive stop operation is completed. While in this mode the user can write to registers that are accessible to both the user and the eTSEC hardware without fear of conflict, and finally clear IEVENT[GRSC]. After DMACTRL[GRS] is cleared, the eTSEC scans the input data stream for the start of a new frame (preamble sequence and start of frame delimiter), it resumes receiving, and the first valid frame received is placed in the next available RxBD.

18.6.2.5 Ethernet Preamble Customization

By default eTSEC generates a standard Ethernet preamble sequence prior to transmitting frames. However, the user can substitute a custom preamble sequence for the purpose of controlling switching equipment at the receiver, particularly at 100/1000Mbps speeds.; in any RMII mode only the standard preamble can be transmitted

eTSEC normally searches for and discards the standard Ethernet preamble sequence upon receiving frames. Part of the received preamble sequence can be optionally recovered and returned as part of the frame data, making it visible to user software. Note however, that , and preamble cannot be recovered in any RMII mode. Note that it is also possible for the first two bytes of custom preamble (PreOct0 and PreOct1) to be lost in during conversion to ten-bit code groups in the PCS sub-layer. Thus is it recommended that any custom preamble start at PreOct2.

18.6.2.5.1 User-Defined Preamble Transmission

To substitute a custom preamble, the user must ensure that:

- MACCFG2[PreAm TxEN] bit is set
- The first TxBD of every frame containing a custom preamble has its PRE bit set

- An 8-byte custom preamble sequence appears before the Ethernet DA field in the first transmit data buffer

The definition of the 8-byte custom preamble sequence is shown in Figure 18-128.

Byte Offsets	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0–1	PreOct0								PreOct1							
2–3	PreOct2								PreOct3							
4–5	PreOct4								PreOct5							
6–7	PreOct6															

Figure 18-128. Definition of Custom Preamble Sequence

The fields of the custom preamble sequence are described in Table 18-139. It should be noted that use of preamble octets matching the standard start of frame delimiter (0xD5) can be expected to trigger premature frame reception by the receiving station.

Table 18-139. Custom Preamble Field Descriptions

Bytes	Bits	Name	Description
0–1	0–7	PreOct0	Octet #0 of custom transmit preamble. This is the first octet of preamble sent.
	8–15	PreOct1	Octet #1 of custom transmit preamble. This is the second octet of preamble sent.
2–3	0–7	PreOct2	Octet #2 of custom transmit preamble. This is the third octet of preamble sent.
	8–15	PreOct3	Octet #3 of custom transmit preamble. This is the fourth octet of preamble sent.
4–5	0–7	PreOct4	Octet #4 of custom transmit preamble. This is the fifth octet of preamble sent.
	8–15	PreOct5	Octet #5 of custom transmit preamble. This is the sixth octet of preamble sent.
6–7	0–7	PreOct6	Octet #6 of custom transmit preamble. This is the seventh octet of preamble sent. The last octet (the start of frame delimiter) is generated by the MAC automatically.
	8–15	—	Reserved; should be cleared.

18.6.2.5.2 User-Visible Preamble Reception

To return the received preamble, the user must ensure that:

- MACCFG2[PreAm RxEN] bit is set
- Space for an 8-byte preamble sequence is allowed before the Ethernet DA field in the first receive data buffer of each frame

The definition of the 8-byte received preamble sequence is shown in Figure 18-129.

Byte Offsets	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0–1	PreOct0								PreOct1							
2–3	PreOct2								PreOct3							
4–5	PreOct4								PreOct5							
6–7	PreOct6															

Figure 18-129. Definition of Received Preamble Sequence

The fields of the received preamble sequence are described in Table 18-140. Should the received preamble be shorter than the 7-octet sequence defined by IEEE Std. 802.3 standard, initial bytes of the received preamble sequence hold undefined values. The standard start of frame delimiter (0xD5) is always omitted. Note that preamble extraction is not possible in RMII mode.

Table 18-140. Received Preamble Field Descriptions

Bytes	Bits	Name	Description
0–1	0–7	PreOct0	Octet #0 of received preamble. This is the first octet of preamble received.
	8–15	PreOct1	Octet #1 of received preamble. This is the second octet of preamble received.
2–3	0–7	PreOct2	Octet #2 of received preamble. This is the third octet of preamble received.
	8–15	PreOct3	Octet #3 of received preamble. This is the fourth octet of preamble received.
4–5	0–7	PreOct4	Octet #4 of received preamble. This is the fifth octet of preamble received.
	8–15	PreOct5	Octet #5 of received preamble. This is the sixth octet of preamble received.
6–7	0–7	PreOct6	Octet #6 of received preamble. This is the seventh octet of preamble received. The last octet (the start of frame delimiter) is discarded.
	8–15	—	Reserved

18.6.2.6 RMON Support

Using promiscuous mode, the eTSEC can automatically gather network statistics required for remote network interface monitoring. The RMON MIB group 1, RMON MIB group 2, RMON MIB group 3, RMON MIB group 9, RMON MIB2, and the IEEE 802.3 Ethernet MIB are supported. For RMON statistics and their corresponding counters, see the memory map.

18.6.2.7 Frame Recognition

The Ethernet controller performs frame recognition using destination address (DA) recognition. A frame can be rejected or accepted based on the outcome.

18.6.2.7.1 Destination Address Recognition and Frame Filtering

The eTSEC can perform layer 2 frame filtering on the basis of destination Ethernet address (DA), as illustrated by the flowchart in Figure 18-130.

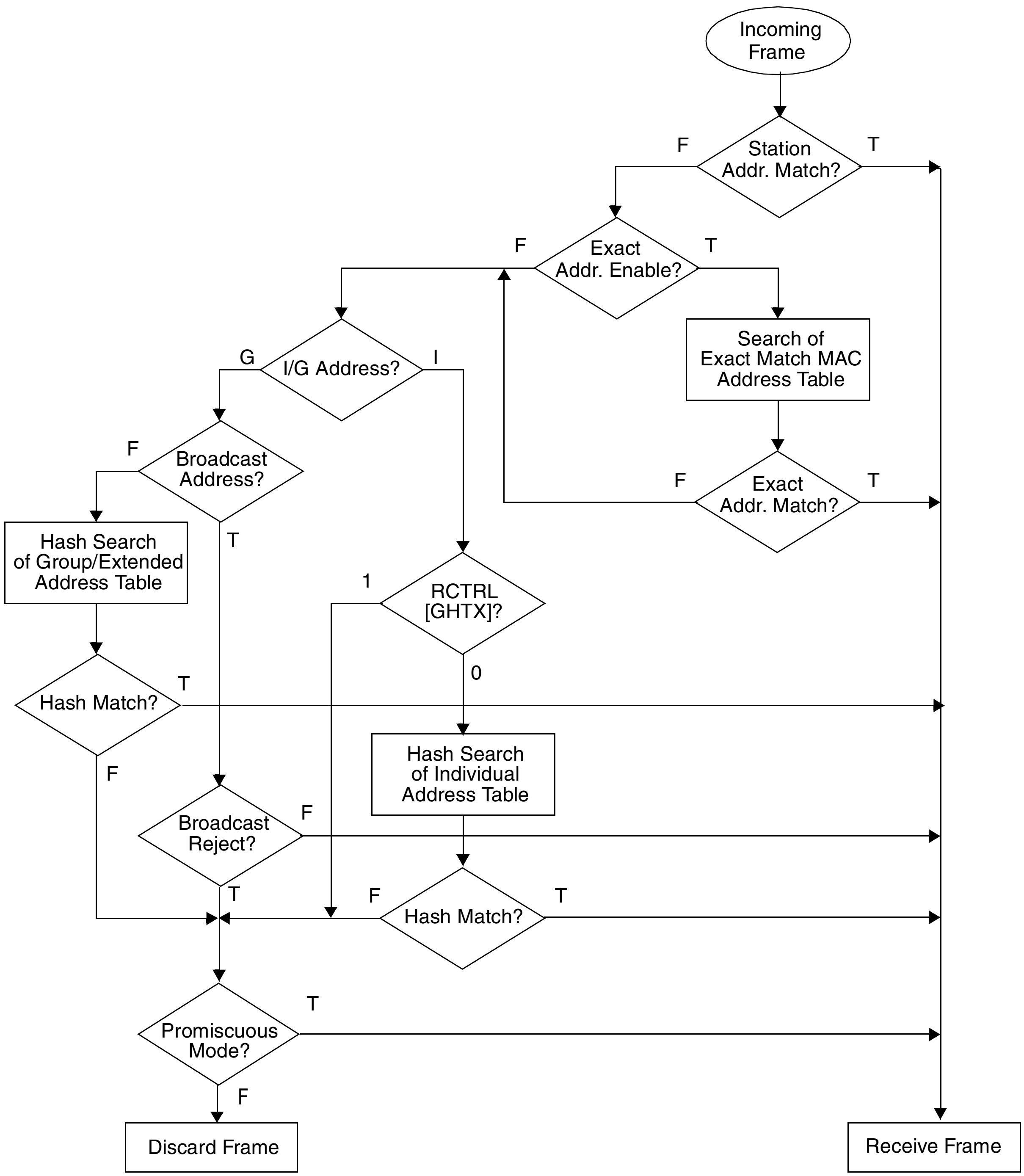


Figure 18-130. Ethernet Address Recognition Flowchart

In promiscuous mode, the eTSEC accepts all received frames regardless of DA. Note, however, that Ethernet frame filtering simply restricts the traffic seen by the receive queue filer. Therefore even in promiscuous mode it remains possible to program the filer to reject frames based on their higher-layer header contents.

In the case of an individual address, the DA field of the received frame is compared with the physical address that the user programs in the station address registers (MACSTNADDR1 and MACSTNADDR2). If the DA does not match the station address, and exact MAC address matching is enabled through RCTRL[EMEN], the controller performs address recognition on the multiple MAC addresses written to the MAC*x*ADDR1 and MAC*x*ADDR2 registers. These virtual addresses give a particular eTSEC the ability to mirror other MACs on the network, which caters for router redundancy protocols, such as HSRP and VRRP.

If exact MAC address matching is not enabled, the eTSEC determines whether DA is a group or individual address. If DA is the standard broadcast address, and broadcast addresses are not rejected, the frame is accepted. If any other group address is received, the eTSEC looks-up the DA by means of the group hash table. The group hash table may be extended to 512 entries if RCTRL[GHTX] = 1. Otherwise, an individual address is hashed into the 256-entry individual hash table when RCTRL[GHTX] = 0.

18.6.2.7.2 Hash Table Algorithm

The hash table process used in the group hash filtering operates as follows. By default, the Ethernet controller maps any 48-bit destination address into one of 256 bins, represented by the 256 bits in IGADDR0–IGADDR7 for individual addresses, and the 256 bits in GADDR0–GADDR7 for group addresses. But in the case where RCTRL[GHTX] is set, both sets of registers are combined into an extended group-only hash table of 512 bits, where IGADDR0–IGADDR7 contain the first 256 bits and GADDR0–GADDR7 contain the last 256 bits. No individual-address table exists in extended mode.

The 48-bit destination address received by the MAC is passed through the Ethernet CRC-32 algorithm to produce a hash value. The CRC polynomial used is:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The MAC initializes its CRC register to 0xFFFFFFFF before computing a CRC on the 6 bit-reversed octets of the DA. A non-optimized sample of C code for computing the DA hash is listed in Figure 18-131. The 9 most significant bits of the raw, uninverted CRC are used as the hash table index, H[8:0]. If RCTRL[GHTX] = 0, bits H[8:6] select one of the 8 IGADDR or GADDR registers, while bits H[5:1] select a bit within the 32-bit register. If RCTRL[GHTX] = 1, bits H[8:5] select one of the 16 registers in the {IGADDR, GADDR} set, while bits H[4:0] select a bit within the 32-bit register. For example, if H[8:5] = 7, IGADDR7 is selected, whereas H[8:5] = 9 selects GADDR1.

```
/* Wrapper macros for 256-bucket and 512-bucket hash tables:
   Pass 6-byte Ethernet MAC address as parameter. */
#define TSEC_HASH256(macaddr) ((crc32(macaddr) >> 24) & 0xff)
#define TSEC_HASH512(macaddr) ((crc32(macaddr) >> 23) & 0x1ff)

/* CRC constants.  Note: CRC-32 polynomial is bit-reversed. */
#define CRC_POLYNOMIAL 0xedb88320
#define CRC_INITIAL    0xffffffff
#define MAC_ADDRLEN    6
#define BITS_PER_BYTE  8

/* crc32() Takes the array of bytes, macaddr[], representing an
   Ethernet MAC address and returns the CRC-32 result over these bytes,
   where each byte is used in bit-reversed form (Ethernet bit order).
   Index 0 of macaddr[] is the first byte of the address on the wire.
   Test case: the result of crc32 on {0x00, 0x01, 0x02, 0x03, 0x04, 0x05}
   should be 0xad0c28f3.
 */
unsigned long crc32(unsigned char macaddr[MAC_ADDRLEN])
{
  unsigned long crc, result;
  int byte, i;

  /* CRC-32 algorithm starts by inverting first 4 bytes */
  crc = CRC_INITIAL;
  /* add each byte to running CRC accumulator */
  for (byte = 0; byte < MAC_ADDRLEN; ++byte) {
    crc ^= macaddr[byte];
    /* shift CRC right to perform but reversal on byte of address */
    for (i = 0; i < BITS_PER_BYTE; ++i)
      if (crc & 1)
     crc = (crc >> 1) ^ CRC_POLYNOMIAL;
      else
     crc >>= 1;
  }
  /* finally, reverse bits of result to get CRC in normal bit order */
  for (result = 0, i = 4*BITS_PER_BYTE-1; i >= 0; crc >>= 1, --i)
    result |= (crc & 1) << i;
  return result;
}
```

Figure 18-131. Sample C Code for Computing eTSEC Hash Table Indices

If the CRC hash table index selects a bit that is set in the hash table, the frame is accepted. If 32 group addresses are stored in the hash table and random group addresses are received, the extended hash table prevents roughly 480/512 (93.8%) of the group address frames from reaching memory. Software must further filter those that reach memory to determine if they contain the correct addresses. Alternatively, small multicast groups can be held in the exact match MAC address registers, which guarantees that only correct frames are admitted.

The effectiveness of the hash table declines as the number of addresses increases. For instance, as the number of addresses stored in the 512-bin hash table increases, the vast majority of the hash table bits are set, preventing only a small fraction of frames from reaching memory.

NOTE

The hash table cannot be used to reject frames that match a set of selected addresses because unintended addresses can map to the same bit in the hash table. The receive queue filer may be used to reject frames with unintended address hits in the hash table.

18.6.2.8 Magic Packet Mode

eTSEC implements the AMD Magic Packet™ specification for LAN-initiated power management. This mode is normally entered with the rest of the system in a low-power sleep mode. Software must enable normal receive function in the Ethernet MAC, and then finally set the MACCFG2[MPEN] bit to enable Magic Packet detection before the system enters a reduced mode. While the rest of the system is operating in low-power mode, the enabled eTSEC continues to receive Ethernet frames, but discards them immediately. Upon receipt of any frame whose contents contain the valid Magic Packet sequence, the eTSEC exits out of Magic Packet mode, thus clearing MACCFG2[MPEN], and raises an error/diagnostic interrupt through IEVENT[MAG], which causes the surrounding system to wake-up. Frames received after Magic Packet mode has exited are received into software buffers as usual. Software can abort Magic Packet mode by writing 0 to MACCFG2[MPEN] at any time.

AMD specify a Magic Packet™ to be any Ethernet frame containing a valid Ethernet header (Destination and Source Addresses) and valid FCS (CRC-32), and whose payload includes the specific Magic Packet byte sequence at any offset from the start of frame. The specific byte sequence comprises an unbroken stream of 102 bytes, the first 6 bytes of which are 0xFFFFFF_FFFFFF, followed by 16 copies of the MAC's unique IEEE station address in the normal byte order for Ethernet addresses. For example, if the station address were 0x112233_445566, then the MAC would have to receive 0xFFFFFF_FFFFFF, 0x112233_445566, ..., 0x112233_445566 in any payload to detect a Magic Packet. Only frames addressed specifically to the MAC's station address or a valid multicast or broadcast address can be examined for the Magic Packet sequence.

18.6.2.9 Flow Control

Because collisions cannot occur in full-duplex mode, gigabit Ethernet can operate at the maximum rate. If the rate becomes too fast for a station's receiver, the station's transmitter can send flow-control frames to reduce the rate. Flow-control instructions are transferred by special frames of minimum frame size. The length/type fields of these frames have a special value.

Table 18-141 lists the flow-control frame structure.

Table 18-141. Flow Control Frame Structure

Size [Octets]	Description	Value	Comment
7	Preamble		—
1	SFD		Start frame delimiter
6	Destination address	01-80-C2-00-00-01	Multicast address reserved for use in MAC frames (or MAC station address)
6	Source address		—
2	Length/type	88-08	Control frame type

Table 18-141. Flow Control Frame Structure (continued)

Size [Octets]	Description	Value	Comment
2	MAC opcode	00-01	Pause command
2	MAC parameter		Pause time as defined by the PTV[PT] field. The pause period is measured in pause_quanta, a speed independent constant of 512 bit-times (unlike slot time). The most-significant octet is transmitted first.
2	Extended MAC parameter		Pause time extended as defined by the PTV[PTE] field. The most significant octet is transmitted first.
40	Reserved	—	—
4	FCS		Frame check sequence (CRC)

If flow-control mode is enabled (MACCFG1[Rx_Flow] is set) and the receiver identifies a pause-flow control frame, transmission stops for the time specified in the control frame. Since the pause timer commences counting immediately upon receipt of a PAUSE frame, regardless of whether transmission is currently in progress, a sufficiently large pause time must be received to stop transmission past a frame of MTU size. During a pause, only a control frame can be sent (TCTRL[TFC_PAUSE] is set). Normal transmission resumes after the pause timer stops counting, or resumes immediately if a pause frame with a zero time-out is received. If another pause-control frame is received during the pause, the period changes to the new value received.

18.6.2.10 Interrupt Handling

The following describes what usually occurs within a eTSEC interrupt handler:

- If an interrupt occurs, read IEVENT to determine interrupt sources. IEVENT bits to be handled in this interrupt handler are normally cleared at this time. There are three kinds of interrupts:
  - Receive data frame interrupts, when bits RXB or RXF in IEVENT are set
  - Transmit data frame interrupts, when bits TXB or TXF in IEVENT are set
  - Error, diagnostic, and special interrupts (all bits in IEVENT other than RXB, RXF, TXB, or TXF)
- Process the TxBDs to reuse them if the IEVENT[TXB, TXF or TXE] were set. Consult register bits TSTAT[TXF0–TXF7] to determine which TxBD rings gave rise to the transmit interrupt in the case of TXF. If the transmit speed is fast or the interrupt delay is long, more than one transmit buffer may have been sent by the eTSEC; thus, it is important to check more than just one TxBD during the interrupt handler. One common practice is to process all TxBDs in the interrupt handler until one is found with R set.
- Obtain data from RxBD rings if IEVENT[RXC, RXB or RXF] is set. Consult register bits RSTAT[RXF0–RXF7] to determine which RxBD rings gave rise to the receive interrupt in the case of RXF. If the receive speed is fast or the interrupt delay is long, the eTSEC may have received more than one RxBD; thus, it is important to check more than just one RxBD during interrupt handling. Typically, all RxBDs in the interrupt handler are processed until one is found with E set. Because the eTSEC pre-fetches BDs, the BD table must be big enough so that there is always another empty BD to pre-fetch, otherwise a BSY error occurs.

- Clear any set halt or frame interrupt bits in TSTAT and RSTAT registers, or DMACTRL[GTS] and DMACTRL[GRS] by writing 1s to these bits.
- Continue normal execution.

Table 18-142. Non-Error Transmit Interrupts

Interrupt	Description	Action Taken by the eTSEC
GTSC	Graceful transmit stop complete: transmitter is put into a pause state after completion of the frame currently being transmitted.	None
TXC	Transmit control: Instead of the next transmit frame, a control frame was sent.	None
TXB	Transmit buffer: A transmit buffer descriptor, that is not the last one in the frame, was updated in one of the enabled TxBD rings.	Programmable 'write with response' TxBD to memory before setting IEVENT[TXB].
TXF	Transmit frame: A frame from an enabled TxBD ring was transmitted and the last transmit buffer descriptor (TxBD) of that frame was updated.	Programmable 'write with response' to memory on the last TxBD before setting IEVENT[TXF].

Table 18-143. Non-Error Receive Interrupts

Interrupt	Description	Action Taken by the eTSEC
GRSC	Graceful receive stop complete: Receiver is put into a pause state after completion of the frame currently being received.	None
RXC	Receive control: A control frame was received. As soon as the transmitter finishes sending the current frame, a pause operation is performed.	None
RXB	Receive buffer: A receive buffer descriptor, that is not the last one of the frame, was updated in one of the enabled RxBD rings.	Programmable 'write with response' RxBD to memory before setting IEVENT[RXB].
RXF	Receive frame: A frame was received to an enabled RxBD ring and the last receive buffer descriptor (RxBD) of that frame was updated.	Programmable 'write with response' to memory on the last RxBD before setting IEVENT[RXF].

18.6.2.10.1 Interrupt Coalescing

Interrupt coalescing offers the user the ability to contour the behavior of the eTSEC with regard to frame interrupts. Separate but identical mechanisms exist for both transmitted frames and received frames. In either case, frame interrupts require that software set the I-bit in RxBDs or TxBDs, and disable buffer interrupts (IEVENT[RXB] or IEVENT[TXB]). Particular rings can remain free of interrupts by ensuring that the I-bit is consistently cleared in all BDs. While interrupt coalescing is enabled, a transmit or receive frame interrupt is raised either when a counter threshold-defined number of frames is received/transmitted or the timer threshold-defined period of time has elapsed, whichever occurs first. Disabling and then re-enabling interrupt coalescing forces reset of the coalescing timers and counters to reflect changes made to the threshold registers.

18.6.2.10.2 Interrupt Coalescing By Frame Count Threshold

To avoid interrupt bandwidth congestion due to frequent, consecutive interrupts, the user may enable and configure interrupt coalescing to deliberately group frame interrupts, reducing the total number of interrupts raised. The number of frames received or transmitted prior to an interrupt being raised is determined by the frame threshold field (ICFT) in the appropriate interrupt coalescing configuration register (RXIC or TXIC). The frame threshold field may be assigned a value between 1 and 255. The internal transmit or receive frame counter decrements from this initial value each time a frame is transmitted or received. Upon reaching zero, an interrupt is raised, the appropriate threshold counter is reset to the value in the ICFT field, and then eTSEC continues counting frames while the interrupt is active. The appropriate threshold counter is also reset to the value in the ICFT field if an interrupt is raised subject to the corresponding threshold timer.

18.6.2.10.3 Interrupt Coalescing By Timer Threshold

To avoid stale frame interrupts, the user may also assign a timer threshold, beyond which any frame interrupts not yet raised are forced. The timer threshold fields of the receive and transmit interrupt coalescing configuration registers (RXIC[ICTT] and TXIC[ICTT]) are defined in units equivalent to 64 interface clocks or system clocks, depending on the setting of the ICCS field in RXIC and TXIC.

After transmitting a frame, the transmit interrupt coalescing threshold time begins counting down from the value in TXIC[ICTT]. An interrupt is raised when the counter reaches zero. In the event of graceful transmit stop completion before the coalescing timer expires, the eTSEC issues two interrupts, the first for GTS, the second for TXF (due to timer expiration of a pending event). To prevent the second interrupt from affecting servicing of the GTS event, it is recommended that the user mask out the TXF event during execution of the service routine. After receiving a frame, the receive interrupt coalescing threshold time begins counting down from the value in RXIC[ICTT]. An interrupt is raised when the counter reaches zero. In the event of graceful receive stop completion before the coalescing timer expires, the eTSEC issues two interrupts, the first for GRS, the second for RXF (due to timer expiration of a pending event). To prevent the second interrupt from affecting servicing of the GRS event, it is recommended that the user mask out the RXF event during execution of the service routine.

The interrupt coalescing timer thresholds (transmit and receive, operating independently) may be values ranging from 0x0001 to 0xFFFF. Table 18-144 specifies the range of possible timing thresholds subject to timer clock source, the interface or system frequency, and the value of the RXIC[ICTT] or TXIC[ICTT] field.

Table 18-144. Interrupt Coalescing Timing Threshold Ranges

ICCS (Clock Source)	eTSEC Interface Format and Frequency or eTSEC System Frequency	Interrupt Coalescing Threshold Time	
		Minimum (ICTT = 0x0001)	Maximum (ICTT = 0xFFFF)
0 (I/F clock)	10Base-T at 2.5 MHz	25.6 µs	1.68 s
0 (I/F clock)	100Base-T at 25 MHz	2.56 µs	168 ms
0 (I/F clock)	1000Base-T at 125 MHz	0.51 µs	33.6 ms
1 (sys. clock)	eTSEC operating at 266 MHz	0.24 µs	15.7 ms
1 (sys. clock)	eTSEC operating at 333 MHz	0.19 µs	12.6 ms

The transmit timer threshold counter is reset to the value in TXIC[ICTT] and begins counting down on transmission of the frame following an interrupt.

The receive timer threshold counter is reset to the value in RXIC[ICTT] and begins counting down on receiving the frame following an interrupt.

18.6.2.11 Inter-Frame Gap Time

If a station must transmit, it waits until the LAN becomes silent for a specified period (inter-frame gap, or IFG). The minimum inter-packet gap (IPG) time for back-to-back transmission is set by IPGIFG[Back-to-Back Inter-Packet-Gap]. The receiver receives back-to-back frames with the minimum interframe gap (IFG) as set in IPGIFG[Minimum IFG Enforcement]. If multiple frames are ready to transmit, the Ethernet controller follows the minimum IPG as long as the following restrictions are met:

- The first TxBD pointer, TBPTR*n*, of any given frame is located at a 16-byte aligned address.
- Each TxBD[Data Length] is greater-than or equal to 64 bytes.

If the first TxBD alignment restriction is not met, the back-to-back IPG may be as many as 32 cycles. If the TxBD size restriction is not met, the back-to-back IPG may be significantly longer.

In half-duplex mode, after a station begins sending, it continually checks for collisions on the LAN. If a collision is detected, the station forces a jam signal (all ones) on its frame and stops transmitting. Collisions usually occur close to the beginning of a packet. The station then waits a random time period (back-off) before attempting to send again. After the back-off completes, the station waits for silence on the LAN (carrier sense negated) and then begins retransmission (retry) on the LAN. Retransmission begins 36 bit times after carrier sense is negated for at least 60 bit times. If the frame is not successfully sent within a specified number of retries, an error is indicated (collision retry limit exceeded).

18.6.2.12 Internal and External Loop Back

Setting MACCFG1[Loop Back] causes the MAC transmit outputs to be looped back to the MAC receive inputs. Clearing this bit results in normal operation. This bit is cleared by default. Clearing this bit results in normal operation.

18.6.2.13 Error-Handling Procedure

The eTSEC reports frame reception and transmission error conditions using the channel BDs, the error counters, and the IEVENT register.

Transmission errors are described in Table 18-145.

Table 18-145. Transmission Errors

Error	Response
Transmitter underrun	Transmitter underrun can occur either after frame transmission has commenced, or in response to an incomplete sequence of TxBDs. In the former case, the controller sends 32 bits that ensure a CRC error, and terminates buffer transmission. In the latter case, the relevant transmit queue is halted. In all cases, the eTSEC closes the buffer, sets TxBD[UN], IEVENT[XFUN], and IEVENT[TXE]. The controller resumes transmission after TSTAT[THLT] is cleared (and DMACTRL[GTS] is cleared).
Retransmission attempts limit expired	The controller terminates buffer transmission, sets TxBD[RL], closes the buffer, IEVENT[CRL], and IEVENT[TXE]. Transmission resumes after TSTAT[THLT] is cleared (and DMACTRL[GTS] is cleared).

Table 18-145. Transmission Errors (continued)

Error	Response
Late collision	The controller terminates buffer transmission, sets TxBD[LC], closes the buffer, IEVENT[LC], and IEVENT[TXE]. The controller resumes transmission after TSTAT[THLT] is cleared (and DMACTRL[GTS] is cleared).
Memory read error	A system bus error occurred during a DMA transaction. The controller sets IEVENT[EBERR], DMA stops sending data to the FIFO which causes an underrun error, and therefore TxBD[UN] is set, but IEVENT[XFUN] is not set. The TSTAT[THLT] is set. Transmits are continued once TSTAT[THLT] is cleared.
Data parity error	Data in the transmit FIFO was potentially corrupted. The controller sets IEVENT[DPE], but otherwise continues transmission until halted explicitly.
Babbling transmit error	A frame is transmitted which exceeds the MAC's Maximum Frame Length and MACCFG2[Huge Frame] is a 0. The controller sets IEVENT[BABT] and continues without interruption.

Reception errors are described in Table 18-146.

Table 18-146. Reception Errors

Error	Description
Overrun error	The Ethernet controller maintains an internal FIFO buffer for receiving data. If a receiver FIFO buffer overrun occurs, the controller sets RxBD[OV], sets RxBD[L], closes the buffer, increments the discarded frame counter (RDRP), and sets IEVENT[RXF], The receiver then enters hunt mode (seeking start of a new frame).
Busy error	A frame is received and discarded due to a lack of buffers. The controller sets IEVENT[BSY] and increments the discarded frame counter (RDRP). In addition, the RSTAT[QHLT*n*] bit is set. RDRP increments for each frame that is received while the receiver is halted due to a busy condition. The halted queue resumes reception once the RSTAT[QHLT*n*] bit is cleared.
Filed frame to invalid queue error	A frame is received and discarded as a result of the filer directing it to an RxBD ring that is currently not enabled. The controller sets IEVENT[FIQ] and increments the discarded frame counter (RDRP).
Parser error	If the receive frame parser is enabled, a parse error can be flagged as a result of inconsistencies discovered between fields of the embedded packet headers. For example, the L2 header may indicate an IPv4 header, but the IP version number fails to match. In the event of a parse error, parsing is terminated at the inconsistent header, and the RxFCB[PERR] field indicates at which layer of the protocol stack the error was discovered. Receiver function continues regardless of parse errors, but IEVENT[PERR] is set. The receive queue filer may operate with reduced or default information in some cases; therefore, filer rule sets should be constructed so as to be tolerant of misformed frames. **Note:** Any values in the length/type field between 1500 and 1536 is treated as a length, however, only illegal packets exist with this length/type since these are not valid lengths and not valid types. These are treated by the MAC logic as out of range. Software must confirm the parser and filer results by checking the type/length field after the packet has been written to memory to see if it falls in this range.
Non-octet error (dribbling bits)	The Ethernet controller handles a nibble of dribbling bits if the receive frame terminates as non-octet aligned and it checks the CRC of the frame on the last octet boundary. If there is a CRC error, the frame non-octet aligned (RxBD[NO]) error is reported, IEVENT[RXF] is set, and the alignment error counter increments. The eTSEC relies on the statistics collector block to increment the receive alignment error counter (RALN). If there is no CRC error, no error is reported.

Table 18-146. Reception Errors (continued)

Error	Description
CRC error	If a CRC error occurs, the controller sets RxBD[CR], closes the buffer, and sets IEVENT[RXF]. This eTSEC relies on the statistics collector block to record the event. After receiving a frame with a CRC error, the receiver then enters hunt mode.
Memory read error	A system bus error occurred during a DMA transaction. The controller sets IEVENT[EBERR] and discards the frame and increments the discarded frame counter (RDRP). In addition the RSTAT[QHLT*n*] bit is set. The halted queue resumes reception once the RSTAT[QHLT*n*] bit is cleared.
Data parity error	Data in the receive FIFO or filer table was potentially corrupted. The controller sets IEVENT[DPE], but otherwise continues reception until halted explicitly.
Babbling receive error	A frame is received that exceeds the MAC's maximum frame length. The controller sets IEVENT[BABR] and continues.

18.6.3 TCP/IP Off-Load

Each eTSEC provides hardware support for accelerating the basic functions of TCP/IP packet transmission and reception. By default, these features are disabled and must be explicitly enabled through RCTRL and TCTRL. In this configuration, the eTSEC processes frames as vanilla Ethernet frames and none of the multi-ring QoS/CoS receive services or per-frame VLAN insertion and deletion are available. Operate eTSEC in this default configuration when using existing TCP/IP stack software that has not been modified to take advantage of TOE.

TOE can be enabled independently for Rx and Tx and at various levels. Receive TOE functions are controlled by RCTRL and transmit functions through a combination of TCTRL[TUCSEN] and the Tx frame control block.

On receive, according to RCTRL[PRSDEP], eTSEC can parse frames at layer 2 of the stack only (Ethernet headers and switching headers), layers 2 to 3 (including IPv4 or IPv6), or layers 2 to 4 (including TCP and UDP). TOE provides protocol header recognition, header verification (IPv4 header checksum verification), and TCP/UDP payload checksum verification including verification of associated pseudo-header checksums. For large frames off-load of checksum verification saves a significant fraction of the CPU cycles that would otherwise be spent by the TCP/IP stack. IP packet fragmentation and re-assembly, and TCP stream establishment and tear-down are not performed in hardware. The frame parser sets RQFPR[IPF] status flag encountering a fragmented frame. The frame parser in eTSEC searches a maximum of 512 bytes from the start of a received frame when attempting to locate headers; headers deeper than 512 bytes are assumed not to exist, and any associated receive status flags in the frame control block remain cleared.

On transmit, TOE provides IPv4 and TCP/UDP header checksum generation. Like receive TOE, checksum generation reduces CPU load significantly for TCP/IP stacks modified to exploit eTSEC TOE functions. The eTSEC does not checksum transmitted packets with IPv6 routing headers or calculate TCP/UDP checksums from IP fragments. If a transmitted TCP segment requires checksum generation but IPv6 extension headers would prevent eTSEC from calculating the pseudo-header checksum, software can calculate just the pseudo-header checksum in advance and supply it to the eTSEC as part of per-frame TOE configuration.

18.6.3.1 Frame Control Blocks

Frame control blocks (FCBs) are 8-byte blocks of TOE control and/or status data that are passed between software (driver and TCP/IP stack) and each eTSEC. A FCB always precedes the frame it applies to, and is present only when TOE functions are being used. As Figure 18-132 shows, the first BD of each frame points to the initial data buffer and the FCB. The initial data buffer must be at least 8 bytes long to contain the FCB without breaking it. Custom or received Ethernet preamble sequences also follow the FCB if preambles are visible.

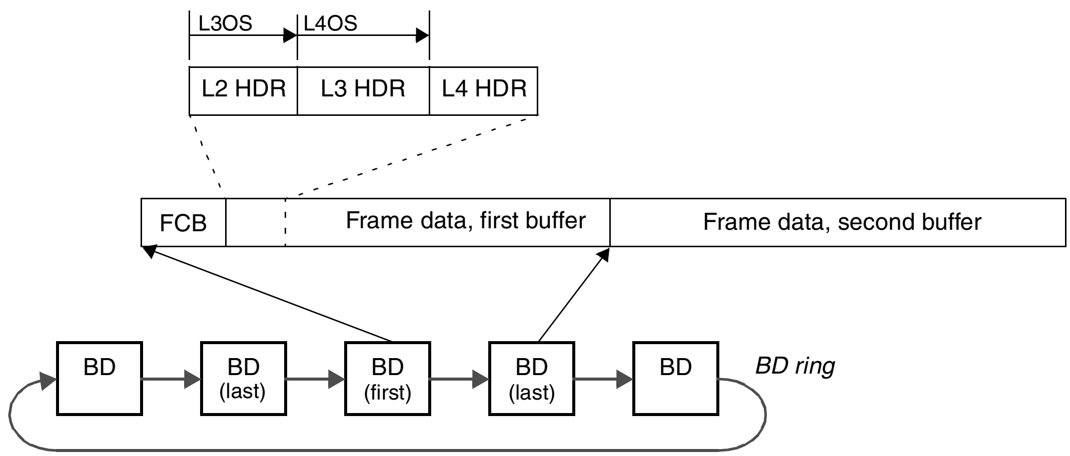


Figure 18-132. Location of Frame Control Blocks for TOE Parameters

For TxBD rings, FCBs are assumed present when the TxBD[TOE/UN] bit is set by user software. The eTSEC ignores the TxBD[TOE/UN] bit in all BDs other than those pointing to initial data buffers, therefore FCBs must not be inserted in second and subsequent data buffers. Since TxBD[TOE/UN] can be set under software discretion, TOE acceleration for transmit may be applied on a frame-by-frame basis.

In the case of RxBD rings, FCBs are inserted by the eTSEC whenever RCTRL[PRSDEP] is set to a non-zero value. Only one FCB is inserted per frame, in the buffer pointed to by the RxBD with bit F set. TOE acceleration for receive is enabled for all frames in this case.

18.6.3.2 Transmit Path Off-Load and Tx PTP Packet Parsing

TOE functions for transmit are defined by the contents of the Tx FCB. Figure 18-133 describes the definition for the Tx FCB.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	VLN	IP	IP6	TUP	UDP	CIP	CTU	NPH								PTP
Offset + 2	L4OS								L3OS							
Offset + 4	PHCS															
Offset + 6	VLCTL															

Figure 18-133. Transmit Frame Control Block

The user instructs the Tx packet to be timestamped via setting bit 15 in the TxFCB to mark a PTP packet. TxFCB[VLCTL] can be translated as the Tx PTP packet identification number. BD[TOE] has to be set to enable transmit PTP packet time stamping. TxFCB[PTP] bit takes precedence over TxFCB[VLN] bit. It disables per packet VLAN tag insertion. On a PTP packet, VLAN tag can be inserted from the DFVLAN register. A proposed TxFCB update for the PTP packet is shown in Figure 18-140.

The contents of the Tx FCB are defined in Table 18-147.

Table 18-147. Tx Frame Control Block Description

Bytes	Bits	Name	Description
0–1	0	VLN	VLAN control word valid. This bit is ignored when the PTP bit is set. VLAN tag is read from the DFVLAN register if PTP=1. 0 Ignore VLCTL field. 1 If VLAN tag insertion is enabled for eTSEC, use the VLCTL field as the VLAN control word.
	1	IP	Layer 3 header is an IP header. 0 Ignore layer 3 and higher headers. 1 Assume that the layer 3 header is an IPv4 or IPv6 header, and take L3OS field as valid.
	2	IP6	IP header is IP version 6. Valid only if IP = 1. 0 IP header version is 4. 1 IP header version is 6.
	3	TUP	Layer 4 header is a TCP or UDP header. 0 Do not process any layer 4 header. 1 Assume that the layer 4 header is either TCP or UDP (see UDP bit), and offload checksumming on the basis that the IP header has no extension headers.
	4	UDP	UDP protocol at layer 4. 0 Layer 4 protocol is either TCP (if TUP = 1) or undefined. 1 Layer 4 protocol is UDP if TUP = 1.
0–1	5	CIP	Checksum IP header enable. 0 Do not generate an IP header checksum. 1 Generate an IPv4 header checksum.
	6	CTU	Checksum TCP or UDP header enable. 0 Do not generate a TCP or UDP header checksum. RFC 768 advises that UDP packets not requiring checksum validation should have their checksum field set to zero. 1 Generate a TCP header checksum if IP = 1 and TUP = 1 and UDP = 0.
	7	NPH	Disable calculation of TCP or UDP pseudo-header checksum. This bit should be set if IP options need to be consulted in forming the pseudo-header checksum, as eTSEC does not examine IP options or extension headers for TCP/IP offload on transmit. 0 Calculate TCP or UDP pseudo-header checksum as normal, assuming that the IP header has no options. 1 Do not calculate a TCP or UDP pseudo-header checksum, but instead use the value in field PHCS when determining the overall TCP or UDP checksum.
	8–14	—	Reserved
	15	PTP	Indication to the transmitter that this is a PTP packet. Enabling PTP disables per packet VLAN tag insertion. Instead, VLAN tag will be read from the DFVLAN when the PTP field is true. 0 Do not attempt to capture transmission event time 1 Valid PTP_ID field. When this packet is transmitted, capture the time of transmission. Must be clear if TMR_CTRL[TE] is clear.

Table 18-147. Tx Frame Control Block Description (continued)

Bytes	Bits	Name	Description
2–3	0–7	L4OS	Layer 4 header offset from start of layer 3 header. The layer 4 header starts L4OS octets after the layer 3 header if it is present. The maximum layer 3 header length supported is thus 255 bytes, which may prevent TCP/IP offload on particularly large IPv6 headers.
	8–15	L3OS	Layer 3 header offset from start of frame not including the 8 bytes for this FCB. The layer 3 header starts L3OS octets from the start of the frame including any custom preamble header that may be present. The maximum layer 2 header length supported is thus 255 bytes.
4–5	0–15	PHCS	Pseudo-header checksum (16-bit one's complement sum with carry wraparound, but without result inversion) for TCP or UDP packets, calculated by software. Valid only if NPH = 1.
6–7	0–15	VLCTL/ PTP_ID	VLAN control word for insertion in the transmitted VLAN tag. Valid only if VLN = 1.Tx PTP packet identification number. This number will be copied into the Tx PTP packet time stamp identification field. PTP field takes precedence over VLN field.

18.6.3.3 Receive Path Off-Load

Upon receive, the Rx FCB returns the status of frame parse and TOE functions applied to the accompanying frame. Figure 18-134 describes the definition for the Rx FCB.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	VLN	IP	IP6	TUP	CIP	CTU	EIP	ETU					PERR			
Offset + 2			RQ						PRO							
Offset + 4																
Offset + 6	VLCTL															

Figure 18-134. Receive Frame Control Block

The contents of the Rx FCB are defined in Table 18-148.

Table 18-148. Rx Frame Control Block Descriptions

Bytes	Bits	Name	Description
0–1	0	VLN	VLAN tag recognized. This bit is set only if RCTRL[VLEX] is set. 0 No VLAN tag recognized. 1 IEEE Std. 802.1Q VLAN tag found; VLAN control word in VLCTL is valid.
	1	IP	IP header found at layer 3.RCTRL[PRSDEP] must be set to 10 or 11 in order to enable IP discovery. See also IP6 bit of FCB. 0 No layer 3 header recognized. 1 An IP header was recognized at layer 3; the IANA protocol identifier for the next header can be found in PRO; see PRO for more information. If S/W is relying on the RxFCB for the parse results, any RxFCB[IP] bits set with the corresponding RxFCB[PRO] = 0xFF indicates a fragmented packet (or that this packet had a back-to-back IPv6 routing extension header). Additionally, RQFPR[IPF] (see Section 18.5.3.3.8, “Receive Queue Filer Table Property Register (RQFPR)”) indicates that the packet was fragmented.
	2	IP6	IP version 6 header found at layer 3. 0 No IPv6 header was found. 1 The layer 3 header was an IPv6 header provided IP = 1.
	3	TUP	TCP or UDP header found at layer 4. RCTRL[PRSDEP] must be set to 10 or 11 in order to enable TCP/UDP discovery. 0 No layer 4 header recognized. 1 The layer 4 header was recognized as either TCP (PRO = 0x06) or UDP (PRO = 0x11).
	4	CIP	IPv4 header checksum checked. RCTRL[PRSDEP] must be set to 10 or 11 in order to enable IPv4 checksum verification. 0 IPv4 header checksum not verified, either because verification was disabled or a valid IPv4 header could not be located. 1 IPv4 header checksum was verified by the eTSEC, and bit EIP indicates result.
	5	CTU	TCP or UDP header checksum checked. RCTRL[PRSDEP] must be set to 11 in order to enable layer 4 checksum verification. 0 TCP or UDP header checksum not verified, either because verification was disabled or a valid TCP or UDP header could not be located. If a UDP header with zero checksum was located, this bit is cleared in accordance with RFC 768. 1 TCP or UDP header checksum was verified by the eTSEC, and ETU indicates result.
	6	EIP	IPv4 header checksum verification error. Not valid unless CIP = 1. 0 No checksum error in IPv4 header. 1 Error in header checksum only if IP = 1 and IP6 = 0.
0–1	7	ETU	TCP or UDP header checksum verification error. Not valid unless CTU = 1. 0 No checksum error in TCP or UDP header. 1 Error in header checksum only if PRO = 0x06 or PRO = 0x11.
	8–11	—	Reserved
	12–13	PER	Parse error. 00 No error in L2 to L4 parse 01 Reserved 10 Inconsistent or unsupported L3 header sequence 11 Reserved
	14	—	Reserved
	15	GPFP	General-purpose filer event packet. This packet was filed based on matching a GPI rule sequence.

Table 18-148. Rx Frame Control Block Descriptions (continued)

Bytes	Bits	Name	Description
2–3	0–1	—	Reserved
	2–7	RQ	Receive queue index. This index was selected by the eTSEC Rx Filer (from a matching Filer rule's RQCTRL[Q] field) when it accepted the associated frame. If filing is not enabled, RQ is zero. Note that the 3 least significant bits of RQ correspond with the RxBD ring index whenever RCTRL[FSQEN] = 0.
	8–15	PRO	If IP = 1, PRO is set as follows: • PRO=0xFF for a fragment header or a back to back route header • PRO=0x*nn* for an unrecognized header, where *nn* is the next protocol field • PRO=(TCP/UDP header), as defined in the IANA specification, if TCP or UDP header is found If IP = 0, PRO is undefined. Note that the eTSEC parser logic stops further parsing when encountering an IP datagram that has indicated that it has fragmented the upper layer protocol. This in general means that there is likely no layer 4 header following the IP header and extension headers. eTSEC leaves the RxFCB[PRO] and RQFPR[L4P] fields 0xFF in this case, which usually means that there was no IP header seen. In this case RxFCB[IP] and optionally RxFCB[IP6] is set. IP header checksumming operates and performs as intended. Most of the time, the eTSEC updates the RxFCB[PRO] field and RQFPR[L4P] fileds with whatever value was found in the protocol field of the IP header. See Section 18.5.3.3.8, "Receive Queue Filer Table Property Register (RQFPR)," for a description of RQFPR.
4–5	0–15	—	Reserved
6–7	0–15	VLCTL	VLAN control word as per IEEE Std. 802.1Q standard. The lower 12 bits comprise the VLAN identifier. Valid only if VLN = 1.

18.6.4 Quality of Service (QoS) Provision

This section describes the quality of service support features of this device. It includes a parser which extracts vital packet properties and passes then to the filer which essentially acts as a frame classifier.

18.6.4.1 Receive Parser

The receive parser parses the incoming frame data and generates filer properties and frame control block (FCB). The receive parser composes of the Ethernet header parser and L3/L4 parser.

The Ethernet header parser parses only L2 (ethertype) headers. It is enabled by RCTRL[PRSDEP] != 0. It has the following key features:

- Extraction of 48-bit MAC destination and source addresses
- Extraction and recognition of the first 2-byte ethertype field
- Extraction and recognition of the final 2-byte ethertype field
- Extraction of 2-byte VLAN control field
- Walk through MPLS stack and find layer 3 protocol
- Walk through VLAN stack and find layer 3 protocol
- Recognition of the following ethertypes for inner layer parsing
  — LLC and SNAP header

— JUMBO and SNAP header
— IPV4
— IPV6
— VLAN
— MPLSU/MPLSM
— PPPOES
— ARP

For stack L2 (that is, more than one ethertypes) header, the Ethernet parser traverses through the header until it finds the last valid ethertype or the ethertype is unsupported. Below is a description of what the Ethernet header parser recognizes for stack L2 header.

Table 18-149. Supported Stack L2 Ethernet Headers

Column—Current L2 Ethertype Row—Next Supported L2 Ethertype	LLC/ SNAP	JUMBO/ SNAP	IPV4	IPV6	VLAN	MPLSU	MPLSM	PPOES	ARP
LLC/SNAP	N	N	Y	Y	Y	Y	Y	Y	Y
JUMBO/SNAP	N	N	Y	Y	Y	Y	Y	Y	Y
IPV4	N	N	N	N	N	N	N	N	N
IPV6	N	N	N	N	N	N	N	N	N
VLAN	Y	Y	Y	Y	Y	Y	Y	Y	Y
MPLSU	N	N	Y*	Y*	N	y	Y	N	N
MPLSM	N	N	Y*	Y*	N	Y	Y	N	N
PPOES	N	N	Y	Y	N	Y	Y	N	N
ARP	N	N	N	N	N	N	N	N	N
Note: * means that it is the next protocol									

The L3 parser is enabled by RCTRL[PRSDEP] = 10 or 11. It begins when the Ethernet parser ends and a valid IPv4/v6 ethertype is found. The L4 header is enabled by RCTRL[PRSDEP] = 11. It begins when the L3 parser ends and a valid TCP/UDP next protocol is found and no fragment frame is found. The primary functionalities of L3(IPv4/6) and L4(TCP/UDP) parsers are as follows:

- IP recognition (v4/v6, ARP, encapsulated protocol)
- IP header checksum verification
- IPv4/6 over IPv4/6 (tunneling)—parse headers and find layer 4 protocol
- IP layer 4 protocol/next header extraction

- Stop parsing on unrecognized next header/protocol
- IPv4 support
  - IPv4 source and destination addresses
  - 8-bit IPv4 type of service
  - IP layer 4 protocol / next header support
    - IPV4
    - IPV4 Fragment. Parser stops after a fragment is found
    - TCP/UDP
- IPv6 support
  - The first 4 bytes of the IPv6 source address extraction
  - The first 4 bytes of the IPv6 destination address extraction
  - IPv6 source address hash for pseudo header calculation
  - IPv6 destination address hash for pseudo header calculation
  - 8-bit IPv6 traffic class field extraction
  - Payload length field extraction
  - IP layer 4 protocol/next header support
    - IPV6
    - IPV6 fragment. Parser stops after a fragment is found
    - IPV6 route
    - IPV6 hop/destination
    - TCP/UDP
- L4 (TCP/UDP) support
  - Extraction of 16-bit source port number extraction
  - Extraction of 16-bit destination port number extraction
  - TCP checksum calculation (including pseudo header)
  - UDP checksum calculation if the checksum field is not zero (including pseudo header)

18.6.4.2 Receive Queue Filer

The receive queue filer receives protocol header properties extracted from the incoming frame by the eTSEC frame parse engine. A property is defined to be a field extracted from a packet header, such as a TCP port number or VLAN identifier. As soon as the last identifiable header has been recognized, the filer commences searching the receive queue filer table, comparing properties in the table against properties extracted from the frame. This table is illustrated in Figure 18-135. Software populates the table with property values, stored to the RQPROP field, and indicates how to match and interpret the properties by setting flags in the RQCTRL field. The eTSEC memory map provides access to these fields by way of an address register (RQFAR) and two porthole registers (RQFCR and RQFPR).

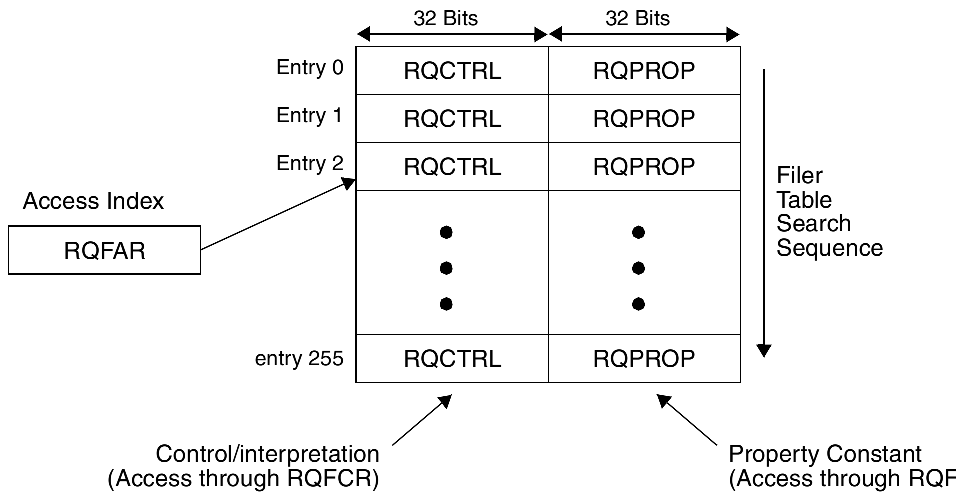


Figure 18-135. Structure of the Receive Queue Filer Table

18.6.4.2.1 Filing Rules

Unless the filer is disabled, every received frame from the Ethernet MAC initiates a search of the receive queue filer table, starting at entry 0. The table search is terminated as soon as an entry is found whose contents match a property of the frame. Accordingly, software must guarantee that at least one entry results in a match—even if only to set a default receive queue index.

Since eTSEC searches the table at a rate of two entries every system clock cycle, all 256 entries can be searched in the time taken to receive a 64-byte Ethernet frame.

Each entry of the receive queue filer table specifies a simple match rule for determining how to process the received frame. The elements of a filing rule, expressed in the RQCTRL and RQPROP fields, are summarized as follows:

- The PID field in RQCTRL identifies what property is being matched against RQPROP. The eTSEC supports 16 properties, some of which are different portions of the same header field. Reserved or unused bits in RQPROP are read as zero. See Section 18.5.3.3.8, "Receive Queue Filer Table Property Register (RQFPR)," on page 18-57 for a list of all properties and their associated PID values.
- The Q field in RQCTRL identifies which one of 64 virtual receive queues the frame should be filed to (sent through DMA) in the event of a filing rule match that accepts the frame. The physical RxBD ring this queue maps to is controlled by the RCTRL[FSQEN] bit. If RCTRL[FSQEN] = 0, the three least significant bits of the Q field indicate which physical RxBD ring hosts the queue. If RCTRL[FSQEN] = 1, RxBD ring 0 hosts all receive queues, but the RxFCB[RQ] field allows software to distinguish queues by ID. In all cases if Q maps to a RxBD ring that is not currently enabled, the frame is discarded with an IEVENT[FIQ] error.
- The REJ field in RQCTRL controls whether the frame is to be rejected (REJ = 1) or filed (REJ = 0) upon a filing rule match. Rejected frames occupy Rx FIFO space, but do not consume memory bus cycles.
- The CMP field in RQCTRL determines how property PID is compared against RQPROP. Equality, inequality, greater-or-equal, and less-than compares are available.

- The AND field in RQCTRL allows more than one comparison in a sequence to be chained together as a Boolean AND condition. Setting AND = 1 defers evaluation of the rule until the next entry has been matched, which may, in turn, have AND set. If any comparison involving AND = 1 fails, the entire chained sequence fails. A typical use for AND is to combine a pair of comparisons in a range match; the first such entry has AND = 1, the second has AND = 0 and its values of Q and REJ take effect.
- The CLE field in RQCTRL offers a way to bracket a set of consecutive—perhaps related—rules into a rule cluster. A cluster must be preceded by a guard rule, which simply determines whether the cluster rules can be evaluated. If the guard rule succeeds and its last entry has both CLE = 1 and AND = 1, the cluster rules that follow are enabled. The cluster ends at the first entry where CLE = 1 and AND = 0, which may also belong to a rule that files or rejects a frame. If the guard rule fails, all rules in the cluster are skipped, including mask_register assignments. Clusters must not be nested.
- The GPI field offers the user the ability to interrupt the core upon matching a rule that causes a frame to be filed to memory. Once the last RxBD corresponding to that frame is written to memory, the IEVENT[FGPI] event will be asserted. This bit will be set regardless of any interrupt coalescing that may be set.

18.6.4.2.2 Comparing Properties with Bit Masks

By default, extracted properties are compared arithmetically according to the CMP field in each RQCTRL word. This permits point value matches in each table entry, and range checks across a pair of table entries combined with the AND attribute in RQCTRL. However, inspection of the parse flags, Ethernet preamble, and IP addresses typically requires “don’t care” bit fields in the properties to be cleared as part of the comparison. The eTSEC provides a dedicated 32-bit register, known as the mask_register, for performing such masking operations. At the start of each table search by the filer, mask_register is reset to 0xFFFF_FFFF, which ensures that no masking occurs.

Filer rules may be configured to assign specific bit patterns to mask_register. Such rules can be configured to either match always (useful for implementing a default rule and specifying an associated receive queue), or fail always (which prevents termination of the filer table search). Once mask_register has been assigned, it retains its value until it is reassigned or the table search terminates. All properties are non-destructively bit-wise ANDed with mask_register prior to comparison in subsequent rules, which allows an entire cluster of rules to make use of a common mask. Individual masks for specific rules can also be created simply by combining a mask_register assignment (match always form) with a regular rule using the AND attribute.

To create a mask_register assignment rule, it is necessary to select PID = 0 in RQCTRL, and choose CMP such that the rule either matches (CMP = 01) or fails (CMP = 11). In this entry, RQPROP is then considered to be the assigned bit vector.

18.6.4.2.3 Special-Case Rules

It is frequently useful to create rules that are guaranteed to succeed or fail, specifically to enforce a default filing decision or act as null entries. Suggested constructions for such rules are shown in Table 18-150.

Table 18-150. Special Filer Rules

Rule Description	RQCTRL Fields						RQPROP Word	RQCTRL Word[1]
	CLE	REJ	AND	Q	CMP	PID		
Default file—Always file frame to ring Q	0	0	0	Q	01	0000	0x0000_0000	0x0000_*qq*20
Default reject—Always discard frame	0	1	0	000_000	01	0000	0x0000_0000	0x0000_0120
Empty rule in AND—Always matches	0/1[2]	0	1	000_000	01	0000	0xFFFF_FFFF	0x0000_00A0
Empty rule in rule set—Always fails	0/1[3]	0	0	000_000	11	0000	0xFFFF_FFFF	0x0000_0060

[1] Hexadecimal digits *qq* denotes field Q shifted left 2 bits.
[2] Set CLE = 1 if the empty rule guards a cluster.
[3] Set CLE = 1 if the empty rule occurs at the end of a cluster.

18.6.4.2.4 Filer Interrupt Events

The filer can produce three interrupt events in IEVENT. Event FIR indicates an error condition where the filer was unable to provide a definite result, either because no rule in the table succeeded, or because frames arrived too rapidly to complete searching of the table. Event FIQ indicates that the filer accepted a frame to a RxBD ring that was not enabled in RQCTRL (this can also occur if the filer is disabled, but RxBD ring 0—default queue or FSQEN mode queue—is not enabled). FIQ is also asserted in the case where no rule in the entire table succeeded. The various combinations of these interrupt events and their interpretation appear in Table 18-151.

Table 18-151. Receive Queue Filer Interrupt Events

IEVENT[FIR]	IEVENT[FIQ]	Description
0	0	No error. The filer successfully rejected or filed a frame.
0	1	Illegal queue error. The filer accepted a frame to a RxBD ring that is disabled (including ring 0 if filing is disabled).
1	0	Partial search error. The filer did not have sufficient time to complete its search of the filer table.
1	1	No matching rule error. The filer searched all 256 entries of the filer table without finding a rule that succeeds.

A functional interrupt is provided via use of the general purpose interrupt (GPI) bit in the filer table. When a property matches the value in the RQPROP entry at this index, and REJ = 0 and AND = 0, the filer will set IEVENT[FGPI] when the corresponding receive frame is written to memory. This allows the user to set up a filer rule where the core will be interrupted upon the reception of 'special' frames.

If the timer is enabled (TMR_CTRL[TE] = 1), then the interrupt dedicated for timer events (in addition to the usual receive, transmit and error interrupts) will be asserted.

18.6.4.2.5 Setting Up the Receive Queue Filer Table

The eTSEC frame parser always provides values for all properties, even where the relevant headers are not available. In the latter case, the filer is given default properties that can be used to avoid conflict with normal, defined property values. Accordingly, the rules in the filer table can be partitioned into rule sets such that if all rules in a given set fail (due to headers being unavailable), lower priority rule sets can be subsequently searched until either a rule set provides a match or a single default—catch-all—rule specifies a definite receive queue. For example, an IEEE 802.1p priority rule set may be followed by an IP TOS rule set, followed by a default rule; thus, if no VLAN tag appears in the received frame, the TOS rules are checked, or the default is activated should no IP header be present.

The rule cluster feature is used to conditionalize evaluation of rule sets. Typically, this avoids evaluating rules based on properties that may not be valid or relevant to the filing or filtering decision. For example, TCP-related rules might be clustered behind a guard rule that checks that a TCP header has appeared and the IP address matches our home address. Property 1—the parse flags property—is provided specifically to check the characteristics of the received frame and the parser error status. The mask_register is typically assigned beforehand to extract specific flags, in which case care should be taken that mask_register be reassigned an appropriate mask vector for following comparisons.

In many cases it is possible to write the entire filer table before using eTSEC, as the rule set is static. However, dynamic rule updates can be supported by pre-allocating partially instantiated rule sets, which software rewrites as necessary. Rules that are not instantiated should be composed of empty entries, as indicated in Table 18-150. In many cases empty entries can be overwritten by software without stopping eTSEC’s receive function.

18.6.4.2.6 Filer Example—802.1p Priority Filing

This example, shown in Table 18-152, illustrates how to file frames according to layer 2 802.1p priority. This matches against property 1001, comparing each specific priority level in order to associate them with a RxBD ring index. Note that if a VLAN tag does not appear in the frame, the parser passes priority 0 to the filer, which always matches the rule at entry 7 and terminate the table search.

Table 18-152. Filer Table Example—802.1p Priority Filing

Table Entry	RQCTRL Fields						RQPROP	Comment	RQCTRL Word
	CLE	REJ	AND	Q	CMP	PID			
0	0	0	0	000_000	00	1001	0x0000_0007	File priority 7 to ring 0	0x0000_0009
1	0	0	0	000_001	00	1001	0x0000_0006	File priority 6 to ring 1	0x0000_0409
2	0	0	0	000_010	00	1001	0x0000_0005	File priority 5 to ring 2	0x0000_0809
3	0	0	0	000_011	00	1001	0x0000_0004	File priority 4 to ring 3	0x0000_0C09
4	0	0	0	000_100	00	1001	0x0000_0003	File priority 3 to ring 4	0x0000_1009
5	0	0	0	000_101	00	1001	0x0000_0002	File priority 2 to ring 5	0x0000_1409

Table 18-152. Filer Table Example—802.1p Priority Filing (continued)

Table Entry	RQCTRL Fields						RQPROP	Comment	RQCTRL Word
	CLE	REJ	AND	Q	CMP	PID			
6	0	0	0	000_110	00	1001	0x0000_0001	File priority 1 to ring 6	0x0000_1809
7	0	0	0	000_111	00	1001	0x0000_0000	File undefined 802.1p or priority 0 to ring 7—Default always matches	0x0000_1C09

18.6.4.2.7 Filer Example—IP Diff-Serv Code Points Filing

This example demonstrates use of rule priority for determining class selector codepoints (RFC 2474) from the IP TOS property. An example filer table is shown in Table 18-153. The example relies on the fact that the first rule matched terminates the search, hence successively lower Diff-Serv codepoint ranges can be compared in each step until the default (zero or greater) range is reached. By default, property 1010 (IP TOS) takes the value 0x00 if no IP headers were recognized, therefore the table search always terminates.

Table 18-153. Filer Table Example—IP Diff-Serv Code Points Filing

Table Entry	RQCTRL Fields						RQPROP	Comment	RQCTRL Word
	CLE	REJ	AND	Q	CMP	PID			
0	0	0	0	001_000	01	1010	0x0000_00E0	File class 7 to queue 8 (TOS >= 0xE0)	0x0000_202A
1	0	0	0	001_001	01	1010	0x0000_00C0	File class 6 to queue 9 (TOS >= 0xC0)	0x0000_242A
2	0	0	0	001_010	01	1010	0x0000_00A0	File class 5 to queue 10 (TOS >= 0xA0)	0x0000_282A
3	0	0	0	001_011	01	1010	0x0000_0080	File class 4 to queue 11 (TOS >= 0x80)	0x0000_2C2A
4	0	0	0	000_100	01	1010	0x0000_0060	File class 3 to queue 4 (TOS >= 0x60)	0x0000_102A
5	0	0	0	001_100	01	1010	0x0000_0040	File class 2 to queue 12 (TOS >= 0x40)	0x0000_302A
6	0	0	0	010_100	01	1010	0x0000_0020	File class 1 to queue 20 (TOS >= 0x20)	0x0000_502A
7	0	0	0	011_100	01	1010	0x0000_0000	File class 0 to queue 28 (TOS >= 0x00) or file to ring 4 by default	0x0000_702A

18.6.4.2.8 Filer Example—TCP and UDP Port Filing

This example demonstrates rule clusters and AND-combined entries for filing packets based on transport protocol and well-known port numbers in a termination application. An example filer table is shown in Table 18-154. The example contains two clusters; the first is entered only for TCP packets, the second is entered only for UDP packets. A default filing rule catches the case where neither TCP nor UDP headers are found. Each cluster compares source port number (property 1111) against a list of server ports, and files the packets accordingly. Note that entries 1 and 2 form an AND rule for checking that the port number >= 20 and port number < 22. Entries 4 and 5 are initially set up to always fail (zero port number), and thus comprise empty entries that can be used at a later time.

Table 18-154. Filer Table Example—TCP and UDP Port Filing

Table Entry	RQCTRL Fields						RQPROP	Comment	RQCTRL Word
	CLE	REJ	AND	Q	CMP	PID			
0	1	0	1	000_000	00	1011	0x0000_0006	Enter cluster if layer 4 is TCP	0x0000_028B
1	0	0	1	000_000	01	1111	0x0000_0014	AND rule—FTP from TCP ports 20 and 21: file to ring 2	0x0000_00AF
2	0	0	0	000_010	11	1111	0x0000_0016		0x0000_086F
3	0	0	0	000_011	00	1111	0x0000_0017	telnet from TCP port 23: file to ring 3	0x0000_0C0F
4	0	0	0	000_000	00	1111	0x0000_0000	*empty entry reserved for future use*	0x0000_000F
5	0	0	0	000_000	00	1111	0x0000_0000	*empty entry reserved for future use*	0x0000_000F
6	1	0	0	000_001	01	0000	0x0000_0000	end cluster; default TCP: file to ring 1	0x0000_0620
7	1	0	1	000_000	00	1011	0x0000_0011	Enter cluster if layer 4 is UDP	0x0000_028B
8	0	0	0	000_101	00	1111	0x0000_0801	NFS from UDP port 2049	0x0000_140F
9	0	0	0	000_111	00	1111	0x0000_0208	Route from UDP port 520	0x0000_000F
10	0	0	0	000_110	00	1111	0x0000_0045	TFTP from UDP port 69	0x0000_180F
11	1	0	0	000_100	01	0000	0x0000_0000	End cluster; default UDP: file to ring 4	0x0000_1220
12	0	0	0	000_000	01	0000	0x0000_0000	By default, file to ring 0	0x0000_0020

18.6.4.3 Transmission Scheduling

Each eTSEC can maintain multiple TxBD rings (or transmission queues) to satisfy QoS requirements. The ability to choose from a number of transmission streams dynamically is especially important during periods of network congestion. Certain application such as voice and video streaming are delay sensitive, but loss insensitive. For instance, VoIP applications require little bandwidth, but are highly sensitive to latency. Conversely, FTP or SMTP protocols are delay insensitive, but loss sensitive.

eTSEC has a transmission scheduler that implements a programmable QoS regime. The scheduler is responsible for choosing which of the prefetched TxBDs shall be processed next, and accordingly issuing DMA requests to service the data stream described by the chosen BD(s). The scheduler cycle is one of:

1. decide on a TxBD queue,
2. transmit exactly one frame from that queue, and
3. return to deciding on another queue, in step 1.

If TCTRL[TXSCHED] is set to 00, no transmission scheduling occurs, and only TxBD ring 0 is polled for new data to transmit, with DMACTRL controlling waiting or polling. TCTRL[TXSCHED], if not zero, can be programmed to invoke one of two scheduling algorithms, namely priority-based queuing (PBQ), and modified weighted round-robin queuing (MWRR). In all cases where TCTRL[TXSCHED] is not zero, the scheduler can choose from among 1 to 8 TxBD rings per eTSEC, with individual rings being enabled by the setting of TQUEUE[EN0–EN7] bits. For example, TxBD rings 3, 4, and 7 may be enabled for scheduling by setting EN3, EN4, and EN7, and clearing all other EN bits.

18.6.4.3.1 Priority-Based Queuing (PBQ)

PBQ is the simplest scheduler decision policy. The enabled TxBD rings are assigned a priority value based on their index. Rings with a lower index have precedence over rings with higher indices. For example, TxBD ring 0 has higher priority than TxBD ring 1, and TxBD ring 1 has higher priority than TxBD ring 2, and so on.

The scheduling decision is then achieved as follows:

```
loop
      priority_ring = null;
      ring = 0;
      while priority_ring == null and ring <= 7 loop
       if enabled(ring) and not ring_empty(ring) then
              priority_ring = ring;
       endif
       ring = ring + 1;
      endloop
      if priority_ring >= 0 then
       while not ring_empty(priority_ring) loop
              transmit_frame(priority_ring);
       endloop
      endif
endloop
```

In practice a protocol stack or device driver can abuse PBQ by attempting to queue too much traffic onto high priority rings. It is recommended that the highest priority ring should normally not be used at all except for frames requiring the utmost urgent transmission. This allows emergency traffic to overtake backlogged queues out of sequence.

18.6.4.3.2 Modified Weighted Round-Robin Queuing (MWRR)

eTSEC implements a modified weighted round-robin scheduling algorithm across all enabled TxBD rings when TCTRL[TXSCHED] = 10. In MWRR, the weights in the TR03WT and TR47WT registers determine the ideal size of each transmit slot, as measured in multiples of 64 bytes. Thus, to set a transmit slot of 512 bytes, a weight of 512/64 or 8 needs to be set for the ring. In this mode TxBD rings 1–7 are selected in round-robin fashion, whereas TxBD ring 0, if enabled with ready data for transmission, is always selected in between other rings so as to expedite transmission from ring 0.

The scheduling decision is then achieved as follows:

```
for ring = 1..7 and enabled(ring) loop
      credit[ring] = 0;
endloop
for ring = 1..7 and enabled(ring) loop
      if not ring_empty(0) then
       credit[0] = credit[0] + weight[0];
       while credit[0] > 0 loop
              transmit_frame(0);
              credit[0] = credit[0] - frame_size;
              if ring_empty(0) then
               credit[0] = 0;
              endif
       endloop
      endif
```

```
        if not ring_empty(ring) then
         credit[ring] = credit[ring] + weight[ring];
        endif
        while credit[ring] > 0 loop
         transmit_frame(ring);
         credit[ring] = credit[ring] - frame_size;
         if ring_empty(ring) then
              credit[ring] = 0;
         endif
        endloop
    endloop
```

The algorithm checks registers TQUEUE[EN0–EN7] for `enabled()`, TSTAT[THLT0–THLT7] for `ring_empty()`, and TR*x*WT for `weight()`. For TxBD ring *k*, having a weight WT*k*, the long term average throughput for that ring is:

rate of queue[k] (K = 1 to 7) = (available bandwidth) × WT*k* ÷ (sum(WT*i*) + 6WT0)

rate of queue(0) = (available bandwidth) × 7 × WT0 ÷ (sum(WT*i*) + 6WT0)

where *i* = 0 to 7

18.6.5 Hardware Assist for IEEE Std. 1588-Compatible Timestamping

There is a push in industrial control applications to use Ethernet as the principal link layer for communications. This requires Ethernet to be used for both data transfer and real-time control. For real-time systems, each node is required to be synchronized to a master clock. The precision of this clock is dictated by the application, but generally needs to be of the order of <1uSec for high-speed machinery (for example, printing presses).

IEEE 1588 [1588] specifies a mechanism for synchronizing multiple nodes to a master clock. Support for 1588 can be done entirely in software running on a host CPU, but applications that require sub 10uSec accuracy will need hardware support for accurate timestamping of incoming packets.

The eTSEC includes a new timer clock module to support the IEEE Std. 1588 timer standard. The following sections describe the features, programming model, and implementation information.

NOTE

IEEE 1588 timestamping is not supported in conjunction with the SGMII 10/100 interface mode.

18.6.5.1 Features

- 64-bit free running timer running from an external oscillator or internal clock
- Programmable timer oscillator clock selection
- Self-correcting precision timer with nano-second resolution
- Time stamp all incoming packets inline
  - Maskable interrupts on received PTP packet’s filer rule match
- Time stamp transmit packets when instructed in the TxFCB
  - Maskable interrupts on transmit timestamp capture

- Two Tx time stamp registers per eTSEC with 16-bit tag for each of them to support burst mode.
- Time stamp capture on two general-purpose external triggers
  - — Maskable interrupts on GPIO timestamp trigger
  - — Programmable polarity of external trigger (GPIO) edge
- Two 64-bit alarm (future time) registers for future time comparison
  - — Maskable interrupts on alarm
- Three programmable timer output pulse period phase aligned with 1588 timer clock
  - — Maskable interrupts associated with each pulse
- Separate maskable timer interrupt event register
- Recognition of incoming PTP packet through filer rule match
- Phase aligned adjustable (divide by N) clock output
- Supports all Ethernet modes supported by the eTSEC, including full- and half-duplex modes
- Supports both master and slave modes
- Supports timestamp of nano-second resolution

18.6.5.2 Timer Logic Overview

The 1588 timer module can be partitioned into four different sub-modules as shown in Figure 18-136.

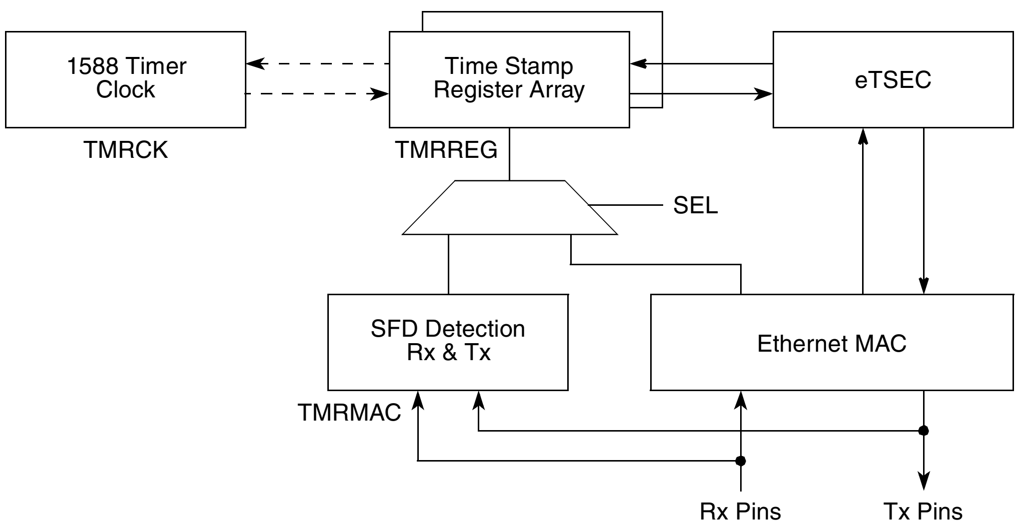


Figure 18-136. 1588 Timer Design Partition

18.6.5.3 Time-Stamp Insertion on the Received Packets

Every incoming packet’s 8-byte time stamp is inserted into the packet data buffer as padding alignment bytes. Time-stamp insertion into the data buffer requires RCTRL[PAL] to be set to a value greater than or equal to 8 and the control bit RCTRL[TS] bit to be set.

18.6.5.3.1 Timestamp Point

The required timestamp point, as specified in the IEEE 1588 Specification Sep-2004 (IEC 61588 First Edition), is shown in Figure 18-137. From this, it is clear that the end of the SFD is the critical point in the MII data stream.

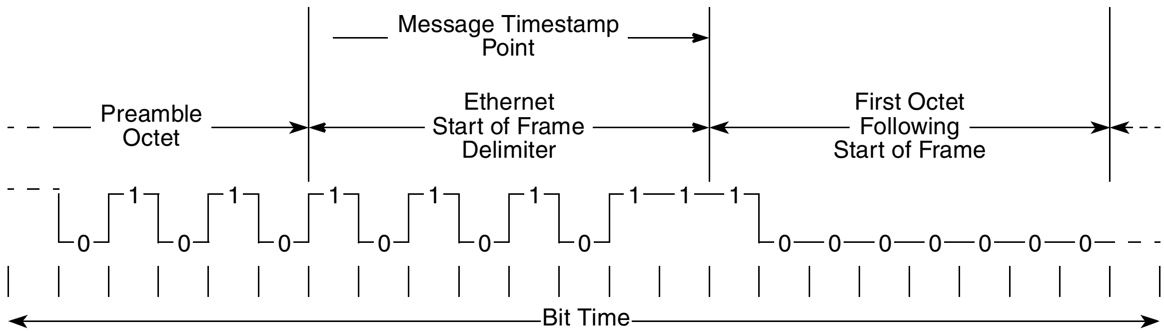


Figure 18-137. Ethernet Sampling Points for 1588

The sample point coincides with the cycle after the SFD (Start of Frame Delimiter) detection by the MAC. For received frames, this will be at least 4 bit times (MII) or 8 bit times (GMII) after the message timestamp point specified in [1588]. For transmission, the eTSEC sample point precedes the sample point specified in [1588] by at least 4-bit times (MII) or 8-bit times (GMII). For a particular mode, the eTSEC sample point is a consistent number of bit times relative to the SFD detection. Thus, the offset from the [1558] specified sample point can be accounted for in the PTP software implementation.

18.6.5.4 PTP Packet Parsing

PTP packets are typically embedded within a UDP payload with special IP source and destination address and special source and destination ports numbers. Special fields of interest of a PTP packet are listed in Table 18-155.

Table 18-155. PTP Payload Special Fields

Layer	Octet (Offset from the SFD)	Field	Value	eTSEC filer PID	Comments
Ethernet	12-13	Length/Packet	0x0800	ETY-RQPFR[PID=0111]	IPv4
IP header	22	Time to live	0x00	RBIFX-choose an arbitrary extraction byte	Must be 0
IP header	23	IP Protocol	0x11	L4P-RQPFR[PID=1011]	UDP
IP header	26-29	Source IP Address IANA defines 4 multicast address for the PTP packet		SIA-RQPFR[PID=1101]	

Table 18-155. PTP Payload Special Fields (continued)

Layer	Octet (Offset from the SFD)	Field	Value	eTSEC filer PID	Comments
IP header	30-33	Destination IP Address IANA defines 4 multicast address for the PTP packet	224.0.1.129 224.0.1.130 224.0.1.131 224.0.1.132	DIA-RQPFR[PID=1100]	DefaultPTPdomain AlternatePTPdomain1 AlternatePTPdomain2 AlternatePTPdomain3
UDP header	34-35	Source port number		SPT-RQPFR[PID=1011]	
UDP header	36-37	Destination port number	319 320	DPT-RQPFR[PID=1011]	EventPort GeneralPort
UDP data	74	Control	0x0 0x1 0x2 0x3 0x4	RBIFX-choose an arbitrary extraction byte	Sync Delay_req Follow_up Delay_resp Management

A representation of the PTP packet is shown in Figure 18-138.

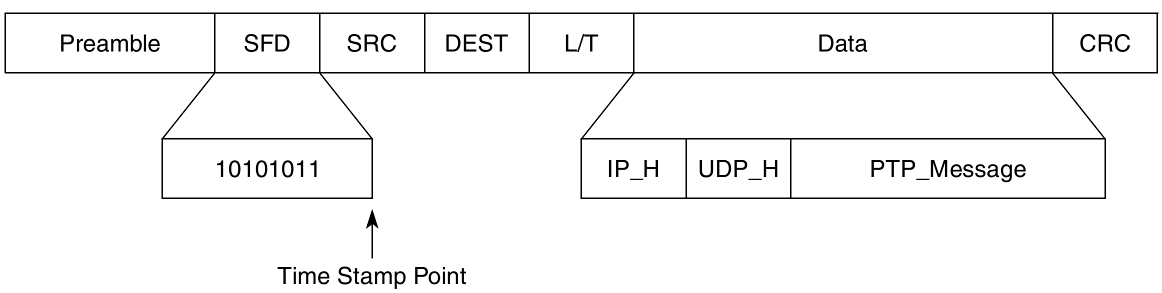


Figure 18-138. PTP Packet Format

18.6.5.4.1 General Purpose Filer Rule

The eTSEC receive filer has been enhanced with the addition of a general-purpose event bit. This event bit can be used in conjunction with filing table rules to identify 1588 packets and indicate these packets by setting special timer status register bits (TMR_STAT). Additionally, 1588 packets can be easily identified by upper-layer software by using the filer to queue all PTP packets to one or more predefined virtual queues. See Section 18.6.4.2.1, “Filing Rules for further information.

18.6.5.5 Time-Stamp Insertion on Transmit Packets

Software has the option to write the time stamp of the transmitted frame to memory in the padding alignment bytes (PAL) located between the TxFCB and the frame data. It is required that a minimum of two TxBDs are used. The first points to the start of the 8 byte TxFCB. The second points to the start of frame data. In memory, the TxFCB, and at least the first 16 bytes of the TxPAL must be adjacent, i.e., located in continguous memory locations, as depicted in Figure 18-139.

The first TxBD[TOE] bit is set. When the TMR_CTRL[Record Time-stamp In PAL Enable] and TxFCB[PTP] bits are set, the timestamp is written to memory location TxBD[Data Buffer Pointer]+16.

The second TxBD's Data Length must either contain the full frame length, or a value greater than the TxThreshold setting. Refer to Table 18-156. When time-stamps are inserted into the TxPAL, the TMR_TXTS*n*_H/L and TMR_TXTS*n*_ID registers still function normally.

18.6.5.5.1 Interrupts

The TxPAL is updated with a time-stamp before closing the second TxBD. The TxBD[I] bit can be set for the second TxBD frame to cause an interrupt (via IEVENT[TXF]) after the time-stamp has been written to the TxPAL.

When time-stamps are inserted into the TxPAL, the TMR_TXTS*n*_H/L and TMR_TXTS*n*_ID registers still function normally. Therefore, the 1588 interrupt can be triggered by using the TMR_PEVENT register bits TXP1, and TXP2.

Table 18-156. Time-Stamp Insertion Programming Requirements

Requirement	Behavior if requirement is not met
TMR_CTRL[RTPE]=1	If TMR_CTRL[RTPE]=0, then no time-stamp is written to a TxPAL.
TxBD[TOE]=1	If TxBD[TOE]=0, then no time-stamp is written to a TxPAL.
First TxBD[Data Buffer Pointer] is 8-byte aligned	The time-stamp will be written to address First TxBD[Data Buffer Pointer] + 0x10 rounded down to the nearest 8-byte aligned address, except at 4K page boundaries, in which case the time-stamp may be invalid, and the Second TxBD close status will be lost.
First TxBD[Data Length]=8, 8 bytes for TxFCB	If L2 or frame data is included in the Length, the buffer immediately following the FCB is transmitted on the line and the frame data stored in memory will be overwritten with a time-stamp value after the frame is transmitted.
TxFCB[PTP]=1	If TxBD[PTP]=0, then no time-stamp is written to a TxPAL.
The TxFCB is followed immediately by a minimum of 16 bytes for the TxPAL	The time-stamp will be written to address First TxBD[Data Buffer Pointer] + 0x10.
Second TxBD[Data Buffer Pointers] points to start of L2 or frame data	If there is only one TxBD used to transfer a PTP frame, then no time-stamp is written to a TxPAL.
Second TxBD[Data Length] >= FIFO_TX_THR or includes the entire frame	If this condition is not true, the time-stamp in TxPAL is invalid.

Figure 18-139 depicts the buffer format requirements for time-stamp insertion on transmit packets.

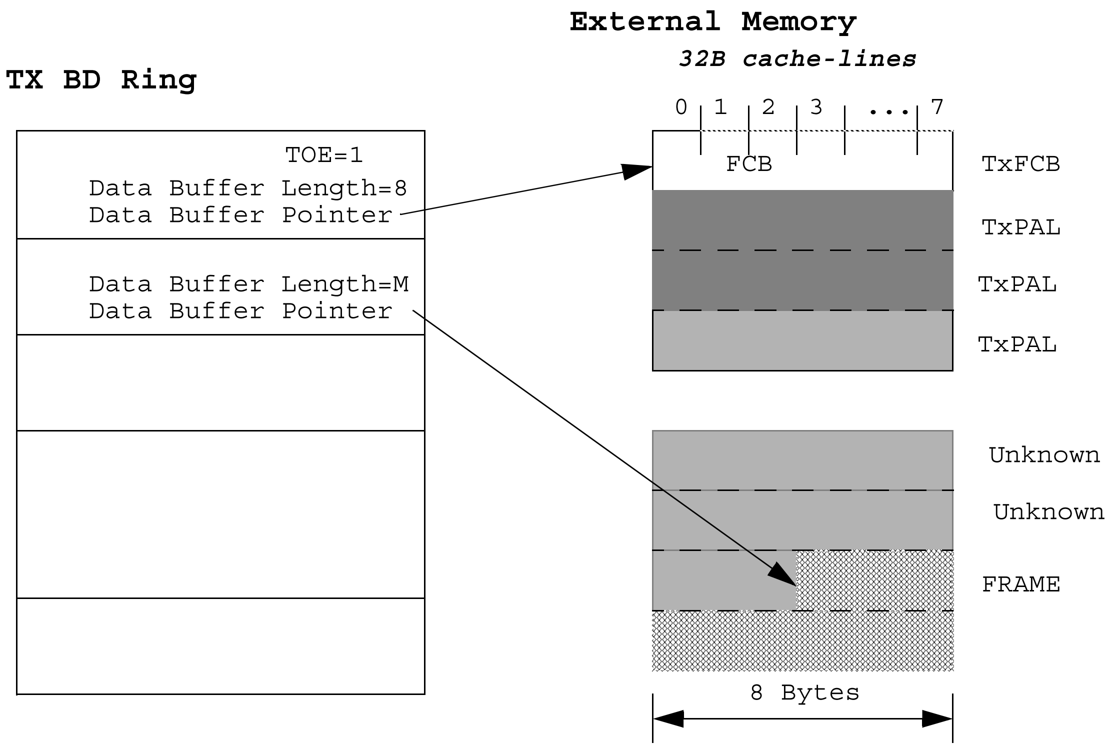


Figure 18-139. Buffer Format for Transmit Time-Stamp Insertion

18.6.5.5.2 Error Condition

When an error is encountered after a PTP packet has begun to be processed, the time-stamp written to the TxPAL is zero. Subsequent frames may be flushed by eTSEC. There will be no time-stamp update to TxPAL for the subsequent flushed frames.

18.6.5.6 Tx PTP Packet Parsing

Software instructs the Tx packet to be timestamped via setting bit 15 in the TxFCB to mark a PTP packet. TxFCB[VLCTL] can be translated as the Tx PTP packet identification number. BD[TOE] must be set to enable transmit PTP packet time stamping. TxFCB[PTP] bit takes precedence over TxFCB[VLN] bit. It disables per packet VLAN tag insertion. On a PTP packet, a VLAN tag can be inserted from the DFVLAN register. The TxFCB for the PTP packet is shown in Figure 18-140.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	VLN	IP	IP6	TUP	UDP	CIP	CTU	NPH								PTP
Offset + 2	L4OS								L3OS							
Offset + 4	PHCS															
Offset + 6	VLCTL/PTP_ID															

Figure 18-140. Transmit Frame Control Block

The contents of the Tx FCB are defined in Table 18-157.

Table 18-157. Tx Frame Control Block Description

Bytes	Bits	Name	Description
0–1	0	VLN	VLAN control word valid. This bit is ignored when the PTP bit is set. VLAN tag is read from the DFVLAN register if PTP=1. 0 Ignore VLCTL field. 1 If VLAN tag insertion is enabled for eTSEC, use the VLCTL field as the VLAN control word.
	1	IP	Layer 3 header is an IP header. 0 Ignore layer 3 and higher headers. 1 Assume that the layer 3 header is an IPv4 or IPv6 header, and take L3OS field as valid.
	2	IP6	IP header is IP version 6. Valid only if IP = 1. 0 IP header version is 4. 1 IP header version is 6.
	3	TUP	Layer 4 header is a TCP or UDP header. 0 Do not process any layer 4 header. 1 Assume that the layer 4 header is either TCP or UDP (see UDP bit), and offload checksumming on the basis that the IP header has no extension headers.
	4	UDP	UDP protocol at layer 4. 0 Layer 4 protocol is either TCP (if TUP = 1) or undefined. 1 Layer 4 protocol is UDP if TUP = 1.
0–1	5	CIP	Checksum IP header enable. 0 Do not generate an IP header checksum. 1 Generate an IPv4 header checksum.
	6	CTU	Checksum TCP or UDP header enable. 0 Do not generate a TCP or UDP header checksum. RFC 768 advises that UDP packets not requiring checksum validation should have their checksum field set to zero. 1 Generate a TCP header checksum if IP = 1 and TUP = 1 and UDP = 0.
	7	NPH	Disable calculation of TCP or UDP pseudo-header checksum. This bit should be set if IP options need to be consulted in forming the pseudo-header checksum, as eTSEC does not examine IP options or extension headers for TCP/IP offload on transmit. 0 Calculate TCP or UDP pseudo-header checksum as normal, assuming that the IP header has no options. 1 Do not calculate a TCP or UDP pseudo-header checksum, but instead use the value in field PHCS when determining the overall TCP or UDP checksum.
	8–14	—	Reserved
	15	PTP	Indication to the transmitter that this is a PTP packet. Enabling PTP disables per packet VLAN tag insertion. Instead, VLAN tag will be read from the DFVLAN when the PTP field is true. 0 Do not attempt to capture transmission event time 1 Valid PTP_ID field. When this packet is transmitted, capture the time of transmission. Must be clear if TMR_CTRL[TE] is clear.
2–3	0–7	L4OS	Layer 4 header offset from start of layer 3 header. The layer 4 header starts L4OS octets after the layer 3 header if it is present. The maximum layer 3 header length supported is thus 255 bytes, which may prevent TCP/IP offload on particularly large IPv6 headers.
	8–15	L3OS	Layer 3 header offset from start of frame not including the 8 bytes for this FCB. The layer 3 header starts L3OS octets from the start of the frame including any custom preamble header that may be present. The maximum layer 2 header length supported is thus 255 bytes.

Table 18-157. Tx Frame Control Block Description (continued)

Bytes	Bits	Name	Description
4–5	0–15	PHCS	Pseudo-header checksum (16-bit one's complement sum with carry wraparound, but without result inversion) for TCP or UDP packets, calculated by software. Valid only if NPH = 1.
6–7	0–15	VLCTL/ PTP_ID	VLAN control word for insertion in the transmitted VLAN tag. Valid only if VLN = 1.Tx PTP packet identification number. This number will be copied into the Tx PTP packet time stamp identification field. PTP field takes precedence over VLN field.

18.6.6 Buffer Descriptors

The eTSEC buffer descriptor (BD) is modeled after the MPC8260 Fast Ethernet controller BD for ease of reuse across the PowerQUICC network processor family. Drawing from the MPC8260 FEC BD programming model, the eTSEC descriptor base registers point to the beginning of BD rings. The eTSEC BD also expands upon the MPC8260 BD model to accommodate the eTSEC's unique features. However, the 8-byte data BD format is designed to be compatible with the existing MPC8260 BD model.

18.6.6.1 Data Buffer Descriptors

Data buffers are used in the transmission and reception of Ethernet frames (see Figure 18-141). Data BDs encapsulate all information necessary for the eTSEC to transmit or receive an Ethernet frame. Within each data BD there is a status field, a data length field, and a data pointer. The BD completely describes an Ethernet packet by centralizing status information for the data packet in the status field of the BD and by containing a data BD pointer to the location of the data buffer. Software is responsible for setting up the BDs in memory. Because of pre-fetching, a minimum of four buffer descriptors per ring are required. This applies to both the transmit and the receive descriptor rings. Transmit rings are limited to a maximum size of 65536 BDs due to BD and frame data prefetching. Software also must have the data pointer pointing to a legal memory location. Within the status field, there exists an ownership bit which defines the current state of the buffer (pointed to by the data pointer). Other bits in the status field of the buffer descriptor are used to communicate status/control information between the eTSEC and the software driver.

Because there is no next BD pointer in the transmit/receive BD (see Figure 18-142), all BDs must reside sequentially in memory. The eTSEC increments the current BD location appropriately to the next BD location to be processed. There is a wrap bit in the last BD that informs the eTSEC to loop back to the beginning of the BD chain. Software must initialize the TBASE and RBASE registers that point to the beginning transmit and receive BDs for eTSEC.

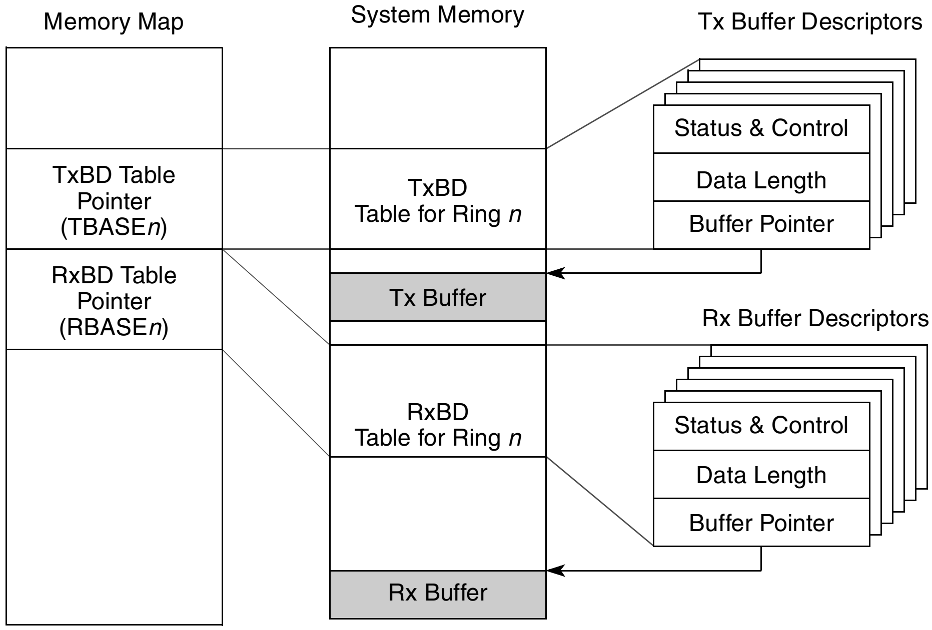


Figure 18-141. Example of eTSEC Memory Structure for BDs

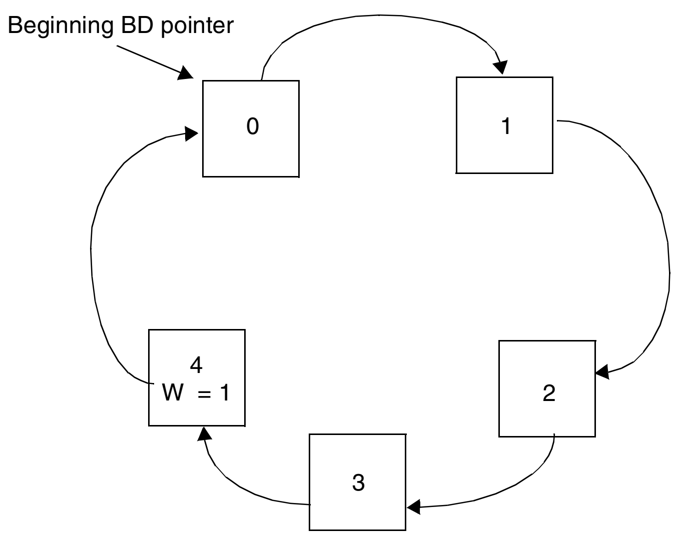


Figure 18-142. Buffer Descriptor Ring

18.6.6.2 Transmit Data Buffer Descriptors (TxBD)

Data is presented to the eTSEC for transmission by arranging it in memory buffers referenced by the TxBDs. In the TxBD the user initializes the R, PAD, W, I, L, TC, PRE, HFE, CF, and TOE bits and the length (in bytes) in the first word, and the buffer pointer in the second word. Unused fields or fields written by the eTSEC must be initialized to zero.

The eTSEC clears the R bit in the first word of the BD after it finishes using the data buffer. The transfer status bits are then updated. Additional transmit frame status can be found in statistic counters in the MIB block.

Software must expect eTSEC to prefetch multiple TxBDs, and for TCP/IP checksumming an entire frame must be read from memory before a checksum can be computed. Accordingly, the R bit of the first TxBD in a frame must not be set until at least one entire frame can be fetched from this TxBD onwards. If eTSEC prefetches TxBDs and fails to reach a last TxBD (with bit L set), it halts further transmission from the current TxBD ring and report an underrun error as IEVENT[XFUN]; this indicates that an incomplete frame was fetched, but remained unprocessed. The relevant TBPTR register points to the next unread TxBD following the error.

Figure 18-143 defines the TxBD.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	R	PAD/CRC	W	I	L	TC	PRE/DEF	0	HFE/LC	CF/RL	RC				TOE/UN	TR
Offset + 2	DATA LENGTH															
Offset + 4	TX DATA BUFFER POINTER															
Offset + 6																

Figure 18-143. Transmit Buffer Descriptor

The TxBD definition is interpreted by eTSEC hardware as if TxBDs mapped to C data structures in the manner illustrated by Figure 18-144.

```
typedef unsigned short uint_16; /* choose 16-bit native type */
typedef unsigned int uint_32; /* choose 32-bit native type */
typedef struct txbd_struct {
        uint_16 flags;
        uint_16 length;
        uint_32 bufptr;
} txbd;
```

Figure 18-144. Mapping of TxBDs to a C Data Structure

The TxBD fields are detailed in Table 18-158.

Table 18-158. Transmit Data Buffer Descriptor (TxBD) Field Descriptions

Offset	Bits	Name	Description
0–1	0	R	Ready, written by eTSEC and user. 0 The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or its associated data buffer. The eTSEC clears this bit after the buffer is transmitted or after an error condition is encountered. 1 The data buffer, which is prepared for transmission by the user, was not transmitted or is currently being transmitted. No fields of this BD may be written by the user once this bit is set.
	1	PAD/CRC	Padding for frames. (Valid only while it is set in the first BD and MACCFG2[PAD enable] is cleared). If MACCFG2[PAD enable] is set, this bit is ignored. 0 Do not add padding to short frames. 1 Add PAD to frames. PAD bytes are inserted until the length of the transmitted frame equals 64 bytes. Unlike the MPC8260 which PADs up to MINFLR value, the eTSEC PADs always up to the IEEE minimum frame length of 64 bytes. CRC is always appended to frames.
	2	W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location. 1 The next buffer descriptor is found at the location defined in TBASE.
	3	I	Interrupt. Written by user. 0 No interrupt is generated after this buffer is serviced. 1 IEVENT[TXB] or IEVENT[TXF] are set after this buffer is serviced. These bits can cause an interrupt if they are enabled (That is, IEVENT[TXBEN] or IEVENT[TXFEN] are set).
	4	L	Last in frame. Written by user. 0 The buffer is not the last in the transmit frame. 1 The buffer is the last in the transmit frame.

Table 18-158. Transmit Data Buffer Descriptor (TxBD) Field Descriptions (continued)

Offset	Bits	Name	Description
0–1	5	TC	Tx CRC. Written by user. (Valid only while it is set in first BD and TxBD[PAD/CRC] is cleared and MACCFG2[PAD/CRC enable] is cleared and MACCFG2[CRC enable] is cleared.) If MACCFG2[PAD/CRC enable] is set or MACCFG2[CRC enable] is set, this bit is ignored in ethernet modes.0End transmission immediately after the last data byte with no hardware generated CRC appended, unless TxBD[PAD/CRC] is set. 1 Transmit the CRC sequence after the last data byte.
	6	PRE	Transmit user-defined Ethernet preamble. Written by user. Valid only if set in the first BD of a frame, and MACCFG2[PreAm TxEN] is set. 0 This frame does not contain Ethernet preamble bytes for transmission. 1 This frame includes a user-defined Ethernet preamble sequence prior to the destination address in the data buffer.
		DEF	Defer indication. The eTSEC updates this bit after transmitting a frame (TxBD[L] is set) 0 This frame was not deferred. 1 This frame did not have a collision before it was sent but it was sent late because of deferring
	7	—	Reserved
	8	HFE	Huge frame enable. Written by user. Valid only if set in the first BD of a frame and MACCFG2[Huge Frame] is cleared. If MACCFG2[Huge Frame] is set, this bit is ignored. 0 Truncate transmit frame if its length is greater than the MAC's maximum frame length. 1 Allow large frames to be transmitted without truncation.
		LC	Late collision. Written by the eTSEC. 0 No late collision. 1 A collision occurred after 64 bytes are sent. The eTSEC terminates the transmission and updates LC.
	9	CF	Control Frame. Written by user. Valid only if set in the first BD of a frame. 0 Regular frame; transmission is deferred when eTSEC is in PAUSE. 1 Control frame; transmission starts even if eTSEC is in PAUSE.
		RL	Retransmission Limit. Written by the eTSEC. 0 Transmission before maximum retry limit is hit. 1 The transmitter failed (max. retry limit + 1) attempts to successfully send a message due to repeated collisions. The eTSEC terminates the transmission and updates RL.
	10–13	RC	Retry Count. Written by the eTSEC. 0 The frame is sent correctly the first time. x One or more attempts where needed to send the transmit frame. If this field is 15, then 15 or more retries were needed. The Ethernet controller updates RC after sending the buffer.

Table 18-158. Transmit Data Buffer Descriptor (TxBD) Field Descriptions (continued)

Offset	Bits	Name	Description
0–1	14	UN	Underrun. Written by the eTSEC. 0 No underrun encountered (data was retrieved from external memory in time to send a complete frame). 1 The Ethernet controller encountered a transmitter underrun condition while sending the associated buffer. This could also have occurred in relation to a bus error causing IEVENT[EBERR]. The eTSEC terminates the transmission and updates UN.
		TOE	TCP/IP off-load enable. Written by user. Valid only if set in the first BD of a frame. 0 No TCP/IP off-load acceleration is applied to the frame prior to transmission. 1 eTSEC looks for a TOE Frame Control Block preceding the frame, and applies TCP/IP off-load acceleration as controlled by the FCB.
	15	TR	Truncation. Written by the eTSEC. Set in the last TxBD (TxBD[L] is set) when IEVENT[BABT] occurs for a frame (a frame length greater than or equal to the value set in the maximum frame length register is encountered, the HFE bit in the BD is cleared, and MACCFG2[Huge Frame] is cleared). The frame is sent truncated.
2–3	0–15	Data Length	Data length is the number of octets the eTSEC should transmit from this BD's data buffer. It is never modified by the eTSEC. This field must be greater than zero, as zero indicates a BD not ready.
4–7	0–31	TX Data Buffer Pointer	The transmit buffer pointer contains the address of the associated data buffer. The data buffer pointer for the first BD of a TxPAL-enabled frame must be aligned on an 8-byte boundary. There are no alignment restrictions for the data buffer pointers of the second or subsequent BDs of a TxPAL-enabled frame, or for non-TxPAL frames.

18.6.6.3 Receive Buffer Descriptors (RxBD)

In the RxBD the user initializes the E, I, and W bits in the first word and the pointer in second word. If the data buffer is used, the eTSEC modifies the E, L, F, M, BC, MC, LG, NO, CR, OV, and TR bits and writes the length of the used portion of the buffer in the first word. The M, BC, MC, LG, NO, CR, OV, and TR bits in the first word of the buffer descriptor are only modified by the eTSEC if the L (last BD in frame) bit is set. The first word of the RxBD contains control and status bits. Its formats are detailed below.

The number of buffer descriptors in a ring is set using the W bit to indicate that the next buffer wraps back to the beginning of the ring. See Section 18.5.3.5.5, "Maximum Frame Length Register (MAXFRM)," for information on setting the size of the buffer ring.

Figure 18-145 defines the RxBD.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Offset + 0	E	RO1	W	I	L	F	0	M	BC	MC	LG	NO	SH	CR	OV	TR
Offset + 2	DATA LENGTH															
Offset + 4	RX DATA BUFFER POINTER															
Offset + 6																

Figure 18-145. Receive Buffer Descriptor

The RxBD definition is interpreted by eTSEC hardware as if RxBDs mapped to C data structures in the manner illustrated by Figure 18-146.

```
typedef unsigned short uint_16; /* choose 16-bit native type */
typedef unsigned int uint_32; /* choose 32-bit native type */
typedef struct rxbd_struct {
        uint_16 flags;
        uint_16 length;
        uint_32 bufptr;
} rxbd;
```

Figure 18-146. Mapping of RxBDs to a C Data Structure

Table 18-159 describes the fields of the RxBD.

Table 18-159. Receive Buffer Descriptor Field Descriptions

Offset	Bits	Name	Description
0–1	0	E	Empty, written by the eTSEC (when cleared) and by the user (when set). 0 The data buffer associated with this BD is filled with received data, or data reception is aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
	1	RO1	Receive software ownership bit. This field is reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware.
	2	W	Wrap, written by user. 0 The next buffer descriptor is found in the consecutive location. 1 The next buffer descriptor is found at the location defined in RBASE.
	3	I	Interrupt, written by user. 0 No interrupt is generated after this buffer is serviced. 1 IEVENT[RXB] or IEVENT[RXF] are set after this buffer is serviced. This bit can cause an interrupt if enabled (IMASK[RXBEN] or IMASK[RXFEN]). If the user wants to be interrupted only if RXF occurs, then the user must disable RXB (IMASK[RXBEN] is cleared) and enable RXF (IMASK[RXFEN] is set).
	4	L	Last in frame, written by the eTSEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
	5	F	First in frame, written by the eTSEC. 0 The buffer is not the first in a frame. 1 The buffer is the first in a frame.
	6	—	Reserved
	7	M	Miss, written by the eTSEC. (This bit is valid only if the L-bit is set and eTSEC is in promiscuous mode.) This bit is set by the eTSEC for frames that were accepted in promiscuous mode, but were flagged as a “miss” by the internal address recognition; thus, while in promiscuous mode, the user can use the M-bit to quickly determine whether the frame was destined to this station. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.

Table 18-159. Receive Buffer Descriptor Field Descriptions (continued)

Offset	Bits	Name	Description
0–1	8	BC	Broadcast. Written by the eTSEC. (Only valid if L is set.) Is set if the DA is broadcast (FF-FF-FF-FF-FF-FF).
	9	MC	Multicast. Written by the eTSEC. (Only valid if L is set.) Is set if the DA is multicast and not BC.
	10	LG	Rx frame length violation, written by the eTSEC (only valid if L is set). A frame length greater than or equal to the maximum frame length was recognized; in this case LG is set regardless of the setting of MACCFG2[Huge Frame]. If MACCFG2[Huge Frame] is cleared, the frame is truncated to the value programmed in the maximum frame length register. This bit is valid only if the L bit is set.
	11	NO	Rx non-octet aligned frame, written by the eTSEC (only valid if L is set). A frame that contained a number of bits not divisible by eight was received.
	12	SH	Short frame, written by the eTSEC (only valid if L is set). A frame length less than the minimum 64B that is defined for ethernet. was recognized, provided RCTRL[RSF] is set.
	13	CR	Rx CRC error, written by the eTSEC (only valid if L is set). This frame contains a CRC error and is an integral number of octets in length.This bit is also set if a receive code group error is detected.
	14	OV	Overrun, written by the eTSEC (only valid if L is set). A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR and TR lose their normal meaning and are zero.
	15	TR	Truncation, written by the eTSEC (only valid if L is set). Set if the receive frame is truncated. This can happen if a frame length greater than the maximum frame length is received and MACCFG2[Huge Frame] is cleared. If this bit is set, the frame must be discarded and the other error bits must be ignored as they may be incorrect.
2–3	0–15	Data Length	Data length, written by the eTSEC. Data length is the number of octets written by the eTSEC into this BD's data buffer if L is cleared (the value is equal to MRBLR), or, if L is set, the length of the frame including CRC, FCB (if RCTRL[PRSDEP > 00), preamble (if MACCFG2[PreAmRxEn]=1), timestamp (if RCTRL[TS]=1) and any padding (RCTRL[PAL]).
4–7	0–31	RX Data Buffer Pointer	Receive buffer pointer, written by the user. The receive buffer pointer, which always points to the first location of the associated data buffer, must be 8-byte aligned. For best performance, use 64-byte aligned receive buffer pointer addresses. The buffer must reside in memory external to the eTSEC.

18.7 Initialization/Application Information

18.7.1 Interface Mode Configuration

This section describes how to configure the eTSEC in different supported interface modes. These include the following:

- MII
- RMII

- RGMII
- SGMII
- RTBI

The pinout, the data registers that must be initialized, as well as speed selection options are described.

18.7.1.1 MII Interface Mode

Table 18-160 describes the signal configurations required for MII interface mode.

Table 18-160. MII Interface Mode Signal Configuration

eTSEC Signals			MII Interface		
			Frequency [MHz] 25		
			Voltage [V] 3.3		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	leave unconnected		
TX_CLK	I	1	TX_CLK	I	1
TxD[0]	O	1	TxD[0]	O	1
TxD[1]	O	1	TxD[1]	O	1
TxD[2]	O	1	TxD[2]	O	1
TxD[3]	O	1	TxD[3]	O	1
TX_EN	O	1	TX_EN	O	1
TX_ER	O	1	TX_ER	O	1
RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RxD[0]	I	1
RxD[1]	I	1	RxD[1]	I	1
RxD[2]	I	1	RxD[2]	I	1
RxD[3]	I	1	RxD[3]	I	1
RX_DV	I	1	RX_DV	I	1
RX_ER	I	1	RX_ER	I	1
COL	I	1	COL	I	1
CRS	I	1	CRS	I	1
Sum		17	**Sum**		16

Table 18-161 describes the shared signals of the MII interface.

Table 18-161. Shared MII Signals

eTSEC Signals	I/O	No. of Signals	MII Signals	I/O	No. of Signals	Function
MDIO	I/O	1	MDIO	I/O	1	Management interface I/O
MDC	O	1	MDC	O	1	Management interface clock
ECGTX_CLK125	I	1	not used	I	0	Reference clock
Sum			**Sum**			

Table 18-162 describes the register initializations required to configure the eTSEC in MII mode.

Table 18-162. MII Mode Register Initialization Steps

Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACCFG2, for MII, half duplex operation. Set I/F Mode bit, MACCFG2[0000_0000_0000_0000_0111_0001_0000_0100] (This example has Full Duplex = 0, Preamble count = 7, PAD/CRC append = 1)
Initialize ECNTRL, ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000] (This example has Statistics Enable = 1)
Initialize MAC Station Address, MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] Set station address to 02_60_8C_87_65_43, for example.
Initialize MAC Station Address, MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] Set station address to 02_60_8C_87_65_43, for example.
Reset the management interface. MIIMCFG[1000_0000_0000_0000_0000_0000_0000_0111]
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101] set source clock divide by 14 for example to insure that MDC clock speed is not greater than 2.5 MHz
Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the eTSEC MII Mgmt bus is idle.
Set up the MII Mgmt for a write cycle to the external PHY Auxiliary Control and Status Register to configure the PHY through the Management interface (overrides configuration signals of the PHY). MIIMADD[0000_0000_0000_0000_0000_0000_0001_1100]
Perform an MII Mgmt write cycle to the external PHY Writing to MII Mgmt Control with 16-bit data intended for the external PHY register, MIIMCON[0000_0000_0000_0000_0000_0000_0000_0100]

Table 18-162. MII Mode Register Initialization Steps (continued)

Check to see if MII Mgmt write is complete Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Set up the MII Mgmt for a write cycle to the external PHY Extended PHY control register #1 to set up the interface mode selection. MIIMADD[0000_0000_0000_0000_0000_0000_0001_0111]
Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY register, MIIMCON[0000_0000_0000_0000_0000_0000_0000_0000]
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Set up the MII Mgmt for a write cycle to the external PHY Mode control register to set up the interface mode selection. MIIMADD[0000_0000_0000_0000_0000_0000_0000_0000]
Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY register, MIIMCON[0000_0000_0000_0000_00uu_00uu_0u00_0000] where u is user defined based on desired configuration.
Check to see if MII Mgmt write is complete Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
If auto-negotiation was enabled in the PHY, check to see if PHY has completed Auto-Negotiation. Set up the MII Mgmt for a read cycle to PHY MII Mgmt register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0000_0000_0000_0001] The PHY Status register is at address 0x1 and in this case the PHY Address is 0x00.
Perform an MII Mgmt read cycle of Status Register. Clear MIIMCOM[Read Cycle]. Set MIIMCOM[Read Cycle]. (Uses the PHY address (0) and Register address (1) placed in MIIMADD register), When MIIMIND[BUSY]=0, read the MIIMSTAT register and check bit 10 (AN Done and Link is up) MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0100] Other information about the link is also returned.(Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability)
Check auto-negotiation attributes in the PHY as necessary.
Clear IEVENT register, IEVENT[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize IMASK (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACnADDR1/2 (Optional) MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n* (Optional) GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize RCTRL (Optional) RCTRL[0000_0000_0000_0000_0000_0000_0000_0000]

Table 18-162. MII Mode Register Initialization Steps (continued)

Initialize DMACTRL (Optional) DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize (Empty) Transmit Descriptor ring and fill buffers with Data Initialize TBASE0–TBASE7, TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE0–RBASE7, RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Transmit Queues Initialize TQUEUE
Enable Receive Queues Initialize RQUEUE
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

18.7.1.2 RGMII Interface Mode

Table 18-163 shows the signals configurations required for RGMII interface mode.

Table 18-163. RGMII Interface Mode Signal Configuration

eTSEC Signals			RGMII Interface		
			Frequency [MHz] 125		
			Voltage [V] 2.5		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1	not used		
TxD[0]	O	1	TxD[0]/TxD[4]	O	1
TxD[1]	O	1	TxD[1]/TxD[5]	O	1
TxD[2]	O	1	TxD[2]/TxD[6]	O	1
TxD[3]	O	1	TxD[3]/TxD[7]	O	1
TX_EN	O	1	TX_CTL (TX_EN/TX_ERR)	O	1
TX_ER	O	1	leave unconnected		
RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RxD[0]/RxD[4]	I	1
RxD[1]	I	1	RxD[1]/RxD[5]	I	1
RxD[2]	I	1	RxD[2]/RxD[6]	I	1
RxD[3]	I	1	RxD[3]/RxD[7]	I	1
RX_DV	I	1	RX_CTL (RX_DV/RX_ERR)	I	1

Table 18-163. RGMII Interface Mode Signal Configuration (continued)

eTSEC Signals			RGMII Interface		
			Frequency [MHz] 125		
			Voltage [V] 2.5		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
RX_ER	I	1	not used		
COL	I	1	not used		
CRS	I	1	not used		
Sum		17	**Sum**		12

Table 18-164 describes the shared signals for the RGMII interface.

Table 18-164. Shared RGMII Signals

eTSEC Signals	I/O	No. of Signals	GMII Signals	I/O	No. of Signals	Function
MDIO	I/O	1	MDIO	I/O	1	Management interface I/O
MDC	O	1	MDC	O	1	Management interface clock
GTX_CLK125	I	1	GTX_CLK125	I	1	Reference clock
Sum			**Sum**			

Table 18-165 describes the register initializations required to configure the eTSEC in RGMII mode.

Table 18-165. RGMII Mode Register Initialization Steps

Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACCFG2, MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101] (I/F Mode = 2, Full Duplex = 1)
Initialize ECNTRL, ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000] (This example has RGMII 10Mbps mode, Statistics Enable = 1)
Initialize MAC Station Address, MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] to 02608C:876543, for example.
Initialize MAC Station Address, MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] to 02608C:876543, for example.

Table 18-165. RGMII Mode Register Initialization Steps (continued)

Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101] Set source clock divide by 14, for example, to insure that TSEC_MDC clock speed is not greater than 2.5 MHz.
Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the eTSEC MII Mgmt bus is idle.
Set up the MII Mgmt for a write cycle to external the PHY AN Advertisement register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0001_0000_0100] The AN Advertisement register is at offset address 0x04 from the external PHY address. (in this case 0x11)
Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY AN Advertisement register, MIIMCON[0000_0000_0000_0000_u0uu_uuuu_uuuu_uuuu] Where u must be selected by the user for proper system configuration.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Set up the MII Mgmt for a write cycle to the external PHY Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0001_0000_0000] The control register (CR) is at offset address 0x00 from the external PHY address. (in this case 0x11)
Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY Control register, MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000] This enables the external PHY to restart Auto-Negotiations using the configuration set in the AN Advertisement register.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Check to see if PHY has completed Auto-Negotiation. Set up the MII Mgmt for a read cycle to the PHY MII Mgmt register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0000_0010_0000_0001] The PHY Status register is at address 0x1 and in this case the PHY Address is 0x2.
Perform an MII Mgmt read cycle of Status Register. Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (2) and Register address (2) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MIIMSTAT register and check bit 10. (AN Done) MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000] Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability)

Table 18-165. RGMII Mode Register Initialization Steps (continued)

Perform an MII Mgmt read cycle of AN Expansion Register. Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0110] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x11) and Register address (6) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MII Mgmt AN Expansion register and check bits 13 and 14. (NP Able and Page Rx'd) MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110]
Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional) Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0101] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x11) and Register address (5) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex) MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_000x_1x10_0000]
Clear IEVENT register, IEVENT[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize IMASK (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACnADDR1/2 (Optional) MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n* (Optional) GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize RCTRL (Optional) RCTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize DMACTRL (Optional) DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize (Empty) Transmit Descriptor ring and fill buffers with Data Initialize TBASE0–TBASE7, TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE0–RBASE7, RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Transmit Queues Initialize TQUEUE
Enable Receive Queues Initialize RQUEUE
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

18.7.1.3 RMII Interface Mode

Table 18-166 shows the signals configurations required for RMII interface mode.

Table 18-166. RMII Interface Mode Signal Configuration

eTSEC Signals			RMII Interface		
			Frequency [MHz] 50		
			Voltage [V] 3.3		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	leave unconnected		
TX_CLK	I	1	REF_CLK	I	1
TxD[0]	O	1	TxD[0]	O	1
TxD[1]	O	1	TxD[1]	O	1
TxD[2]	O	1	leave unconnected		
TxD[3]	O	1	leave unconnected		
TX_EN	O	1	TX_EN	O	1
TX_ER	O	1	leave unconnected		
RX_CLK	I	1	leave unconnected		
RxD[0]	I	1	RxD[0]	I	1
RxD[1]	I	1	RxD[1]	I	1
RxD[2]	I	1	not used		
RxD[3]	I	1	not used		
RX_DV	I	1	CRS_DV	I	1
RX_ER	I	1	RX_ER	I	1
COL	I	1	not used		
CRS	I	1	not used		
Sum		17	**Sum**		8

Table 18-167 describes the shared signals for the RMII interface.

Table 18-167. Shared RMII Signals

eTSEC Signals	I/O	No. of Signals	GMII Signals	I/O	No. of Signals	Function
MDIO	I/O	1	MDIO	I/O	1	Management interface I/O
MDC	O	1	MDC	O	1	Management interface clock
TX_CLK	I	1	REF_CLK	I	1	Reference clock
Sum		3	**Sum**		3	

Table 18-168 describes the register initializations required to configure the eTSEC in RMII mode.

Table 18-168. RMII Mode Register Initialization Steps

Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACCFG2, MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101] (I/F Mode = 2, Full Duplex = 1)
Initialize ECNTRL, ECNTRL[0000_0000_0000_0000_0001_0000_0001_0000] (Used to setup Reduced-Pin mode = 1, and TBIM = 0,statistics enable = 1)
Initialize MAC Station Address MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] to 02608C:876543 for example
Initialize MAC Station Address MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] to 02608C:876543 for example
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_0000_0000_0000_1101] set system clock divide by 14 for example to insure that MDC clock speed = 2.5 MHz
Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the eTSEC MII Mgmt bus is idle.
Set up the MII Mgmt for a write cycle to external the PHY AN Advertisement register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0001_0000_0100] The AN Advertisement register is at offset address 0x04 from the external PHY address. (in this case 0x11)
Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY AN Advertisement register, MIIMCON[0000_0000_0000_0000_u0uu_uuuu_uuuu_uuuu] Where u must be selected by the user for proper system configuration.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Set up the MII Mgmt for a write cycle to the external PHY Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0001_0000_0000] The control register is at offset address 0x00 from the external PHY address. (in this case 0x11)
Perform an MII Mgmt write cycle to the external PHY. Write to MII Mgmt Control with 16-bit data intended for the external PHY Control register, MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000] This enables the external PHY to restart Auto-Negotiations using the configuration set in the AN Advertisement register.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.

Table 18-168. RMII Mode Register Initialization Steps (continued)

Check to see if PHY has completed Auto-Negotiation. Set up the MII Mgmt for a read cycle to the PHY MII Mgmt register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0000_0010_0000_0001] The PHY Status register is at address 0x1 and in this case the PHY Address is 0x2.
Perform an MII Mgmt read cycle of Status Register. Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (2) and Register address (1) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MIIMSTAT register and check bit 10. (AN Done) MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000] Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability)
Perform an MII Mgmt read cycle of AN Expansion Register. Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0110] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x11) and Register address (6) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MII Mgmt AN Expansion register and check bits 13 and 14. (NP Able and Page Rx'd) MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110]
Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional) Setup MIIMADD[0000_0000_0000_0000_0001_0001_0000_0101] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x11) and Register address (5) placed in MIIMADD register) When MIIMIND[BUSY]=0, read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex) MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_000x_x110_0000]
Setting up the MII Mgmt for a write cycle to TBI MII Mgmt register (write the TBI's address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_1011] the TBI control register is at offset address 0x11 from TBIPA
Perform an MII Mgmt write cycle Writing to MII Mgmt Control with 16-bit data intended for TBI's MII Mgmt control register (TBI control), MIIMCON[0000_0000_0000_0000_0000_0010_0001_0000] This configures the TBI control to GMII mode and AN sense
Check to see if MII Mgmt write is complete Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicate that the write cycle was completed
Perform an MII Mgmt read cycle (0ptional) Set MIIMCOM[Read Cycle] (Uses the TBI address and Register address placed in MIIMADD register), read the MIIMSTAT register and verify that MIIMSTAT ---> [0000_0000_0000_0000_0000_0010_0001_0000]
Check to see if PHY has completed Auto-Negotiation Setting up the MII Mgmt for a read cycle to PHY's MII Mgmt register (write the PHY's address and Register address), MIIMADD[0000_0000_0000_0000_0000_0010_0000_0010] the PHY Status control register is at address 0x2 and lets say the PHY Address is 0x2

Table 18-168. RMII Mode Register Initialization Steps (continued)

Perform an MII Mgmt read cycle of Status Register Set MIIMCOM[Read Cycle] (Uses the PHY address (2) and Register address (2) placed in MIIMADD register), read the MIIMSTAT register and check bit 10 (AN Done) MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000] other information about the link is also returned (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability)
Perform an MII Mgmt read cycle of AN Expansion Register MIIMADD[0000_0000_0000_0000_0000_0010_0000_0110] Set MIIMCOM[Read Cycle] (Uses the PHY address (2) and Register address (6) placed in MIIMADD register), read the MII Mgmt AN Expansion register and check bits 13 and 14 (NP Able and Page Rx'd) MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110]
Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register (Optional) MIIMADD[0000_0000_0000_0000_0000_0010_0000_0101] Set MIIMCOM[Read Cycle] (Uses the PHY address (2) and Register address (5) placed in MIIMADD register), read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10 (Half and Full Duplex) MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_00X_1110_0000]
Clear IEVENT register, IEVENT[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize IMASK (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n* (Optional) GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize RCTRL (Optional) RCTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize DMACTRL (Optional) DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize (Empty) Transmit Descriptor ring and fill buffers with Data Initialize TBASE0–TBASE7, TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE0–RBASE7, RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Transmit Queues Initialize TQUEUE
Enable Receive Queues Initialize RQUEUE
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

18.7.1.4 RTBI Interface Mode

Table 18-169 describes the signal configurations required for RTBI interface mode.

Table 18-169. RTBI Interface Mode Signal Configuration

eTSEC Signal s			RTBI Interface		
			Frequency [MHz] 125		
			Voltage [V] 2.5		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
GTX_CLK	O	1	GTX_CLK	O	1
TX_CLK	I	1	not used		
TxD[0]	O	1	TCG[0]/TCG[5]	O	1
TxD[1]	O	1	TCG[1]/TCG[6]	O	1
TxD[2]	O	1	TCG[2]/TCG[7]	O	1
TxD[3]	O	1	TCG[3]/TCG[8]	O	1
TX_EN	O	1	TCG[4]/TCG[9]	O	1
TX_ER	O	1	leave unconnected		
RX_CLK	I	1	RX_CLK	I	1
RxD[0]	I	1	RCG[0]/RCG[5]	I	1
RxD[1]	I	1	RCG[1]/RCG[6]	I	1
RxD[2]	I	1	RCG[2]/RCG[7]	I	1
RxD[3]	I	1	RCG[3]/RCG[8]	I	1
RX_DV	I	1	RCG[4]/RCG[9]	I	1
RX_ER	I	1	not used		
COL	I	1	not used		
CRS	I	1	not used		
Sum		17	**sum**		12

Table 18-170 describes the shared signals for the RTBI interface.

Table 18-170. Shared RTBI Signals

eTSEC Signals	I/O	No. of Signals	GMII Signals	I/O	No. of Signals	Function
MDIO	I/O	1	MDIO	I/O	1	Management interface I/O
MDC	O	1	MDC	O	1	Management interface clock
ECGTX_CLK125	I	1	GTX_CLK125	I	1	Reference clock
Sum		3	Sum		3	

Table 18-171 describes the register initializations required to configure the eTSEC in RTBI mode.

Table 18-171. RTBI Mode Register Initialization Steps

Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACCFG2, MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101] (I/F Mode = 2, Full Duplex = 1)
Initialize ECNTRL, ECNTRL[0000_0000_0000_0000_0001_0000_0000_0000] (This example has Statistics Enable = 1)
Initialize MAC Station Address, MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] to 02608C:876543, for example.
Initialize MAC Station Address, MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] to 02608C:876543, for example.
Assign a Physical address to the TBI, TBIPA[0000_0000_0000_0000_0000_0000_0001_0000] set to 16, for example.
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_0000_0000_0000_0101] Set source clock divide by 14, for example, to insure that TSEC_MDC clock speed is not greater than 2.5 MHz.
Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the eTSEC MII Mgmt bus is idle.
Set up the MII Mgmt for a read cycle to TBI's Control register (write the TBI's address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000] The control register (CR) is at offset address 0x0 from TBIPA.
Perform an MII Mgmt read cycle to verify state of TBI Control Register(0ptional) Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the TBI address and Register address placed in MIIMADD register), When MIIMIND[BUSY]=0, read the MIIMSTAT and look for AN Enable and other bit information.
Set up the MII Mgmt for a write cycle to TBI's AN Advertisement register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0100] The AN Advertisement register is at offset address 0x04 from the TBI's address. (in this case 0x10)
Perform an MII Mgmt write cycle to TBI. Write to MII Mgmt Control with 16-bit data intended for TBI's AN Advertisement register, MIIMCON[0000_0000_0000_0000_0000_0001_1010_0000] This advertises to the Link Partner that the TBI supports PAUSE and Full Duplex mode and does not support Half Duplex mode.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.

Table 18-171. RTBI Mode Register Initialization Steps (continued)

Set up the MII Mgmt for a write cycle to TBI's Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000] The control register (CR) is at offset address 0x00 from the TBI's address. (in this case 0x10)
Perform an MII Mgmt write cycle to TBI. Writing to MII Mgmt Control with 16-bit data intended for TBI's Control register, MIIMCON[0000_0000_0000_0000_0001_0010_0000_0000] This enables the TBI to restart Auto-Negotiations using the configuration set in the AN Advertisement register.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Check to see if PHY has completed Auto-Negotiation. Set up the MII Mgmt for a read cycle to the PHY MII Mgmt register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0001] The PHY Status control register is at address 0x1 and in this case the PHY Address is 0x10.
Perform an MII Mgmt read cycle of Status Register. Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (2) and Register address (2) placed in MIIMADD register), When MIIMIND[BUSY]=0, read the MIIMSTAT register and check bit 10 (AN Done) MIIMSTAT ---> [0000_0000_0000_0000_0000_0000_0010_0000] Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability)
Perform an MII Mgmt read cycle of AN Expansion Register. Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0110] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x10) and Register address (6) placed in MIIMADD register), When MIIMIND[BUSY]=0, read the MII Mgmt AN Expansion register and check bits 13 and 14. (NP Able and Page Rx'd) MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110]
Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional) Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0101] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x10) and Register address (5) placed in MIIMADD register), When MIIMIND[BUSY]=0, read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex) MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_00x_x110_0000]
Clear IEVENT register, IEVENT[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize IMASK (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACnADDR1/2 (Optional) MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n* (Optional) GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]

Table 18-171. RTBI Mode Register Initialization Steps (continued)

Initialize RCTRL (Optional) RCTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize DMACTRL (Optional) DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize (Empty) Transmit Descriptor ring and fill buffers with Data Initialize TBASE0–TBASE7, TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE0–RBASE7, RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Transmit Queues Initialize TQUEUE
Enable Receive Queues Initialize RQUEUE
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

18.7.1.5 SGMII Interface Support

Table 18-172. SGMII Interface Signal Configuration (4-Wire)

SerDes Signals			**SGMII Interface**		
Frequency [MHz] 1250			**Frequency [MHz] 1250**		
Voltage [V] LVDS			**Voltage [V] LVDS**		
Signals	**I/O**	**No. of Signals**	**Signals**	**I/O**	**No. of Signals**
TX*n*/$\overline{\text{TX}n}$	O	2	TXD	O	2
RX*n*/$\overline{\text{RX}n}$	I	2	RXD	I	2
Sum		4	**Sum**		4

SGMII mode initialization sequence is very similar to TBI mode initialization. Additional initialization is required for the SerDes. An example of SGMII mode initialization sequence is shown in Table 18-173.

NOTE

SGMII mode utilizes the internal TBI PHY. The internal TBI PHY only auto-negotiates at 1 Gbps. However, 10 Mbps and 100 Mbps speeds are supported in SGMII mode. It is recommended that the external PHY inform the MAC if the desired link speed is not 1 Gbps. Software can perform MII management cycles to determine the external PHY link speed and program ECNTRL and MACCFG2 accordingly.

Table 18-173. SGMII Mode Register Initialization Steps

Initialize SerDes to select SGMII. The initialization sequence should be prepended with SerDes initialization.
Set Soft_Reset, MACCFG1[1000_0000_0000_0000_0000_0000_0000_0000]
Clear Soft_Reset, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACCFG2, MACCFG2[0000_0000_0000_0000_0111_0010_0000_0101] (I/F Mode = 2, Full Duplex = 1) (Set I/F mode = 1 in SGMII 10/100 Mbps speed)
Initialize ECNTRL, ECNTRL[0000_0000_0000_0000_0001_0000_0010_0010] (This example has Statistics Enable = 1, TBIM = 1, SGMIIM = 1) (Set R100M = 1 in SGMII 100 Mbps speed)
Initialize MAC Station Address MACSTNADDR2[0110_0000_0000_0010_0000_0000_0000_0000] to 02608C:876543, for example.
Initialize MAC Station Address MACSTNADDR1[0100_0011_0110_0101_1000_0111_1000_1100] to 02608C:876543, for example.
Assign a Physical address to the TBI, TBIPA[0000_0000_0000_0000_0000_0000_0001_0000] set to 16, for example.
Setup the MII Mgmt clock speed, MIIMCFG[0000_0000_0000_0000_000_0000_0000_0101] set source clock divide by 14 for example to insure that MDC clock speed is not greater than 2.5 MHz
Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the eTSEC MII Mgmt bus is idle.
Set up the MII Mgmt for a read cycle to TBI's Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000] the control register (CR) is at offset address 0x00 from the TBI's address.
Perform an MII Mgmt read cycle to verify state of TBI Control Register (optional) Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the TBI address and Register address placed in MIIMADD register), When MIIMIND[BUSY] = 0, read the MIIMSTAT and look for AN Enable and other bit information.
Set up the MII Mgmt for a write cycle to TBICON register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0001_0001] The TBICON register is at offset address 0x11 from the TBI's address.
Perform an MII Mgmt write cycle to TBI. Writing to MII Mgmt Control with 16-bit data intended for TBICON register, MIIMCON[0000_0000_0000_0000_0000_0000_0010_0000] This sets TBI in single clock mode and MII Mode off to enable communication with SerDes.

Table 18-173. SGMII Mode Register Initialization Steps (continued)

Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Set up the MII Mgmt for a write cycle to TBI's AN Advertisement register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0100] The AN Advertisement register is at offset address 0x04 from the TBI's address.
Perform an MII Mgmt write cycle to TBI. Writing to MII Mgmt Control with 16-bit data intended for TBI's AN Advertisement register, MIIMCON[0000_0000_0000_0000_0000_0001_1010_0000] This advertises to the Link Partner that the TBI supports PAUSE and Full Duplex mode and does not support Half Duplex mode.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Additional SerDes setup as required
Set up the MII Mgmt for a write cycle to TBI's Control register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0000] the control register (CR) is at offset address 0x00 from the TBI's address.
Perform an MII Mgmt write cycle to TBI. Writing to MII Mgmt Control with 16-bit data intended for TBI's Control register, MIIMCON[0000_0000_0000_0000_0001_0011_0100_0000] This enables the TBI to restart Auto-Negotiations using the configuration set in the AN Advertisement register.
Check to see if MII Mgmt write is complete. Read MII Mgmt Indicator register and check for Busy = 0, MIIMIND ---> [0000_0000_0000_0000_0000_0000_0000_0000] This indicates that the write cycle was completed.
Check to see if PHY has completed Auto-Negotiation. Set up the MII Mgmt for a read cycle to PHY MII Mgmt register (write the PHY address and Register address), MIIMADD[0000_0000_0000_0000_0001_0000_0000_0001] The PHY Status control register is at address 0x1 and in this case the PHY Address is 0x10.
Perform an MII Mgmt read cycle of Status Register. Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (2) and Register address (2) placed in MIIMADD register), When MIIMIND[BUSY] = 0, read the MIIMSTAT register and check bit 10 (AN Done) MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110] Other information about the link is also returned. (Extend Status, No pre, Remote Fault, An Ability, Link status, extend Ability)
Perform an MII Mgmt read cycle of AN Expansion Register. Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0110] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x10) and Register address (6) placed in MIIMADD register), When MIIMIND[BUSY] = 0, read the MII Mgmt AN Expansion register and check bits 13 and 14 (NP Able and Page Rx'd) MII Mgmt AN Expansion ---> [0000_0000_0000_0000_0000_0000_0000_0110]

Table 18-173. SGMII Mode Register Initialization Steps (continued)

Perform an MII Mgmt read cycle of AN Link Partner Base Page Ability Register. (Optional) Setup MIIMADD[0000_0000_0000_0000_0001_0000_0000_0101] Clear MIIMCOM[Read Cycle] Set MIIMCOM[Read Cycle] (Uses the PHY address (0x10) and Register address (5) placed in MIIMADD register), When MIIMIND[BUSY] = 0, read the MII Mgmt AN Link Partner Base Page Ability register and check bits 9 and 10. (Half and Full Duplex) MII Mgmt AN Link Partner Base Page Ability ---> [0000_0000_0000_0000_0000_000x_1110_0000]
Clear IEVENT register, IEVENT[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize IMASK (Optional) IMASK[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize MACnADDR1/2 (Optional) MACnADDR1/2[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize GADDR*n* (Optional) GADDR*n*[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize RCTRL (Optional) RCTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize DMACTRL (Optional) DMACTRL[0000_0000_0000_0000_0000_0000_0000_0000]
Initialize (Empty) Transmit Descriptor ring and fill buffers with Data Initialize TBASE0–TBASE7, TBASE0–TBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Initialize (Empty) Receive Descriptor ring and fill with empty buffers Initialize RBASE0–RBASE7, RBASE0–RBASE7[LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_LLLL_L000]
Enable Transmit Queues Initialize TQUEUE
Enable Receive Queues Initialize RQUEUE
Enable Rx and Tx, MACCFG1[0000_0000_0000_0000_0000_0000_0000_0101]

Chapter 19
SerDes PHY

19.1 Introduction

The SerDes PHY block includes the following components:

- SerDes PHY
- Protocol multiplexer and converter per protocol
- Control registers and control logic
- Power-down/reset state machine for cold (power-on) or warm (software-initiated) reset of the SerDes, PCVTR, and controllers
- Interface with the clock controls

19.1.1 Overview

Figure 19-1 is a block diagram showing the functional blocks inside the SerDes PHY block and its connections to other modules in the processor.

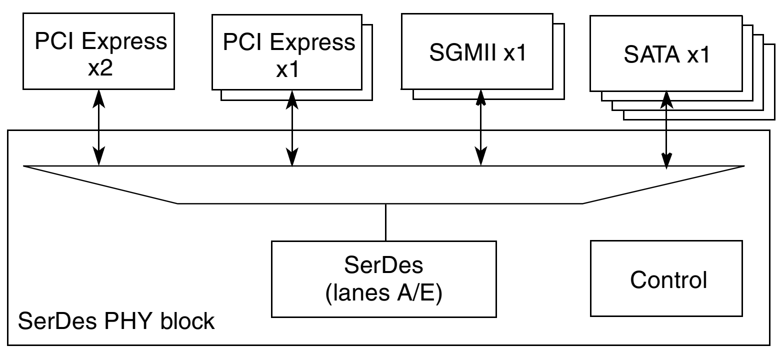


Figure 19-1. SerDes PHY Block Diagram

19.1.2 Features

The SerDes PHY block has the following features:

- Support for one ×2 PCI Express, two ×1 PCI Express, two ×1 SGMII, and four ×1 SATA
  — Gen1i, Gen1m, Gen2i, and Gen2m electrical specifications are supported in SATA mode, compliant to *Serial ATA 2.5 Specification*
- Link-layer interfaces to IP controller
- Memory-mapped registers with 256-byte address region

- SerDes power-down/reset state machine for cold (power-on) or warm (software-initiated) reset of SerDes, PHY, and controllers
- Provides *reset_done* indication
- Interface to clock controls

19.1.3 Modes of Operation

The SerDes PHY block supports the following modes of operation:

- SerDes1
  - — Two lanes running ×1 SGMII at 1.25 Gbps (MPC8378E)
  - — Two lanes running ×1 SATA at 1.5 or 3.0 Gbps (MPC8377E, MPC8379E)
- SerDes2
  - — Two lanes running ×1 PCI Express at 2.5 Gbps (MPC8377E, MPC8378E)
  - — One lane running ×2 PCI Express at 2.5 Gbps (MPC8377E, MPC8378E)
  - — Two lanes running ×1 SATA at 1.5 or 3.0 Gbps (MPC8379E)

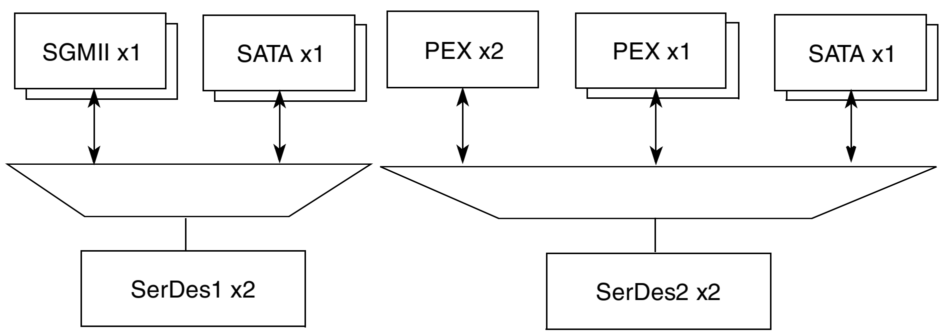


Figure 19-2. Modes of Operation

19.1.4 Clock

The SerDes control has one clock which is running at platform speed. This is an internally generated clock based off of the system clock.

19.2 External Signals

Table 19-1 describes the external signals to the SerDes PHY block.

Table 19-1. SerDes External Signals—Detailed Signal Descriptions

Signal	I/O	Description
L1_SD_IMP_CAL_RX L2_SD_IMP_CAL_RX	I	Receiver impedance calibration control signal. This pin requires an external resistor to ground to set the differential input impedance of the receivers. **State Meaning** Assertion/Negation—The SerDesacts as an integrated active impedance calibration circuit to ensure the best possible impedance control of the receiver's link termination resistors. The calibration circuit uses an externally established impedance against which internal impedance is calibrated. The user connects a 200 Ω, 1% tolerance resistor between the sd_imp_cal_rx input and ground. If the user wishes to set the impedance control to its nominal value, the sd_imp_cal_rx input maybe tied directly to the xcorevdd supply.
L1_SD_IMP_CAL_TX L2_SD_IMP_CAL_TX	I	Transmitter impedance calibration control signal. This pin requires an external resistor to ground to set the differential output impedance of the transmitters. **State Meaning** Asserted/Negated—The SerDes acts as an integrated active impedance calibration circuit to ensure the best possible impedance control of the transmitter's output impedance resistors. The calibration circuit uses and externally established impedance against which internal impedance is calibrated. The user connects a 100 Ω, 1% tolerance resistor between the sd_imp_cal_tx input and ground. If the user wishes to set the impedance control to its nominal value, the sd_imp_cal_tx input maybe tied directly to the xpadvdd supply.
L1_SD_REF_CLK L2_SD_REF_CLK	I	SerDes PLL reference clock, along with $\overline{\text{L1\_SD\_REF\_CLK}}$, is used by the SerDes PLL to generate all of the necessary clocks for the SerDes. **Timing** Assertion/Negation—Choices of input reference clock values are limited by specification, output bit rate and PLL functionality.
$\overline{\text{L1\_SD\_REF\_CLK}}$ $\overline{\text{L2\_SD\_REF\_CLK}}$	I	SerDes PLL reference clock complement, along with L1_SD_REF_CLK, is used by the SerDes PLL to generate all of the necessary clocks for the SerDes. **Timing** Assertion/Negation—Choices of input reference clock values are limited by specification, output bit rate and PLL functionality.
L1_SD_RXA L2_SD_RXA	I	Serial receiver data, lane A, positive. **Timing** Assertion/Negation—Serial differential receiver input which can be configured to meet either the PCI Express, SATA, or SGMII specifications.
$\overline{\text{L1\_SD\_RXA}}$ $\overline{\text{L2\_SD\_RXA}}$	I	Serial receiver data, lane A, complement. **Timing** Assertion/Negation—Serial differential receiver input which can be configured to meet either the PCI Express, SATA, or SGMII specifications.
L1_SD_RXE L2_SD_RXE	I	Serial receiver data, lane E, positive. **Timing** Assertion/Negation—Serial differential receiver input which can be configured to meet either the PCI Express, SATA, or SGMII specifications.
$\overline{\text{L1\_SD\_RXE}}$ $\overline{\text{L2\_SD\_RXE}}$	I	Serial receiver data, lane E, complement. **Timing** Assertion/Negation—Serial differential receiver input which can be configured to meet either the PCI Express, SATA, or SGMII specifications.

Table 19-1. SerDes External Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description
L1_SD_TXA L2_SD_TXA	O	Serial transmitter data, lane A, positive. **Timing** Assertion/Negation—Serial differential transmitter output which can be configured to meet either the PCI Express, SATA, or SGMII specifications.
$\overline{\text{L1\_SD\_TXA}}$ $\overline{\text{L2\_SD\_TXA}}$	O	Serial transmitter data, lane A, complement. **Timing** Assertion/Negation—Serial differential transmitter output which can be configured to meet either the PCI Express, SATA, or SGMII specifications.
L1_SD_TXE L2_SD_TXE	O	Serial transmitter data, lane E, positive. **Timing** Assertion/Negation—Serial differential transmitter output which can be configured to meet either the PCI Express, SATA, or SGMII specifications.
$\overline{\text{L1\_SD\_TXE}}$ $\overline{\text{L2\_SD\_TXE}}$	O	Serial transmitter data, lane E, complement. **Timing** Assertion/Negation—Serial differential transmitter output which can be configured to meet either the PCI Express, SATA, or SGMII specifications.

19.3 Memory Map/Registers

Table 19-2 lists the SerDes PHY block registers and their addresses. Reading undefined portions of the memory map returns all zeros; writing has no effect. All the registers are 32 bits wide.

Table 19-2. SerDes PHY Block Memory Map

Offset	Register	Access	Reset	Section/Page
0xE_3000	SRDS1CR0—SerDes1 Control Register 0	R/W	0x1100_CC30	19.3.1/19-5
0xE_3004	SRDS1CR1—SerDes1 Control Register 1	R/W	0x0000_0040	19.3.2/19-9
0xE_3008	SRDS1CR2—SerDes1 Control Register 2	R/W	0x0080_1C1C	19.3.3/19-11
0xE_300C	SRDS1CR3—SerDes1 Control Register 3	R/W	0x0101_0000	19.3.4/19-13
0xE_3010	SRDS1CR4—SerDes1 Control Register 4	R/W	0x*nn*00_0*n*0*n*	19.3.5/19-16
0xE_3014– 0xE_301C	Reserved	—	—	—
0xE_3020	SRDS1RSTCTL—SerDes1 Reset Control Register	R/W	0x0044_4500	19.3.6/19-17
0xE_3024– 0xE_30FC	Reserved	—	—	—
0xE_3100	SRDS2CR0—SerDes2 Control Register 0	R/W	0x1100_CC30	19.3.1/19-5
0xE_3104	SRDS2CR1—SerDes2 Control Register 1	R/W	0x0000_0040	19.3.2/19-9
0xE_3108	SRDS2CR2—SerDes2 Control Register 2	R/W	0x0080_1C1C	19.3.3/19-11
0xE_310C	SRDS2CR3—SerDes2 Control Register 3	R/W	0x0101_0000	19.3.4/19-13
0xE_3110	SRDS2CR4—SerDes2 Control Register 4	R/W	0x*nn*00_0*n*0*n*	19.3.5/19-16
0xE_3114– 0xE_311C	Reserved	—	—	—

Table 19-2. SerDes PHY Block Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0xE_3120	SRDS2RSTCTL—SerDes2 Reset Control Register	R/W	0x0044_4500	19.3.6/19-17
0xE_3124–0xE_31FC	Reserved	—	—	—

NOTE

Reserved bits should always be written with the value they return when read. That is, the register should be programmed by reading the value, modifying appropriate fields, and writing back the value.

19.3.1 SerDes*n* Control Register 0 (SRDS*n*CR0)

SRDS*n*CR0, shown in Figure 19-3, contains the functional control bits for the SerDes logic.

Offset 0x000 Access: Read/write

	0	1	2	3	4	5	6	7	8			11	12	13	14	15
R/W	TLCCA	—	RXEQA		TLCCE	—	RXEQE		—				EACCA	EACCE	—	
Reset	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0

	16	17		19	20	21		23	24	25	26	27	28	29	30	31
R/W	DPPA	TXEQA			DPPE	TXEQE			SDPD	—	IACCA	IACCE	—		RXEIA	RXEIE
Reset	1	1	0	0	1	1	0	0	0	0	1	1	0	0	0	0

Figure 19-3. SerDes*n* Control Register 0 (SRDS*n*CR0)

Table 19-3 defines the bit fields of SRDS*n*CR0.

Table 19-3. SRDS*n*CR0 Field Descriptions

Bits	Name	Description
0	TLCCA	Tracking loop centering control for lane A. When enabled, it centers the first-stage digital filter after the second-stage filter moves transition point. 0 Enable recentering algorithm 1 Disable recentering algorithm Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0
1	—	Reserved

Table 19-3. SRDS*n*CR0 Field Descriptions (continued)

Bits	Name	Description
2–3	RXEQA	Receive equalization selection bus for lane A—when asserted in PCI Express mode: 00 No equalization 01 2 dB of equalization 10 4 dB of equalization 11 Reserved RXEQA[2] when asserted in SGMII mode: 0 If $V_{diffp-p}$ on the receiver < 30 mV, the receiver detects loss of signal or if $V_{diffp-p}$ is > 100 mV, the receiver detects valid data. Therefore, if a signal comes across the link and 30 mV < Vd_{iffp-p} < 100 mV, the results from the receiver are indeterminate. 1 If $V_{diffp-p}$ on the receiver < 65 mV. the receiver detects loss of signal or if $V_{diffp-p}$ is > 175 mV, the receiver detects valid data. Basically if a signal comes across the link and 65 mV < $V_{diffp-p}$ < 175 mV, the results from the receiver are indeterminate. RXEQA[3] when asserted in SGMII mode: 0 No equalization 1 ~2 dB of equalization at 1.25 GHz Recommended setting per protocol: • SATA: 01 • PCI Express: 01 • SGMII: 01
4	TLCCE	Tracking loop centering control for lane E. When enabled, it centers the first-stage digital filter after the second-stage filter moves transition point. 0 Enable recentering algorithm 1 Disable recentering algorithm Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0
5	—	Reserved
6–7	RXEQE	Receive equalization selection bus for lane E—when asserted in PCI Express mode: 00 No equalization 01 2 dB of equalization 10 4 dB of equalization 11 Reserved RXEQA[2] when asserted in SGMII mode: 0 If $V_{diffp-p}$ on the receiver < 30 mV, the receiver detects loss of signal or if $V_{diffp-p}$ is > 100 mV, the receiver detects valid data. Therefore, if a signal comes across the link and 30 mV < $V_{diffp-p}$ < 100 mV, the results from the receiver are indeterminate. 1 If $V_{diffp-p}$ on the receiver < 65 mV, the receiver detects loss of signal or if $V_{diffp-p}$ is > 175 mV, the receiver detects valid data. Therefore, if a signal comes across the link and 65 mV < $V_{diffp-p}$ < 175 mV, the results from the receiver are indeterminate. RXEQA[3] when asserted in SGMII mode: 0 No equalization 1 ~2 dB of equalization at 1.25 GHz Recommended setting per protocol: • SATA: 01 • PCI Express: 01 • SGMII: 01
8–11	—	Reserved

Table 19-3. SRDS*n*CR0 Field Descriptions (continued)

Bits	Name	Description
12	EACCA	Used to set DC common mode bias on the receiver in lane A. Sets $V_{CM} = 0.7 \times V_{DD}$. Disables 50 Ω to ground termination. 0 Disable DC common mode bias 1 Set DC common mode bias Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0
13	EACCE	Used to set DC common mode bias on the receiver in lane E. Sets $V_{CM} = 0.7 \times V_{DD}$. Disables 50 Ω to ground termination. 0 Disable DC common mode bias 1 Set DC common mode bias Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0
14–15	—	Reserved
16	DPPA	Diff pk-pk swing for lane A. Sets the peak value for output swing of transmitters and the amount of transmit equalization for lane A. 0 $V_{DD\text{-diff-pk-pk}}$ 1 5/6 V_{DD}-diff-pk-pk Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0
17–19	TXEQA	Sets the peak value for output swing of transmitters and the amount of transmit equalization for lane A. Transmit equalization selection bus for lane A. If register field SRDSCR3[21:23] = 000, then the equalization definitions are: 000 No equalization 001 1.09x relative amplitude 010 1.2x relative amplitude 011 1.33x relative amplitude 100 1.5x relative amplitude 101 1.71x relative amplitude 110 2.0x relative amplitude 111 Reserved If register field SRDSCR3[21:23] = 110, then the equalization definitions are: 000 No equalization 001 1.2x relative amplitude 010 1.5x relative amplitude 011 2.0x relative amplitude 100 Reserved 101 Reserved 110 Reserved 111 Reserved Recommended setting per protocol: • SATA: don’t care • PCI Express: 100 • SGMII: 100

Table 19-3. SRDS*n*CR0 Field Descriptions (continued)

Bits	Name	Description
20	DPPE	Diff pk-pk swing for lane E. Sets the peak value for output swing of transmitters and the amount of transmit equalization for lane E. 0 $V_{DD\text{-diff-pk-pk}}$ 1 5/6 $V_{DD\text{-diff-pk-pk}}$ Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0
21-23	TXEQE	Sets the peak value for output swing of transmitters and the amount of transmit equalization for lane E. Transmit equalization selection bus for lane E. If register field SRDSCR3[29:31] = 000, then the equalization definitions are: 000 No equalization 001 1.09x relative amplitude 010 1.2x relative amplitude 011 1.33x relative amplitude 100 1.5x relative amplitude 101 1.71x relative amplitude 110 2.0x relative amplitude 111 Reserved If register field SRDSCR3[29:31]= 110, then the equalization definitions are: 000 No equalization 001 1.2x relative amplitude 010 1.5x relative amplitude 011 2.0x relative amplitude 100 Reserved 101 Reserved 110 Reserved 111 Reserved Recommended setting per protocol: • SATA: don't care • PCI Express: 100 • SGMII: 100
24	SDPD	SerDes power down. This power down signal shuts down the PLL, all of the receiver amplifiers, all of the samplers and places the transmitters in 3-state. 0 Application mode 1 Block power down
25	—	Reserved
26	IACCA	Used to set on-chip AC coupling in the receiver in lane A. 0 Disable on-chip AC coupling 1 Enable on-chip AC coupling Recommended setting per protocol: • SATA: 1 • PCI Express: 1 • SGMII: 1

Table 19-3. SRDS*n*CR0 Field Descriptions (continued)

Bits	Name	Description
27	IACCE	Used to set on-chip AC coupling in the receiver in lane E. 0 Disable on-chip AC coupling 1 Enable on-chip AC coupling Recommended setting per protocol: • SATA: 1 • PCI Express: 1 • SGMII: 1
28–29	—	Reserved
30	RXEIA	When asserted, places lane A into receiver electrical idle state. 0 Lane A is not ‘forced’ into receive electrical idle state 1 Place lane A into electrical idle state Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0
31	RXEIE	When asserted, places lane Einto receiver electrical idle state. 0 Lane E is not ‘forced’ into receive electrical idle state 1 Place lane E into electrical idle state Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0

19.3.2 SerDes*n* Control Register 1 (SRDS*n*CR1)

SRDS*n*CR1, shown in Figure 19-4, contains the functional control bits for the SerDes logic.

Individual lanes can be powered down using SRDS*n*CR1[0] and SRDS*n*CR1[4]. The entire SerDes must be reset to activate a lane from power-down. Refer to Section 19.3.6, “SerDesn Reset Control Register (SRDSnRSTCTL).”

Offset 0x04 Access: Read/write

Bits	0	1–3	4	5–7	8	9	10	11–13	14	15
R/W	PDA	—	PDE	—	PDTCK	—	X3SA	—	X3SE	—
Reset	All zeros									

Bits	16–17	18	19–20	21–24	25	26	27	28	29	30	31
R/W	—	X3STCK	—	LBSEL	PLLBW	PLLRST	SDRST	—	—	RPTA	RPTE
Reset	0 0	0	0 0	0 0 0 0	1	0	0	0	0	0	0

Figure 19-4. SerDes*n* Control Register 1 (SRDS*n*CR1)

Table 19-4 describes the SRDS*n*CR1.

Table 19-4. SRDS*n*CR1 Field Descriptions

Bits	Name	Description
0	PDA	Lane A power down. 0 Normal 1 Power down lane A
1–3	—	Reserved
4	PDE	Lane E power down. 0 Normal 1 Power down lane E
5–7	—	Reserved
8	PDTCK	Power down test clock.
9	—	Reserved
10	X3SA	Lane A transmitter three-state. 0 Normal 1 The transmitter output is disabled and place in a three-state condition
11–13	—	Reserved
14	X3SE	Lane E transmitter three-state. 0 Normal 1 The transmitter output is disabled and place in a three-state condition
15–17	—	Reserved
18	X3STCK	Test clock three state.
19-20	—	Reserved
21–24	LBSEL	Select type loop-back. 0000 Application mode 0001 Digital loopback both lanes 0010Analog loopback both lanes 0011 External loopback, both lanes 0101 Digital loopback, lane A only 0110 Analog loopback, lane A only 0111 External loopback, lane A only 1001 Digital loopback, lane E only 1010 Analog loopback, lane E only 1011 External loopback, lane E only 11xx Reserved
25	PLLBW	Selects the SerDes PLL bandwidth. 0 If PCI Express is not enabled 1 If PCI Express is enabled
26	PLLRST	PLL master reset for SerDes. Resets the PLL and the impedance calibration circuitry. software needs to set and clear this bit. 0 Application mode 1 Reset **Note:** PLLRST can also be done by setting SRDS*n*RSTCTL[RST]. In this case, PLLRST can self-clear.

Table 19-4. SRDS*n*CR1 Field Descriptions (continued)

Bits	Name	Description
27	SDRST	Master reset for SerDes logic. Resets all logic in all lanes. Software needs to set and clear this bit. 0 Application mode 1 Reset **Note:** SDRST can also be done by setting SRDS*n*RSTCTL[RST]. In this case, SDRST can self-clear.
28	—	Reserved
29	—	Reserved
30	RPTA	To enable repeater mode on lane A. Enables data received on serial inputs to be repeated back through to the transmitter outputs after data sampling and transition recovery on lane A. 0 Repeater mode disabled 1 Enable repeater mode on lane A
31	RPTE	To enable repeater mode on lane E. Enables data received on serial inputs to be repeated back through to the transmitter outputs after data sampling and transition recovery on lane E. 0 Repeater mode disabled 1 Enable repeater mode on lane E

19.3.3 SerDes*n* Control Register 2 (SRDS*n*CR2)

SRDS*n*CR2, shown in Figure 19-5, contains the functional control bits used for the SerDes logic.

Offset 0x08 — Access: Read-write

	0–13	14	15
R/W	—	TIMPCALS	RIMPCALS
Reset	0 0 0 0 0 0 0 0 1 0 0 0 0 0	0	0

	16–18	19–21	22–23	24–26	27–29	30–31
R/W	—	SEICA	PEICA	—	SEICE	PEICE
Reset	0 0 0	1 1 1	0 0	0 0 0	1 1 1	0 0

Figure 19-5. SerDes*n* Control Register 2 (SRDS*n*CR2)

Table 19-5 describes the SRDS*n*CR2.

Table 19-5. SRDS*n*CR2 Field Descriptions

Bits	Name	Description
0–13	—	Reserved
14	TIMPCALS	Transmitter impedance calibration stop command. Allows user to stop calibration of transmitter impedances. 0 Run transmit impedance calibration 1 Stop transmit impedance calibration Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0

Table 19-5. SRDS*n*CR2 Field Descriptions (continued)

Bits	Name	Description
15	RIMPCALS	Receiver impedance calibration stop command. Allows user to stop calibration of receiver impedances. 0 Run receive impedance calibration 1 Stop receive impedance calibration Recommended setting per protocol: • SATA: 0 • PCI Express: 0 • SGMII: 0
16–18	—	Reserved
19–21	SEICA	Receiver electrical idle detection control for lane A. 000 Loss of signal detect function is disabled 001 Default SGMII levels (low = 30 mV, high = 100 mV) 010 Intermediate level (low = 38 mV, high = 120 mV) 011 Intermediate level (low = 50mV, high = 150 mV) 100 Default PCI Express, SATA1 levels (low = 65 mV, high = 175 mV) 101 Default SATA2 levels (low = 75 mV, high = 200 mV) 110 Intermediate level (low = 88 mV, high = 225 mV) 111 Intermediate level (low = 100 mV, high = 250 mV) Recommended setting per protocol: • SATA: 101 • PCI Express: 100 • SGMII: 001
22–23	PEICA	PCI Express, SGMII, and SATA receiver electrical idle detection control for lane A. For PCI Express and SGMII: 00 Exit from idle ~88 UI and unexpected idle detect ~1 µs (application mode) 01 Exit from idle ~88 UI and unexpected idle detect ~10 µs 10 Exit from idle ~48 UI and unexpected idle detect ~1 µs 11 Bypass For SATA: 00 20 consecutive UI with no glitch to change 01 40 consecutive UI with no glitch to change 10 80 consecutive UI with no glitch to change 11 1 µs more bad data than good data to assert los_result Recommended setting per protocol: • PCI Express: 00 • SGMII: 00 • SATA: 00
24–26	—	Reserved

Table 19-5. SRDS*n*CR2 Field Descriptions (continued)

Bits	Name	Description
27–29	SEICE	Receiver electrical idle detection control for lane E. 000 Loss of signal detect function is disabled. 001 Default SGMII levels (low = 30 mV, high = 100 mV) 010 Intermediate level (low = 38 mV, high = 120 mV) 011 Intermediate level (low = 50 mV, high = 150 mV) 100 Default PCI Express, SATA1 levels (low = 65 mV, high = 175 mV) 101 Default SATA2 levels (low = 75 mV, high = 200 mV) 110 Intermediate level (low = 88 mV, high = 225 mV) 111 Intermediate level (low = 100mV, high = 250 mV) Recommended setting per protocol: • SATA: 101 • PCI Express: 100 • SGMII: 001
30–31	PEICE	PCI Express, SGMII, and SATA receiver electrical idle detection control for lane E. For PCI Express and SGMII: 00 Exit from Idle ~88 UI and unexpected idle detect ~1 µs (application mode) 01 Exit from Idle ~88 UI and unexpected idle detect ~10 µs 10 Exit from Idle ~48 UI and unexpected idle detect ~1 µs 11 Bypass For SATA: 00 20 consecutive UI with no glitch to change 01 40 consecutive UI with no glitch to change 10 80 consecutive UI with no glitch to change 11 1 µs more bad data than good data to assert los_result Recommended setting per protocol: • PCI Express: 00 • SGMII: 00 • SATA: 00

19.3.4 SerDes*n* Control Register 3 (SRDS*n*CR3)

SRDS*n*CR3, shown in Figure 19-6, contains the functional control bits for the SerDes logic.

Offset 0x0C — Access: Read only

Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	—		SDFMA						—		SDFME					
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Bits	16–17	18–23	24–25	26–31
R/W	—	SDTXLA	—	SDTXLE
Reset	All zeros			

Figure 19-6. SerDes*n* Control Register 3 (SRDS*n*CR3)

Table 19-6 describes the SRDS*n*CR3.

Table 19-6. SRDS*n*CR3 Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–3	KFRA	Selects the gain ’Kfr’ in the CDR for lane A. 00 2^{-5} 01 2^{-6} 10 Reserved 11 Reserved Recommended setting per protocol: • SATA: 01 • PCI Express: don’t care • SGMII: don’t care
4–5	KPHA	Selects the gain ’Kph’ in the CDR for lane A. 00 Reserved 01 2^{-7} 10 2^{-8} 11 Reserved Recommended setting per protocol: • SATA: 01 • PCI Express: don’t care • SGMII: don’t care
6–7	SDFMA	Sets the bandwidth of the digital filter to optimize for given frequency offset specification for lane A. 00 200 ppm (SGMII) 01 600 ppm (PCI Express or SATA) 10 Reserved 11 Reserved Recommended setting per protocol: • SATA: 01 • PCI Express: 01 • SGMII: 00
8–9	—	Reserved
10–11	KFRE	Selects the gain ’Kfr’ in the CDR for lane E. 00 2^{-5} 01 2^{-6} 10 Reserved 11 Reserved Recommended setting per protocol: • SATA: 01 • PCI Express: don’t care • SGMII: don’t care
12–13	KPHE	Selects the gain ’Kph’ in the CDR for lane E. 00 Reserved 01 2^{-7} 10 2^{-8} 11 Reserved Recommended setting per protocol: • SATA: 01 • PCI Express: don’t care • SGMII: don’t care

Table 19-6. SRDS*n*CR3 Field Descriptions (continued)

Bits	Name	Description
14–15	SDFME	Sets the bandwidth of the digital filter to optimize for given frequency offset specification for lane E. 00 200 ppm (SGMII) 01 600 ppm (PCI Express or SATA) 10 Reserved 11 Reserved Recommended setting per protocol: • SATA: 01 • PCI Express: 01 • SGMII: 00
16–17	—	Reserved[
18–20	—	Reserved
21–23	SDTXLA	Controls lane A transmitter amplitude levels. 000 No amplitude reduction 001 0.916 × full swing 010 0.833 × full swing 011 0.750 × full swing 100 0.666 × full swing 101 0.583 × full swing 110 0.500 × full swing 111 Reserved Recommended setting per protocol: • SATA: 101 • PCI Express: 000 • SGMII: 000
24–25	—	Reserved
26–31	SDTXLE	Controls lane E transmitter amplitude levels. [5:3] = Reserved 000 No amplitude reduction 001 0.916 × full swing 010 0.833 × full swing 011 0.750 × full swing 100 0.666 × full swing 101 0.583 × full swing 110 0.500 × full swing 111 Reserved Recommended setting per protocol (SRDS*n*CR0[DPPA/E] = 0): • SATA: 101 • PCI Express: 000 • SGMII: 000 Recommended setting per protocol (SRDS*n*CR0[DPPA/E] = 1): • SATA: don't care • PCI Express: don't care • SGMII: don't care

19.3.5 SerDes*n* Control Register 4 (SRDS*n*CR4)

SRDS*n*CR4, shown in Figure 19-7, contains the functional control bits for the SerDes logic.

NOTE

The protocol select configuration should be identical for both lane A and lane E. The user can only select identical protocol on both lanes. To power down a lane, use SRDS*n*CR1[0] or SRDS*n*CR1[4] to power down individual lanes (refer to Section 19.3.2, “SerDesn Control Register 1 (SRDSnCR1)”).

Valid combinations for protocol select:

- PCI Express mode (PROTA/PROTE = 0001) can be configured to either x1 or x2 lanes. The reference clock can be either 100 MHz or 125 MHz (150 MHz is not supported).
- SGMII mode (PROTA/PROTE = 0101) is only a x1 lane. The reference clock can be either 100 or 125 MHz(150 MHz is not supported).
- SATA mode (PROTA/PROTE = 1000) is only a x1 lane. The reference clock can be either 100 MHz, 125 MHz or 150 MHz.

Offset 0x10 Access: Read/write

	0–1	2–3	4–6	7	8–15
R/W	—	RFCKS	—	PLANE	—
Reset	0 0	n n	0 0 0	n	0 0 0 0 0 0 0 0

	16–19	20–23	24–27	28–31
R/W	—	PROTA	—	PROTE
Reset	0 0 0 0	n n n n	0 0 0 0	n n n n

Figure 19-7. SerDes*n* Control Register 4 (SRDS*n*CR4)

Table 19-7 describes the SRDS*n*CR4 bit fields.

Table 19-7. SRDS*n*CR4 Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–3	RFCKS	SerDes reference clock selection. 00 100 MHz 01 125 MHz 10 Reserved 11 150 MHz
4–6	—	Reserved
7	PLANE	Number of PCI Express lanes. 0 PCI Express1 controller uses lane A, PCI Express2 controller uses lane E 1 PCI Express1 controller uses both lanes A and E in x2 mode, PCI Express2 controller is not used **Note:** Not relevant for SerDes1.

Table 19-7. SRDS*n*CR4 Field Descriptions (continued)

Bits	Name	Description
8–19	—	Reserved
20–23	PROTA	Lane A protocol select (PCI Express, SGMII, SATA) 0001 PCI Express at 2.5 Gbps (not valid for SRDS1, only valid for SRDS2) 0101 SGMII (of eTSEC1) at 1.25 Gbps (not valid option for SRDS2, only valid for SRDS1) 1000 SATA1 or SATA3 at 3.0 Gbps or 1.5 Gbps (speed determined automatically by link negotiation). Note that SATA1 is connected to SerDes1 and SATA3 is connected to SerDes2. All other modes are reserved.
24–27	—	Reserved
28–31	PROTE	Lane E protocol select (PCI Express, SGMII, SATA) 0001 PCI Express at 2.5 Gbps (not valid for SRDS1, only valid for SRDS2) 0101 SGMII (of eTSEC2) at 1.25 Gbps (not valid option for SRDS2, only valid for SRDS1) 1000 SATA2 or SATA4 at 3.0 Gbps or 1.5 Gbps (speed determined automatically by link negotiation). Note that SATA2 is connected to SerDes1 and SATA4 is connected to SerDes2. All other modes are reserved.

19.3.6 SerDes*n* Reset Control Register (SRDS*n*RSTCTL)

SRDS*n*RSTCTL, shown in Figure 19-8, contains the control for SerDes reset state machine counter values.

Offset 0x20 Access: Read/write

	0	1	2–15
R	RST	RDONE	—
W	RST		—
Reset	0	0	0 0 0 0 0 0 0 1 0 0 0 1 0 0

	16–23	24	25	26–27	28	29	30	31
R	—	DRXRTA	DRXRTE	—	RXRTA	RXRTE	TXRTA	TXRTE
W	—	DRXRTA	DRXRTE	—	RXRTA	RXRTE	TXRTA	TXRTE
Reset	0 1 0 0 0 1 0 1	0	0	0 0	0	0	0	0

Figure 19-8. SerDes*n* Reset Control Register (SRDS*n*RSTCTL)

Table 19-8 describes the SRDS*n*RSTCTL register.

Table 19-8. SRDS*n*RSTCTL Field Descriptions

Bits	Name	Description
0	RST	To initiate SerDes soft reset software writes a 1. SerDes reset state machine clears bit when reset is done. Software can only assert this bit but not clear it. If cleared in the middle, the reset state machine ignores the change.
1	RDONE	SerDes reset done from SerDes state machine. When this bit is set, the SerDes is ready to start link training in concert with the protocol controller.
2–3	—	Reserved
4–23	—	Reserved. The default value must be preserved.

Table 19-8. SRDS*n*RSTCTL Field Descriptions (continued)

Bits	Name	Description
24	DRXRTA	Disable SATA lane A receive reset request
25	DRXRTE	Disable SATA lane E receive reset request
26–27	—	Reserved
28	RXRTA	SATA lane A receive reset to SerDes
29	RXRTE	SATA lane E receive reset to SerDes
30	TXRTA	SATA lane A transmit reset to SerDes
31	TXRTE	SATA lane E transmit reset to SerDes

19.4 Initialization Sequence and Reset

SerDes and PHY can be initialized by software anytime by doing a reset_req. This is done by setting the SRDS*n*RSTCTL[0] register field. Setting this register field starts the warm reset state machine. Software does not need to clear this bit.

Register fields to configure before software reset request:

- SRDS*n*RSTCTL register
  - Configure prescale and counter values (refer to Figure 19-8)
  - SRDS*n*RSTCTL[28:31]—in SATA mode, set and then clear these bits; skip this stage in other modes
- SRDS*n*CR1 register
  - SRDS*n*CR1[25] (PLL bandwidth)—configure to 0x1 in PCI Express and 0x0 in SGMII and SATA modes
- SRDS*n*CR2 register
  - SRDS*n*CR2[19:21], SRDS*n*CR2[27:29]—configure to 0x100 in PCI Express, 0x101 in SATA, and 0x001 in SGMII mode
- SRDS*n*CR3 Register
  - SRDS1CR3[2:3], SRDS1CR3[10:11]—configure to 0x01 in SATA mode, “don’t care” for PCI Express and SGMII modes
  - SRDS1CR3[4:5], SRDS1CR3[12:13]—configure to 0x01 in SATA mode, “don’t care” for PCI Express and SGMII modes
  - SRDS*n*CR3[6:7], SRDS*n*CR3[14:15]—configure to 0x01 in PCI Express and SATA modes, 0x00 in SGMII mode
  - SRDS*n*CR3[18:23], SRDS*n*CR3[26:31]—configure to 0x000000 in PCI Express and SGMII modes, 0x000101 in SATA mode
- SRDS*n*CR4 register
  - Configure protocol select, reference clock frequency, and PCI Express x1 or x2 lanes (refer to Figure 19-7)

Valid combinations for protocol select:

- PCI Express mode (AD_PROTO_SEL/EH_PROTO_SEL[3:0] = 0001) can be configured to either ×1 or ×2 lanes. The reference clock can be either 100 MHz or 125 MHz (150 MHz is not supported).
- SGMII mode (AD_PROTO_SEL/EH_PROTO_SEL[3:0] = 0101) is only a ×1 lane. The reference clock can be either 100 MHz or 125 MHz (150 MHz is not supported).
- SATA mode (AD_PROTO_SEL/EH_PROTO_SEL[3:0] = 1000) is only a ×1 lane. The reference clock can be either 100 MHz or 125 MHz or 150 MHz.

NOTE

The entire SerDes need to be reset in order to activate a lane from power-down. However, in SATA mode, software must take the controller offline (by writing 0 to the SATA HControl register) before triggering SerDes reset, and then wait for the reset bit to be cleared by hardware before bringing the controller back online.

19.5 Power Management: Power Down

The SerDes is capable of several different power management states depending on the settings of the protocol selection and power down signals.

By setting the register field SRDSCR0[24] powers down the entire SerDes and is comparable to the PCI Express L2 low power link state. The steps for powering down the SerDes are as follows:

1. Apply power to all XCOREVDD, XPADVDD, SDAVDD supplies.
2. Be sure all XCOREVSS, XPADVSS, and SDAVSS supplies are grounded.
3. Assert the SRDSCR0[24] control from the chip logic to whichever SerDes block is not in use. This safely parks all the analog circuitry and stops all SerDes-generated clocks.
4. Tie all unused SD_RX*n* and $\overline{\text{SD\_RX}n}$ serial differential inputs to XCOREVSS.
5. Float all unused SD_TX*n* and $\overline{\text{SD\_TX}n}$ serial differential outputs.
6. If a SerDes block is not being used and has its own receiver and transmitter calibration external resistors, then these can be tied to XCOREVDD for the receiver (SD_IMP_CAL_RX) and XPADVDD for the transmitter (SD_IMP_CAL_TX).
7. Tie SD_REF_CLK and $\overline{\text{SD\_REF\_CLK}}$ both to XCOREVSS.

NOTE

If the entire SerDes is powered down, or even if parts of the SerDes is powered down, all power pads in the SerDes must be powered.

Powering down the SerDes includes the following steps:

1. Disable the PLL and place its output clocks into a known, static state.
2. Power down the receiver termination and amplifier cells. In PCI Express mode, there is still a differential termination, but its value is no longer calibrated. Also, there is no longer a termination to ground. InSATA, SGMII mode, ,there is still a termination to ground and its value is calibrated.

3. Power down the transmitter and receiver impedance control amplifiers so that the termination impedances are uncalibrated.
4. Power down the transmitter cell. The $V_{TX\text{-}DIFFp}$ < 20 mV. The DC common mode voltage is not held.

Chapter 20
Universal Serial Bus Interface

This chapter describes the universal serial bus (USB) interface of the device. The USB interface implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, it is left to the reader to refer to the governing specifications.

The following documents are available from the USB Implementers Forum web page at http://www.usb.org/developers/docs/.

- *Universal Serial Bus Revision 2.0 Specification*
- *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*

The following documents are available from the Intel USB Specifications web page at http://www.intel.com/technology/usb/spec.htm.

- *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*

The following documents are available from the ULPI web page at http://www.ulpi.org/.

- *UTMI+ Specification, Revision 1.0*
- *UTMI Low Pin-Count Interface (ULPI) Specification, Revision 1.0*

20.1 Introduction

The device implements a a dual-role (DR) USB module. This module may be connected to an external port. Collectively the module and external port are called the USB interface. The USB interface is shown in Figure 20-1.

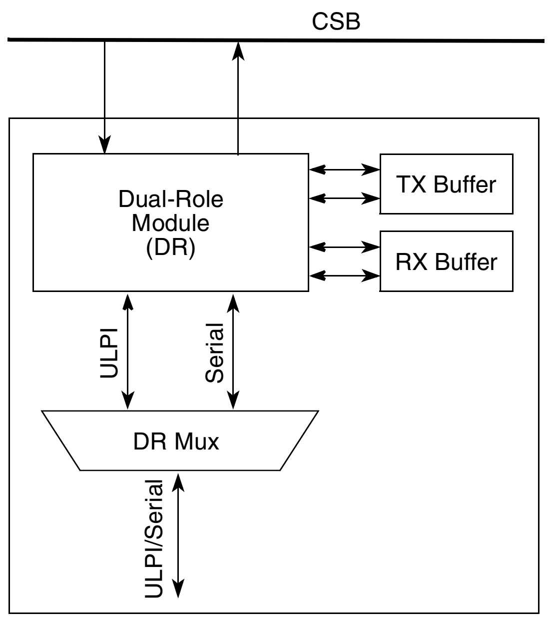


Figure 20-1. USB Interface Block Diagram

20.1.1 Overview

The USB DR module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures for the module are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The DR module can act as a device or host controller. Interfaces to negotiate the host or device role on the bus in compliance with the On-The-Go (OTG) supplement to the USB specification are also provided.

The DR module supports the required signaling for UTMI low pin count interface (ULPI) transceivers (PHYs) or for a full-speed (FS) or low-speed (LS) serial interface. The PHY interfacing to the ULPI or serial interface is an external PHY.

The module contains a chaining DMA (direct memory access) engine that reduces the interrupt load on the application processor and reduces the total system bus bandwidth that must be dedicated to servicing the USB interface requirements.

20.1.2 Features

The USB DR module includes the following features:

- Complies with USB specification rev 2.0
- Supports operation as a standalone USB host controller

— Supports enhanced host controller interface (EHCI)

- Supports high-speed (480 Mbps), full-speed (12 Mbps), and low-speed (1.5 Mbps) operation. Low speed is only supported in host mode.
- Supports external PHY with ULPI (UTMI + low-pin interface) and serial interfaces
- Supports operation as a standalone USB device
  — Supports one upstream facing port
  — Supports six bidirectional USB endpoints
- Host and device support
- OTG (on-the-go) support, which includes both device and host capability, with external PHY (ULPI)
- Host mode supports direct connect of FS/LS devices

20.1.3 Modes of Operation

The USB DR module has three basic operating modes: host, device, and OTG. The module can be configured to use one of two different PHY interfaces: ULPI or FS/LS serial transceiver.

NOTE

Only high-speed and full-speed operations are supported in device mode.

20.2 External Signals

This section contains detailed descriptions of all the USB dual-role controller signals. Many of the signals for the PHY interfaces are muxed onto the same pins in order to reduce pin count. Table 20-1 describes the signals, indicating which interface supports each signal. Note that the selection between ULPI/Serial modes of operation per USB port is done by programming the USB DR port status and control registers (PORTSC1/PORTSC2). Refer to Section 20.3.2.14, “Port Status and Control Register (PORTSC)” for more details.

Table 20-1. USB External Signals

Signal	I/O	Description
USBDR_D0_ENABLEN	I/O	ULPI—Use as USBDR_D0 Serial—Use as USBDR_ENABLEN
USBDR_D1_SER_TXD	I/O	ULPI—Use as USBDR_D1 Serial—Use as USBDR_SER_TXD
USBDR_D2_VMO_SE0	I/O	ULPI—Use as USBDR_D2 Serial—Use as USBDR_VMO_SE0
USBDR_D3_SPEED	I/O	ULPI—Use as USBDR_D3 Serial—Use as USBDR_SPEED

Table 20-1. USB External Signals (continued)

Signal	I/O	Description
USBDR_D4_DP	I/O	ULPI—Use as USBDR_D4 Serial—Use as USBDR_DP
USBDR_D5_DM	I/O	ULPI—Use as USBDR_D5 Serial—Use as USBDR_DM
USBDR_D6_SER_RCV	I/O	ULPI—Use as USBDR_D6 Serial—Use as USBDR_SER_RCV
USBDR_D7_DRVVBUS	I/O	ULPI—Use as USBDR_D7 Serial—Use as USBDR_DRVVBUS
USBDR_SESS_VLD_NXT	I	ULPI—Use as USBDR_NXT Serial—Use as USBDR_SESS_VLD
USBDR_DIR_DPPULLUP	I/O	ULPI—Use as USBDR_DIR Serial—Use as USBDR_DIR_DPPULLUP
USBDR_STP_SUSPEND	O	ULPI—Use as USBDR_STP Serial—Use as USBDR_SUSPEND
USBDR_PWRFAULT	I	ULPI—Use as USBDR_PWRFAULT Serial—Use as USBDR_PWRFAULT
USBDR_PCTL0	O	ULPI—Use as USBDR_PCTL0 Serial—Use as USBDR_PCTL0
USBDR_PCTL1	O	ULPI—Use as USBDR_PCTL1 Serial—Use as USBDR_PCTL1
USBDR_CLK	I	ULPI—Use as USBDR_CLK Serial—Use as USBDR_CLK

20.2.1 ULPI Interface

The ULPI (UTMI low pin count interface) is a reduced pin-count (12 signals) extension of the UTMI+ specification. Pin count is reduced by converting relatively static signals to register bits, and providing a bidirectional, generic data bus that carries USB and register data. This interface minimizes pin count requirements for external PHYs. Table 20-2 describes the signals for the ULPI interface.

Table 20-2. ULPI Signal Descriptions

Signal	I/O	Description	
USBDR_DIR	I	Direction. USBDR_DIR controls the direction of the data bus. When the PHY has data to transfer to USB port, it drives USBDR_DIR high to take ownership of the bus. When the PHY has no data to transfer it drives USBDR_DIR low and monitors the bus for link activity. The PHY pulls USBDR_DIR high whenever the interface cannot accept data from the link.	
		State Meaning	Asserted—PHY has data to transfer to the link. Negated—PHY has no data to transfer.
		Timing	Synchronous to PHY_CLK.
USBDR_NXT	I	Next data. The PHY asserts USBDR_NXT to throttle the data. When USB port is sending data to the PHY, USBDR_NXT indicates when the current byte has been accepted by the PHY. The USB port places the next byte on the data bus in the following clock cycle. When the PHY is sending data to USB port, USBDR_NXT indicates when a new byte is available for USB port to consume.	
		State Meaning	Asserted—PHY is ready to transfer byte. Negated—PHY is not ready.
		Timing	Synchronous to PHY_CLK.
USBDR_STP	O	Stop. USBDR_STP indicates the end of a transfer on the bus.	
		State Meaning	Asserted—USB asserts this signal for 1 clock cycle to stop the data stream currently on the bus. If USB port is sending data to the PHY, USBDR_STP indicates the last byte of data was previously on the bus. If the PHY is sending data to USB port, USBDR_STP forces the PHY to end its transfer, negate USBDR_DIR and relinquish control of the data bus to the USB port. Negated—Indicates normal operation.
		Timing	Synchronous to PHY_CLK.
USBDR_PWRFAULT	I	Power fault. USBDR_PWRFAULT indicates whether a power fault occurred on the USB port Vbus.	
		State Meaning	Asserted—Indicates that a Vbus fault occurred. Applications that support power switching must shut down Vbus power. Negated—Indicates normal operation.
		Timing	Synchronous to PHY_CLK.
USBDR_PCTL0	O	Port control 0. USBDR_PCTL0 controls the port status indicator LED 0 when in host mode.	
		State Meaning	Asserted—LED on. Negated—LED off.
		Timing	Synchronous to PHY_CLK.
USBDR_PCTL1	O	Port control 1. USBDR_PCTL1 controls the port status indicator LED 1 when in host mode.	
		State Meaning	Asserted—LED on. Negated—LED off.
		Timing	Synchronous to PHY_CLK.

Table 20-2. ULPI Signal Descriptions (continued)

Signal	I/O	Description	
USBDR_TXRXD[7:0]	I/O	Data bit *n*. USBDR_TXRXD*n* is bit *n* of the 8-bit (USBDR_TXRXD7–USBDR_TXRXD0), uni-directional data bus used to carry USB, register, and interrupt data between the PHY and the USB controller.	
		State Meaning	Asserted—Data bit *n* is 1. Negated—Data bit *n* is 0.
		Timing	Synchronous to PHY_CLK.

20.2.2 Serial Interface

For full-speed/low-speed USB applications, the operating frequency is low enough to allow data recovery to be handled digitally and level translation through a simple transceiver (PHY). The serial transceiver interface signals are brought out of the IP block to support just such legacy USB1.1 FS/LS transceivers. Table 20-3 describes the signals for the serial interface. Note that the timing of the serial interface is asynchronous, however, the USB controller must be connected to a 60-MHz input clock through the USBDR_CLK signal in order to generate the serial output data or to over-sample the serial input data.

NOTE

Some of the serial signals are muxed with the ULPI. This section describes the function when in serial mode; refer to the other PHY sections for descriptions of the functionally in those modes.

Table 20-3. Serial Detailed Signal Descriptions

Signal	I/O	Description	
USBDR_SER_RCV	I	Receive. USB FS/LS serial differential input signal in single-ended form.	
		State Meaning	Asserted—J state; Data 1 Negated—K state; Data 0
		Timing	Asynchronous.
$\overline{\text{USBDR\_ENABLEN}}$	O	Enable. Active low output enable signal for a FS/LS transceiver (PHY).	
		State Meaning	Asserted (low)—PHY is in transmit mode Negated (high)—PHY is in receive mode
		Timing	Asynchronous.
USBDR_SER_TXD	O	Transmit data. Supplies the single-ended data value to the PHY. This signal is valid only when single-ended zero (USBDR_VMO_SE0) is low.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.
USBDR_VMO_SE0	O	Single-ended zero. Indicates to the FS/LS transceiver that is should send a single-ended zero.	
		State Meaning	Asserted—Single-ended zero Negated—Normal data
		Timing	Asynchronous.

Table 20-3. Serial Detailed Signal Descriptions (continued)

Signal	I/O	Description	
USBDR_DP	I	Data plus. Reflects the state of the USB DP signal.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.
USBDR_DM	I	Data minus. Reflects the state of the USB DM signal.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous.
USBDR_SPEED	O	Speed. Used with some FS/LS transceivers to select LS and FS edge rates.	
		State Meaning	Asserted—Full speed Negated—Low speed
		Timing	Asynchronous.
USBDR_DPPULLUP	O	Data plus pull-up. Controls the pull-up termination on the USB DP signal in FS only applications.	
		State Meaning	Asserted—Pull up enabled Negated—Pull up disabled
		Timing	Asynchronous.
USBDR_PWRFAULT	I	Power fault. Indicates whether a Vbus fault has occurred. Controllers that support power switching must shut down Vbus power. This signal is only used in host mode.	
		State Meaning	Asserted—Over current condition on Vbus Negated—Normal operation
		Timing	Asynchronous.
USBDR_PCTL0	O	Port control 0. Controls the port status indicator LED 0 when in host mode.	
		State Meaning	Asserted—LED on Negated—LED off
		Timing	Asynchronous.
USBDR_PCTL1	O	Port control 1. Controls the port status indicator LED 1 when in host mode.	
		State Meaning	Asserted—LED on Negated—LED off
		Timing	Asynchronous.
USBDR_SESS_VLD	I	Session valid. Indicates whether the Vbus voltage is at a valid level for operation. Note that this signal is only applicable to the DR module.	
		State Meaning	Asserted— Vbus voltage is in the valid range as specified in the USB2.0 specification. Negated—Vbus voltage is outside the valid range as specified in the USB2.0 specification.
		Timing	Asynchronous.

Table 20-3. Serial Detailed Signal Descriptions (continued)

Signal	I/O	Description	
USBDR_SUSPEND	O	Suspend (active high). Used to place the FS/LS transceivers in low power mode when a suspend condition is detected. This signal is active high in serial mode.	
		State Meaning	Asserted—Indicates PHY should enter low power mode. Negated—Indicates PHY should be in normal operation.
		Timing	Asynchronous.
USBDR_DRVVBUS	O	Drive Vbus. Used to enable/disable power (VBus) on devices that support port power switching. This signal is not applicable to device-only applications.	
		State Meaning	Asserted—Vbus power enabled. Negated—Vbus power disabled.
		Timing	Asynchronous.

20.2.3 PHY Clocks

The USBDR_CLK input provides the clocking signal for both ULPI and serial PHY interfaces. The clock is 60 MHz. Detailed clock specifications are given in the appropriate hardware specifications document.

20.3 Memory Map/Register Definitions

This section provides the memory map and detailed descriptions of all USB interface registers. The memory map of the USB interface is shown in Table 20-4.

Table 20-4. USB Interface Memory Map

Offset	Register	Access	Reset	Section/Page
USB DR Controller Registers				
0x2_3000–0x2_30FF	Reserved, should be cleared	—	—	—
0x2_3100	CAPLENGTH—Capability register length	R	0x40	20.3.1.1/20-10
0x2_3102	HCIVERSION—Host interface version number[1]	R	0x0100	20.3.1.2/20-11
0x2_3104	HCSPARAMS—Host crtl. structural parameters[1]	R	0x0001_0011	20.3.1.3/20-11
0x2_3108	HCCPARAMS—Host crtl. capability parameters[1]	R	0x0000_0006	20.3.1.4/20-12
0x2_3120	DCIVERSION—Device interface version number	R	0x0001	20.3.1.5/20-13
0x2_3124	DCCPARAMS—Device controller parameters	R	0x0000_0186	20.3.1.6/20-13
0x2_3140	USBCMD—USB command	Mixed	0x0008_0000	20.3.2.1/20-14
0x2_3144	USBSTS—USB status	Mixed	0x0000_0000	20.3.2.2/20-17
0x2_3148	USBINTR—USB interrupt enable	R/W	0x0000_0000	20.3.2.3/20-19
0x2_314C	FRINDEX—USB frame index	R/W	0x0000_*nnnn*	20.3.2.4/20-20
0x2_3154	PERIODICLISTBASE—Frame list base address[1]	R/W	0x*nnnn*_0000	20.3.2.6/20-22
	DEVICEADDR—USB device address	R/W	0x0000_0000	20.3.2.7/20-22

Table 20-4. USB Interface Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2_3158	ASYNCLISTADDR—Next asynchronous list addr (host mode)[1]	R/W	0x0000_0000	20.3.2.8/20-23
	ENDPOINTLISTADDR—Address at endpoint list (device mode)	R/W	0x0000_0000	20.3.2.9/20-23
0x2_3160	BURSTSIZE—Programmable burst size	R/W	0x0000_1010	20.3.2.10/20-24
0x2_3164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0000_0000	20.3.2.11/20-24
0x2_3170	ULPI VIEWPORT—ULPI Register Access	Mixed	0x0000_0000	20.3.2.12/20-26
0x2_3180	CONFIGFLAG—Configured flag register	R	0x0000‘_0001	20.3.2.13/20-28
0x2_3184	PORTSC—Port status/control	Mixed	0x8000_0010	20.3.2.14/20-28
0x2_31A4	OTGSC—On-The-Go status and control[1]	Mixed	0x200C_0000	20.3.2.15/20-33
0x2_31A8	USBMODE—USB device mode	R/W	0x0000_0000	20.3.2.16/20-36
0x2_31AC	ENDPTSETUPSTAT—Endpoint setup status	R/W	0x0000_0000	20.3.2.17/20-37
0x2_31B0	ENDPOINTPRIME—Endpoint initialization	R/W	0x0000_0000	20.3.2.18/20-37
0x2_31B4	ENDPTFLUSH—Endpoint de-initialize	R/W	0x0000_0000	20.3.2.19/20-38
0x2_31B8	ENDPTSTATUS—Endpoint status	R	0x0000_0000	20.3.2.20/20-39
0x2_31BC	ENDPTCOMPLETE—Endpoint complete	w1c	0x0000_0000	20.3.2.21/20-39
0x2_31C0	ENDPTCTRL0—Endpoint control 0	Mixed	0x0080_0080	20.3.2.22/20-40
0x2_31C4	ENDPTCTRL1—Endpoint control 1	R/W	0x0000_0000	20.3.2.23/20-41
0x2_31C8	ENDPTCTRL2—Endpoint control 2	R/W	0x0000_0000	20.3.2.23/20-41
0x2_31CA	ENDPTCTRL3—Endpoint control 3	R/W	0x0000_0000	20.3.2.23/20-41
0x2_31D0	ENDPTCTRL4—Endpoint control 4	R/W	0x0000_0000	20.3.2.23/20-41
0x2_31D4	ENDPTCTRL5—Endpoint control 5	R/W	0x0000_0000	20.3.2.23/20-41
0x2_3400	SNOOP1—Snoop 1	R/W	0x0000_0000	20.3.2.24/20-42
0x2_3404	SNOOP2—Snoop 2	R/W	0x0000_0000	20.3.2.24/20-42
0x2_3408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	20.3.2.25/20-43
0x2_340C	PRI_CTRL—Priority control	R/W	0x0000_0000	20.3.2.26/20-45
0x2_3410	SI_CTRL—System interface control	R/W	0x0000_0000	20.3.2.27/20-45
0x2_3500	CONTROL—Control	R/W	0x0000_0000	20.3.2.28/20-46
0x2_3504–0x2_3FFF	Reserved, should be cleared	—	—	—

1 This register has separate functions for the host and device operation; the host function is listed first in the table.

The following sections provide details about the registers in the USB memory map.

NOTE

Memory may be viewed from either a big-endian or little-endian byte ordering perspective depending on the processor configuration. In big-endian mode, the most-significant byte of word 0 is located at address 0 and the least-significant byte of word 0 is located at address 3. In little-endian mode, the least-significant byte of word 0 is located at address 0 and the most-significant byte of word 0 is located at address 3. Within registers, bits are numbered within a word starting with bit 31 as the most-significant bit. By convention USB registers use little-endian byte ordering. In the USB DR module, these are the registers from offsets 0x00 to 0x1FF. The registers associated with the internal system interface (0x400 and above) use big-endian byte ordering.

20.3.1 Capability Registers

The capability registers specify the software limits, restrictions, and capabilities of the host/device controller implementation. Most of these registers are defined by the EHCI specification. Registers that are not defined by the EHCI specification are noted in their descriptions.

20.3.1.1 Capability Registers Length (CAPLENGTH)

CAPLENGTH is used as an offset to add to the register base address to find the beginning of the operational register space, that is, the location of the USBCMD register. Figure 20-2 shows CAPLENGTH.

Offset 0x2_3100 Access: Read-only

	7							0
R	CAPLENGTH							
W								
Reset	0	1	0	0	0	0	0	0

Figure 20-2. Capability Registers Length (CAPLENGTH)

Table 20-5 provides bit descriptions for the CAPLENGTH register.

Table 20-5. CAPLENGTH Register Field Descriptions

Bits	Name	Description
7–0	CAPLENGTH	Capability registers length. Value is 0x40.

20.3.1.2 Host Controller Interface Version (HCIVERSION)

HCIVERSION contains a BCD encoding of the EHCI revision number supported by this host controller. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision. Figure 20-3 shows the HCIVERSION register.

Offset 0x2_3102 Access: Read-only

	15–0
R	HCIVERSION
W	
Reset	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

Figure 20-3. Host Controller Interface Version (HCIVERSION)

Table 20-6 provides bit descriptions for the HCIVERSION register.

Table 20-6. HCIVERSION Register Field Descriptions

Bits	Name	Description
15–0	—	EHCI revision number. Value is 0x0100 indicating version 1.0.

20.3.1.3 Host Controller Structural Parameters (HCSPARAMS)

HCSPARAMS contains structural parameters such as the number of downstream ports. Figure 20-4 shows the HCSPARAMS register.

Offset 0x2_3104 Access: Read-only

	31–28	27–24	23–20	19–17	16	15–12	11–8	7–5	4	3–0
R	—	N_TT	N_PTT	—	PI	N_CC	N_PCC	—	PPC	N_PORTS
W										
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0	1	0 0 0 0	0 0 0 0	0 0 0	1	0 0 0 1

Figure 20-4. Host Controller Structural Parameters (HCSPARAMS)

Table 20-7 provides bit descriptions for the HCSPARAMS register.

Table 20-7. HCSPARAMS Register Field Descriptions

Bits	Name	Description
31–28	—	Reserved, should be cleared.
27–24	N_TT	Number of transaction translators. This is a non-EHCI field. This field indicates the number of embedded transaction translators associated the module. Always 1. See Section 20.9.1, "Embedded Transaction Translator Function."
23–20	N_PTT	Ports per transaction translator. This is a non-EHCI field. The number of ports assigned to each transaction translator. This is equal to N_PORTS.
19–17	—	Reserved, should be cleared.
16	PI	Port indicators. Indicates whether the ports support port indicator control. Always 1. 1 The port status and control registers include a R/W field for controlling the state of the port indicator.
15–12	N_CC	Number of companion controllers associated with the DR controller. Always 0.

Table 20-7. HCSPARAMS Register Field Descriptions (continued)

Bits	Name	Description
11–8	N_PCC	Number ports per CC. This field indicates the number of ports supported per internal companion controller. Always 0.
7–5	—	Reserved, should be cleared.
4	PPC	Power port control. Indicates whether the host controller supports port power control. Always 1. 1 Ports have power port switches.
3–0	N_PORTS	Number of ports. Number of physical downstream ports implemented for host applications. The value of this field determines how many port registers are addressable in the operational register. Always 1.

20.3.1.4 Host Controller Capability Parameters (HCCPARAMS)

HCCPARAMS identifies multiple mode control (time-base bit functionality) addressing capability. Figure 20-5 shows the HCCPARAMS register.

Offset 0x2_3108 — Access: Read-only

Bits	31–16	15–8	7–4	3	2	1	0
R	—	EECP	IST	—	ASP	PFL	ADC
W							
Reset	0000 0000 0000 0000	0000 0000	0000	0	1	1	0

Figure 20-5. Host Control Capability Parameters (HCCPARAMS)

Table 20-8 provides bit descriptions for the HCCPARAMS register.

Table 20-8. HCCPARAMS Register Field Descriptions

Bits	Name	Description
31–16	—	Reserved, should be cleared.
15–8	EECP	EHCI extended capabilities pointer. Indicates the existence of a capabilities list. A value of 0x00 indicates no extended capabilities are implemented. A non-zero value in this register indicates the offset in PCI configuration space of the first EHCI extended capability. The pointer value must be 0x40 or greater if implemented to maintain the consistency of the PCI header defined for this class of device. This field is always 0.
7–4	IST	Isochronous scheduling threshold. Indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit 7 is zero, the value of the least significant 3 bits indicates the number of microframes a host controller can hold a set of isochronous data structures (one or more) before flushing the state. When bit 7 is a one, then host software assumes the host controller may cache an isochronous data structure for an entire frame. This field is always 0.
3	—	Reserved, should be cleared.
2	ASP	Asynchronous schedule park capability. Indicates whether the USB DR module supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level by using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register. This field is always 1 (park feature supported).

Table 20-8. HCCPARAMS Register Field Descriptions (continued)

Bits	Name	Description
1	PFL	Programmable frame list flag. Indicates whether system software can specify and use a frame list length less that 1024 elements. Frame list size is configured via the USBCMD register frame list size field. The frame list must always be aligned on a 4K page boundary. This requirement ensures that the frame list is always physically contiguous. This field is always 1.
0	ADC	64-bit addressing capability. Always 0; 64-bit addressing is not supported. 0 Data structures use 32-bit address memory pointers

20.3.1.5 Device Controller Interface Version (DCIVERSION)—Non-EHCI

This register is not defined in the EHCI specification. DCIVERSION is a two-byte register containing a BCD encoding of the device controller interface. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision. Figure 20-6 shows the DCIVERSION register.

Offset 0x2_3120 Access: Read-only

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DCIVERSION															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 20-6. Device Interface Version (DCIVERSION)

Table 20-9 provides bit descriptions for the DCIVERSION register.

Table 20-9. DCIVERSION Register Field Descriptions

Bits	Name	Description
15–0	DCIVERSION	Device interface revision number.

20.3.1.6 Device Controller Capability Parameters (DCCPARAMS)—Non-EHCI

This register is not defined in the EHCI specification. This register describes the overall host/device capability of the DR module. Figure 20-7 shows the DCCPARAMS register.

Offset 0x2_3124 Access: Read-only

Bits	31–9	8	7	6–5	4–0
R	—	HC	DC	—	DEN
W					
Reset	0000_0000_0000_0000_0000_000	1	1	0 0	0 0 1 1 0

Figure 20-7. Device Control Capability Parameters (DCCPARAMS)

Table 20-10 provides bit descriptions for the DCCPARAMS register.

Table 20-10. DCCPARAMS Register Field Descriptions

Bits	Name	Description
31–9	—	Reserved, should be cleared.
8	HC	Host capable. Always 1, indicating the USB DR controller can operate as an EHCI compatible USB 2.0 host
7	DC	Device capable. Always 1, indicating the USB DR controller can operate as an USB 2.0 device. 1 Device capability. 0 No device capability (host only).
6–5	—	Reserved, should be cleared.
4–0	DEN	Device endpoint number. Indicates the number of endpoints built into the device controller. Always 0x6.

20.3.2 Operational Registers

The operational registers are comprised of dynamic control or status registers that may be read-only, read/write, or read/write-1-to-clear. The following sections define the operational registers.

20.3.2.1 USB Command Register (USBCMD)

The module executes the command indicated in this register.

Offset 0x2_3140 Access: Mixed

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	—								ITC							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FS2	ATDTW	SUTW	—	ASPE	—	ASP		LR	IAA	ASE	PSE	FS1	FS0	RST	RS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-8. USB Command Register (USBCMD)

Table 20-11. USBCMD Register Field Descriptions

Bits	Name	Description
31–24	—	Reserved, should be cleared.
23–16	ITC	Interrupt threshold control. The system software uses this field to set the maximum rate at which the USB DR module will issue interrupts. ITC contains the maximum interrupt interval measured in microframes. Valid values are shown below. 0x00 Immediate (no threshold) 0x01 1 microframe 0x02 2 microframes 0x04 4 microframes 0x08 8 microframes 0x10 16 microframes 0x20 32 microframes 0x40 40 microframes
15	FS2	See bits 3–2 below. This is a non-EHCI bit.
14	ATDTW	Add dTD TripWire. This is a non-EHCI bit. Used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit shall also be cleared by hardware when is state machine is hazard region where adding a dTD to a primed endpoint may go unrecognized. More information on the use of this bit is described in Section 20.9.2, "Device Operation."
13	SUTW	Setup tripwire. This is a non-EHCI bit. Used as a semaphore when the 8 bytes of setup data read extracted from a QH by the DCD. If the setup lockout mode is off (See USBMODE) then there exists a hazard when new setup data arrives and the DCD is copying setup from the QH for a previous setup packet. This bit is set and cleared by software and will be cleared by hardware when a hazard exists. More information on the use of this bit is described in Section 20.9.2, "Device Operation."
12	—	Reserved, should be cleared.
11	ASPE	Asynchronous schedule park mode enable. This bit defaults to a 1 and is R/W. Software uses this bit to enable or disable park mode. 0 Disabled 1 Enabled
10	—	Reserved, should be cleared.
9–8	ASP	Asynchronous schedule park mode count. This field defaults to 0x3 and is R/W. It contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the Asynchronous schedule before continuing traversal of the Asynchronous schedule. Valid values are 0x1H to 0x3H. Software must not write a zero to this field when ASPE is set as this will result in undefined behavior.
7	LR	Light host/device controller reset (OPTIONAL). Not implemented. Always 0.
6	IAA	Interrupt on async advance doorbell. Used as a doorbell by software to tell the USB DR controller to issue an interrupt the next time it advances asynchronous schedule. Software must write a 1 to this bit to ring the doorbell. When the controller has evicted all appropriate cached schedule states, it sets USBSTS[AAI]. If USBINTR[AAE] is set, the host controller will assert an interrupt at the next interrupt threshold. The controller clears this bit after it has set USBSTS[AAI]. Software should not set this bit when the asynchronous schedule is inactive. Doing so will yield undefined results. This bit is only used in host mode. Setting this bit when the USB DR module is in device mode is selected will result in undefined results.

Table 20-11. USBCMD Register Field Descriptions (continued)

Bits	Name	Description
5	ASE	Asynchronous schedule enable. Controls whether the controller skips processing the asynchronous schedule. Only used in host mode. 0 Do not process the asynchronous schedule 1 Use the ASYNCLISTADDR register to access the asynchronous schedule.
4	PSE	Periodic schedule enable. Controls whether the controller skips processing the periodic schedule. Only used in host mode. 0 Do not process the periodic schedule. 1 Use the PERIODICLISTBASE register to access the periodic schedule.
3–2	FS	Frame list size. Together with bit 15 these bits make the FS[2:0] field. This field is read/write only if programmable frame list flag in the HCCPARAMS registers is set to 1. This field specifies the size of the frame list that controls which bits in FRINDEX should be used for the frame list current index. Only used in host mode. Note that values below 256 elements are not defined in the EHCI specification. 000 1024 elements (4096 bytes) 001 512 elements (2048 bytes) 010 256 elements (1024 bytes) 011 128 elements (512 bytes) 100 64 elements (256 bytes) 101 32 elements (128 bytes) 110 16 elements (64 bytes) 111 8 elements (32 bytes)
1	RST	Controller reset. Software uses this bit to reset the controller. This bit is cleared by the controller when the reset process is complete. Software cannot terminate the reset process early by writing a zero to this register. Host mode: • When software sets this bit, the host controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. A USB reset is not driven on downstream ports. Software should not set this bit when USBSTS[HCH] is a zero. Attempting to reset an actively running host controller will result in undefined behavior. Device mode: • When software sets this bit, the USB DR controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction currently in progress on USB is immediately terminated. Writing a one to this bit in device mode is not recommended.
0	RS	Run/Stop. Host mode: • When this bit is set, the controller proceeds with the execution of the schedule. The controller continues execution as long as this bit is set. When this bit is set to 0, the host controller completes the current transaction on the USB and then halts. The USBSTS[HCH] bit indicates when the USB DR controller has finished the transaction and has entered the stopped state. Software should not write a one to this field unless the controller is in the halted state (that is, USBSTS[HCH] is a one). Device mode: • Setting this bit will cause the USB DR controller to enable a pull-up on D+ and initiate an attach event. This control bit is not directly connected to the pull-up enable, as the pull-up will become disabled upon transitioning into high-speed mode. Software should use this bit to prevent an attach event before the controller has been properly initialized. Clearing this bit will cause a detach event. 0 Stop 1 Run

20.3.2.2 USB Status Register (USBSTS)

This register indicates various states of the USB DR module and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them (indicated by a w1c in the bit's W cell in Figure 20-9).

Offset 0x2_3144 Access: Mixed

	31–16
R	—
W	
Reset	All zeros

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AS	PS	RCL	HCH	—	ULPII	—	SLI	SRI	URI	AAI	SEI	FRI	PCI	UEI	UI
W								w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros															

Figure 20-9. USB Status Register (USBSTS)

Table 20-12. USBSTS Register Field Descriptions

Bits	Name	Description
31–16	—	Reserved, should be cleared.
15	AS	Asynchronous schedule status. Reports the current real status of the asynchronous schedule. The USB DR controller is not required to immediately disable or enable the asynchronous schedule when software transitions USBCMD[ASE]. When this bit and USBCMD[ASE] have the same value, the asynchronous schedule is either enabled (1) or disabled (0). Only used in host mode. 0 Disabled 1 Enabled
14	PS	Periodic schedule status. Reports the current real status of the periodic schedule. The USB DR controller is not required to immediately disable or enable the periodic schedule when software transitions USBCMD[PSE]. When this bit and USBCMD[PSE] have the same value, the periodic schedule is either enabled (1) or disabled (0). Only used in host mode. 0 Disabled 1 Enabled
13	RCL	Reclamation. Used to detect an empty asynchronous schedule. Only used by the host mode. 0 Non-empty asynchronous schedule 1 Empty asynchronous schedule
12	HCH	HC halted. This bit is a zero whenever USBCMD[RS] is a one. The USB DR controller sets this bit to one after it has stopped executing because of USBCMD[RS] being cleared, either by software or by the host controller hardware (for example, internal error). Only used in host mode. 0 Running 1 Halted
11	—	Reserved, should be cleared.
10	ULPII	ULPI interrupt. An event completion to the viewport register sets this bit. If the ULPI enables the USBINTR[ULPIE] to be set, the USB interrupt (UI) will occur.
9	—	Reserved, should be cleared.

Table 20-12. USBSTS Register Field Descriptions (continued)

Bits	Name	Description
8	SLI	DCSuspend. This is a non-EHCI bit. When a device controller enters a suspend state from an active state, this bit is set. The device controller clears the bit upon exiting from a suspend state. Only used by the device controller. 0 Active 1 Suspended
7	SRI	Host mode: • This is a non-EHCI status bit. In host mode, this bit will be set every 125 us, provided the PHY clock is present and running (for example, the port is NOT suspended), and can be used by the host controller driver as a time base. Device mode: • SOF received. When the USB DR controller detects a Start Of (Micro)Frame, this bit will be set. When a SOF is extremely late, the DR controller will automatically set this bit to indicate that an SOF was expected. Therefore, this bit will be set roughly every 1 msec in device FS mode and every 125 msec in HS mode and will be synchronized to the actual SOF that is received. Because the controller is initialized to FS before connect, this bit will be set at an interval of 1 msec during the prelude to the connect and chirp. Software writes a 1 to this bit to clear it.
6	URI	USB reset received. This is a non-EHCI bit. When the USB DR controller detects a USB reset and enters the default state, this bit will be set. Software can write a 1 to this bit to clear the USB reset received status bit. Only used by the device mode. 0 No reset received 1 Reset received
5	AAI	Interrupt on async advance. System software can force the controller to issue an interrupt the next time the USB DR controller advances the asynchronous schedule by writing a one to USBCMD[IAA]. This status bit indicates the assertion of that interrupt source. Only used by the host mode. 0 No async advance interrupt 1 Async advance interrupt
4	SEI	System error. This bit is set whenever an error is detected on the system bus. If USBINTR[SEE] is set, an interrupt will be generated. The interrupt and status bits will remain asserted until cleared by writing a 1 to this bit. Additionally, when in host mode, USBCMD[RS] is cleared, effectively disabling the USB DR controller. For the USB DR controller in device mode, an interrupt is generated, but no other action is taken. 0 Normal operation 1 Error
3	FRI	Frame list rollover. The controller sets this bit to a one when the frame list index rolls over from its maximum value to zero. The exact value at which the rollover occurs depends on the frame list size. For example. If the frame list size (as programmed in USBCMD[FS]) is 1024, FRINDEX rolls over every time FRINDEX [1 3] toggles. Similarly, if the size is 512, the USB DR controller sets this bit to a one every time FHINDEX [12] toggles. Only used by the host mode.
2	PCI	Host mode: • Port change detect. The controller sets this bit when a connect status occurs on any port, a port enable/disable change occurs, an over current change occurs, or PORTSC[FPR] is set as the result of a J-K transition on the suspended port. Device mode: • The USB DR controller sets this bit when it enters the full or high-speed operational state. When the it exits the full or high-speed operation states due to reset or suspend events, the notification mechanisms are USBSTS[URI] and USBSTS[SLI], respectively. This bit is not EHCI compatible.

Table 20-12. USBSTS Register Field Descriptions (continued)

Bits	Name	Description
1	UEI (USBERRINT)	USB error interrupt (USBERRINT). When completion of a USB transaction results in an error condition, this bit is set by the controller. This bit is set along with the UI, if the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set. See Section 4.15.1 in EHCI for a complete list of host error interrupt conditions. Also see Table 20-92 in this chapter for more information on device error matrix. For the USB DR controller in device mode, only resume signaling is detected, all others are ignored. 0 No error 1 Error detected
0	UI (USBINT)	USB interrupt (USBINT). This bit is set by the controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set. This bit is also set by the controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes.

20.3.2.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with this register. An interrupt is generated when a bit is set and the corresponding interrupt is active. The USB status register (USBSTS) still shows interrupt sources even if they are disabled by the USBINTR register, allowing polling of interrupt events by the software.

Offset 0x2_3148 Access: Read/Write

	31–16
R/W	—
Reset	All zeros

	15–11	10	9	8	7	6	5	4	3	2	1	0
R/W	—	ULPIE	—	SLE	SRE	URE	AAE	SEE	FRE	PCE	UEE	UE
Reset	All zeros											

Figure 20-10. USB Interrupt Enable (USBINTR)

Table 20-13. USBINTR Register Field Descriptions

Bits	Name	Description
31–11	—	Reserved, should be cleared.
10	ULPIE	ULPI interrupt enable. An event completion to the viewport register sets the USBSTS[ULPII]. If the ULPI enables ULPIE bit to be set, then the USBINT (USBSTS[UI]) will occur. 0 Disable 1 Enable
9	—	Reserved, should be cleared.
8	SLE	Sleep enable. This is a non-EHCI bit. When this bit is a one, and USBSTS[SLI] transitions, the USB DR controller will issue an interrupt. The interrupt is acknowledged by software writing a one to USBSTS[SLI]. Only used in device mode. 0 Disable 1 Enable

Table 20-13. USBINTR Register Field Descriptions (continued)

Bits	Name	Description
7	SRE	SOF received enable. This is a non-EHCI bit. When this bit is a one, and USBSTS[SRI] is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[SRI]. 0 Disable 1 Enable
6	URE	USB reset enable. This is a non-EHCI bit. When this bit is a one, USBSTS[URI] is a one, the device controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[URI] bit. Only used in device mode. 0 Disable 1 Enable
5	AAE	Interrupt on async advance enable. When this bit is a one, and USBSTS[AAI] is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[AAI]. Only used in host mode. 0 Disable 1 Enable
4	SEE	System error enable. When this bit is a one, and USBSTS[SEI] is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[SEI]. 0 Disable 1 Enable
3	FRE	Frame list rollover enable. When this bit is a one, and USBSTS[FRI] is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[FRI]. Only used by the host mode. 0 Disable 1 Enable
2	PCE	Port change detect enable. When this bit is a one, and USBSTS[PCI] is a one, the controller will issue an interrupt. The interrupt is acknowledged by software clearing USBSTS[PCI]. 0 Disable 1 Enable
1	UEE	USB error interrupt enable. When this bit is a one, and USBSTS[UEI] is a one, the controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[UEI]. 0 Disable 1 Enable
0	UE	USB interrupt enable. When this bit is a one, and USBSTS[UI] is a one, the DR controller will issue an interrupt at the next interrupt threshold. The interrupt is acknowledged by software clearing USBSTS[UI]. 0 Disable 1 Enable

20.3.2.4 Frame Index Register (FRINDEX)

In host mode, this register is used by the controller to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits N–3 are used to select a particular entry in the periodic frame list during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in USBCMD[FS].

This register must be written as a DWord. Byte writes produce-undefined results. This register cannot be written unless the USB DR controller is in the Halted state as indicated by the USBSTS[HCH]. A write to this register while USBCMD[RS] is set produces undefined results. Writes to this register also affect the SOF value.

In device mode, this register is read-only and, the USB DR controller updates the FRINDEX[13–3] register from the frame number indicated by the SOF marker. Whenever a SOF is received by the USB bus, FRINDEX[13–3] is checked against the SOF marker. If FRINDEX[13–3] is different from the SOF marker, FRINDEX[13–3] is set to the SOF value and FRINDEX[2–0] is cleared (that is, SOF for 1 msec frame). If FRINDEX[13–3] is equal to the SOF value, FRINDEX[2–0] is incremented (that is, SOF for 125-µsec microframe.)

Offset 0x2_314C Access: Read/Write

	31–14	13–0
R W	—	FRINDEX
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	*n n n n n n n n n n n n n n*

Figure 20-11. USB Frame Index (FRINDEX)

Table 20-14. FRINDEX Register Field Descriptions

Bits	Name	Description
31–14	—	Reserved, should be cleared.
13–0	FRINDEX	Frame index. The value in this register increments at the end of each time frame (for example, microframe). Bits N–3 are used for the Frame List current index. This means that each location of the frame list is accessed 8 times (frames or microframes) before moving to the next index. In device mode, the value is the current frame number of the last frame transmitted. It is not used as an index. In either mode, bits 2–0 indicate the current microframe.

Table 20-15 illustrates values of N based on the value of the Frame List Size in the USBCMD register, when used in host mode.

Table 20-15. FRINDEX N Values

USBCMD[FS]	Frame List Size	FRINDEX N value
000	1024 elements (4096 bytes)	12
001	512 elements (2048 bytes)	11
010	256 elements (1024 bytes)	10
011	128 elements (512 bytes)	9
100	64 elements (256 bytes)	8
101	32 elements (128 bytes)	7
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

20.3.2.5 Control Data Structure Segment Register (CTRLDSSEGMENT)

The CTRLDSSEGMENT register is not implemented on the MPC8379E.

20.3.2.6 Periodic Frame List Base Address Register (PERIODICLISTBASE)

This register contains the beginning address of the Periodic Frame List in the system memory. The host controller driver loads this register prior to starting the schedule execution by the controller. The memory structure referenced by this physical memory pointer is assumed to be 4-Kbyte aligned. The contents of this register are combined with the frame index register (FRINDEX) to enable the controller to step through the Periodic Frame List in sequence.

Note that this register is shared between the host and device mode functions. In host mode, it is the PERIODICLISTBASE register; in device mode, it is the DEVICEADDR register. See Section 20.3.2.7, "Device Address Register (DEVICEADDR)—Non-EHCI," for more information.

Offset 0x2_3154 Access: Read/Write

	31–12	11–0
R/W	PERBASE	—
Reset	*n n n n n n n n n n n n n n n n n n n n*	0 0 0 0 0 0 0 0 0 0 0 0

Figure 20-12. Periodic Frame List Base Address (PERIODICLISTBASE)

Table 20-16. PERIODICLISTBASE Register Field Descriptions

Bits	Name	Description
31–12	PERBASE	Base address. Correspond to memory address signal [31:12]. Only used in the host mode.
11–0	—	Reserved, should be cleared.

20.3.2.7 Device Address Register (DEVICEADDR)—Non-EHCI

This register is not defined in the EHCI specification. In device mode, the upper seven bits of this register represent the device address. After any controller reset or a USB reset, the device address is set to the default address (0). The default address will match all incoming addresses. Software shall reprogram the address after receiving a SET_ADDRESS descriptor.

Note that this register is shared between the host and device mode functions. In device mode, it is the DEVICEADDR register; in host mode, it is the PERIODICLISTBASE register. See Section 20.3.2.6, "Periodic Frame List Base Address Register (PERIODICLISTBASE)," for more information.

Offset 0x2_3154 Access: Read/Write

	31–25	24–0
R/W	USBADR	—
Reset	All zeros	

Figure 20-13. Device Address (DEVICEADDR)

Table 20-17. DEVICEADDR Register Field Descriptions

Bits	Name	Description
31–25	USBADR	Device address. This field corresponds to the USB device address.
24–0	—	Reserved, should be cleared.

20.3.2.8 Current Asynchronous List Address Register (ASYNCLISTADDR)

This 32-bit register contains the address of the next asynchronous queue head to be executed by the host. Bits 4–0 of this register cannot be modified by the system software and always return zeros when read.

Note that this register is shared between the host and device mode functions. In host mode, it is the ASYNCLISTADDR register; in device mode, it is the ENDPOINTLISTADDR register. See Section 20.3.2.9, “Endpoint List Address Register (ENDPOINTLISTADDR)—Non-EHCI,” for more information.

Offset 0x2_3158 Access: Read/Write

	31–5	4–0
R/W	ASYBASE	—
Reset	All zeros	

Figure 20-14. Current Asynchronous List Address (ASYNCLISTADDR)

Table 20-18. ASYNCLISTADDR Register Field Descriptions

Bits	Name	Description
31–5	ASYBASE	Link pointer low (LPL). These bits correspond to memory address signals [31:5]. This field may only reference a queue head (QH). Only used by the host controller.
4–0	—	Reserved, should be cleared.

20.3.2.9 Endpoint List Address Register (ENDPOINTLISTADDR)—Non-EHCI

This register is not defined in the EHCI specification. In device mode, this register contains the address of the top of the endpoint list in system memory. Bits 10–0 of this register cannot be modified by the system software and always return zeros when read. The memory structure referenced by this physical memory pointer is assumed to be 64-bytes. The queue head is actually a 48-byte structure, but must be aligned on 64-byte boundary. However, the ENDPOINTLISTADDR[EPBASE] has a granularity of 2 Kbytes, so in practice the queue head should be 2-Kbyte aligned.

Note that this register is shared between the host and device mode functions. In device mode, it is the ENDPOINTLISTADDR register; in host mode, it is the ASYNCLISTADDR register. See Section 20.3.2.8, “Current Asynchronous List Address Register (ASYNCLISTADDR),” for more information.

Offset 0x2_3158 Access: Read/Write

	31–11	10–0
R/W	EPBASE	—
Reset	All zeros	

Figure 20-15. Endpoint List Address (ENDPOINTLISTADDR)

Table 20-19. ENDPOINTLISTADDR Register Field Descriptions

Bits	Name	Description
31–11	EPBASE	Endpoint list address. Address of the top of the endpoint list.
10–0	—	Reserved, should be cleared.

20.3.2.10 Master Interface Data Burst Size Register (BURSTSIZE)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on the initiator (master) interface.

Offset 0x2_3160 Access: Read/Write

	31–16	15–8	7–0
R/W	—	TXPBURST	RXPBURST
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0	0 0 0 1 0 0 0 0

Figure 20-16. Master Interface Data Burst Size (BURSTSIZE)

Table 20-20. BURSTSIZE Register Field Descriptions

Bits	Name	Description
31–16	—	Reserved, should be cleared.
15–8	TXPBURST	Programable TX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from system memory to the USB bus. Must not be set to greater that 16.
7–0	RXPBURST	Programable RX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from the USB bus to system memory. Must not be set to greater than 16.

20.3.2.11 Transmit FIFO Tuning Controls Register (TXFILLTUNING)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to control and dynamically change the burst size used during data movement on device DMA transfers. It is only used in host mode.

The fields in this register control performance tuning associated with how the USB DR module posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include the how much data to post into the FIFO and an estimate for how long that operation should take in the target system.

Definitions:

T_0	Standard packet overhead
T_1	Time to send data payload
T_s	Total Packet Flight Time (send-only) packet ($T_s = T_0 + T_1$)
T_{ff}	Time to fetch packet into TX FIFO up to specified level.
T_p	Total Packet Time (fetch and send) packet ($T_p = T_{ff} + T_s$)

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, host controller checks to ensure T_p remains before the end of the [micro]frame. If so it proceeds to pre-fill the TX FIFO. If at any time during the pre-fill operation the time remaining the [micro]frame is < T_s then the packet attempt ceases and the packet is tried at a later time. Although this is not an error condition and the module eventually recovers, a mark is made in the scheduler health counter to note the occurrence of a back-off event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic that will begin after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and thus should be minimized (not necessarily eliminated). Back-offs can be minimized with use of the TSCHHEALTH (T_{ff}) parameter described below.

Offset 0x2_3164 Access: Read/Write

Bits	31–22	21–16	15–13	12–8	7–0
R/W	—	TXFIFOTHRES	—	TXSCHHEALTH	TXSCHOH
Reset	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0	0 0 0	0 0 0 0 0	0 0 0 0 0 0 0 0

Figure 20-17. Transmit FIFO Tuning Controls (TXFILLTUNING)

Table 20-21. TXFILLTUNING Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared.
21–16	TXFIFOTHRES	FIFO burst threshold. Control the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on to the bus. The minimum value is 2 and this value should be a low as possible to maximize USB performance. A higher value can be used in systems with unpredictable latency and/or insufficient bandwidth where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can be replenished from system memory. This value is ignored if USBMODE[SDIS] (stream disable bit) is set. When USBMODE[SDIS] is set, the host controller behaves as if TXFIFOTHRES is set to the maximum value.
15–13	—	Reserved, should be cleared.
12–8	TXSCHHEALTH	Scheduler health counter. Increment when the host controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next Start-Of-Frame. This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register clears the counter and this counter stops counting after reaching the maximum of 31.

Table 20-21. TXFILLTUNING Register Field Descriptions (continued)

Bits	Name	Description
7–0	TXSCHOH	Scheduler overhead. These bits add an additional fixed offset to the schedule time estimator described above as T_{ff} As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH to less than 10 per second in a highly utilized bus. Choosing a value that is too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is 1.267µs when a device is connected in high-speed mode. The time unit represented in this register is 6.333µs when a device is connected in low-/full-speed mode. For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: TXFIFOTHRES × (BURSTSIZE × 4 bytes-per-word) ÷ (40 × *TimeUnit*), always rounded to the next higher integer. *TimeUnit* is either 1.267 or 6.333 as noted earlier in this description. For example, if TXFIFOTHRES is 5 and BURSTSIZE is 8, then set TXSCHOH to 5×(8×4)÷(40×1.267)=4 for a high-speed link. If this value of TXSCHOH results in a TXSCHHEALTH count of 0 per second, try lowering the value by 1 if optimizing performance is desired. If TXSCHHEALTH exceeds 10 per second, try raising the value by 1. If streaming mode is disabled via the USBMODE register, treat TXFIFOTHRES as the maximum value for purposes of the TXSCHOH calculation.

20.3.2.12 ULPI Register Access (ULPI VIEWPORT)

The register provides indirect access to the ULPI PHY register set. Although the controller modules perform access to the ULPI PHY register set, there may be extraordinary circumstances where software may need direct access. Be advised that writes to the ULPI through the ULPI viewport can substantially harm standard USB operations. Currently no usage model has been defined where software should need to execute writes directly to the ULPI. Note that executing read operations though the ULPI viewport should have no harmful side effects to standard USB operations. Also note that if the ULPI interface is not enabled, this register will always read zeros.

ULPI VIEWPORT is shown in Figure 20-18.

Offset 0x2_3170 Access: Mixed

	31	30	29	28	27	26–24	23–16
R	ULPIWU	ULPIRUN	ULPIRW	—	ULPISS	ULPIPORT	ULPIADDR
W							
Reset	All zeros						

	15–8	7–0
R	ULPIDATRD	ULPIDTWR
W		
Reset	All zeros	

Figure 20-18. ULPI Register Access (ULPI VIEWPORT)

Table 20-22. ULPI VIEWPORT Field Descriptions

Bits	Name	Description
31	ULPIWU	ULPI Wake Up. Writing 1 to this bit begins the wakeup operation. This bit automatically transitions to 0 after the wakeup is complete. Once this bit is set, it can not be cleared by software. **Note:** The driver must never execute a wakeup and a read/write operation at the same time.
30	ULPIRUN	ULPI Run. Writing 1 to this bit begins a read/write operation. This bit automatically transitions to 0 after the read/write is complete. Once this bit is set, it can not be cleared by software. **Note:** The driver must never execute a wakeup and a read/write operation at the same time.
29	ULPIRW	This bit selects between running a read or write operation to the ULPI. 0 Read 1 Write
28	—	Reserved, should be cleared.
27	ULPISS	This bit represents the state of the ULPI interface. Before reading this bit, the ULPIPORT field should be set accordingly if used with the multi-port host. Otherwise, this field should always remain 0. 0 Any other state (that is, carkit, serial, low power). 1 Normal Sync State.
26–24	ULPIPORT	For wakeup or read/write operations this value selects the port number to which the ULPI PHY is attached. Valid values are 0 and 1.
23–16	ULPIADDR	When a read or write operation is commanded, the address of the operation is written to this field.
15–8	ULPIDATRD	After a read operation completes, the result is placed in this field.
7–0	ULPIDTWR	When a write operation is commanded, the data to be sent is written to this field.

There are two operations that can be performed with the ULPI viewport, wakeup and read /write operations. The wakeup operation is used to put the ULPI interface into normal operation mode and re-enable the clock if necessary. A wakeup operation is required before accessing the registers when the ULPI interface is operating in low power mode, serial mode, or carkit mode. The ULPI state can be determined by reading the sync state bit (ULPISS). If this bit is set, then the ULPI interface is running in normal operation mode and can accept read/write operations. If the ULPISS is cleared, then read/write operations will not be able execute. Undefined behavior results if a read or write operation is performed when ULPISS is cleared. To execute a wakeup operation, write all 32-bits of the ULPI Viewport where ULPIPORT is constructed appropriately and the ULPIWU bit is set and the ULPIRUN bit is cleared. Poll the ULPI Viewport until ULPIWU is cleared for the operation to complete.

To execute a read or write operation, write all 32-bits of the ULPI Viewport where ULPIDATWR, ULPIADDR, ULPIPORT, ULPIRW are constructed appropriately and the ULPIRUN bit is set. Poll the ULPI Viewport until ULPIRUN is cleared for the operation to complete. For read operations, ULPIDATRD is valid once ULPIRUN is cleared.

The polling method above can be replaced with interrupts using the ULPI interrupt defined in the USBSTS and USBINTR registers. When a wakeup or read/write operation completes, the ULPI interrupt is set.

20.3.2.13 Configure Flag Register (CONFIGFLAG)

This EHCI register is not used in this implementation. A read from this register returns a constant of a 0x0000_0001 to indicate that all port routings default to this host controller.

Offset 0x2_3180 Access: Read only

	31–1	0
R	—	CF
W		
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1

Figure 20-19. Configure Flag Register (CONFIGFLAG)

Table 20-23. CONFIGFLAG Register Field Descriptions

Bits	Name	Description
31–1	—	Reserved.
0	CF	Configure flag. Always 1 indicating all port routings default to this host.

20.3.2.14 Port Status and Control Register (PORTSC)

The port status and control (PORTSC) register is only reset when power is initially applied or in response to a controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remaivPns until software applies power to the port by setting port power to one.

In device mode, the USB DR controller does not support power control. Port control in device mode is only used for status port reset, suspend, and current connect status. It is also used to initiate test mode or force signaling and allows software to put the PHY into low power suspend mode and disable the PHY clock.

Offset 0x2_3184 Access: Mixed

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PTS		—	—	PSPD		—	PFSC	PHCD	WKOC	WKDS	WLCN	PTC			
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIC		PO	PP	LS		—	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS
W											w1c		w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 20-20. Port Status and Control (PORTSC)

Table 20-24. PORTSC Register Field Descriptions

Bits	Name	Description
31–30	PTS	Port transceiver select. This register bit is used to control which parallel transceiver interface is selected. 00 Reserved 01 Reserved, should be cleared 10 ULPI parallel interface 11 FS/LS serial interface This bit is not defined in the EHCI specification.
29	—	Reserved, should be cleared
28	—	Reserved
27–26	PSPD	Port speed. This read-only register field indicates the speed at which the port is operating. This bit is not defined in the EHCI specification. 00 Full-speed 01 Low-speed 10 High-speed 11 Undefined
25	—	Reserved, should be cleared
24	PFSC	Port force full-speed connect. Used to disable the chirp sequence that allows the port to identify itself as a HS port. This is useful for testing FS configurations with a HS host, hub or device. 0 Allow the port to identify itself as high speed. 1 Force the port to only connect at full speed. This bit is not defined in the EHCI specification. This bit is for debugging purposes.
23	PHCD	PHY low power suspend. This bit is not defined in the EHCI specification. Host mode: • The PHY can be put into low power suspend – when the downstream device has been put into suspend mode or when no downstream device is connected. Low power suspend is completely under the control of software. Device mode: • The PHY can be put into low power suspend – when the device is not running (USBCMD[RS] = 0b) or suspend signaling is detected on the USB. Low power suspend will be cleared automatically when the resume signaling has been detected or when forcing port resume. 0 Normal PHY operation. 1 Signal the PHY to enter low power suspend mode Reading this bit indicates the status of the PHY. **Note:** If there is no clock connected to the USBDR_CLK signals, PHCD must be set and the following registers should not be written: DEVICE_ADDR/PERIODICLISTBASE, PORTSC, ENDPTCTRL0, ENDPTCTRL1, ENDPTCTRL2, ENDPTCTRL3, ENDPTCTRL4, ENDPTCTRL5.
22	WKOC	Wake on over-current enable. Writing this bit to a one enables the port to be sensitive to over-current conditions as wake-up events. This field is zero if Port Power (PP) is zero. This bit is (OTG/host mode only) for use by an external power control circuit.
21	WKDS	Wake on disconnect enable. Writing this bit to a one enables the port to be sensitive to device disconnects as wake-up events. This field is zero if Port Power(PP) is zero or in device mode. This bit is (OTG/host mode only) for use by an external power control circuit.

Table 20-24. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
20	WLCN	Wake on connect enable. Writing this bit to a one enables the port to be sensitive to device connects as wake-up events. This field is zero if Port Power(PP) is zero or in device mode. This bit is (OTG/host mode only) for use by an external power control circuit.
19–16	PTC	Port test control. Any other value than zero indicates that the port is operating in test mode. 0000 Not Enabled 0001 J_STATE 0010 K_STATE 0011 SEQ_NAK 0100 Packet 0101 FORCE_ENABLE 0110–1111 Reserved, should be cleared Refer to Chapter 7 of the USB Specification Revision 2.0 [3] for details on each test mode.
15–14	PIC	Port indicator control. Control the link indicator signals. These signals are valid for host mode only. 00 Off 01 Amber 10 Green 11 Undefined Refer to the USB Specification Revision 2.0 [3] for a description on how these bits are to be used. This field is output from the module on the USB port control signals for use by an external LED driving circuit.
13	PO	Port owner. Unconditionally goes to a 0 when the configured bit in the CONFIGFLAG register makes a 0 to 1 transition. This bit unconditionally goes to 1 whenever the Configured bit is zero. System software uses this field to release ownership of the port to a selected the module (in the event that the attached device is not a high-speed device). Software writes a one to this bit when the attached device is not a high-speed device. A one in this bit means that an internal companion controller owns and controls the port. Port owner hand-off is not implemented in this design, therefore this bit is always 0.
12	PP	Port power. Represents the current setting of the switch (0=off, 1=on). When power is not available on a port (that is, PP equals a 0), the port is non-functional and will not report attaches, detaches, etc. When an over-current condition is detected on a powered port, the PP bit in each affected port is transitioned by the host controller driver from a one to a zero (removing power from the port). This feature is implemented in the host/OTG controller (PPC = 1). In a device-only implementation port power control is not necessary, thus PPC and PP = 0.
11–10	LS	Line status. Reflect the current logical levels of the USB D+ (bit 11) and D– (bit 10) signal lines. The use of line status by the host controller driver is not necessary (unlike EHCI), because the connection of FS and LS is managed by hardware. 00 SE0 10 J-state 01 K-state 11 Undefined
9	—	Reserved, should be cleared

Table 20-24. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
8	PR	Port reset. Host mode: • When software writes a one to this bit the bus-reset sequence as defined in the USB Specification Revision 2.0 is started. This bit will automatically change to zero after the reset sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the reset duration is timed in the driver. Device mode: • This bit is a read only status bit. Device reset from the USB bus is also indicated in the USBSTS register. 1 Port is in reset. 0 Port is not in reset. This field is zero if Port Power(PP) is zero.
7	SUSP	Suspend. Host mode: • The port enabled bit (PE) and suspend (SUSP) bit define the port states as follows: 0x Disable 10 Enable 11 Suspend • When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. Note that the bit status does not change until the port is suspended and that there may be a delay in suspending a port if there is a transaction currently in progress on the USB. • The module unconditionally sets this bit to zero when software clears the FPR bit. A write of zero to this bit is ignored by the host controller. If host software sets this bit to a one when the port is not enabled (that is, port enabled bit is a zero) the results are undefined. • This field is zero if Port Power (PP) is zero in host mode. Device mode: 1 Port in suspend state. 0 Port not in suspend state. Default. In device mode this bit is a read-only status bit.
6	FPR	Force port resume. This bit is not-EHCI compatible. 1 Resume detected/driven on port. 0 No resume (K-state) detected/driven on port. Host mode: • Software sets this bit to one to drive resume signaling. The controller sets this bit to one if a J-to-K transition is detected while the port is in the Suspend state. When this bit transitions to a one a J-to-K transition is detected, USBSTS[PCI] (port change detect) is also set. This bit will automatically change to zero after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to set this bit to a zero after the resume duration is timed in the driver. • Note that when the controller owns the port, the resume sequence follows the defined sequence documented in the USB Specification Revision 2.0. The resume signaling (Full-speed 'K') is driven on the port as long as this bit remains a one. This bit will remain a one until the port has switched to the high-speed idle. Writing a zero has no affect because the port controller will time the resume operation clear the bit the port control state switches to HS or FS idle. • This field is zero if Port Power (PP) is zero in host mode. Device mode: • After the device has been in Suspend State for 5 msec or more, software must set this bit to one to drive resume signaling before clearing. The USB DR controller will set this bit to one if a J-to-K transition is detected while the port is in the Suspend state. The bit will be cleared when the device returns to normal operation. Also, when this bit transitions to a one because a J-to-K transition detected, USBSTS[PCI] is also set.

Table 20-24. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
5	OCC	Over-current change. This bit gets set when there is a change to over-current active. Software clears this bit by writing a one to this bit position. Host/OTG mode: • The user can provide over-current detection to the USB*n*_PWRFAULT signal for this condition. Device mode: • This bit must always be 0. 1 Over current detect. 0 No over current.
4	OCA	Over-current active. This bit will automatically transition from one to zero when the over current condition is removed. Host/OTG mode: • The user can provide over-current detection to the USB*n*_PWRFAULT signal for this condition. Device mode: • This bit must always be 0. 1 Port currently in over-current condition. 0 Port not in over-current condition.
3	PEC	Port enable/disable change. For the root hub, this bit gets set only when a port is disabled due to disconnect on the port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the USB Specification). Software clears this by writing a one to it.[In device mode: • The device port is always enabled. (This bit will be zero.) 1 Port disabled. 0 No change. This field is zero if Port Power(PP) is zero.
2	PE	Port enabled/disabled. Host mode: • Ports can only be enabled by the controller as a part of the reset and enable. Software cannot enable a port by writing a one to this field. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by the host software. Note that the bit status does not change until the port state actually changes. There may be a delay in disabling or enabling a port due to other host and bus events. • When the port is disabled, (0) downstream propagation of data is blocked except for reset. • This field is zero if Port Power(PP) is zero in host mode. Device mode: • The device port is always enabled. (This bit will be one.)

Table 20-24. PORTSC Register Field Descriptions (continued)

Bits	Name	Description
1	CSC	Connect change status. Host mode: • This bit indicates a change has occurred in the port's Current Connect Status. the controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, hub hardware will be 'setting' an already-set bit (i.e., the bit will remain set). Software clears this bit by writing a one to it. 1 Connect Status has changed. 0 No change. • This field is zero if Port Power(PP) is zero. Device mode: • This bit is undefined.
0	CCS	Current connect status. Host mode: 1 Device is present 0 No device present. This field is zero if Port Power(PP) is zero in host mode. In device mode: 1 Attached 0 Not attached. A one indicates that the device successfully attached and is operating in either high-speed or full-speed as indicated by the High Speed Port bit in this register. A zero indicates that the device did not attach successfully or was forcibly disconnected by the software writing a zero to USBCMD[RS] (run bit). It does not state the device being disconnected or suspended.

20.3.2.15 On-The-Go Status and Control (OTGSC)—Non-EHCI

This register is not defined in the EHCI specification. The USB DR module implements one On-The-Go (OTG) status and control register corresponding to Port 0.

The OTGSC register has four sections:

- OTG interrupt enables (Read/Write)
- OTG interrupt status (Read/Write to Clear)
- OTG status inputs (Read Only)
- OTG controls (Read/Write)

The status inputs are de-bounced using a 1-msec time constant. Values on the status inputs that do not persist for more than 1 msec will not cause an update of the status inputs, or cause and OTG interrupt.

Offset 0x2_31A4 Access: Mixed

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	—	DPIE	1msE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE	—	DPIS	1msS	BSEIS	BSVIS	ASVIS	AVVIS	IDIS
W										w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	—	DPS	1msT	BSE	BSV	ASV	AVV	ID	—			DP	OT	—	VC	VD
W																
Reset	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0

Figure 20-21. OTG Status Control (OTGSC)

Table 20-25. OTGSC Register Field Descriptions

Bits	Name	Description
31	—	Reserved, should be cleared.
30	DPIE	Data pulse interrupt enable 1 Enable 0 Disable
29	1msE	1-millisecond timer Interrupt enable 1 Enable 0 Disable
28	BSEIE	B session end interrupt enable 1 Enable 0 Disable
27	BSVIE	B session valid interrupt enable 1 Enable 0 Disable
26	ASVIE	A session valid interrupt enable 1 Enable 0 Disable
25	AVVIE	A VBus valid interrupt enable 1 Enable 0 Disable
24	IDIE	USB ID interrupt enable. 1 Enable 0 Disable
23	—	Reserved, should be cleared.
22	DPIS	Data pulse interrupt status. Set when data bus pulsing occurs on DP or DM. Data bus pulsing is only detected when USBMODE[CM] = Host (11) and PORTSC[PP] (port power) = Off (0). Software must write a one to clear this bit.
21	1msS	1-millisecond timer interrupt status. Set once every millisecond. Software must write a one to clear this bit.

Table 20-25. OTGSC Register Field Descriptions (continued)

Bits	Name	Description
20	BSEIS	B session end interrupt status. Set when VBus has fallen below the B session end threshold. Software must write a one to clear this bit.
19	BSVIS	B session valid interrupt status. Set when VBus has either risen above or fallen below the B session valid threshold (0.8 VDC). Software must write a one to clear this bit.
18	ASVIS	A session valid interrupt status. Set when VBus has either risen above or fallen below the A session valid threshold (0.8 VDC). Software must write a one to clear this bit.
17	AVVIS	A VBus valid interrupt status. Set when VBus has either risen above or fallen below the VBus valid threshold (4.4 VDC) on an A device. Software must write a one to clear this bit.
16	IDIS	USB ID interrupt status. Set when a change on the ID input has been detected. Software must write a one to clear this bit.
15	—	Reserved, should be cleared.
14	DPS	Data bus pulsing status 1 Pulsing detected on port 0 No pulsing on port
13	1msT	1 millisecond timer toggle. This bit toggles once per millisecond.
12	BSE	B session end 1 VBus is below the B session end threshold. 0 VBus is above the B session end threshold.
11	BSV	B session valid 1 VBus is above the B session valid threshold. 0 VBus is below the B session valid threshold.
10	ASV	A session valid 1 VBus is above the A session valid threshold. 0 VBus is below the A session valid threshold.
9	AVV	A VBus valid 1 VBus is above the A VBus valid threshold. 0 VBus is below the A VBus valid threshold.
8	ID	USB ID 1 B device 0 A device
7–5	—	Reserved, should be cleared.
4	DP	Data pulsing 1 The pullup on DP is asserted for data pulsing during SRP. 0 The pullup on DP is not asserted.
3	OT	OTG termination. This bit must be set when the OTG device is in device mode. 1 Enable pulldown on DM 0 Disable pulldown on DM
2	—	Reserved, should be cleared.

Table 20-25. OTGSC Register Field Descriptions (continued)

Bits	Name	Description
1	VC	VBUS charge. Setting this bit causes the VBus line to be charged. This is used for VBus pulsing during SRP.
0	VD	VBUS discharge. Setting this bit causes VBus to discharge through a resistor.

20.3.2.16 USB Mode Register (USBMODE)—Non-EHCI

This register is not defined in the EHCI specification. This register controls the operating mode of the module.

Offset 0x2_31A8 Access: Read/Write

	31–5	4	3	2	1–0
R/W	—	SDIS	SLOM	—	CM
Reset	All zeros				

Figure 20-22. USB Mode (USBMODE)

Table 20-26. USBMODE Register Field Descriptions

Bits	Name	Description
31–5	—	Reserved, should be cleared.
4	SDIS	Stream disable In host mode, setting this bit ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring the TX latency is filled to capacity before the packet is launched onto the USB. Note that time duration to pre-fill the FIFO becomes significant when stream disable is active. See TXFILLTUNING to characterize the adjustments needed for the scheduler when using this feature. Also note that in systems with high system bus utilization, setting this bit will ensure no overruns or underruns during operation, at the expense of link utilization. For those who desire optimal link performance, SDIS can be left clear, and the rules used under the description of the TXFILLTUNING register to limit underruns/overruns. 1 Active. 0 Inactive. In device mode, setting this bit disables double priming on both RX and TX for low bandwidth systems. This mode ensures that when the RX and TX buffers are sufficient to contain an entire packet that the standard double buffering scheme is disabled to prevent overruns/underruns in bandwidth limited systems. Note that in high-speed mode, all packets received will be responded to with a NYET handshake when stream disable is active.
3	SLOM	Setup lockout mode. In device mode, this bit controls behavior of the setup lock mechanism. See Section 20.8.3.5, “Control Endpoint Operation Model.” 1 Setup lockouts off. DCD requires use of setup data buffer tripwire in USBCMD (SUTW). 0 Setup lockouts on

Table 20-26. USBMODE Register Field Descriptions (continued)

Bits	Name	Description
2	—	Reserved, should be cleared.
1–0	CM	Controller mode This register can only be written once after reset. If it is necessary to switch modes, software must reset the controller by writing to USBCMD[RST] before reprogramming this register. 00 Idle (default for combination host/device). 01 Reserved, should be cleared. 10 Device controller (default for device only controller). 11 Host controller (default for host only controller). Defaults to the idle state and needs to be initialized to the desired operating mode after reset.

20.3.2.17 Endpoint Setup Status Register (ENDPTSETUPSTAT)—Non-EHCI

This register is not defined in the EHCI specification. This register contains the endpoint setup status. It is only used in device mode.

Offset 0x2_31AC — Access: Read/Write

	31–6	5–0
R/W	—	ENDPTSETUPSTAT
Reset	All zeros	

Figure 20-23. Endpoint Setup Status (ENDPTSETUPSTAT)

Table 20-27. ENDPTSETUPSTAT Register Field Descriptions

Bits	Name	Description
31–6	—	Reserved, should be cleared.
5–0	ENDPTSETUP STAT	Setup endpoint status. For every setup transaction that is received, a corresponding bit in this register is set. Software must clear or acknowledge the setup transfer by writing a one to a respective bit after it has read the setup data from queue head. The response to a setup packet as in the order of operations and total response time is crucial to limit bus time outs while the setup lockout mechanism is engaged. This register is only used in device mode.

20.3.2.18 Endpoint Initialization Register (ENDPTPRIME)—Non-EHCI

This register is not defined in the EHCI specification. This register is used to initialize endpoints. It is only used in device mode.

Offset 0x2_31B0 — Access: Read/Write

	31–22	21–16	15–6	5–0
R/W	—	PETB	—	PERB
Reset	All zeros			

Figure 20-24. Endpoint Initialization (ENDPTPRIME)

Table 20-28. ENDPTPRIME Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared.
21–16	PETB	Prime endpoint transmit buffer. For each endpoint a corresponding bit is used to request that a buffer prepared for a transmit operation in order to respond to a USB IN/INTERRUPT transaction. Software should write a one to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PETB[5] (bit 21 of the register) corresponds to endpoint 5. Note that these bits will be momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.
15–6	—	Reserved, should be cleared.
5–0	PERB	Prime endpoint receive buffer. For each endpoint, a corresponding bit is used to request a buffer prepare for a receive operation in order to respond to a USB OUT transaction. Software should write a one to the corresponding bit whenever posting a new transfer descriptor to an endpoint. Hardware will automatically use this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. Hardware will clear this bit when the associated endpoint(s) is (are) successfully primed. PERB[5] corresponds to endpoint 5. Note that these bits will be momentarily set by hardware during hardware re-priming operations when a dTD is retired, and the dQH is updated.

20.3.2.19 Endpoint Flush Register (ENDPTFLUSH)—Non-EHCI

This register is not defined in the EHCI specification. This register is only used in device mode.

Offset 0x2_31B4 Access: Read/Write

	31–22	21–16	15–6	5–0
R W	—	FETB	—	FERB
Reset	All zeros			

Figure 20-25. Endpoint Flush (ENDPTFLUSH)

Table 20-29. ENDPTFLUSH Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared.
21–16	FETB	Flush endpoint transmit buffer. Writing a one to a bit(s) in this register will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FETB[5] (bit 21 of the register) corresponds to endpoint 5.
15–6	—	Reserved, should be cleared.
5–0	FERB	Flush endpoint receive buffer. Writing a one to a bit(s) will cause the associated endpoint(s) to clear any primed buffers. If a packet is in progress for one of the associated endpoints, then that transfer will continue until completion. Hardware will clear this register after the endpoint flush operation is successful. FERB[5] corresponds to endpoint 5.

20.3.2.20 Endpoint Status Register (ENDPTSTATUS)—Non-EHCI

This register is not defined in the EHCI specification. This register is only used in device mode.

Offset 0x2_31B8 Access: Read only

	31–22	21–16	15–6	5–0
R	—	ETBR	—	ERBR
W				
Reset	All zeros			

Figure 20-26. Endpoint Status (ENDPTSTATUS)

Table 20-30. ENDPTSTATUS Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared
21–16	ETBR	Endpoint transmit buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ETBR[5] (bit 21 of the register) corresponds to endpoint 5. Note that these bits will be momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.
15–6	—	Reserved, should be cleared
5–0	ERBR	Endpoint receive buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. This bit is set by the hardware as a response to receiving a command from a corresponding bit in the ENDPTPRIME register. There will always be a delay between setting a bit in the ENDPTPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the ENDPTPRIME register. Buffer ready is cleared by USB reset, by the USB DMA system, or through the ENDPTFLUSH register. ERBR[5] corresponds to endpoint 5. Note that these bits will be momentarily cleared by hardware during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.

20.3.2.21 Endpoint Complete Register (ENDPTCOMPLETE)—Non-EHCI

This register is not defined in the EHCI specification. This register is only used in device mode.

Offset 0x2_31BC Access: w1c

	31–22	21–16	15–6	5–0
R	—	ETCE	—	ERCE
W		w1c		w1c
Reset	All zeros			

Figure 20-27. Endpoint Complete (ENDPTCOMPLETE)

Table 20-31. ENDPTCOMPLETE Register Field Descriptions

Bits	Name	Description
31–22	—	Reserved, should be cleared
21–16	ETCE	Endpoint transmit complete event. Each bit indicates a transmit event (IN/INTERRUPT) occurred and software should read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ETCE[5] (bit 21 of the register) corresponds to endpoint 5.
15–6	—	Reserved, should be cleared
5–0	ERCE	Endpoint receive complete event. Each bit indicates a received event (OUT/SETUP) occurred and software should read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the Transfer Descriptor, then this bit will be set simultaneously with the USBINT. Writing a one will clear the corresponding bit in this register. ERCE[5] corresponds to endpoint 5.

20.3.2.22 Endpoint Control Register 0 (ENDPTCTRL0)—Non-EHCI

This register is not defined in the EHCI specification. Every device will implement endpoint 0 as a control endpoint.

Offset 0x2_31C0 Access: Mixed

	31–24	23	22–20	19–18	17	16	15–8	7	6–4	3–2	1	0
R	—	TXE	—	TXT	—	TXS	—	RXE	—	RXT	—	RXS
W												
Reset	0 0 0 0 0 0 0 0	1	0 0 0	0 0	0	0	0 0 0 0 0 0 0 0	1	0 0 0	0 0	0	0

Figure 20-28. Endpoint Control 0 (ENDPTCTRL0)

Table 20-32. ENDPTCTRL0 Register Field Descriptions

Bits	Name	Description
31–24	—	Reserved, should be cleared.
23	TXE	TX endpoint enable. Endpoint zero is always enabled. 0 Disable 1 Enable
22–20	—	Reserved, should be cleared.
19–18	TXT	TX endpoint type. Endpoint zero is always a control endpoint (00).
17	—	Reserved, should be cleared.
16	TXS	TX endpoint stall. Software can write a one to this bit to force the endpoint to return a STALL handshake to the Host. It will continue returning STALL until the bit is cleared by software or it will automatically be cleared upon receipt of a new SETUP request. 1 Endpoint stalled 0 Endpoint OK
15–8	—	Reserved, should be cleared.
7	RXE	RX endpoint enable. Endpoint zero is always enabled. 0 Disabled 1 Enabled

Table 20-32. ENDPTCTRL0 Register Field Descriptions (continued)

Bits	Name	Description
6–4	—	Reserved, should be cleared.
3–2	RXT	RX endpoint type. Endpoint zero is always a control endpoint (00).
1	—	Reserved, should be cleared.
0	RXS	RX endpoint stall Software can write a one to this bit to force the endpoint to return a STALL handshake to the host. It will continue returning STALL until the bit is cleared by software or it will automatically be cleared upon receipt of a new SETUP request. 1 Endpoint stalled 0 Endpoint OK

20.3.2.23 Endpoint Control Register *n* (ENDPTCTRL*n*)—Non-EHCI

These registers are not defined in the EHCI specification. There is an ENDPTCTRL*n* register of each endpoint in a device.

Offset 0x2_31C4 (ENDPTCTRL1), 0x2_31C8 (ENDPTCTRL2), 0x2_31CA (ENDPTCTRL3), 0x2_31D0 (ENDPTCTRL4), 0x2_31D4 (ENDPTCTRL5) — Access: Read/Write

	31–24	23	22	21	20	19–18	17	16	15–8	7	6	5	4	3–2	1	0
R/W	—	TXE	TXR	TXI	—	TXT	TXD	TXS	—	RXE	RXR	RXI	—	RXT	RXD	RXS
Reset	All zeros															

Figure 20-29. Endpoint Control 1 to 5 (ENDPTCTRL*n*)

Table 20-33. ENDPTCTRL*n* Register Field Descriptions

Bits	Name	Description
31–24	—	Reserved, should be cleared
23	TXE	TX endpoint enable 0 Disabled 1 Enabled
22	TXR	TX data toggle reset. Whenever a configuration event is received for this endpoint, software must write a one to this bit in order to synchronize the data PID's between the Host and device.
21	TXI	TX data toggle inhibit. Used only for test and should always be written as zero. Writing a one to this bit will cause this endpoint to ignore the data toggle sequence and always transmit DATA0 for a data packet. 0 PID sequencing enabled 1 PID sequencing disabled
20	—	Reserved, should be cleared
19–18	TXT	TX endpoint type 00 Control 01 Isochronous 10 Bulk 11 Interrupt **Note:** When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.

Table 20-33. ENDPTCTRL*n* Register Field Descriptions (continued)

Bits	Name	Description
17	TXD	TX endpoint data source. This bit should always be written as 0, which selects the dual port memory/DMA engine as the source.
16	TXS	TX endpoint stall. This bit will be set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It will be cleared automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint. Software can write a one to this bit to force the endpoint to return a STALL handshake to the host. It will continue to returning STALL until this bit is either cleared by software or automatically cleared as above. 0 Endpoint OK 1 Endpoint stalled
15–8	—	Reserved, should be cleared
7	RXE	RX endpoint enable 0 Disabled 1 Enabled
6	RXR	RX data toggle reset. Whenever a configuration event is received for this endpoint, software must write a one to this bit in order to synchronize the data PID's between the Host and device.
5	RXI	RX data toggle inhibit. This bit is only used for test and should always be written as zero. Writing a one to this bit will cause this endpoint to ignore the data toggle sequence and always accept data packets regardless of their data PID. 1 PID sequencing enabled 0 PID sequencing disabled
4	—	Reserved, should be cleared
3–2	RXT	RX endpoint type 00 Control 01 Isochronous 10 Bulk 11 Interrupt **Note:** When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.
1	RXD	RX endpoint data sink. This bit should always be written as 0, which selects the dual port memory/DMA engine as the sink.
0	RXS	RX endpoint stall. This bit will be set automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It will be cleared automatically upon receipt a SETUP request if this endpoint is configured as a control endpoint, Software can write a one to this bit to force the endpoint to return a STALL handshake to the host. It will continue to returning STALL until this bit is either cleared by software or automatically cleared as above, 1 Endpoint stalled 0 Endpoint OK

20.3.2.24 SNOOP1 and SNOOP2—Non-EHCI

Note that these registers use big-endian byte ordering and are not defined in the EHCI specification. The SNOOP1 and SNOOP2 registers provide snooping control and address range selection function. Transactions that hit a snooping window will generate cache coherent transactions on the internal CSB bus. When the five lower bits (SNOOP*n*[27–31]) are equal to 00000, snooping is always disabled on the CSB for all DMA transfers. When SNOOP*n*[27–31] is 01011 through 11110, the twenty upper bits (SNOOP*n*[0–19]) provide the starting base address for which transactions are snooped. These twenty bits are compared to the twenty upper bits of the address provided by the DMA block of the USB controller. When a match occurs, the five lower bits are decoded as shown below. This provides a snooping region of 4 Kbytes to 2 Gbytes within each starting base address that is programmed by the core. The SNOOP*n*[20–26] are not used.

Offset 0x2_3400 (SNOOP1), 0x2_3404 (SNOOP2) Access: Read/Write

	0–19	20–26	27–31
R W	Snoop Address	—	Snoop Enables
Reset	All zeros		

Figure 20-30. Snoop 1 and Snoop 2 (SNOOP*n*)

Table 20-34. SNOOP*n* Register Field Descriptions

Bits	Name	Description
0–19	Snoop address	The starting base address for which transactions are snooped.
20–26	—	Reserved, should be cleared
27–31	Snoop Enables	0x00 Snooping disabled 0x0B 4-Kbyte snoop range starting at the value defined by SNOOP*n*[0–19] 0x0C 8-Kbyte snoop range starting at the value defined by SNOOP*n*[0–18] 0x0D 16-Kbyte snoop range starting at the value defined by SNOOP*n*[0–17] 0x0E 32-Kbyte snoop range starting at the value defined by SNOOP*n*[0–16] 0x0F 64-Kbyte snoop range starting at the value defined by SNOOP*n*[0–15] 0x10 128-Kbyte snoop range starting at the value defined by SNOOP*n*[0–14] 0x11 256-Kbyte snoop range starting at the value defined by SNOOP*n*[0–13] 0x12 512-Kbyte snoop range starting at the value defined by SNOOP*n*[0–12] 0x13 1-Mbyte snoop range starting at the value defined by SNOOP*n*[0–11] 0x14 2-Mbyte snoop range starting at the value defined by SNOOP*n*[0–10] 0x15 4-Mbyte snoop range starting at the value defined by SNOOP*n*[0–9] 0x16 8-Mbyte snoop range starting at the value defined by SNOOP*n*[0–8] 0x17 16-Mbyte snoop range starting at the value defined by SNOOP*n*[0–7] 0x18 32-Mbyte snoop range starting at the value defined by SNOOP*n*[0–6] 0x19 64-M byte snoop range starting at the value defined by SNOOP*n*[0–5] 0x1A 31-Mbyte snoop range starting at the value defined by SNOOP*n*[0–4] 0x1B 256-Mbyte snoop range starting at the value defined by SNOOP*n*[0–3] 0x1C 512-Mbyte snoop range starting at the value defined by SNOOP*n*[0–2] 0x1D 1-Gbyte snoop range starting at the value defined by SNOOP*n*[0–1] 0x1E 2-Gbyte snoop range starting at the value defined by SNOOP*n*[0]

20.3.2.25 Age Count Threshold Register (AGE_CNT_THRESH)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The age count threshold (AGE_CNT_THRESH) register provides the aging counter threshold value used to determine the priority state of the USB DR controller's internal system interface. This is used to increase the priority state of the module's system interface from zero to one. The actual priority level on the system bus for each state is defined by the PRI_CTRL register. See Section 6.3.1.1, "Address Bus Arbitration with PRIORITY[0:1]," for more details on bus priority. The threshold value is in units of *csb_clk* cycles. This register should be written during system initialization or during normal system operation when the system bus interface is idle. It can be read at any time.

If the aging counter is less than the AGE_CNT_THRESH value, priority state zero is chosen. If the aging counter is greater than or equal to the AGE_CNT_THRESH value, priority state one is chosen.

The aging counter begins to count from zero when a bus access is requested. It increments every bus cycle until the bus transaction completes. At the completion of a bus transaction, the counter is synchronously reset to zero. If there are any outstanding bus requests, the aging counter will then begin counting immediately.

The AGE_CNT_THRESH is compared against the value of the aging counter during each clock cycle of the current transaction. If AGE_CNT_THRESH is equal to zero, priority state one is always chosen. If the aging counter is less than the AGE_CNT_THRESH value, priority state zero is selected. If the aging counter is greater than or equal to the AGE_CNT_THRESH value, priority state one is selected.

The two priority states of the aging counter function each have corresponding register bits which are programmed by the CPU. Thus, when the aging counter function is at priority state zero, PRI_CTRL[30–31] are selected and used to drive bus priority levels. When the aging counter function is at priority state one, PRI_CTRL[28–29] are selected and used to drive the priority.

Offset 0x2_3408 Access: Read/Write

	0–17	18–31
R W	—	Threshold
Reset	All zeros	

Figure 20-31. Age Count Threshold (AGE_CNT_THRESH)

Table 20-35. AGE_CNT_THRESH Register Field Descriptions

Bits	Name	Description
0–17	—	Reserved, should be cleared
18–31	Threshold	Aging counter threshold value.

The setting of AGE_CNT_THRESH is highly dependent on both the mix of other controllers operating on the system bus as well as the kind of traffic moving through the USB controller. A recommended approach is first to try leaving the aging mechanism disabled and see if the USB meets performance requirements. If USB performance does not meet application requirements, try the following settings:

- Set PRI_CTRL[pri_lvl0] to 0.
- Set tPRI_CTRL[pri_lvl1] to 3.
- Set AGE_CNT_THRESH to 40.

This combination works for a wide variety of applications. If this combination still does not meet application requirements, try lowering AGE_CNT_THRESH by 5. On the contrary, the setting 40 may be too conservative for some applications. If USB performance is acceptable at 40, try raising the value in increments of 5. Raising AGE_CNT_THRESH benefits the other controllers on the system bus by reducing the frequency that this USB controller raises its priority to the arbiter.

20.3.2.26 Priority Control Register (PRI_CTRL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The priority control (PRI_CTRL) register sets the priority level for each of two priority states. The priority state is determined by the value programmed in the AGE_CNT_THRESH register and the number of *csb_clk* cycles that a particular transaction takes to complete.

Offset 0x2_340C — Access: Read/Write

	0–27	28–29	30–31
R/W	—	pri_lvl1	pri_lvl0
Reset	All zeros		

Figure 20-32. Priority Control (PRI_CTRL)

Table 20-36. PRI_CTRL Register Field Descriptions

Bits	Name	Description
0–27	—	Reserved, should be cleared
28–29	pri_lvl1	Priority level for priority state 1.
30–31	pri_lvl0	Priority level for priority state 0.

20.3.2.27 System Interface Control Register (SI_CTRL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The system interface control register (SI_CTRL) controls various functions pertaining to the internal system interface.

Offset 0x2_3410 — Access: Read/Write

	0–26	27	28–30	31
R/W	—	err_ disable	—	rd_prefetch _val
Reset	All zeros			

Figure 20-33. System Interface Control Register (SI_CTRL)

Table 20-37. SI_CTRL Register Field Descriptions

Bits	Name	Description
0–26	—	Reserved, should be cleared
27	err_disable	When this bit is set, it causes the controller to ignore system bus errors. If cleared the controller responds according to the values set in USBSTS[SEI] and USBINT[SEE]. 0 enable 1 disable

Table 20-37. SI_CTRL Register Field Descriptions (continued)

Bits	Name	Description
28–30	—	Reserved, should be cleared
31	rd_prefetch_val	Selects whether 32 bytes or 64 bytes are fetched during burst read transactions at the system interface. When this input is LOW 64 bytes are fetched and when it is HIGH 32 bytes are fetched. The setting of rd_prefetch_val must match the setting of the larger of TXPBURST and RXPBURST fields in the BURSTSIZE register. If either of these fields is 64 bytes, then rd_prefetch_val must be left cleared. Otherwise, this value should be set. 0 64-byte fetch 1 32-byte fetch

20.3.2.28 USB General Purpose Register (CONTROL)—Non-EHCI

Note that this register uses big-endian byte ordering and is not defined in the EHCI specification. The USB general purpose (CONTROL) register contains the general-purpose IP control register outputs and is shown in Figure 20-34.

Offset 0x2_3500 Access: Read/Write

	0–14	15
R	—	—
W		
Reset	All zeros	

	16–28	29	30	31
R	—	USB_EN	—	ULPI_INT_EN
W				
Reset	All zeros			

Figure 20-34. USB General-Purpose Register (CONTROL)

Table 20-38. CONTROL Field Descriptions

Bits	Name	Description
0–14	—	Reserved
15	—	Reserved
16–28	—	Reserved
29	USB_EN	UTMI mode: This bit is used to enable the USB interface. It must be set before setting RS bit in USB CMD register. 1 Enable 0 Disable ULPI mode: In safe mode, all USB interface signals are put into input mode or driven inactive, except for SUSPEND_STP which is driven high. Also, the input signal DIR is forced to appear high to the controller. This prevents any start-up problems that otherwise could occur if the PHY and the controller take significantlly different times to complete power-on reset. 1 Normal operation 0 Safe mode

Table 20-38. CONTROL Field Descriptions (continued)

Bits	Name	Description
30	—	Reserved
31	ULPI_INT_EN	Used to enable the ULPI low power wakeup interrupt from the PHY when the PHY is in low power mode only. 0 ULPI low power wakeup interrupt disabled 1 ULPI low power wakeup interrupt enabled **Note:** PORTSC[PHCD] bit must be set

20.4 Functional Description

The USB DR module can be broken down into functional sub-blocks, which are described below.

20.4.1 System Interface

The system interface block contains all the control and status registers that allow a processor to interface to the USB DR module. These registers allow the processor to control the configuration of the module, ascertain the capabilities of the module, and control the module's operation. It also has registers to control snoopability and priority of the DMA interface.

20.4.2 DMA Engine

The module contains a local DMA engine. The DMA engine interfaces internally to the CSB. It is responsible for moving all of the data to be transferred over the USB between the module and buffers in system memory. Like the system interface block, the DMA engine block uses a simple synchronous bus signaling protocol that eases connections to a number of different standard buses.

The DMA controller must access both control information and packet data from system memory. The control information is contained in link list–based queue structures. The DMA controller has state machines that are able to parse data structures defined in the EHCI specification. In host mode, the data structures are EHCI compliant and represent queues of transfers to be performed by the host controller, including the split-transaction requests that allow an EHCI controller to direct packets to FS and LS devices. In device mode, the data structures are designed to be similar to those in the EHCI specification and are used to allow device responses to be queued for each of the active pipes in the device.

20.4.3 FIFO RAM Controller

The FIFO RAM controller is used for context information and to control FIFOs between the protocol engine and the DMA controller. These FIFOs decouple the system processor/memory bus requests from the extremely tight timing required by USB.

The use of the FIFO buffers differs between host and device mode operation. In host mode, a single data channel is maintained in each direction through the buffer memory. In device mode, multiple FIFO channels are maintained for each of the active endpoints in the system.

In host mode, the USB DR module uses a 512-byte Tx buffer and a 512-byte Rx buffer. Device operation uses a single 512-byte Rx buffer and a 512-byte Tx buffer for each endpoint. The 512-byte buffers allow the module to buffer a complete HS bulk packet.

20.4.4 PHY Interface

The USB DR module interfaces to any ULPI-compatible PHY, as well as the FS/LS serial transceivers. The primary function of the port controller block is to isolate the rest of the module from the transceiver, and to move all of the transceiver signaling into the primary clock domain of the module. This allows the module to run synchronously with the system processor and its associated resources.

Due to pincount limitations the module only supports certain combinations of PHY interfaces and USB functionality. Refer to Table 20-39 for more information.

Table 20-39. Supported PHY Interfaces

PHY	Function
ULPI, Serial	Host/Device/OTG

20.5 Host Data Structures

This section defines the interface data structures used to communicate control, status, and data between HCD (software) and the Enhanced Host Controller (hardware). The data structure definitions in this section support a 32-bit memory buffer address space. The interface consists of a periodic schedule, periodic frame list, asynchronous schedule, isochronous transaction descriptors, split-transaction isochronous transfer descriptors, queue heads, and queue element transfer descriptors.

The periodic frame list is the root of all periodic (isochronous and interrupt transfer type) support for the host controller interface. The asynchronous list is the root for all the bulk and control transfer type support. Isochronous data streams are managed using isochronous transaction descriptors. Isochronous split-transaction data streams are managed with split-transaction isochronous transfer descriptors. All interrupt, control, and bulk data streams are managed with queue heads and queue element transfer descriptors. These data structures are optimized to reduce the total memory footprint of the schedule and to reduce (on average) the number of memory accesses needed to execute a USB transaction.

Note that software must ensure that no interface data structure reachable by the EHCI host controller spans a 4-K page boundary.

The data structures defined in this section are (from the host controller's perspective) a mix of read-only and read/writable fields. The host controller must preserve the read-only fields on all data structure writes.

20.5.1 Periodic Frame List

Figure 20-35 shows the organization of the periodic schedule. This schedule is for all periodic transfers (isochronous and interrupt). The periodic schedule is referenced from the operational registers space using the PERIODICLISTBASE address register and the FRINDEX register. The periodic schedule is based on an array of pointers called the periodic frame list. The PERIODICLISTBASE address register is combined with the FRINDEX register to produce a memory pointer into the frame list. The periodic frame list implements a sliding window of work over time.

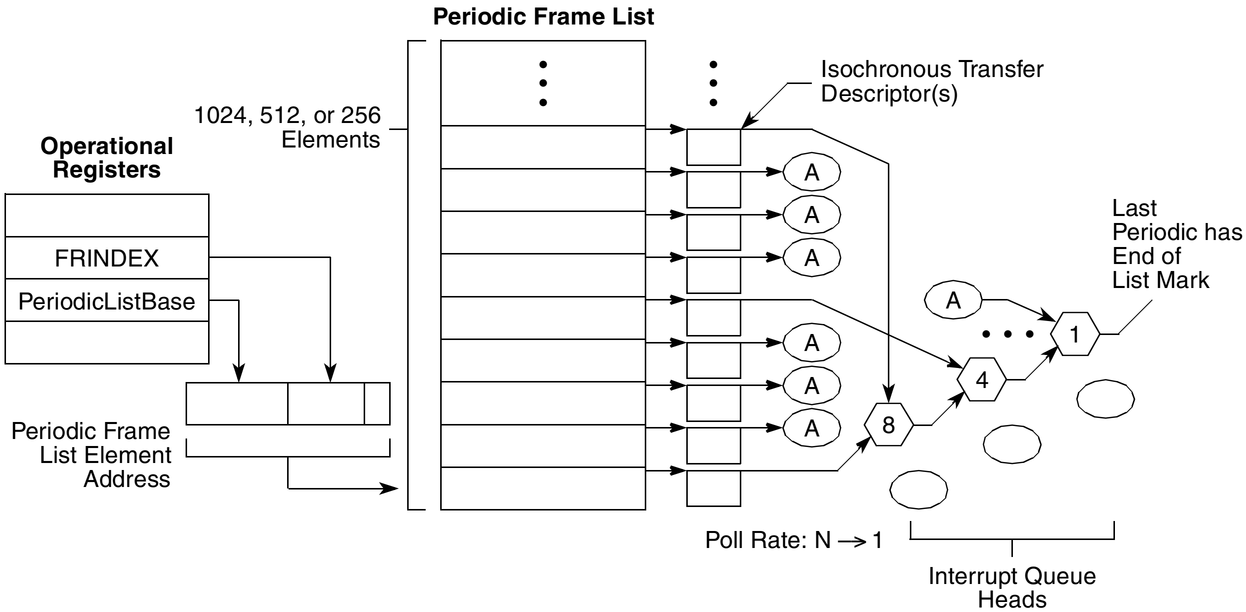


Figure 20-35. Periodic Schedule Organization

Split transaction interrupt, bulk and control are also managed using queue heads and queue element transfer descriptors.

The periodic frame list is a 4-K page aligned array of frame list link pointers. The length of the frame list may be programmable. The programmability of the periodic frame list is exported to system software through the HCCPARAMS register. If nonprogrammable, the length is 1024 elements. If programmable, the length can be selected by system software as one of 256, 512, or 1024 elements. An implementation must support all three sizes. Programming the size (that is, the number of elements) is accomplished by system software writing the appropriate value into frame list size field in the USBCMD register.

Frame list link pointers direct the host controller to the first work item in the frame's periodic schedule for the current micro-frame. The link pointers are aligned on DWord boundaries within the frame list. Figure 20-36 shows the format for the frame list link pointer.

31–5	4–3	2–1	0
Frame List Link Pointer	00	Typ	T

Figure 20-36. Frame List Link Pointer Format

Frame list link pointers always reference memory objects that are 32-byte aligned. The referenced object may be an isochronous transfer descriptor for high-speed devices, a split-transaction isochronous transfer descriptor (for full-speed isochronous endpoints), or a queue head (used to support high-, full- and low-speed interrupt). System software should not place non-periodic schedule items into the periodic schedule. The least-significant bits in a frame list pointer are used to key the host controller in as to the type of object the pointer is referencing.

The least-significant bit is the T bit (bit 0). When this bit is set, the host controller never uses the value of the frame list pointer as a physical memory pointer. The Typ field indicates the exact type of data structure being referenced by this pointer. The value encodings for the Typ field are given in Table 20-40.

Table 20-40. Typ Field Encodings

Typ	Description
00	Isochronous transfer descriptor
01	Queue head
10	Split transaction isochronous transfer descriptor
11	Frame span traversal node

20.5.2 Asynchronous List Queue Head Pointer

The asynchronous transfer list (based at the ASYNCLISTADDR register) is where all the control and bulk transfers are managed. Host controllers use this list only when it reaches the end of the periodic list, the periodic list is disabled, or the periodic list is empty.

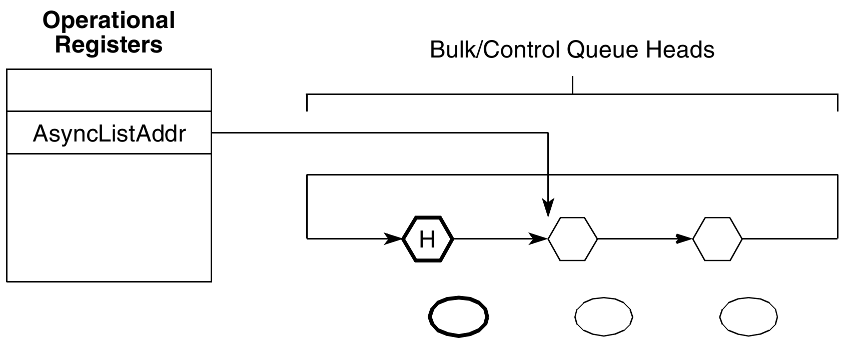


Figure 20-37. Asynchronous Schedule Organization

The asynchronous list is a simple circular list of queue heads. The ASYNCLISTADDR register is simply a pointer to the next queue head. This implements a pure round-robin service for all queue heads linked into the asynchronous list.

20.5.3 Isochronous (High-Speed) Transfer Descriptor (iTD)

Figure 20-38 illustrates the format of an isochronous transfer descriptor. This structure is used only for high-speed isochronous endpoints. All other transfer types should use queue structures. iTDs must be aligned on a 32-byte boundary.

<table>
<tr><th>31–28</th><th>27–16</th><th>15</th><th>14–12</th><th>11</th><th>10–8</th><th>7</th><th>6</th><th>5</th><th>4–3</th><th>2–1</th><th>0</th><th>offset</th></tr>
<tr><td colspan="9">Next Link Pointer</td><td>00</td><td>Typ</td><td>T</td><td>0x00</td></tr>
<tr><td>Status[1]</td><td>Transaction 0 Length[1]</td><td>ioc</td><td>PG[2]</td><td colspan="8">Transaction 0 Offset[2]</td><td>0x04</td></tr>
<tr><td>Status[1]</td><td>Transaction 1 Length[1]</td><td>ioc</td><td>PG[2]</td><td colspan="8">Transaction 1 Offset[2]</td><td>0x08</td></tr>
<tr><td>Status[1]</td><td>Transaction 2 Length[1]</td><td>ioc</td><td>PG[2]</td><td colspan="8">Transaction 2 Offset[2]</td><td>0x0C</td></tr>
<tr><td>Status[1]</td><td>Transaction 3 Length[1]</td><td>ioc</td><td>PG[2]</td><td colspan="8">Transaction 3 Offset[2]</td><td>0x10</td></tr>
<tr><td>Status[1]</td><td>Transaction 4 Length[1]</td><td>ioc</td><td>PG[2]</td><td colspan="8">Transaction 4 Offset[2]</td><td>0x14</td></tr>
<tr><td>Status[1]</td><td>Transaction 5 Length[1]</td><td>ioc</td><td>PG[2]</td><td colspan="8">Transaction 5 Offset[2]</td><td>0x18</td></tr>
<tr><td>Status[1]</td><td>Transaction 6 Length[1]</td><td>ioc</td><td>PG[2]</td><td colspan="8">Transaction 6 Offset[2]</td><td>0x1C</td></tr>
<tr><td>Status[1]</td><td>Transaction 7 Length[1]</td><td>ioc</td><td>PG[2]</td><td colspan="8">Transaction 7 Offset[2]</td><td>0x20</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 0)</td><td colspan="2">EndPt</td><td>R</td><td colspan="5">Device Address</td><td>0x24</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 1)</td><td>I/O</td><td colspan="7">Maximum Packet Size</td><td>0x28</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 2)</td><td colspan="7">Reserved</td><td>Mult</td><td>0x2C</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 3)</td><td colspan="8">Reserved</td><td>0x30</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 4)</td><td colspan="8">Reserved</td><td>0x34</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 5)</td><td colspan="8">Reserved</td><td>0x38</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 6)</td><td colspan="8">Reserved</td><td>0x3C</td></tr>
</table>

Figure 20-38. Isochronous Transaction Descriptor (iTD)

[1] Host controller read/write; all others read-only.

[2] These fields may be modified by the host controller if the I/O field indicates an OUT.

20.5.3.1 Next Link Pointer

The first DWord of an iTD is a pointer to the next schedule data structure.

Table 20-41. Next Schedule Element Pointer

Bits	Name	Description
31–5	Link Pointer	Correspond to memory address signals [31:5], respectively. This field points to another isochronous transaction descriptor (iTD/siTD) or queue head (QH).
4–3	—	Reserved, should be cleared. These bits are reserved and their value has no effect on operation. Software should initialize this field to zero.

Table 20-41. Next Schedule Element Pointer (continued)

Bits	Name	Description
2–1	Typ	Indicates to the host controller whether the item referenced is an iTD, siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. Value encodings are: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate 1 Link Pointer field is not valid. 0 Link Pointer field is valid.

20.5.3.2 iTD Transaction Status and Control List

DWords 1–8 constitute eight slots of transaction control and status. Each transaction description includes:

- Status results field
- Transaction length (bytes to send for OUT transactions and bytes received for IN transactions).
- Buffer offset. The PG and Transaction *n* Offset fields are used with the buffer pointer list to construct the starting buffer address for the transaction.

The host controller uses the information in each transaction description, plus the endpoint information contained in the first three DWords of the buffer page pointer list, to execute a transaction on the USB.

Table 20-42. iTD Transaction Status and Control

Bits	Name	Description
31–28	Status	Records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding: 31 Active. Set by software to enable the execution of an isochronous transaction by the host controller. When the transaction associated with this descriptor is completed, the host controller clears this bit indicating that a transaction for this element should not be executed when it is next encountered in the schedule. 30 Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (underrun). If an overrun condition occurs, no action is necessary. 29 Babble detected. Set by the host controller during status update when" babble" is detected during the transaction generated by this descriptor. 28 Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit may only be set for isochronous IN transactions.
27–16	Transaction *n* Length	For an OUT, this field is the number of data bytes the host controller will send during the transaction. The host controller is not required to update this field to reflect the actual number of bytes transferred during the transfer. For an IN, the initial value of the endpoint to deliver. During the status update, the host controller writes back the field is the number of bytes the host expects the number of bytes successfully received. The value in this register is the actual byte count (for example, 0 zero length data, 1 one byte, 2 two bytes, etc.). The maximum value this field may contain is 0xC00 (3072).
15	ioc	Interrupt on complete. If this bit is set, it specifies that when this transaction completes, the host controller should issue an interrupt at the next interrupt threshold.

Table 20-42. iTD Transaction Status and Control (continued)

Bits	Name	Description
14–12	PG	These bits are set by software to indicate which of the buffer page pointers the offset field in this slot should be concatenated to produce the starting memory address for this transaction. The valid range of values for this field is 0 to 6.
11–0	Transaction *n* Offset	This field is a value that is an offset, expressed in bytes, from the beginning of a buffer. This field is concatenated onto the buffer page pointer indicated in the adjacent PG field to produce the starting buffer address for this transaction.

20.5.3.3 iTD Buffer Page Pointer List (Plus)

DWords 9–15 of an isochronous transaction descriptor are nominally page pointers (4-K aligned) to the data buffer for this transfer descriptor. This data structure requires the associated data buffer to be contiguous (relative to virtual memory), but allows the physical memory pages to be non-contiguous. Seven page pointers are provided to support the expression of eight isochronous transfers. The seven pointers allow for 3 (transactions) × 1024 (maximum packet size) × 8 (transaction records) = 24576 bytes to be moved with this data structure, regardless of the alignment offset of the first page.

Since each pointer is a 4-K aligned page pointer, the least-significant 12 bits in several of the page pointers are used for other purposes.

Table 20-43. Buffer Pointer Page 0 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 0)	A 4-K aligned pointer to physical memory. Corresponds to memory address bits 31–12.
11–8	EndPt	Selects the particular endpoint number on the device serving as the data source or sink.
7	—	Reserved, should be cleared. Reserved for future use and should be initialized by software to zero.
6–0	Device Address	This field selects the specific device serving as the data source or sink.

Table 20-44. iTD Buffer Pointer Page 1 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 1)	This is a 4-K aligned pointer to physical memory. Corresponds to memory address bits 31–12.
11	I/O	Direction (I/O). This field encodes whether the high-speed transaction should use an IN or OUT PID. 0 OUT 1 IN
10–0	Maximum Packet Size	This directly corresponds to the maximum packet size of the associated endpoint (*wMaxPacketSize*). This field is used for high-bandwidth endpoints where more than one transaction is issued per transaction description (for example, per micro-frame). This field is used with the *Multi* field to support high-bandwidth pipes. This field is also used for all IN transfers to detect packet babble. Software should not set a value larger than 1024 (0x400). Any value larger yields undefined results.

Table 20-45. Buffer Pointer Page 2 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 2)	This is a 4-K aligned pointer to physical memory. Corresponds to memory address bits 31–12.
11–2	—	Reserved, should be cleared. This bit reserved for future use and should be cleared.
1–0	Mult	Indicates to the host controller the number of transactions that should be executed per transaction description (for example, per micro-frame). 00 Reserved, should be cleared. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame

Table 20-46. Buffer Pointer Page 3–6

Bits	Name	Description
31–12	Buffer Pointer	This is a 4-K aligned pointer to physical memory. Corresponds to memory address bits 31–12.
11–2	—	Reserved, should be cleared. These bits reserved for future use and should be cleared.

20.5.4 Split Transaction Isochronous Transfer Descriptor (siTD)

All full-speed isochronous transfers through the internal transaction translator are managed using the siTD data structure. This data structure satisfies the operational requirements for managing the split transaction protocol.

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 offset

								offset
Next Link Pointer	00	Typ	T					0x00
I/O	Port Number	0	Hub Address	0000	EndPt	0	Device Address	0x04
0000_0000_0000_00000	µFrame C-mask	µFrame S-mask						0x08
ioc	P[1]	0000	Total Bytes to Transfer[1]	µFrame C-prog-mask[1]	Status[1]			0x0C
Buffer Pointer (Page 0)	Current Offset[1]							0x10
Buffer Pointer (Page 1)	000_0000	TP[1]	T-count[1]					0x14
Back Pointer	0000	T						0x18

Figure 20-39. Split-Transaction Isochronous Transaction Descriptor (siTD)

[1] Host controller read/write; all others read-only.

20.5.4.1 Next Link Pointer

DWord0 of a siTD is a pointer to the next schedule data structure.

Table 20-47. Next Link Pointer

Bits	Name	Description
31–5	Next Link Pointer	This field contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4–3	—	Reserved, should be cleared. These bits must be written as zeros.
2–1	Typ	Indicates to the host controller whether the item referenced is an iTD/siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. Value encodings are: 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor 11 FSTN (frame span traversal node)
0	T	Terminate. 0 Link pointer is valid. 1 Link pointer field is not valid.

20.5.4.2 siTD Endpoint Capabilities/Characteristics

DWords 1 and 2 specify static information about the full-speed endpoint, the addressing of the parent Companion Controller, and micro-frame scheduling control.

Table 20-48. Endpoint and Transaction Translator Characteristics

Bits	Name	Description
31	I/O	Direction (I/O). This field encodes whether the full-speed transaction should be an IN or OUT. 0 OUT 1 IN
30–24	Port Number	This field is the port number of the recipient transaction translator.
23	—	Reserved, should be cleared. Bit reserved and should be cleared.
22–16	Hub Address	This field holds the device address of the companion controllers' hub.
15–12	—	Reserved, should be cleared. Field reserved and should be cleared.
11–8	EndPt	Endpoint Number. Selects the particular endpoint number on the device serving as the data source or sink.
7	—	Reserved, should be cleared. Bit is reserved for future use. It should be cleared.
6–0	Device Address	Selects the specific device serving as the data source or sink.

Table 20-49. Micro-Frame Schedule Control

Bits	Name	Description
31–16	—	Reserved, should be cleared. This field reserved for future use. It should be cleared.
15–8	μFrame C-mask	Split completion mask. This field (along with the Active and SplitX- state fields in the status byte) is used to determine during which micro-frames the host controller should execute complete-split transactions. When the criteria for using this field is met, an all-zeros value has undefined behavior. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the μFrame C-Mask field is a one, this siTD is a candidate for transaction execution. There may be more than one bit in this mask set.
7–0	μFrame S-mask	Split start mask. This field (along with the Active and SplitX-state fields in the Status byte) is used to determine during which micro-frames the host controller should execute start-split transactions. The host controller uses the value of the three low-order bits of the FRINDEX register to index into this bit field. If the FRINDEX register value indexes to a position where the μFrame S-mask field is a one, then this siTD is a candidate for transaction execution. An all zeros value in this field, in combination with existing periodic frame list has undefined results.

20.5.4.3 siTD Transfer State

DWords 3–6 manage the state of the transfer.

Table 20-50. siTD Transfer Status and Control

Bits	Name	Description
31	ioc	Interrupt on complete 0 Do not interrupt when transaction is complete. 1 Do interrupt when transaction is complete. When the host controller determines that the split transaction has completed it will assert a hardware interrupt at the next interrupt threshold.
30	P	Page select. Indicates which data page pointer should be concatenated with the CurrentOffset field to construct a data buffer pointer 0 Selects Page 0 pointer 1 Selects Page 1 pointer The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero).
29–26	—	Reserved, should be cleared. This field reserved for future use and should be cleared.
25–16	Total Bytes to Transfer	This field is initialized by software to the total number of bytes expected in this transfer. Maximum value is 1023 (3FFh)
15–8	μFrame C-prog-mask	Split complete progress mask. This field is used by the host controller to record which split-completes have been executed.

Table 20-50. siTD Transfer Status and Control (continued)

<table>
<tr><th>Bits</th><th>Name</th><th colspan="2">Description</th></tr>
<tr><td rowspan="10">7–0</td><td rowspan="10">Status</td><td colspan="2">This field records the status of the transaction executed by the host controller for this slot. This field is a bit vector with the following encoding:</td></tr>
<tr><th>Status Bits</th><th>Definition</th></tr>
<tr><td>7</td><td>Active. Set by software to enable the execution of an isochronous split transaction by the host controller.</td></tr>
<tr><td>6</td><td>ERR. Set by the host controller when an ERR response is received from the companion controller.</td></tr>
<tr><td>5</td><td>Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). In the case of an under run, the host controller will transmit an incorrect CRC (thus invalidating the data at the endpoint). If an overrun condition occurs, no action is necessary.</td></tr>
<tr><td>4</td><td>Babble detected. Set by the host controller during status update when” babble” is detected during the transaction generated by this descriptor.</td></tr>
<tr><td>3</td><td>Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (Time-out, CRC, Bad PID, etc.). This bit will only be set for IN transactions.</td></tr>
<tr><td>2</td><td>Missed micro-frame. The host controller detected that a host-induced hold- off caused the host controller to miss a required complete-split transaction.</td></tr>
<tr><td>1</td><td>Split transaction state (SplitXstate). The bit encodings are:
0 Do start split. This value directs the host controller to issue a Start split transaction to the endpoint when a match is encountered in the S-mask.
1 Do complete split. This value directs the host controller to issue a Complete split transaction to the endpoint when a match is encountered in the C-mask.</td></tr>
<tr><td>0</td><td>Reserved, should be cleared. Bit reserved for future use and should be cleared.</td></tr>
</table>

20.5.4.4 siTD Buffer Pointer List (Plus)

DWords 4 and 5 are the data buffer page pointers for the transfer. This structure supports one physical page cross. The most-significant 20 bits of each DWord in this section are the 4-K (page) aligned buffer pointers. The least-significant 12 bits of each DWord are used as additional transfer state.

Table 20-51. siTD Buffer Pointer Page 0 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 0)	Bits 31–12 are 4K page-aligned, physical memory addresses. These bits correspond to physical address bits 31–12 respectively. The field P specifies the current active pointer
11–0	Current Offset	The 12 least-significant bits of the Page 0 pointer is the current byte offset for the current page pointer (as selected with the page indicator bit (P field)). The host controller is not required to write this field back when the siTD is retired (Active bit transitioned from a one to a zero).

Table 20-52. siTD Buffer Pointer Page 1 (Plus)

Bits	Name	Description
31–12	Buffer Pointer (Page 1)	Bits 31–12 are 4K page-aligned, physical memory addresses. These bits correspond to physical address bits 31–12 respectively. The field P specifies the current active pointer
11–5	—	Reserved, should be cleared.
4–3	TP	Transaction position. This field is used with T-count to determine whether to send all, first, middle, or last with each outbound transaction payload. System software must initialize this field with the appropriate starting value. The host controller must correctly manage this state during the lifetime of the transfer. The bit encodings are: 00 All. The entire full-speed transaction data payload is in this transaction (that is, less than or equal to 188 bytes). 01 Begin. This is the first data payload for a full-speed transaction that is greater than 188 bytes. 10 Mid. This is the middle payload for a full-speed OUT transaction that is larger than 188 bytes. 11 End. This is the last payload for a full-speed OUT transaction that was larger than 188 bytes.
2–0	T-Count	Transaction count. Software initializes this field with the number of OUT start-splits this transfer requires. Any value larger than 6 is undefined.

20.5.4.5 siTD Back Link Pointer

DWord 6 of a siTD is simply another schedule link pointer. This pointer is always zero, or references a siTD. This pointer cannot reference any other schedule data structure.

Table 20-53. siTD Back Link Pointer

Bits	Name	Description
31–5	Back Pointer	A physical memory pointer to an siTD
4–1	—	Reserved, should be cleared. This field is reserved for future use. It should be cleared.
0	T	Terminate 0 siTD Back Pointer field is valid 1 siTD Back Pointer field is not valid

20.5.5 Queue Element Transfer Descriptor (qTD)

This data structure is only used with a queue head. This data structure is used for one or more USB transactions. This data structure is used to transfer up to 20480 (5 × 4096) bytes. The structure contains two structure pointers used for queue advancement, a DWord of transfer state, and a five-element array of data buffer pointers. This structure is 32 bytes (or one 32-byte cache line). This data structure must be physically contiguous.

The buffer associated with this transfer must be virtually contiguous. The buffer may start on any byte boundary. A separate buffer pointer list element must be used for each physical page in the buffer, regardless of whether the buffer is physically contiguous.

Host controller updates (host controller writes) to stand-alone qTDs only occur during transfer retirement. References in the following bit field definitions of updates to the qTD are to the qTD portion of a queue head.

<table>
<tr><th>31</th><th>30–16</th><th>15</th><th>14–12</th><th>11–10</th><th>9–8</th><th>7–5</th><th>4–1</th><th>0</th><th>offset</th></tr>
<tr><td colspan="7">Next qTD Pointer</td><td>0000</td><td>T</td><td>0x00</td></tr>
<tr><td colspan="7">Alternate Next qTD Pointer</td><td>0000</td><td>T</td><td>0x04</td></tr>
<tr><td>dt[1]</td><td>Total Bytes to Transfer[1]</td><td>ioc</td><td>C_Page[1]</td><td>Cerr[1]</td><td>PID Code</td><td colspan="3">Status[1]</td><td>0x08</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 0)</td><td colspan="5">Current Offset[1]</td><td>0x0C</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 1)</td><td colspan="5">0000_0000_0000</td><td>0x10</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 2)</td><td colspan="5">0000_0000_0000</td><td>0x14</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 3)</td><td colspan="5">0000_0000_0000</td><td>0x18</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 4)</td><td colspan="5">0000_0000_0000</td><td>0x1C</td></tr>
</table>

Figure 20-40. Queue Element Transfer Descriptor (qTD)

[1] Host controller read/write; all others read-only.

Queue element transfer descriptors must be aligned on 32-byte boundaries.

20.5.5.1 Next qTD Pointer

The first DWord of an element transfer descriptor is a pointer to another transfer element descriptor.

Table 20-54. qTD Next Element Transfer Pointer (DWord 0)

Bits	Name	Description
31–5	Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed and corresponds to memory address signals [31:5], respectively.
4–1	—	Reserved, should be cleared. These bits are reserved and their value has no effect on operation.
0	T	Terminate. Indicates to the host controller that there are no more valid entries in the queue. 0 Pointer is valid (points to a valid transfer element descriptor) 1 Pointer is invalid

20.5.5.2 Alternate Next qTD Pointer

The second DWord of a queue element transfer descriptor is used to support hardware-only advance of the data stream to the next client buffer on short packet. To be more explicit the host controller will always use this pointer when the current qTD is retired due to short packet.

Table 20-55. qTD Alternate Next Element Transfer Pointer (DWord 1)

Bits	Name	Description
31–5	Alternate Next qTD Pointer	This field contains the physical memory address of the next qTD to be processed in the event that the current qTD execution encounters a short packet (for an IN transaction). The field corresponds to memory address signals [31:5], respectively.
4–1	—	Reserved, should be cleared. These bits are reserved and their value has no effect on operation.
0	T	Terminate. Indicates to the host controller that there are no more valid entries in the queue. 0 Pointer is valid (points to a valid transfer element descriptor) 1 Pointer is invalid

20.5.5.3 qTD Token

The third DWord of a queue element transfer descriptor contains most of the information the host controller requires to execute a USB transaction (the remaining endpoint-addressing information is specified in the queue head). Note that some of the field descriptions in Table 20-56 reference fields are defined in the queue head. See Section 20.5.6, “Queue Head,” for more information on these fields.

Table 20-56. qTD Token (DWord 2)

Bits	Name	Description
31	dt	Data toggle. This is the data toggle sequence bit. The use of this bit depends on the setting of the Data Toggle Control bit in the queue head.
30–16	Total Bytes to Transfer	Total bytes to transfer. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction, only on the successful completion of the transaction. The maximum value software may store in this field is 5 × 4K (0x5000). This is the maximum number of bytes 5 page pointers can access. If the value of this field is zero when the host controller fetches this transfer descriptor (and the active bit is set), the host controller executes a zero-length transaction and retires the transfer descriptor. It is not a requirement for OUT transfers that total bytes to transfer be an even multiple of QH[Maximum Packet Length]. If software builds such a transfer descriptor for an OUT transfer, the last transaction will always be less than QH[Maximum Packet Length]. Although it is possible to create a transfer up to 20K this assumes the page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K. Therefore, the maximum recommended transfer is 16K (0x4000).
15	ioc	Interrupt on complete. If this bit is set, the host controller should issue an interrupt at the next interrupt threshold when this qTD is completed.
14–12	C_Page	Current rage. This field is used as an index into the qTD buffer pointer list. Valid values are in the range 0x0 to 0x4. The host controller is not required to write this field back when the qTD is retired.

Table 20-56. qTD Token (DWord 2) (continued)

<table>
<tr><th>Bits</th><th>Name</th><th colspan="2">Description</th></tr>
<tr><td rowspan="7">11–10</td><td rowspan="7">Cerr</td><td colspan="2">Error counter. 2-bit down counter that keeps track of the number of consecutive errors detected while executing this qTD. If this field is programmed with a non-zero value during setup, the host controller decrements the count and writes it back to the qTD if the transaction fails. If the counter counts from one to zero, the host controller marks the qTD inactive, sets the Halted bit to a one, and error status bit for the error that caused Cerr to decrement to zero. An interrupt will be generated if USBINTR[UEE] is set. If the host controller driver (HCD) software programs this field to zero during setup, the host controller will not count errors for this qTD and there will be no limit on the retries of this qTD. Note that write-backs of intermediate execution state are to the queue head overlay area, not the qTD.</td></tr>
<tr><th>Error</th><th>Decrement Counter</th></tr>
<tr><td>Transaction Error</td><td>Yes</td></tr>
<tr><td>Data Buffer Error</td><td>No. Data buffer errors are host problems. They don't count against the device's retries. Note that software must not program Cerr to a value of zero when the EPS field is programmed with a value indicating a full- or low-speed device. This combination could result in undefined behavior.</td></tr>
<tr><td>Stalled</td><td>No. Detection of babble or stall automatically halts the queue head. Thus, count is not decremented</td></tr>
<tr><td>Babble Detected</td><td>No. Detection of babble or stall automatically halts the queue head. Thus, count is not decremented</td></tr>
<tr><td>No Error</td><td>No. If the EPS field indicates a HS device or the queue head is in the asynchronous schedule (and PIDCode indicates an IN or OUT) and a bus transaction completes and the host controller does not detect a transaction error, then the host controller should reset Cerr to extend the total number of errors for this transaction. For example, Cerr should be reset with maximum value (0b11) on each successful completion of a transaction. The host controller must never reset this field if the value at the start of the transaction is 0b00.</td></tr>
<tr><td>9–8</td><td>PID Code</td><td colspan="2">This field is an encoding of the token, which should be used for transactions associated with this transfer descriptor. Encodings are:
00 OUT Token generates token (E1H)
01 IN Token generates token (69H)
10 SETUP Token generates token (2DH) (undefined if endpoint is an Interrupt transfer type, for example. µFrame S-mask field in the queue head is non-zero.)
11 Reserved, should be cleared</td></tr>
</table>

Table 20-56. qTD Token (DWord 2) (continued)

Bits	Name	Description	
7–0	Status	This field is used by the host controller to communicate individual command execution states back to the host controller driver (HCD) software. This field contains the status of the last transaction performed on this qTD. The bit encodings are:	
		Bits	**Status Field Description**
		7	Active. Set by software to enable the execution of transactions by the host controller.
		6	Halted. Set by the host controller during status updates to indicate that a serious error has occurred at the device/endpoint addressed by this qTD. This can be caused by babble, the error counter counting down to zero, or reception of the STALL handshake from the device during a transaction. Any time that a transaction results in the Halted bit being set, the Active bit is also cleared.
		5	Data buffer error. Set by the host controller during status update to indicate that the host controller is unable to keep up with the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run). If an overrun condition occurs, the host controller will force a time-out condition on the USB, invalidating the transaction at the source. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
		4	Babble detected. Set by the host controller during status update when babble is detected during the transaction. In addition to setting this bit, the host controller also sets the Halted bit to a one. Since babble is considered a fatal error for the transfer, setting the Halted bit to a one insures that no more transactions occur because of this descriptor.
		3	Transaction error (XactErr). Set by the host controller during status update in the case where the host did not receive a valid response from the device (time-out, CRC, bad PID). If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.
		2	Missed micro-frame. This bit is ignored unless the QH[EPS] field indicates a full- or low-speed endpoint and the queue head is in the periodic list. This bit is set when the host controller detected that a host-induced hold-off caused the host controller to miss a required complete-split transaction. If the host controller sets this bit to a one, then it remains a one for the duration of the transfer.

Table 20-56. qTD Token (DWord 2) (continued)

Bits	Name	Description	
		1	Split transaction state (SplitXstate). This bit is ignored by the host controller unless the QH[EPS] field indicates a full- or low-speed endpoint. When a full- or low-speed device, the host controller uses this bit to track the state of the split- transaction. The functional requirements of the host controller for managing this state bit and the split transaction protocol depends on whether the endpoint is in the periodic or asynchronous schedule. The bit encodings are: 0 Do start split. This value directs the host controller to issue a start split transaction to the endpoint. 1 Do complete split. This value directs the host controller to issue a Complete split transaction to the endpoint.
		0	Ping state (P)/ERR. If the QH[EPS] field indicates a high-speed device and the PID Code indicates an OUT endpoint, then this is the state bit for the Ping protocol. The bit encodings are: 0 Do OUT. This value directs the host controller to issue an OUT PID to the endpoint. 1 Do Ping. This value directs the host controller to issue a PING PID to the endpoint. If the QH[EPS] field does not indicate a high-speed device, then this field is used as an error indicator bit. It is set by the host controller whenever a periodic split-transaction receives an ERR handshake.

20.5.5.4 qTD Buffer Page Pointer List

The last five DWords of a queue element transfer descriptor make up an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

System software initializes the Current Offset field to the starting offset into the current page, where current page is selected with the value in the C_Page field.

Table 20-57. qTD Buffer Pointer

Bits	Name	Description
31–12	Buffer Pointer (page *n*)	Each element in the list is a 4K page aligned physical memory address. The lower 12 bits in each pointer are reserved (except for the first one), as each memory pointer must reference the start of a 4K page. The field C_Page specifies the current active pointer. When the transfer element descriptor is fetched, the starting buffer address is selected using C_Page (similar to an array index to select an array element). If a transaction spans a 4K buffer boundary, the host controller must detect the page-span boundary in the data stream, increment C_Page and advance to the next buffer pointer in the list, and conclude the transaction via the new buffer pointer.
11–0	Current Offset (Page 0)/ — (Pages 1–4)	Reserved in all pointers except the first one (that is, Page 0). The host controller should ignore all reserved bits. For the page 0 current offset interpretation, this field is the byte offset into the current page (as selected by C_Page). The host controller is not required to write this field back when the qTD is retired. Software should ensure the reserved fields are initialized to zeros.

20.5.6 Queue Head

Figure 20-41 shows the queue head structure.

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 offset

Queue Head Horizontal Link Pointer	00	Typ	T						0x00
RL	C	Maximum Packet Length	H	dtc	EPS	EndPt	I	Device Address	0x04[1]
Mult	Port Number	Hub Addr	µFrame C-mask	µFrame S-mask					0x08[1]
Current qTD Pointer[2]	00000								0x0C
Next qTD Pointer[2]	0000	T[2]							0x10[3]
Alternate Next qTD Pointer[2]	NakCnt[2]	T[2]							0x14[3,4]
dt[1]	Total Bytes to Transfer[2]	ioc[2]	C_Page[2]	Cerr[2]	PID Code[2]	Status[2]			0x18[3,4]
Buffer Pointer (Page 0)[2]	Current Offset[2]								0x1C[3,4]
Buffer Pointer (Page 1)[2]	0000	C-prog-mask[2]							0x20[3,4]
Buffer Pointer (Page 2)[2]	S-bytes[2]	FrameTag[2]							0x24[3,4]
Buffer Pointer (Page 3)[2]	0000_0000_0000								0x28[3]
Buffer Pointer (Page 4)[2]	0000_0000_0000								0x2C[3]

Figure 20-41. Queue Head Layout

1 Offsets 0x04 through 0x0B contain the static endpoint state.
2 Host controller read/write; all others read-only.
3 Offsets 0x10 through 0x2F contain the transfer overlay.
4 Offsets 0x14 through 0x27 contain the transfer results.

20.5.6.1 Queue Head Horizontal Link Pointer

The first DWord of a queue head contains a link pointer to the next data object to be processed after any required processing in this queue has been completed, as well as the control bits defined below.

This pointer may reference a queue head or one of the isochronous transfer descriptors. It must not reference a queue element transfer descriptor.

Table 20-58. Queue Head DWord 0

Bits	Name	Description
31–5	QHLP	Queue head horizontal link pointer. This field contains the address of the next data object to be processed in the horizontal list and corresponds to memory address signals [31:5], respectively.
4–3	—	Reserved, should be cleared. These bits must be written as zeros.

Table 20-58. Queue Head DWord 0 (continued)

Bits	Name	Description
2–1	Typ	Indicates to the hardware whether the item referenced by the link pointer is an iTD, siTD or a QH. This allows the host controller to perform the proper type of processing on the item after it is fetched. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 1 Last QH (pointer is invalid). 0 Pointer is valid. If the queue head is in the context of the periodic list, a one bit in this field indicates to the host controller that this is the end of the periodic list. This bit is ignored by the host controller when the queue head is in the asynchronous schedule. Software must ensure that queue heads reachable by the host controller always have valid horizontal link pointers.

20.5.6.2 Endpoint Capabilities/Characteristics

The second and third DWords of a Queue Head specify static information about the endpoint. This information does not change over the lifetime of the endpoint. There are three types of information in this region:

- Endpoint characteristics. These are the USB endpoint characteristics, which include addressing, maximum packet size, and endpoint speed.
- Endpoint capabilities. These are adjustable parameters of the endpoint. They affect how the endpoint data stream is managed by the host controller.
- Split transaction characteristics. This data structure manages full- and low-speed data streams for bulk, control, and interrupt with split transactions to USB 2.0 Hub transaction translator. Additional fields exist for addressing the hub and scheduling the protocol transactions (for periodic).

The host controller must not modify the bits in this region.

Table 20-59. Endpoint Characteristics: Queue Head DWord 1

Bits	Name	Description
31–28	RL	Nak count reload. This field contains a value, which is used by the host controller to reload the Nak Counter field.
27	C	Control endpoint flag. If the QH[EPS] field indicates the endpoint is not a high-speed device, and the endpoint is a control endpoint, then software must set this bit to a one. Otherwise, it should always set this bit to a zero.
26–16	Maximum Packet Length	This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
15	H	Head of reclamation list flag. This bit is set by system software to mark a queue head as being the head of the reclamation list.

Table 20-59. Endpoint Characteristics: Queue Head DWord 1 (continued)

Bits	Name	Description
14	dtc	Data toggle control (DTC). Specifies where the host controller should get the initial data toggle on an overlay transition. 0 Ignore DT bit from incoming qTD. Host controller preserves DT bit in the queue head. 1 Initial data toggle comes from incoming qTD DT bit. Host controller replaces DT bit in the queue head from the DT bit in the qTD.
13–12	EPS	Endpoint speed. This is the speed of the associated endpoint. 00 Full-speed (12 Mbps) 01 Low-speed (1.5 Mbps) 10 High-speed (480 Mbps) 11 Reserved, should be cleared This field must not be modified by the host controller.
11–8	EndPt	Endpoint number. Selects the particular endpoint number on the device serving as the data source or sink.
7	I	Inactivate on next transaction. This bit is used by system software to request that the host controller set the Active bit to zero. This field is only valid when the queue head is in the periodic schedule and the EPS field indicates a full- or low-speed endpoint. Setting this bit when the queue head is in the asynchronous schedule or the EPS field indicates a high-speed device yields undefined results.
6–0	Device Address	Selects the specific device serving as the data source or sink.

Table 20-60. Endpoint Capabilities: Queue Head DWord 2

Bits	Name	Description
31–30	Mult	High-bandwidth pipe multiplier. This field is a multiplier used to key the host controller as the number of successive packets the host controller may submit to the endpoint in the current execution. The host controller makes the simplifying assumption that software properly initializes this field (regardless of location of queue head in the schedules or other run time parameters). 00 Reserved, should be cleared. A zero in this field yields undefined results. 01 One transaction to be issued for this endpoint per micro-frame 10 Two transactions to be issued for this endpoint per micro-frame 11 Three transactions to be issued for this endpoint per micro-frame
29–23	Port Number	This field is ignored by the host controller unless the EPS field indicates a full- or low-speed device. The value is the port number identifier on the USB 2.0 hub (for hub at device address Hub Addr below), below which the full- or low-speed device associated with this endpoint is attached. This information is used in the split-transaction protocol.
22–16	Hub Addr	This field is ignored by the host controller unless the EPS field indicates a full-or low-speed device. The value is the USB device address of the USB 2.0 hub below which the full- or low-speed device associated with this endpoint is attached. This field is used in the split-transaction protocol.

Table 20-60. Endpoint Capabilities: Queue Head DWord 2 (continued)

Bits	Name	Description
15–8	μFrame C-mask	This field is ignored by the host controller unless the EPS field indicates this device is a low- or full-speed device and this queue head is in the periodic list. This field (along with the Active and SplitX-state fields) is used to determine during which micro-frames the host controller should execute a complete-split transaction. When the criteria for using this field are met, a zero value in this field has undefined behavior. This field is used by the host controller to match against the three low-order bits of the FRINDEX register. If the FRINDEX register bits decode to a position where the μFrame C- mask field is a one, then this queue head is a candidate for transaction execution. There may be more than one bit in this mask set.
7–0	μFrame S-mask	Interrupt schedule mask. This field is used for all endpoint speeds. Software should set this field to a zero when the queue head is on the asynchronous schedule. A non-zero value in this field indicates an interrupt endpoint. The host controller uses the value of the three low-order bits of the FRINDEX register as an index into a bit position in this bit vector. If the μFrame S-mask field has a one at the indexed bit position then this queue head is a candidate for transaction execution. If the EPS field indicates the endpoint is a high-speed endpoint, then the transaction executed is determined by the PID_Code field contained in the execution area. This field is also used to support split transaction types: Interrupt (IN/OUT). This condition is true when this field is non-zero and the EPS field indicates this is either a full- or low-speed device. A zero value in this field, in combination with existing in the periodic frame list has undefined results.

20.5.6.3 Transfer Overlay

The nine DWords in this area represent a transaction working space for the host controller. The general operational model is that the host controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it follows the queue head horizontal link pointer to the next queue head. The host controller will never follow the next transfer queue element or alternate queue element pointers unless it is actively attempting to advance the queue. For the duration of the transfer, the host controller keeps the incremental status of the transfer in the overlay area. When the transfer is complete, the results are written back to the original queue element.

The DWord3 of a queue head contains a pointer to the source qTD currently associated with the overlay. The host controller uses this pointer to write back the overlay area into the source qTD after the transfer is complete.

Table 20-61. Current qTD Link Pointer

Bits	Name	Description
31–5	Current qTD Pointer	Current element transaction descriptor link pointer. This field contains the address Of the current transaction being processed in this queue and corresponds to memory address signals [31:5], respectively.
4–0	—	Reserved, should be cleared. These bits are ignored by the host controller when using the value as an address to write data. The actual value may vary depending on the usage.

The DWords 4–11 of a queue head are the transaction overlay area. This area has the same base structure as a queue element transfer descriptor. The queue head utilizes the reserved fields of the page pointers to implement tracking the state of split transactions.

This area is characterized as an overlay because when the queue is advanced to the next queue element, the source queue element is merged onto this area. This area serves an execution cache for the transfer.

Table 20-62. Host-Controller Rules for Bits in Overlay (DWords 5, 6, 8, and 9)

DWord	QH Offset	Bits	Name	Description
5	0x14	4–1	NakCnt	Nak counter—RW. This field is a counter the host controller decrements whenever a transaction for the endpoint associated with this queue head results in a Nak or Nyet response. This counter is reloaded from RL before a transaction is executed during the first pass of the reclamation list (relative to an Asynchronous List Restart condition). It is also loaded from RL during an overlay.
6	0x18	31	dt	Data toggle. The Data toggle control controls whether the host controller preserves this bit when an overlay operation is performed.
6	0x18	15	ioc	Interrupt on complete. The ioc control bit is always inherited from the source qTD when the overlay operation is performed.
6	0x18	11–10	Cerr	Error counter. Copied from the qTD during the overlay and written back during queue advancement.
6	0x18	0	Status[0]	Ping state (P)/ERR. If the EPS field indicates a high-speed endpoint, then this field should be preserved during the overlay operation.
8	0x20	7–0	C-prog-mask	Split-transaction complete-split progress. Initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.
9	0x24	11–5	S-bytes	Software must ensure that the S-bytes field in a qTD is zero before activating the qTD. Keeps track of the number of bytes sent or received during an IN or OUT split transaction.
9	0x24	4–0	FrameTag	Split-transaction frame tag. Initialized to zero during any overlay. This field is used to track the progress of an interrupt split-transaction.

20.5.7 Periodic Frame Span Traversal Node (FSTN)

The periodic frame span traversal node (FSTN) data structure is to be used only for managing full- and low-speed transactions that span a host-frame boundary. Software must not use an FSTN in the asynchronous schedule. An FSTN in the asynchronous schedule results in undefined behavior. Software must not use the FSTN feature with a host controller whose HCIVERSION register indicates a revision implementation under 0x0096. Note that FSTNs were not defined for EHCI implementations before Revision 0.96 of the EHCI Specification and their use may yield undefined results.

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5	4 3	2 1	0	offset
Normal Path Link Pointer	00	Typ	T	0x00
Back Path Link Pointer	00	Typ	T	0x04

Figure 20-42. Frame Span Traversal Node Structure

20.5.7.1 FTSN Normal Path Pointer

The first DWord of an FSTN contains a link pointer to the next schedule object. This object can be of any valid periodic schedule data type.

Table 20-63. FTSN Normal Path Pointer

Bits	Name	Description
31–5	NPLP	Normal path link pointer. Contains the address of the next data object to be processed in the periodic list and corresponds to memory address signals [31:5], respectively.
4–3	—	Reserved, should be cleared. These bits must be written as 0s.
2–1	Typ	Indicates to the host controller whether the item referenced is a iTD/siTD, QH, or FSTN. This allows the host controller to perform the proper type of processing on the item after it is fetched. 00 iTD (isochronous transfer descriptor) 01 QH (queue head) 10 siTD (split transaction isochronous transfer descriptor) 11 FSTN (frame span traversal node)
0	T	Terminate. 0 Link pointer is valid. 1 Link pointer field is not valid.

20.5.7.2 FSTN Back Path Link Pointer

The second DWord of an FTSN node contains a link pointer to a queue head. If the T-bit in this pointer is a zero, then this FSTN is a Save-Place indicator. Its Typ field must be set by software to indicate the target data structure is a queue head. If the T-bit in this pointer is set, then this FSTN is the Restore indicator. When the T-bit is a one, the host controller ignores the Typ field.

Table 20-64. FSTN Back Path Link Pointer

Bits	Name	Description
31–5	BPLP	Back path link pointer. Contains the address of a queue head. This field corresponds to memory address signals [31:5], respectively.
4–3	—	Reserved, should be cleared. These bits must be written as 0s.
2–1	Typ	Software must ensure this field is set to indicate the target data structure is a Queue Head (01). Any other value in this field yields undefined results.
0	T	Terminate. 0 Link pointer is valid (that is, the host controller may use bits 31–5 (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address). This value also indicates that this FSTN is a Save-Place indicator. 1 Link pointer field is not valid (that is, the host controller must not use bits 31–5 (in combination with the CTRLDSSEGMENT register if applicable) as a valid memory address). This value also indicates that this FSTN is a Restore indicator.

20.6 Host Operations

The general operational model for the USB DR module in host mode is defined by the Enhanced Host Controller Interface (EHCI) Specification. The EHCI specification describes the register-level interface for a host controller for the USB Revision 2.0. It includes a description of the hardware/software interface between system software and host controller hardware. Information concerning the initialization of the USB module is included in the following section; however, the full details of the EHCI specification are beyond the scope of this document.

20.6.1 Host Controller Initialization

After initial power-on or host controller reset (hardware or through USBCMD[RST]), all of the operational registers will be at their default values. After a hardware reset, only the operational registers will be at their default values.

In order to initialize the USB DR module, software should perform the following steps

1. Set the controller mode to host mode. Optionally set USBMODE[SDIS] (streaming disable)

NOTE

Transitioning from device mode to host mode requires a host controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.
3. Program the PTS field of the PORTSC register if using a non-ULPI PHY.
4. Set CONTROL[USB_EN].
5. Write the appropriate value to the USBINTR register to enable the appropriate interrupts.
6. Write the base address of the periodic frame list to the PERIODICLIST BASE register. If there are no work items in the periodic schedule, all elements of the periodic frame list should have their T-Bits set.
7. Write the USBCMD register to set the desired interrupt threshold, frame list size (if applicable) and turn the controller by setting the RS bit.

At this point, the USB DR module is up and running and the port registers begin reporting device connects. System software can enumerate a port through the reset process (where the port is in the enabled state). At this point, the port is active with SOFs occurring down the enabled port enabled high-speed ports, but the schedules have not yet been enabled. The EHCI host controller will not transmit SOFs to enabled Full- or Low-speed ports.

In order to communicate with devices via the asynchronous schedule, system software must write the ASYNDLISTADDR register with the address of a control or bulk queue head. Software must then enable the asynchronous schedule by writing a one to USBCMD[ASE]. In order to communicate with devices via the periodic schedule, system software must enable the periodic schedule by writing a one to USBCMD[PSE]. Note that the schedules can be turned on before the first port is reset (and enabled).

Any time the USBCMD register is written, system software must ensure the appropriate bits are preserved, depending on the intended operation.

20.6.2 Power Port

The HCSPARAMS[PPC] bit indicates whether the USB 2.0 host controller has port power control. When the PPC bit is set, the host controller supports port power switches. Each available switch has an output enable. PPE is controlled based on the state of the combination bits PPC bit, EHCI Configured (CF)-bit and individual Port Power (PP) bits.

20.6.3 Reporting Over-Current

Host ports by definition are power providers on USB. Whether the ports are considered high- or low-powered is a platform implementation issue. The EHCI PORTSC register has an over-current status and over-current change bit. The functionality of these bits is specified in the USB Specification Revision 2.0.

The over current detection and limiting logic resides outside the DR logic. The over-current condition effects the following bits in the PORTSC register on the EHCI port:

- Over-current active bit (OCA) is set. When the over-current condition goes away, the OCA will transition from a one to a zero.
- Over-current change bit (OCC) is set. On every transition of OCA, the controller will set OCC to a one. Software sets OCC to a zero by writing a one to this bit.
- Port enabled/disabled bit (PE) is cleared. When this change bit gets set, USBSTS[PCI] (the port change detect bit) is set.
- Port power (PP) bit may optionally be cleared. There is no requirement in USB that a power provider shut off power in an over current condition. It is sufficient to limit the current and leave power applied. When OCC transitions from a zero to a one, the controller also sets USBSTS[PCI] to a one. In addition, if the Port Change Interrupt Enable bit, USBINTR[PCE], is a one, the controller issues an interrupt to the system. Refer to Table 20-65 for summary of behavior for over-current detection when the controller is halted (suspended from a device component point of view).

20.6.4 Suspend/Resume

The host controller provides an equivalent suspend and resume model as that defined for individual ports in a USB 2.0 hub. Control mechanisms are provided to allow system software to suspend and resume individual ports. The mechanisms allow the individual ports to be resumed completely through software initiation. Other control mechanisms are provided to parameterize the host controller's response (or sensitivity) to external resume events. In this discussion, host-initiated, or software-initiated resumes are called Resume Events/Actions; bus-initiated resume events are called wake-up events. The classes of wakeup events are:

- Remote-wakeup enabled device asserts resume signaling. In similar kind to USB 2.0 hubs, when in host mode the host controller responds to explicit device resume signaling and wake up the system (if necessary).
- Port connect and disconnect and over-current events. Sensitivity to these events can be turned on or off by using the port control bits in the PORTSC register.

Selective suspend is a feature supported by the PORTSC register. It is used to place specific ports into a suspend mode. This feature is used as a functional component for implementing the appropriate power management policy implemented in a particular operating system. When system software intends to suspend the bus, it should suspend the enabled port, then shut off the controller by setting the USBCMD[RS] to a zero.

When a wake event occurs the system will resume operation and system software must set the RS bit to a one and resume the suspended port.

20.6.4.1 Port Suspend/Resume

System software places the USB into suspend mode by writing a one into the appropriate PORTSC Suspend bit. Software must only set the Suspend bit when the port is in the enabled state (Port Enabled bit is a one).

The host controller may evaluate the Suspend bit immediately or wait until a micro-frame or frame boundary occurs. If evaluated immediately, the port is not suspended until the current transaction (if one is executing) completes. Therefore, there may be several micro-frames of activity on the port until the host controller evaluates the Suspend bit. The host controller must evaluate the Suspend bit at least every frame boundary.

System software can initiate a resume on the suspended port by writing a one to PORTSC[FPR]. Software should not attempt to resume a port unless the port reports that it is in the suspended state. If system software sets PORTSC[FPR] when the port is not in the suspended state, the resulting behavior is undefined. In order to assure proper USB device operation, software must wait for at least 10 milliseconds after a port indicates that it is suspended (Suspend bit is a one) before initiating a port resume through PORTSC[FPR]. When PORTSC[FPR] is set, the host controller sends resume signaling down the port. System software times the duration of the resume (nominally 20 milliseconds) then clears PORTSC[FPR]. When the host controller receives the write to transition PORTSC[FPR] to zero, it completes the resume sequence as defined in the USB specification, and clears both PORTSC[FPR] and PORTSC[SUSP]. Software-initiated port resumes do not affect the port change detect bit (USBSTS[PCI]) nor do they cause an interrupt if USBINTR[PCE] (port change interrupt enable) is a one. When a wake event occurs on a suspended port, the resume signaling is detected by the port and the resume is reflected downstream within 100 µsec. The port's PORTSC[FPR] bit is set and USBSTS[PCI] is set. If USBINTR[PCE] is a one, the host controller issues a hardware interrupt.

System software observes the resume event on the port, delays a port resume time (nominally 20 milliseconds), then terminates the resume sequence by clearing PORTSC[FPR] in the port. The host controller receives the write of zero to PORTSC[FPR], terminates the resume sequence and clears PORTSC[FPR] and PORTSC[SUSP]. Software can determine that the port is enabled (not suspended) by sampling the PORTSC register and observing that the SUSP and FPR bits are zero. Software must ensure that the host controller is running (that is, USBSTS[HCH] is a zero), before terminating a resume by clearing the port's PORTSC[FPR] bit. If HCH is a one when PORTSC[FPR] is cleared, then SOFs will not occur down the enabled port and the device will return to suspend mode in a maximum of 10 milliseconds.

Table 20-65 summarizes the wake-up events. Whenever a resume event is detected, USBSTS[PCI] is set. If USBINTR[PCE] (port change interrupt enable) is a one, the host controller also generates an interrupt on the resume event. Software acknowledges the resume event interrupt by clearing the USBSTS[PCI].

Table 20-65. Behavior During Wake-Up Events

Port Status and Signaling Type	Signaled Port Response	Device State	
		D0	not D0
Port disabled, resume K-State received	No effect	N/A	N/A
Port suspended, Resume K-State received	Resume reflected downstream on signaled port. PORTSC[FPR] is set. USBSTS[PCI] is set.	[1], [2]	[2]
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit, PORTSC[WKDS], is set. A disconnect is detected.	Depending on the initial port state, the PORTSC Connect (CCS) and Enable (PE) status bits are cleared, and the Connect Change status bit (CSC) is set. USBSTS[PCI] is set.	[1], [2]	[2]
Port is enabled, disabled or suspended, and the port's WKDSCNNT_E bit, PORTSC[WKDS], is cleared. A disconnect is detected.	Depending on the initial port state, the PORTSC Connect (CCS) and Enable (PE) status bits are cleared, and the Connect Change status bit (CSC) is set. USBSTS[PCI] is set.	[1], [3]	[3]
Port is not connected and the port's WKCNNT_E bit is a one. A connect is detected.	PORTSC Connect Status (CCS) and Connect Status Change (CSC) bits are set. USBSTS[PCI] is set.	[1], [2]	[2]
Port is not connected and the port's WKCNNT_E bit is a zero. A connect is detected.	PORTSC Connect Status (CCS) and Connect Status Change (CSC) bits are set. USBSTS[PCI] is set.	[1], [3]	[3]
Port is connected and the port's WKOC_E bit is a one. An over-current condition occurs.	PORTSC Over-current Active (OCA), Over-current Change (OCC) bits are set. If Port Enable/Disable bit (PE) is a one, it is cleared. USBSTS[PCI] is set	[1], [2]	[2]
Port is connected and the port's WKOC_E bit is a zero. An over-current condition occurs.	PORTSC Over-current Active (OCA), Over-current Change (OCC) bits are set. If Port Enable/Disable bit (PE) is a one, it is cleared. USBSTS[PCI] is set.	[1], [3]	[3]

[1] Hardware interrupt issued if USBINTR[PCE] (port change interrupt enable) is set.

[2] PME# asserted if enabled (Note: PME Status must always be set).

[3] PME# not asserted.

20.6.5 Schedule Traversal Rules

The host controller executes transactions for devices using a simple, shared-memory schedule. The schedule is comprised of a few data structures, organized into two distinct lists. The data structures are designed to provide the maximum flexibility required by USB, minimize memory traffic and hardware/software complexity.

System software maintains two schedules for the host controller: a periodic schedule and an asynchronous schedule. The root of the periodic schedule is the PERIODICLISTBASE register. See Section 20.3.2.6, "Periodic Frame List Base Address Register (PERIODICLISTBASE)," for more information. The PERIODICLISTBASE register is the physical memory base address of the periodic frame list. The periodic frame list is an array of physical memory pointers. The objects referenced from the frame list must be valid schedule data structures as defined in Section 20.5, "Host Data Structures." In each micro-frame, if the periodic schedule is enabled (see) then the host controller must execute from the periodic schedule before executing from the asynchronous schedule. It will only execute from the asynchronous schedule after it encounters the end of the periodic schedule. The host controller traverses the periodic schedule by constructing an array offset reference from the PERIODICLISTBASE and the FRINDEX registers (see Figure 20-43). It fetches the element and begins traversing the graph of linked schedule data structures.

The end of the periodic schedule is identified by a next link pointer of a schedule data structure having its T-bit set. When the host controller encounters a T-Bit set during a horizontal traversal of the periodic list, it interprets this as an End-Of-Periodic-List mark. This causes the host controller to cease working on the periodic schedule and transitions immediately to traversing the asynchronous schedule. Once this transition is made, the host controller executes from the asynchronous schedule until the end of the micro-frame.

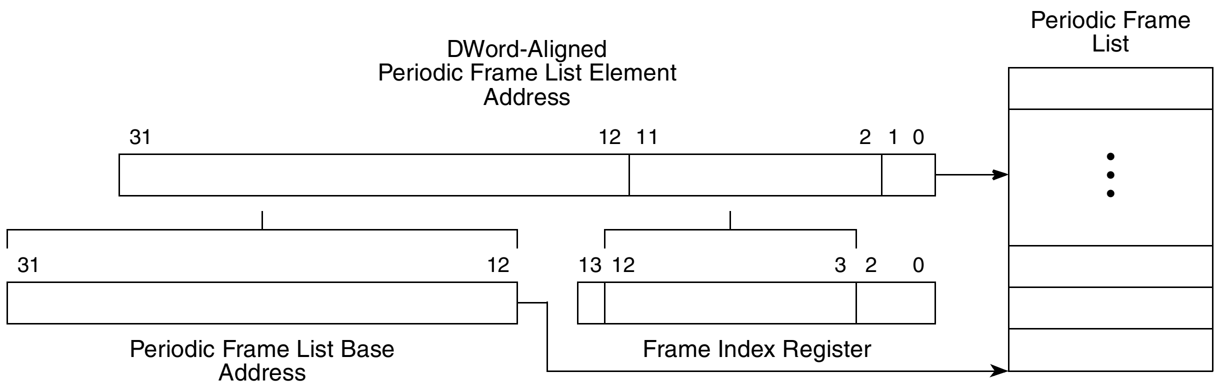


Figure 20-43. Derivation of Pointer into Frame List Array

When the host controller determines that it is time to execute from the asynchronous list, it uses the operational register ASYNCLISTADDR to access the asynchronous schedule, as shown in Figure 20-44.

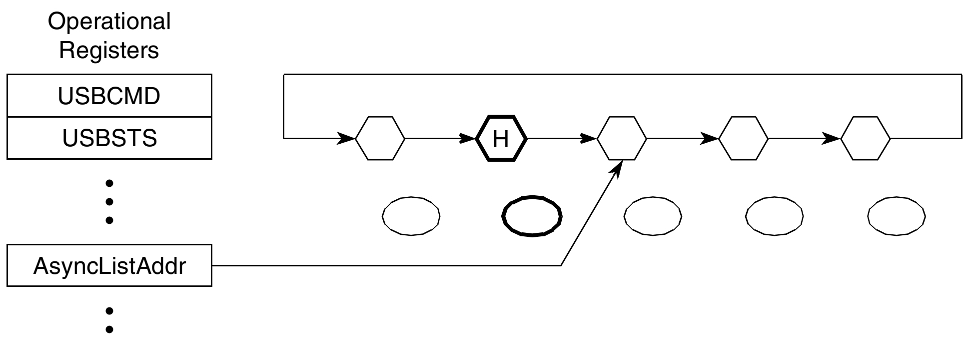


Figure 20-44. General Format of Asynchronous Schedule List

The ASYNCLISTADDR register contains a physical memory pointer to the next queue head. When the host controller makes a transition to executing the asynchronous schedule, it begins by reading the queue head referenced by the ASYNCLISTADDR register. Software must set queue head horizontal pointer T-bits to a zero for queue heads in the asynchronous schedule.

20.6.6 Periodic Schedule Frame Boundaries vs. Bus Frame Boundaries

The USB Specification Revision 2.0 requires that the frame boundaries (SOF frame number changes) of the high-speed bus and the full- and low-speed bus(es) below USB 2.0 hubs be strictly aligned. Super-imposed on this requirement is that USB 2.0 hubs manage full- and low-speed transactions via a micro-frame pipeline (see start- (SS) and complete- (CS) splits illustrated in Figure 20-45). A simple, direct projection of the frame boundary model into the host controller interface schedule architecture creates tension (complexity for both hardware and software) between the frame boundaries and the scheduling mechanisms required to service the full- and low-speed transaction translator periodic pipelines.

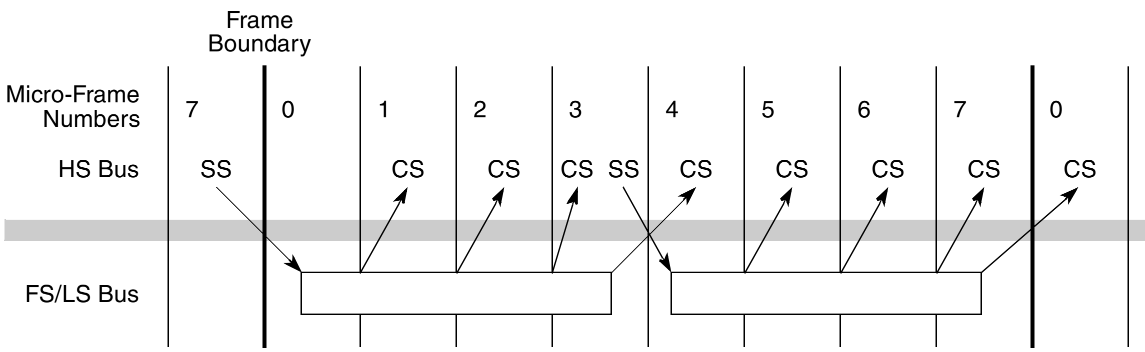


Figure 20-45. Frame Boundary Relationship Between HS Bus and FS/LS Bus

The simple projection, as Figure 20-45 illustrates, introduces frame-boundary wrap conditions for scheduling on both the beginning and end of a frame. In order to reduce the complexity for hardware and software, the host controller is required to implement a one micro-frame phase shift for its view of frame boundaries. The phase shift eliminates the beginning of frame and frame-wrap scheduling boundary conditions.

The implementation of this phase shift requires that the host controller use one register value for accessing the periodic frame list and another value for the frame number value included in the SOF token. These two values are separate, but tightly coupled. The periodic frame list is accessed via the Frame List Index Register (FRINDEX). Bits FRINDEX[2–0], represent the micro-frame number. The SOF value is coupled to the value of FRINDEX[13–3]. Both FRINDEX[13–3] and the SOF value are incremented based on FRINDEX[2–0]. It is required that the SOF value be delayed from the FRINDEX value by one micro-frame. The one micro-frame delay yields a host controller periodic schedule and bus frame boundary relationship as illustrated in Figure 20-46. This adjustment allows software to trivially schedule the periodic start and complete-split transactions for full-and low-speed periodic endpoints, using the natural alignment of the periodic schedule interface.

Figure 20-46 illustrates how periodic schedule data structures relate to schedule frame boundaries and bus frame boundaries. To aid the presentation, two terms are defined. The host controller's view of the

1-millisecond boundaries is called H-Frames. The high-speed bus's view of the 1-millisecond boundaries is called B-Frames.

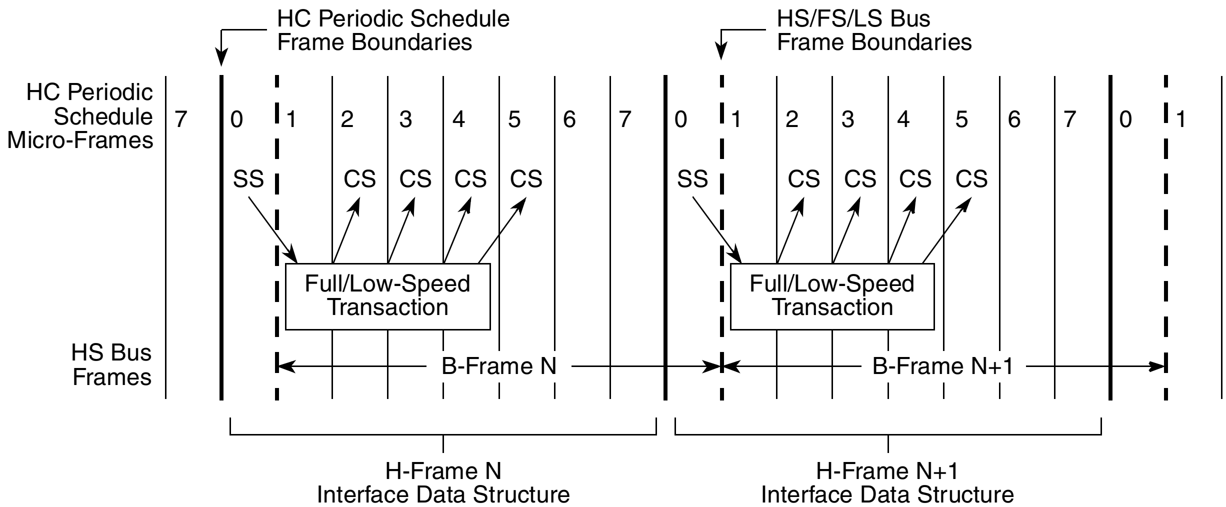


Figure 20-46. Relationship of Periodic Schedule Frame Boundaries to Bus Frame Boundaries

H-Frame boundaries for the host controller correspond to increments of FRINDEX[13–3]. Micro-frame numbers for the H-Frame are tracked by FRINDEX[2–0]. B-Frame boundaries are visible on the high-speed bus by means of changes in the SOF token's frame number. Micro-frame numbers on the high-speed bus are only derived from the SOF token's frame number (that is, the high-speed bus will see eight SOFs with the same frame number value). H-Frames and B-Frames have the fixed relationship (that is, B-Frames lag H-Frames by one micro-frame time) illustrated in Figure 20-46. The host controller's periodic schedule is naturally aligned to H-Frames. Software schedules transactions for full- and low-speed periodic endpoints relative the H-Frames. The result is these transactions execute on the high-speed bus at exactly the right time for the USB 2.0 hub periodic pipeline.

As described in Section 20.3.2.4, "Frame Index Register (FRINDEX)," the SOF Value can be implemented as a shadow register (in this example, called SOFV), which lags the FRINDEX register bits [13–3] by one micro-frame count. Table 20-66 illustrates the required relationship between the value of FRINDEX and the value of SOFV. This lag behavior can be accomplished by incrementing FRINDEX[13–3] based on carry-out on the 7 to 0 increment of FRINDEX[2–0] and incrementing SOFV based on the transition of 0 to 1 of FRINDEX[2–0].

Software is allowed to write to FRINDEX. Section 20.3.2.4, "Frame Index Register (FRINDEX)," provides the requirements that software should adhere when writing a new value in FRINDEX.

Table 20-66. Operation of FRINDEX and SOFV (SOF Value Register)

Current			Next		
FRINDEX[13–3]	SOFV	FRINDEX[2–0]	FRINDEX[13–3]	SOFV	FRINDEX[2–0]
N	N	111	N+1	N	000
N+1	N	000	N+1	N+1	001
N+1	N+1	001	N+1	N+1	010
N+1	N+1	010	N+1	N+1	011
N+1	N+1	011	N+1	N+1	100
N+1	N+1	100	N+1	N+1	101
N+1	N+1	101	N+1	N+1	110
N+1	N+1	110	N+1	N+1	111

20.6.7 Periodic Schedule

The periodic schedule traversal is enabled or disabled through USBCMD[PSE] (periodic schedule enable). If USBCMD[PSE] is cleared, then the host controller simply does not try to access the periodic frame list via the PERIODICLISTBASE register. Likewise, when USBCMD[PSE] is a one, then the host controller does use the PERIODICLISTBASE register to traverse the periodic schedule. The host controller will not react to modifications to USBCMD[PSE] immediately. In order to eliminate conflicts with split transactions, the host controller evaluates USBCMD[PSE] only when FRINDEX[2–0] is zero. System software must not disable the periodic schedule if the schedule contains an active split transaction work item that spans the 0b000 micro-frame. These work items must be removed from the schedule before USBCMD[PSE] is cleared. USBSTS[PS] (periodic schedule status) indicates status of the periodic schedule. System software enables (or disables) the periodic schedule by setting (or clearing) USBCMD[PSE]. Software then can poll USBSTS[PS] to determine when the periodic schedule has made the desired transition. Software must not modify USBCMD[PSE] unless the value of USBCMD[PSE] equals that of USBSTS[PS].

The periodic schedule is used to manage all isochronous and interrupt transfer streams. The base of the periodic schedule is the periodic frame list. Software links schedule data structures to the periodic frame list to produce a graph of scheduled data structures. The graph represents an appropriate sequence of transactions on the USB. Figure 20-47 illustrates isochronous transfers (using iTDs and siTDs) with a period of one are linked directly to the periodic frame list. Interrupt transfers (are managed with queue heads) and isochronous streams with periods other than one are linked following the period-one iTD/siTDs. Interrupt queue heads are linked into the frame list ordered by poll rate. Longer poll rates are linked first (for example, closest to the periodic frame list), followed by shorter poll rates, with queue heads with a poll rate of one, on the very end.

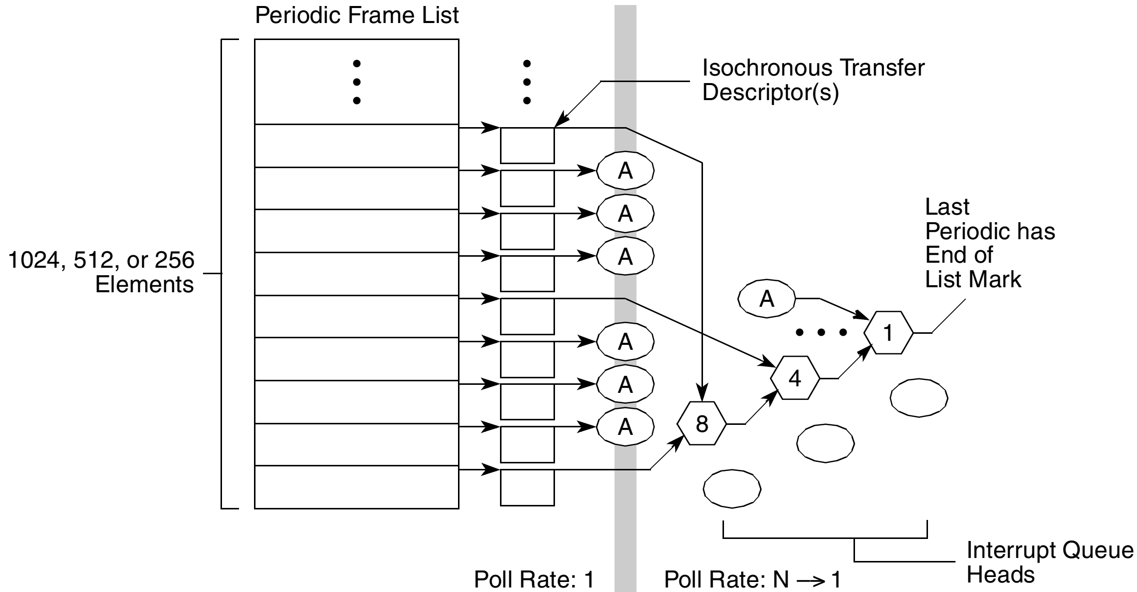


Figure 20-47. Example Periodic Schedule

20.6.8 Managing Isochronous Transfers Using iTDs

The structure of an iTD is presented in Isochronous (High-Speed) Transfer Descriptor (iTD). There are four distinct sections to an iTD:

- The first field is the Next Link Pointer. This field is for schedule linkage purposes only.
- Transaction description array. This area is an eight-element array. Each element represents control and status information for one micro-frame's worth of transactions for a single high-speed isochronous endpoint.
- The buffer page pointer array is a 7-element array of physical memory pointers to data buffers. These are 4K aligned pointers to physical memory.
- Endpoint capabilities. This area utilizes the unused low-order 12 bits of the buffer page pointer array. The fields in this area are used across all transactions executed for this iTD, including endpoint addressing, transfer direction, maximum packet size and high-bandwidth multiplier.

20.6.8.1 Host Controller Operational Model for iTDs

The host controller uses FRINDEX register bits 12–3 to index into the periodic frame list. This means that the host controller visits each frame list element eight consecutive times before incrementing to the next periodic frame list element. Each iTD contains eight transaction descriptions, which map directly to FRINDEX register bits 2–0. Each iTD can span 8 micro-frames worth of transactions. When the host controller fetches an iTD, it uses FRINDEX register bits 2–0 to index into the transaction description array. When the first iTD in the periodic list is traversed after periodic schedule is enabled, the value of FRINDEX[2:0] may be other then 0, so the first transaction issued by the controller may be any of the eight available active transactions. If the active bit in the Status field of the indexed transaction description is cleared, the host controller ignores the iTD and follows the Next pointer to the next schedule data structure.

When the indexed active bit is a one the host controller continues to parse the iTD. It stores the indexed transaction description and the general endpoint information (device address, endpoint number, maximum packet size, etc.). It also uses the Page Select (PG) field to index the buffer pointer array, storing the selected buffer pointer and the next sequential buffer pointer. For example, if PG field is a 0, then the host controller will store Page 0 and Page 1.

The host controller constructs a physical data buffer address by concatenating the current buffer pointer (as selected using the current transaction description's PG field) and the transaction description's Transaction Offset field. The host controller uses the endpoint addressing information and I/O-bit to execute a transaction to the appropriate endpoint. When the transaction is complete, the host controller clears the active bit and writes back any additional status information to the Status field in the currently selected transaction description.

The data buffer associated with the iTD must be virtually contiguous memory. Seven page pointers are provided to support eight high-bandwidth transactions regardless of the starting packet's offset alignment into the first page. A starting buffer pointer (physical memory address) is constructed by concatenating the page pointer (example: page 0 pointer) selected by the active transaction descriptions' PG (example value: 0b00) field with the transaction offset field. As the transaction moves data, the host controller must detect when an increment of the current buffer pointer will cross a page boundary. When this occurs the host controller simply replaces the current buffer pointer's page portion with the next page pointer (example: page 1 pointer) and continues to move data. The size of each bus transaction is determined by the value in the Maximum Packet Size field. An iTD supports high-bandwidth pipes by means of the Mult (multiplier) field. When the Mult field is 1, 2, or 3, the host controller executes the specified number of Maximum Packet sized bus transactions for the endpoint in the current micro-frame. In other words, the Mult field represents a transaction count for the endpoint in the current micro-frame. If the Mult field is zero, the operation of the host controller is undefined. The transfer description is used to service all transactions indicated by the Mult field.

For OUT transfers, the value of the Transaction *n* Length field represents the total bytes to be sent during the micro-frame. The Mult field must be set by software to be consistent with Transaction *n* Length and Maximum Packet Size. The host controller will send the bytes in Maximum Packet Sized portions. After each transaction, the host controller decrements it's local copy of Transaction *n* Length by Maximum Packet Size. The number of bytes the host controller sends is always Maximum Packet Size or Transaction *n* Length, whichever is less. The host controller advances the transfer state in the transfer description, updates the appropriate record in the iTD and moves to the next schedule data structure. The maximum sized transaction supported is 3×1024 bytes.

For IN transfers, the host controller issues Mult transactions. It is assumed that software has properly initialized the iTD to accommodate all of the possible data. During each IN transaction, the host controller must use Maximum Packet Size to detect packet babble errors. The host controller keeps the sum of bytes received in the Transaction *n* Length field. After all transactions for the endpoint have completed for the micro-frame, Transaction *n* Length contains the total bytes received. If the final value of Transaction *n* Length is less than the value of Maximum Packet Size, then less data than was allowed for was received from the associated endpoint. This short packet condition does not set USBSTS[UI] (USB interrupt). The host controller will not detect this condition. If the device sends more than Transaction *n* Length or

Maximum Packet Size bytes (whichever is less), then the host controller will set the Babble Detected bit and clear the Active bit. Note, that the host controller is not required to update the iTD field Transaction *n* Length in this error scenario. If the Mult field is greater than one, then the host controller will automatically execute the value of Mult transactions. The host controller will not execute all Mult transactions if:

- The endpoint is an OUT and Transaction *n* Length goes to zero before all the Mult transactions have executed (ran out of data), or
- The endpoint is an IN and the endpoint delivers a short packet, or an error occurs on a transaction before Mult transactions have been executed. The end of micro-frame may occur before all of the transaction opportunities have been executed. When this happens, the transfer state of the transfer description is advanced to reflect the progress that was made, the result written back to the iTD and the host controller proceeds to processing the next micro-frame.

20.6.8.2 Software Operational Model for iTDs

A client buffer request to an isochronous endpoint may span 1 to N micro-frames. When N is larger than one, system software may have to use multiple iTDs to read or write data with the buffer (if N is larger than eight, it must use more than one iTD).

Figure 20-48 illustrates the simple model of how a client buffer is mapped by system software to the periodic schedule (that is, the periodic frame list and a set of iTDs). On the right is the client description of its request. The description includes a buffer base address plus additional annotations to identify which portions of the buffer should be used with each bus transaction. In the middle is the iTD data structures used by the system software to service the client request. Each iTD can be initialized to service up to 24 transactions, organized into eight groups of up to three transactions each. Each group maps to one micro-frame's worth of transactions. The EHCI controller does not provide per-transaction results within a micro-frame. It treats the per-micro-frame transactions as a single logical transfer. On the left is the host controller’s frame list. System software establishes references from the appropriate locations in the frame list to each of the appropriate iTDs. If the buffer is large, then system software can use a small set of iTDs to service the entire buffer. System software can activate the transaction description records (contained in each iTD) in any pattern required for the particular data stream.

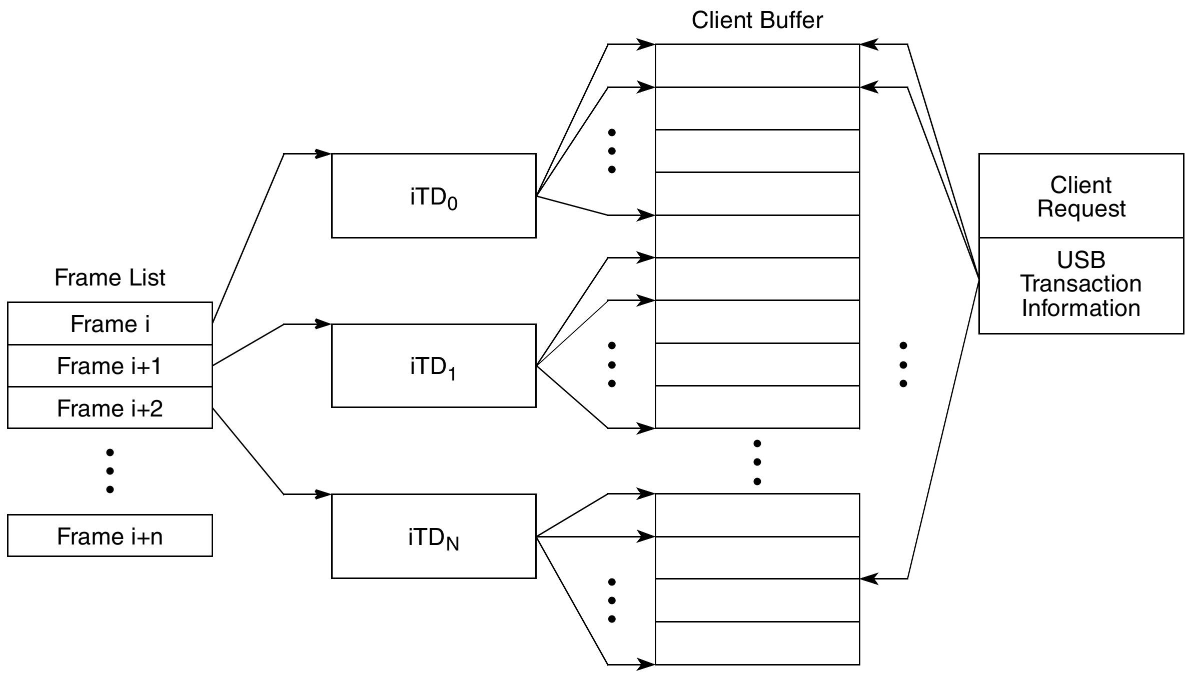


Figure 20-48. Example Association of iTDs to Client Request Buffer

As noted above, the client request includes a pointer to the base of the buffer and offsets into the buffer to annotate which buffer sections are to be used on each bus transaction that occurs on this endpoint. System software must initialize each transaction description in an iTD to ensure it uses the correct portion of the client buffer. For example, for each transaction description, the PG field is set to index the correct physical buffer page pointer and the Transaction Offset field is set relative to the correct buffer pointer page (for example, the same one referenced by the PG field). When the host controller executes a transaction it selects a transaction description record based on FRINDEX[2–0]. It then uses the current Page Buffer Pointer (as selected by the PG field) and concatenates to the transaction offset field. The result is a starting buffer address for the transaction. As the host controller moves data for the transaction, it must watch for a page wrap condition and properly advance to the next available Page Buffer Pointer. System software must not use the Page 6 buffer pointer in a transaction description where the length of the transfer will wrap a page boundary. Doing so yields undefined behavior. The host controller hardware is not required to alias the page selector to page zero. USB 2.0 isochronous endpoints can specify a period greater than one. Software can achieve the appropriate scheduling by linking iTDs into the appropriate frames (relative to the frame list) and by setting appropriate transaction description elements active bits to a one.

20.6.8.2.1 Periodic Scheduling Threshold

The Isochronous Scheduling Threshold field in the HCCPARAMS capability register is an indicator to system software as to how the host controller pre-fetches and effectively caches schedule data structures. It is used by system software when adding isochronous work items to the periodic schedule. The value of this field indicates to system software the minimum distance it can update isochronous data (relative to the current location of the host controller execution in the periodic list) and still have the host controller process them.

The iTD and siTD data structures each describe 8 micro-frames worth of transactions. The host controller is allowed to cache one (or more) of these data structures in order to reduce memory traffic. There are three basic caching models that account for the fact the isochronous data structures span 8 micro-frames. The three caching models are: no caching, micro-frame caching and frame caching.

When software is adding new isochronous transactions to the schedule, it always performs a read of the FRINDEX register to determine the current frame and micro-frame the host controller is currently executing. Of course, there is no information about where in the micro-frame the host controller is, so a constant uncertainty factor of one micro-frame has to be assumed. Combining the knowledge of where the host controller is executing with the knowledge of the caching model allows the definition of simple algorithms for how closely software can reliably work to the executing host controller.

No caching is indicated with a value of zero in the Isochronous Scheduling Threshold field. The host controller may pre-fetch data structures during a periodic schedule traversal (per micro-frame) but will always dump any accumulated schedule state at the end of the micro-frame. At the appropriate time relative to the beginning of every micro-frame, the host controller always begins schedule traversal from the frame list. Software can use the value of the FRINDEX register (plus the constant 1 uncertainty-factor) to determine the approximate position of the executing host controller. When no caching is selected, software can add an isochronous transaction as near as 2 micro-frames in front of the current executing position of the host controller.

Frame caching is indicated with a non-zero value in bit [7] of the Isochronous Scheduling Threshold field. In the frame-caching model, system software assumes that the host controller caches one (or more) isochronous data structures for an entire frame (8 micro-frames). Software uses the value of the FRINDEX register (plus the constant 1 uncertainty) to determine the current micro-frame/frame (assume modulo 8 arithmetic in adding the constant 1 to the micro-frame number). For any current frame N, if the current micro-frame is 0 to 6, then software can safely add isochronous transactions to Frame N + 1. If the current micro-frame is 7, then software can add isochronous transactions to Frame N + 2.

Micro-frame caching is indicated with a non-zero value in the least-significant 3 bits of the Isochronous Scheduling Threshold field. System software assumes the host controller caches one or more periodic data structures for the number of micro-frames indicated in the Isochronous Scheduling Threshold field. For example, if the count value were 2, then the host controller keeps a window of 2 micro-frames worth of state (current micro-frame, plus the next) on-chip. On each micro-frame boundary, the host controller releases the current micro-frame state and begins accumulating the next micro-frame state.

20.6.9 Asynchronous Schedule

The asynchronous schedule traversal is enabled or disabled through USBCMD[ASE] (asynchronous schedule enable). If USBCMD[ASE] is cleared, then the host controller simply does not try to access the asynchronous schedule via the ASYNCLISTADDR register. Likewise, if USBCMD[ASE] is set, the host controller does use the ASYNCLISTADDR register to traverse the asynchronous schedule. Modifications to USBCMD[ASE] are not necessarily immediate. Rather the new value of the bit will only be taken into consideration the next time the host controller needs to use the value of the ASYNCLISTADDR register to get the next queue head.

USBSTS[AS] indicates status of the asynchronous schedule. System software enables (or disables) the asynchronous schedule by writing a one (or zero) to USBCMD[ASE]. Software then can poll USBSTS[AS] to determine when the asynchronous schedule has made the desired transition. Software must not modify USBCMD[ASE] unless the value of USBCMD[ASE] equals that of the USBSTS[AS] (asynchronous schedule status).

The asynchronous schedule is used to manage all Control and Bulk transfers. Control and Bulk transfers are managed using queue head data structures. The asynchronous schedule is based at the ASYNCLISTADDR register. The default value of the ASYNCLISTADDR register after reset is undefined and the schedule is disabled when USBCMD[ASE] is cleared.

Software may only write this register with defined results when the schedule is disabled, for example, USBCMD[ASE] and the USBSTS[AS] are cleared. System software enables execution from the asynchronous schedule by writing a valid memory address (of a queue head) into this register. Then software enables the asynchronous schedule by setting USBCMD[ASE]. The asynchronous schedule is actually enabled when USBSTS[AS] is set.

When the host controller begins servicing the asynchronous schedule, it begins by using the value of the ASYNCLISTADDR register. It reads the first referenced data structure and begins executing transactions and traversing the linked list as appropriate. When the host controller completes processing the asynchronous schedule, it retains the value of the last accessed queue head's horizontal pointer in the ASYNCLISTADDR register. Next time the asynchronous schedule is accessed, this is the first data structure that is serviced. This provides round-robin fairness for processing the asynchronous schedule.

A host controller completes processing the asynchronous schedule when one of the following events occur:

- The end of a micro-frame occurs.
- The host controller detects an empty list condition
- The schedule has been disabled through USBCMD[ASE].

The queue heads in the asynchronous list are linked into a simple circular list as shown in Figure 20-44. Queue head data structures are the only valid data structures that may be linked into the asynchronous schedule. An isochronous transfer descriptor (iTD or siTD) in the asynchronous schedule yields undefined results.

The maximum packet size field in a queue head is sized to accommodate the use of this data structure for all non-isochronous transfer types. The USB Specification, Revision 2.0 specifies the maximum packet sizes for all transfer types and transfer speeds. System software should always parameterize the queue head data structures according to the core specification requirements.

20.6.9.1 Adding Queue Heads to Asynchronous Schedule

This is a software requirement section. There are two independent events for adding queue heads to the asynchronous schedule. The first is the initial activation of the asynchronous list. The second is inserting a new queue head into an activated asynchronous list.

Activation of the list is simple. System software writes the physical memory address of a queue head into the ASYNCLISTADDR register, then enables the list by setting USBCMD[ASE] to a one.

When inserting a queue head into an active list, software must ensure that the schedule is always coherent from the host controllers' point of view. This means that the system software must ensure that all queue head pointer fields are valid. For example qTD pointers have T-Bits set or reference valid qTDs and the Horizontal Pointer references a valid queue head data structure. The following algorithm represents the functional requirements:

```
InsertQueueHead (pQHeadCurrent, pQueueHeadNew)
    --
    -- Requirement: all inputs must be properly initialized.
    --
    -- pQHeadCurrent is a pointer to a queue head that is
    -- already in the active list
    -- pQHeadNew is a pointer to the queue head to be added
    --
    -- This algorithm links a new queue head into a existing
    -- list
    --
    pQueueHeadNew.HorizontalPointer = pQueueHeadCurrent.HorizontalPointer
    pQueueHeadCurrent.HorizontalPointer = physicalAddressOf(pQueueHeadNew)
End InsertQueueHead
```

20.6.9.2 Removing Queue Heads from Asynchronous Schedule

This is a software requirement section. There are two independent events for removing queue heads from the asynchronous schedule. The first is shutting down (deactivating) the asynchronous list. The second is extracting a single queue head from an activated list. Software deactivates the asynchronous schedule by setting USBCMD[ASE] to a zero. Software can determine when the list is idle when USBSTS[AS] is cleared. The normal mode of operation is that software removes queue heads from the asynchronous schedule without shutting it down. Software must not remove an active queue head from the schedule. Software should first deactivate all active qTDs, wait for the queue head to go inactive, then remove the queue head from the asynchronous list. Software removes a queue head from the asynchronous list using the following algorithm. Software merely must ensure all of the link pointers reachable by the host controller are kept consistent.

```
UnlinkQueueHead (pQHeadPrevious, pQueueHeadToUnlink, pQHeadNext)
    --
    -- Requirement: all inputs must be properly initialized.
    --
    -- pQHeadPrevious is a pointer to a queue head that
    -- references the queue head to remove
    -- pQHeadToUnlink is a pointer to the queue head to be
    -- removed
    -- pQheadNext is a pointer to a queue head still in the
    -- schedule. Software provides this pointer with the
    -- following strict rules:
    -- if the host software is one queue head, then
    -- pQHeadNext must be the same as
    -- QueueheadToUnlink.HorizontalPointer. If the host
    -- software is unlinking a consecutive series of
    -- queue heads, QHeadNext must be set by software to
    -- the queue head remaining in the schedule.
    --
    -- This algorithm unlinks a queue head from a circular list
    --
```

```
        pQueueHeadPrevious.HorizontalPointer = pQueueHeadToUnlink.HorizontalPointer
        pQueueHeadToUnlink.HorizontalPointer = pQHeadNext
    End UnlinkQueueHead
```

If software removes the queue head with the H-bit set, it must select another queue head still linked into the schedule and set its H-bit. This should be completed before removing the queue head. The requirement is that software keep one queue head in the asynchronous schedule, with its H-bit set. At the point software has removed one or more queue heads from the asynchronous schedule, it is unknown whether the host controller has a cached pointer to them. Similarly, it is unknown how long the host controller might retain the cached information, as it is implementation dependent and may be affected by the actual dynamics of the schedule load. Therefore, once software has removed a queue head from the asynchronous list, it must retain the coherency of the queue head (link pointers). It cannot disturb the removed queue heads until it knows that the host controller does not have a local copy of a pointer to any of the removed data structures.

The method software uses to determine when it is safe to modify a removed queue head is to handshake with the host controller. The handshake mechanism allows software to remove items from the asynchronous schedule, then execute a simple, lightweight handshake that is used by software as a key that it can free (or reuse) the memory associated the data structures it has removed from the asynchronous schedule.

The handshake is implemented with three bits in the host controller. The first bit is a command bit (USBCMD[IAA]—interrupt on async advance doorbell) that allows software to inform the host controller that something has been removed from its asynchronous schedule. The second bit is a status bit (USBSTS[AAI]—interrupt on async advance) that the host controller sets after it has released all on-chip state that may potentially reference one of the data structures just removed. When the host controller sets this status bit, it also clears the command bit. The third bit is an interrupt enable (USBINTR[AAE]—interrupt on async advance enable) that is matched with the status bit. If the status bit is set and the interrupt enable bit is set, then the host controller asserts a hardware interrupt.

Figure 20-49 illustrates a general example where consecutive queue heads (B and C) are unlinked from the schedule using the algorithm above. Before the unlink operation, the host controller has a copy of queue head A.

The unlink algorithm requires that as software unlinks each queue head, the unlinked queue head is loaded with the address of a queue head that will remain in the asynchronous schedule.

When the host controller observes that doorbell bit being set, it makes a note of the local reachable schedule information. In this example, the local reachable schedule information includes both queue heads (A & B). It is sufficient that the host controller can set the status bit (and clear the doorbell bit) as soon as it has traversed beyond current reachable schedule information (that is, traversed beyond queue head (B) in this example).

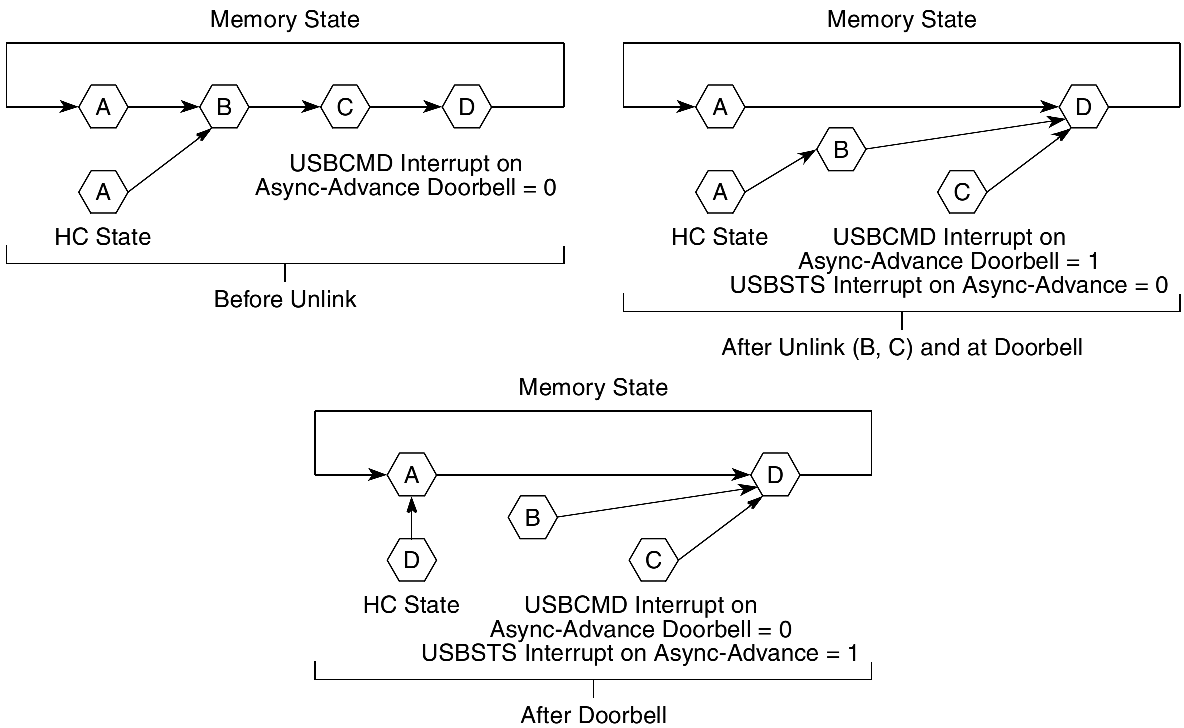


Figure 20-49. Generic Queue Head Unlink Scenario

Alternatively, a host controller implementation is allowed to traverse the entire asynchronous schedule list (for example, observed the head of the queue (twice)) before setting USBSTS[AAI].

Software may re-use the memory associated with the removed queue heads after it observes USBSTS[AAI] is set, following assertion of the doorbell. Software should acknowledge the interrupt on async advance status as indicated in the USBSTS register, before using the doorbell handshake again

20.6.9.3 Empty Asynchronous Schedule Detection

EHCI uses two bits to detect when the asynchronous schedule is empty. The queue head data structure (see Figure 20-41) defines an H-bit in the queue head, which allows software to mark a queue head as being the head of the reclaim list. host controller also keeps a 1-bit flag in the USBSTS register (Reclamation) that is cleared when the host controller observes a queue head with the H-bit set. The reclamation flag in the status register is set when any USB transaction from the asynchronous schedule is executed (or whenever the asynchronous schedule starts, see Section 20.6.9.4, “Asynchronous Schedule Traversal: Start Event.”

If the controller ever encounters an H-bit of one and a Reclamation bit of zero, the controller simply stops traversal of the asynchronous schedule.

An example illustrating the H-bit in a schedule is shown in Figure 20-50.

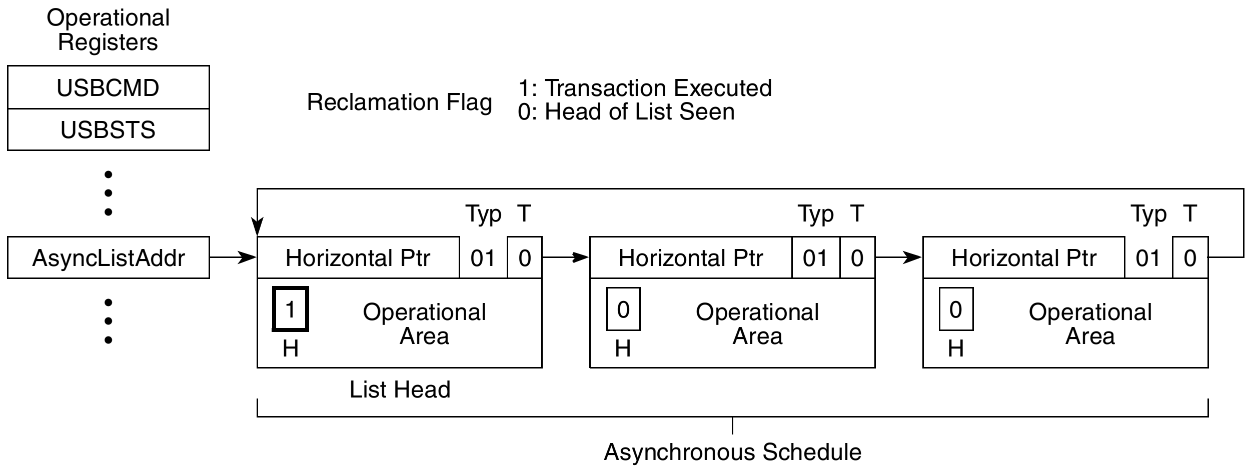


Figure 20-50. Asynchronous Schedule List with Annotation to Mark Head of List

20.6.9.4 Asynchronous Schedule Traversal: Start Event

Once the host controller has idled itself using the empty schedule detection, it naturally activates and begins processing from the Periodic Schedule at the beginning of each micro-frame. In addition, it may have idled itself early in a micro-frame. When this occurs (idles early in the micro-frame) the host controller must occasionally reactivate during the micro-frame and traverse the asynchronous schedule to determine whether any progress can be made. Asynchronous schedule Start Events are defined to be:

- Whenever the host controller transitions from the periodic schedule to the asynchronous schedule. If the periodic schedule is disabled and the asynchronous schedule is enabled, then the beginning of the micro-frame is equivalent to the transition from the periodic schedule, or
- The asynchronous schedule traversal restarts from a sleeping state.

20.6.9.5 Reclamation Status Bit (USBSTS Register)

The operation of the empty asynchronous schedule detection feature depends on the proper management of the Reclamation bit (RCL) in the USBSTS register. The host controller tests for an empty schedule just after it fetches a new queue head while traversing the asynchronous schedule. The host controller sets USBSTS[RCL] whenever an asynchronous schedule traversal Start Event occurs. USBSTS[RCL] is also set whenever the host controller executes a transaction while traversing the asynchronous schedule.The host controller clears USBSTS[RCL] whenever it finds a queue head with its H-bit set. Software should only set a queue head's H-bit if the queue head is in the asynchronous schedule. If software sets the H-bit in an interrupt queue head, the resulting behavior is undefined. The host controller may clear USBSTS[RCL] when executing from the periodic schedule.

20.6.10 Managing Control/Bulk/Interrupt Transfers via Queue Heads

This section presents an overview of how the host controller interacts with queuing data structures.

Queue heads use the Queue Element Transfer Descriptor (qTD) structure defined in Section 20.5.5, "Queue Element Transfer Descriptor (qTD)."

One queue head is used to manage the data stream for one endpoint. The queue head structure contains static endpoint characteristics and capabilities. It also contains a working area from where individual bus transactions for an endpoint are executed. Each qTD represents one or more bus transactions, which is defined in the context of the EHCI specification as a transfer.

The general processing model for the host controller's use of a queue head is simple:

- Read a queue head,
- Execute a transaction from the overlay area,
- Write back the results of the transaction to the overlay area
- Move to the next queue head.

If the host controller encounters errors during a transaction, the host controller will set one of the error reporting bits in the queue head's Status field. The Status field accumulates all errors encountered during the execution of a qTD (that is, the error bits in the queue head Status field are sticky until the transfer (qTD) has completed). This state is always written back to the source qTD when the transfer is complete. On transfer (for example, buffer or halt conditions) boundaries, the host controller must auto-advance (without software intervention) to the next qTD. Additionally, the hardware must be able to halt the queue so no additional bus transactions will occur for the endpoint and the host controller will not advance the queue.

20.6.10.1 Buffer Pointer List Use for Data Streaming with qTDs

A qTD has an array of buffer pointers, which is used to reference the data buffer for a transfer. The EHCI specification requires that the buffer associated with the transfer be virtually contiguous. This means that if the buffer spans more than one physical page, it must obey the following rules:

- The first portion of the buffer must begin at some offset in a page and extend through the end of the page.
- The remaining buffer cannot be allocated in small chunks scattered around memory. For each 4K chunk beyond the first page, each buffer portion matches to a full 4K page. The final portion, which may only be large enough to occupy a portion of a page, must start at the top of the page and be contiguous within that page.

Figure 20-51 illustrates these requirements.

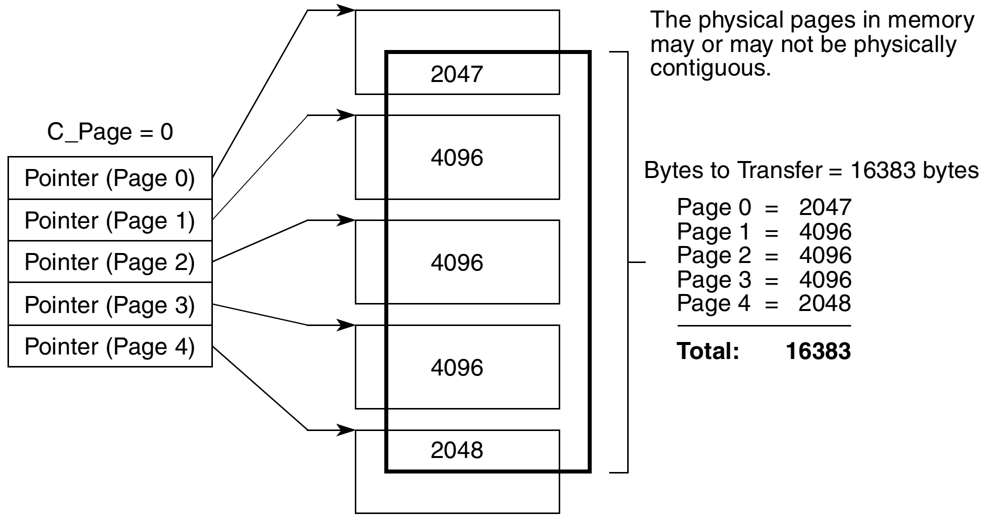


Figure 20-51. Example Mapping of qTD Buffer Pointers to Buffer Pages

The buffer pointer list in the qTD is long enough to support a maximum transfer size of 20K bytes. This case occurs when all five buffer pointers are used and the first offset is zero. A qTD handles a 16Kbyte buffer with any starting buffer alignment.

The host controller uses the C_Page field as an index value to determine which buffer pointer in the list should be used to start the current transaction. The host controller uses a different buffer pointer for each physical page of the buffer. This is always true, even if the buffer is physically contiguous.

The host controller must detect when the current transaction spans a page boundary and automatically move to the next available buffer pointer in the page pointer list. The next available pointer is reached by incrementing C_Page and pulling the next page pointer from the list. Software must ensure there are sufficient buffer pointers to move the amount of data specified in the Bytes to Transfer field.

Figure 20-51 illustrates a nominal example of how System software would initialize the buffer pointers list and the C_Page field for a transfer size of 16383 bytes. C_Page is cleared. The upper 20-bits of Page 0 references the start of the physical page. Current Offset (the lower 12-bits of queue head Dword 7) holds the offset in the page for example, 2049 (for example, 4096-2047). The remaining page pointers are set to reference the beginning of each subsequent 4K page.

For the first transaction on the qTD (assuming a 512-byte transaction), the host controller uses the first buffer pointer (page 0 because C_Page is cleared) and concatenates the Current Offset field. The 512 bytes are moved during the transaction, the Current Offset and Total Bytes to Transfer are adjusted by 512 and written back to the queue head working area.

During the 4th transaction, the host controller needs 511 bytes in page 0 and one byte in page 1. The host controller will increment C_Page (to 1) and use the page 1 pointer to move the final byte of the transaction. After the 4th transaction, the active page pointer is the page 1 pointer and Current Offset has rolled to one, and both are written back to the overlay area. The transactions continue for the rest of the buffer, with the host controller automatically moving to the next page pointer (that is, C_Page) when necessary. There are three conditions for how the host controller handles C_Page.

- The current transaction does not span a page boundary. The value of C_Page is not adjusted by the host controller.
- The current transaction does span a page boundary. The host controller must detect the page cross condition and advance to the next buffer while streaming data to/from the USB.
- The current transaction completes on a page boundary (that is, the last byte moved for the current transaction is the last byte in the page for the current page pointer). The host controller must increment C_Page before writing back status for the transaction.

Note that the only valid adjustment the host controller may make to C_Page is to increment by one.

20.6.10.2 Adding Interrupt Queue Heads to the Periodic Schedule

The link path(s) from the periodic frame list to a queue head establishes in which frames a transaction can be executed for the queue head. Queue heads are linked into the periodic schedule so they are polled at the appropriate rate. System software sets a bit in a queue head's S-Mask to indicate which micro-frame within a 1 millisecond period a transaction should be executed for the queue head. Software must ensure that all queue heads in the periodic schedule have S-Mask set to a non-zero value. An S-mask with a zero value in the context of the periodic schedule yields undefined results.

If the desired poll rate is greater than one frame, system software can use a combination of queue head linking and S-Mask values to spread interrupts of equal poll rates through the schedule so that the periodic bandwidth is allocated and managed in the most efficient manner possible. Some examples are illustrated in Table 20-67.

Table 20-67. Example Periodic Reference Patterns for Interrupt Transfers

Frame # Reference Sequence	Description
0, 2, 4, 6, 8, ... S-Mask = 0x01	A queue head for the bInterval of 2 milliseconds (16 micro-frames) is linked into the periodic schedule so that it is reachable from the periodic frame list locations indicated in the previous column. In addition, the S-Mask field in the queue head is set to 0x01, indicating that the transaction for the endpoint should be executed on the bus during micro-frame 0 of the frame.
0, 2, 4, 6, 8, ... S-Mask = 0x02	Another example of a queue head with a bInterval of 2 milliseconds is linked into the periodic frame list at exactly the same interval as the previous example. However, the S-Mask is set to 0x02 indicating that the transaction for the endpoint should be executed on the bus during micro-frame 1 of the frame.

20.6.10.3 Managing Transfer Complete Interrupts from Queue Heads

The host controller sets an interrupt to be signaled at the next interrupt threshold when the completed transfer (qTD) has an Interrupt on Complete (IOC) bit set, or whenever a transfer (qTD) completes with a short packet. If system software needs multiple qTDs to complete a client request (that is, like a control transfer) the intermediate qTDs do not require interrupts. System software may only need a single interrupt to notify it that the complete buffer has been transferred. System software may set IOC's to occur more frequently. A motivation for this may be that it wants early notification so that interface data structures can be re-used in a timely manner.

20.6.11 Ping Control

USB 2.0 defines an addition to the protocol for high-speed devices called Ping. Ping is required for all USB 2.0 High-speed bulk and control endpoints. Ping is not allowed for a split-transaction stream. This extension to the protocol eliminates the bad side-effects of Naking OUT endpoints. The Status field has a Ping State bit, which the host controller uses to determine the next actual PID it will use in the next transaction to the endpoint (see Table 20-56). The Ping State bit is only managed by the host controller for queue heads that meet all of the following criteria:

- The queue head is not an interrupt
- The EPS field equals High-Speed
- The PIDCode field equals OUT

Table 20-68 illustrates the state transition table for the host controller's responsibility for maintaining the PING protocol. Refer to Chapter 8 in the *USB Specification, Revision 2.0* for detailed description on the Ping protocol.

Table 20-68. Ping Control State Transition Table

	Event		
Current	**Host**	**Device**	**Next**
Do Ping	PING	Nak	Do Ping
Do Ping	PING	Ack	Do OUT
Do Ping	PING	XactErr[1]	Do Ping
Do Ping	PING	Stall	N/C[2]
Do OUT	OUT	Nak	Do Ping
Do OUT	OUT	Nyet	Do Ping[3]
Do OUT	OUT	Ack	Do OUT
Do OUT	OUT	XactErr[1]	Do Ping
Do OUT	OUT	Stall	N/C[2]

[1] Transaction Error (XactErr) is any time the host misses the handshake.

[2] No transition change required for the Ping State bit. The Stall handshake results in the endpoint being halted (for example, Active cleared and Halt set). Software intervention is required to restart queue.

[3] A Nyet response to an OUT means that the device has accepted the data, but cannot receive any more at this time. Host must advance the transfer state and additionally, transition the Ping State bit to Do Ping.

The Ping State bit is described in Table 20-56. The defined ping protocol allows the host to be imprecise on the initialization of the ping protocol (that is, start in Do OUT when we don't know whether there is space on the device or not). The host controller manages the Ping State bit. System software sets the initial value in the queue head when it initializes a queue head. The host controller preserves the Ping State bit across all queue advancements. This means that when a new qTD is written into the queue head overlay area, the previous value of the Ping State bit is preserved.

20.6.12 Split Transactions

USB 2.0 defines extensions to the bus protocol for managing USB 1.x data streams through USB 2.0 hubs. This section describes how the host controller uses the interface data structures to manage data streams with full- and low-speed devices, connected below a USB 2.0 hub, utilizing the split transaction protocol. Refer to the USB 2.0 Specification for the complete definition of the split transaction protocol. Full- and low-speed devices are enumerated identically as high-speed devices, but the transactions to the full- and low-speed endpoints use the split-transaction protocol on the high-speed bus. The split transaction protocol is an encapsulation of (or wrapper around) the full- or low-speed transaction. The high-speed wrapper portion of the protocol is addressed to the USB 2.0 hub and transaction translator below which the full- or low-speed device is attached.

EHCI uses dedicated data structures for managing full-speed isochronous data streams. Control, Bulk and Interrupt are managed using the queuing data structures. The interface data structures need to be programmed with the device address and the transaction translator number of the USB 2.0 hub operating as the low-/full-speed host controller for this link. The following sections describe the details of how the host controller processes and manages the split transaction protocol.

20.6.12.1 Split Transactions for Asynchronous Transfers

A queue head in the asynchronous schedule with an EPS field indicating a full-or low-speed device indicates to the host controller that it must use split transactions to stream data for this queue head. All full-speed bulk and full-, low-speed control are managed via queue heads in the asynchronous schedule.

Software must initialize the queue head with the appropriate device address and port number for the transaction translator that is serving as the full-/low-speed host controller for the links connecting the endpoint. Software must also initialize the split transaction state bit (SplitXState) to Do-Start-Split. Finally, if the endpoint is a control endpoint, then system software must set the Control Transfer Type (C) bit in the queue head to a one. If this is not a control transfer type endpoint, the C bit must be initialized by software to be a zero. This information is used by the host controller to properly set the Endpoint Type (ET) field in the split transaction bus token. When the C bit is a zero, the split transaction token's ET field is set to indicate a bulk endpoint. When the C bit is a one, the split transaction token's ET field is set to indicate a control endpoint. Refer to Chapter 8 of *USB Specification, Revision 2.0* for details.

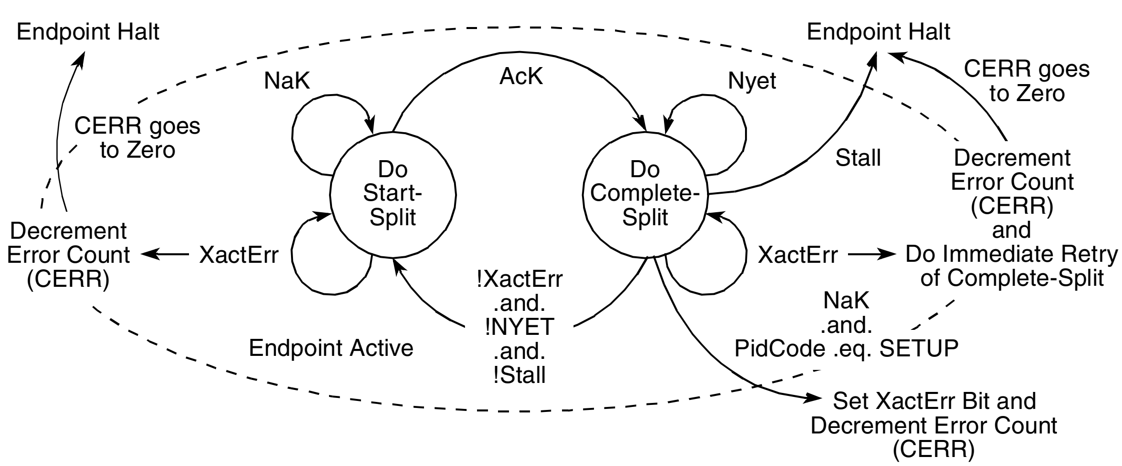


Figure 20-52. Host Controller Asynchronous Schedule Split-Transaction State Machine

20.6.12.1.1 Asynchronous—Do-Start-Split

Do-Start-Split is the state which software must initialize a full- or low-speed asynchronous queue head. This state is entered from the Do-Complete-Split state only after a complete-split transaction receives a valid response from the transaction translator that is not a Nyet handshake.

For queue heads in this state, the host controller executes a start-split transaction to the transaction translator. If the bus transaction completes without an error and PID Code indicates an IN or OUT transaction, then the host controller reloads the error counter (Cerr). If it is a successful bus transaction and the PID Code indicates a SETUP, the host controller will not reload the error counter. If the transaction translator responds with a Nak, the queue head is left in this state, and the host controller proceeds to the next queue head in the asynchronous schedule.

If the host controller times out the transaction (no response, or bad response) the host controller decrements Cerr and proceeds to the next queue head in the asynchronous schedule.

20.6.12.1.2 Asynchronous—Do-Complete-Split

This state is entered from the Do-Start-Split state only after a start-split transaction receives an Ack handshake from the transaction translator.

For queue heads in this state, the host controller executes a complete-split transaction to the transaction translator. If the transaction translator responds with a Nyet handshake, the queue head is left in this state, the error counter is reset and the host controller proceeds to the next queue head in the asynchronous schedule. When a Nyet handshake is received for a bus transaction where the queue head's PID Code indicates an IN or OUT, the host controller reloads the error counter (Cerr). When a Nyet handshake is received for a complete-split bus transaction where the queue head's PID Code indicates a SETUP, the host controller must not adjust the value of Cerr.

Independent of PID Code, the following responses have the indicated effects:

- Transaction Error (XactErr). Timeout/data CRC failure. The error counter (Cerr) is decremented by one and the complete split transaction is immediately retried (if possible). If there is not enough time in the micro-frame to execute the retry, the host controller ensures that the next time the host controller begins executing from the Asynchronous schedule, it must begin executing from this queue head. If another start-split (for some other endpoint) is sent to the transaction translator before the complete-split is really completed, the transaction translator could dump the results (which were never delivered to the host). This is why the core specification states the retries must be immediate. When the host controller returns to the asynchronous schedule in the next micro-frame, the first transaction from the schedule will be the retry for this endpoint. If Cerr went to zero, the host controller halts the queue.
- NAK. The target endpoint Nak’d the full- or low-speed transaction. The state of the transfer is not advanced and the state is exited. If the PID Code is a SETUP, then the Nak response is a protocol error. The XactErr status bit is set and the Cerr field is decremented.
- STALL. The target endpoint responded with a STALL handshake. The host controller sets the halt bit in the status byte, retires the qTD but does not attempt to advance the queue.

If the PID Code indicates an IN, then any of following responses are expected:

- DATA0/1. On reception of data, the host controller ensures the PID matches the expected data toggle and checks CRC. If the packet is good, the host controller advances the state of the transfer (for example, moves the data pointer by the number of bytes received, decrements the BytesToTransfer field by the number of bytes received, and toggles the dt bit). The host controller then exits this state. The response and advancement of transfer may trigger other processing events, such as retirement of the qTD and advancement of the queue.

  If the data sequence PID does not match the expected, the data is ignored, the transfer state is not advanced and this state is exited.

If the PID Code indicates an OUT/SETUP, then any of following responses are expected:

- ACK. The target endpoint accepted the data, so the host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The Bytes To Transfer field is decremented by the same amount and the data toggle bit (dt) is toggled. The host controller then exits this state.

  Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.

20.6.12.2 Split Transaction Interrupt

Split-transaction Interrupt-IN/OUT endpoints are managed using the same data structures used for high-speed interrupt endpoints. They both co-exist in the periodic schedule. Queue heads/qTDs offer the set of features required for reliable data delivery, which is characteristic to interrupt transfer types. The split-transaction protocol is managed completely within this defined functional transfer framework. For example, for a high-speed endpoint, the host controller will visit a queue head, execute a high-speed transaction (if criteria are met) and advance the transfer state (or not) depending on the results of the entire transaction. For low- and full-speed endpoints, the details of the execution phase are different (that is, takes more than one bus transaction to complete), but the remainder of the operational framework is intact.

20.6.12.2.1 Split Transaction Scheduling Mechanisms for Interrupt

Full- and low-speed Interrupt queue heads have an EPS field indicating full- or low-speed and have a non-zero S-mask field. The host controller can detect this combination of parameters and assume the endpoint is a periodic endpoint. Low- and full-speed interrupt queue heads require the use of the split transaction protocol. The host controller sets the Endpoint Type (ET) field in the split token to indicate the transaction is an interrupt. These transactions are managed through a transaction translator's periodic pipeline. Software should not set these fields to indicate the queue head is an interrupt unless the queue head is used in the periodic schedule.

System software manages the per/transaction translator periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each endpoint will occur. The characteristics of the transaction translator are such that the high-speed transaction protocol must execute during explicit micro-frames, or the data or response information in the pipeline is lost. Figure 20-53 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule and queue head data structure. The S and C_n labels indicate micro-frames where software can schedule start-splits and complete splits (respectively).

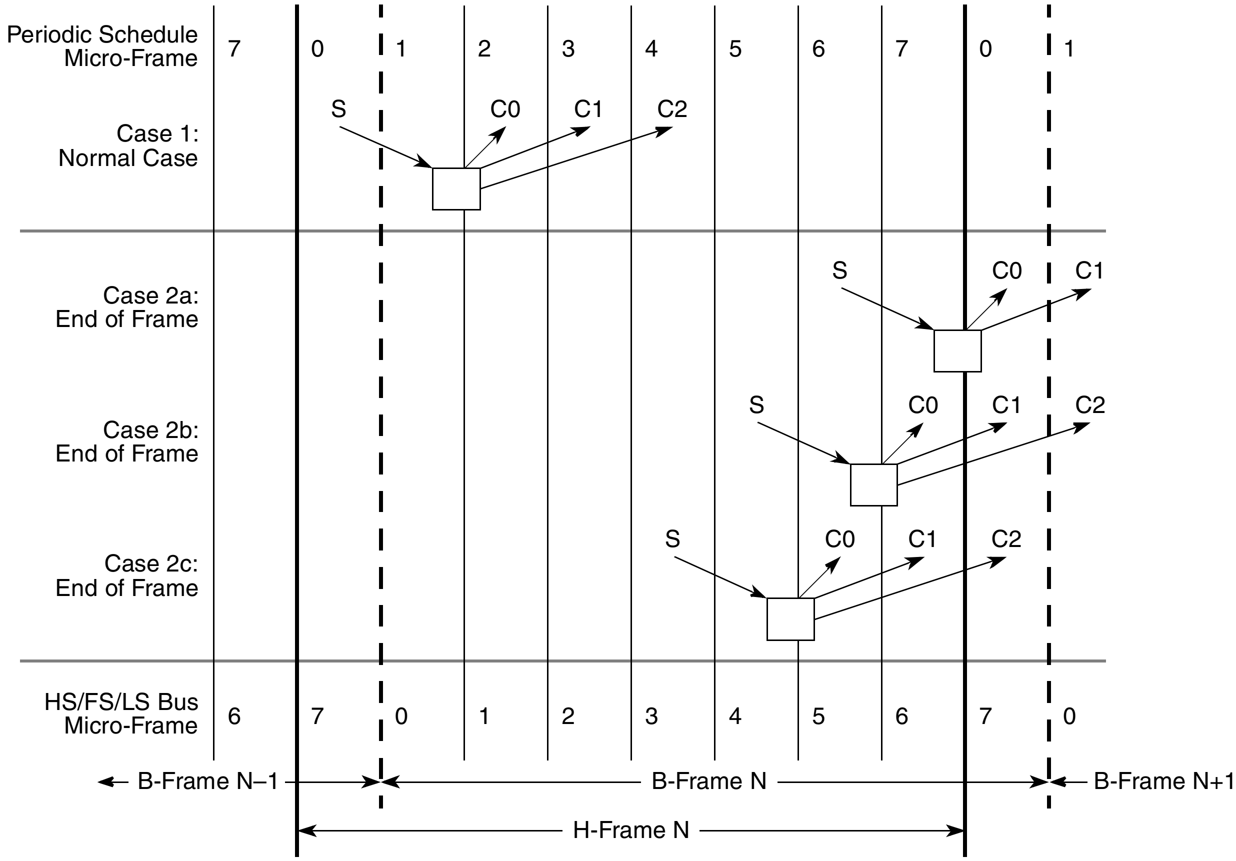


Figure 20-53. Split Transaction, Interrupt Scheduling Boundary Conditions

The scheduling cases are:

- Case 1: The normal scheduling case is where the entire split transaction is completely bounded by a frame (H-Frame in this case).
- Case 2a through Case 2c: The USB 2.0 hub pipeline rules states clearly, when and how many complete-splits must be scheduled to account for earliest to latest execution on the full/low-speed link. The complete-splits may span the H-Frame boundary when the start-split is in micro-frame 4 or later. When this occurs, the H-Frame to B-Frame alignment requires that the queue head be reachable from consecutive periodic frame list locations. System software cannot build an efficient

schedule that satisfies this requirement unless it uses FSTNs. Figure 20-54 illustrates the general layout of the periodic schedule.

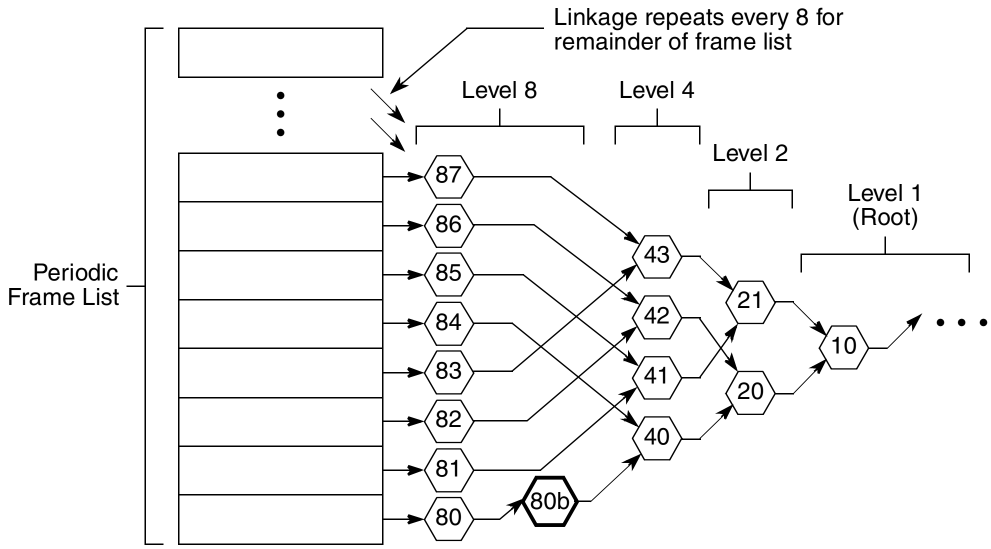


Figure 20-54. General Structure of EHCI Periodic Schedule Utilizing Interrupt Spreading

The periodic frame list is effectively the leaf level a binary tree, which is always traversed leaf to root. Each level in the tree corresponds to a 2^N poll rate. Software can efficiently manage periodic bandwidth on the USB by spreading interrupt queue heads that have the same poll rate requirement across all the available paths from the frame list. For example, system software can schedule eight poll rate 8 queue heads and account for them once in the high-speed bus bandwidth allocation.

When an endpoint is allocated an execution footprint that spans a frame boundary, the queue head for the endpoint must be reachable from consecutive locations in the frame list. An example would be if 8_{0b} were such an endpoint. Without additional support on the interface, to get 8_{0b} reachable at the correct time, software would have to link 8_1 to 8_{0b}. It would then have to move 4_1 and everything linked after into the same path as 4_0. This upsets the integrity of the binary tree and disallows the use of the spreading technique.

FSTN data structures are used to preserve the integrity of the binary-tree structure and enable the use of the spreading technique. Section 20.5.7, “Periodic Frame Span Traversal Node (FSTN),” defines the hardware and software operational model requirements for using FSTNs.

The following queue head fields are initialized by system software to instruct the host controller when to execute portions of the split-transaction protocol.

- SplitXState. This is a single bit residing in the Status field of a queue head (Table 20-56). This bit is used to track the current state of the split transaction.
- Frame S-mask. This is a bit-field where-in system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 20-53, case one, the S-mask would have a value of 0b0000_0001 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates

Do_Start, and the current micro-frame as indicated by FRINDEX[2–0] is 0, then execute a start-split transaction.

- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit in the Status field of the queue head. For example, referring to Figure 20-53, case one, the C-mask would have a value of 0b0001_1100 indicating that if the queue head is traversed by the host controller, and the SplitXState indicates Do_Complete, and the current micro-frame as indicated by FRINDEX[2–0] is 2, 3, or 4, then execute a complete-split transaction. It is software's responsibility to ensure that the translation between H-Frames and B-Frames is correctly performed when setting bits in S-mask and C-mask.

20.6.12.2.2 Host Controller Operational Model for FSTNs

The FSTN data structure is used to manage Low/Full-speed interrupt queue heads that need to be reached from consecutive frame list locations (that is, boundary cases 2a through 2c). An FSTN is essentially a back pointer, similar in intent to the back pointer field in the siTD data structure.

This feature provides software a simple primitive to save a schedule position, redirect the host controller to traverse the necessary queue heads in the previous frame, then restore the original schedule position and complete normal traversal.

There are four components to the use of FSTNs:

- FSTN data structure, defined in Section 20.5.7, “Periodic Frame Span Traversal Node (FSTN).”
- A Save Place indicator; this is always an FSTN with its Back Path Link Pointer[T] bit cleared.
- A Restore indicator; this is always an FSTN with its Back Path Link Pointer[T] bit set.
- Host controller FSTN traversal rules.

When the host controller encounters an FSTN during micro-frames 2 through 7 it simply follows the node's Normal Path Link Pointer to access the next schedule data structure. Note that the FSTN's Normal Path Link Pointer[T] bit may set, which the host controller must interpret as the end of periodic list mark.

When the host controller encounters a Save-Place FSTN in micro-frames 0 or 1, it saves the value of the Normal Path Link Pointer and sets an internal flag indicating that it is executing in Recovery Path mode. Recovery Path mode modifies the host controller's rules for how it traverses the schedule and limits which data structures are considered for execution of bus transactions. The host controller continues executing in Recovery Path mode until it encounters a Restore FSTN or it determines that it has reached the end of the micro-frame.

The rules for schedule traversal and limited execution while in Recovery Path mode are:

- Always follow the Normal Path Link Pointer when it encounters an FSTN that is a Save-Place indicator. The host controller must not recursively follow Save-Place FSTNs. Therefore, while executing in Recovery Path mode, it must never follow an FSTN's Back Path Link Pointer.
- Do not process an siTD or iTD data structure; simply follow its Next Link Pointer.
- Do not process a QH (Queue Head) whose EPS field indicates a high-speed device; simply follow its Horizontal Link Pointer.

- When a QH's EPS field indicates a Full/Low-speed device, the host controller only considers it for execution if its SplitXState is DoComplete (note: this applies whether the PID Code indicates an IN or an OUT). Refer to the *EHCI Specification* for a complete list of additional conditions that must be met in general for the host controller to issue a bus transaction. Note that the host controller must not execute a Start-split transaction while executing in Recovery Path mode. Refer to the *EHCI Specification* for special handling when in Recovery Path mode.
- Stop traversing the recovery path when it encounters an FSTN that is a Restore indicator. The host controller unconditionally uses the saved value of the Save-Place FSTN's Normal Path Link Pointer when returning to the normal path traversal. The host controller must clear the context of executing a Recovery Path when it restores schedule traversal to the Save-Place FSTN's Normal Path Link Pointer.

  If the host controller determines that there is not enough time left in the micro-frame to complete processing of the periodic schedule, it abandons traversal of the recovery path, and clears the context of executing a recovery path. The result is that at the start of the next consecutive micro-frame, the host controller starts traversal at the frame list.

An example traversal of a periodic schedule that includes FSTNs is illustrated in Figure 20-55.

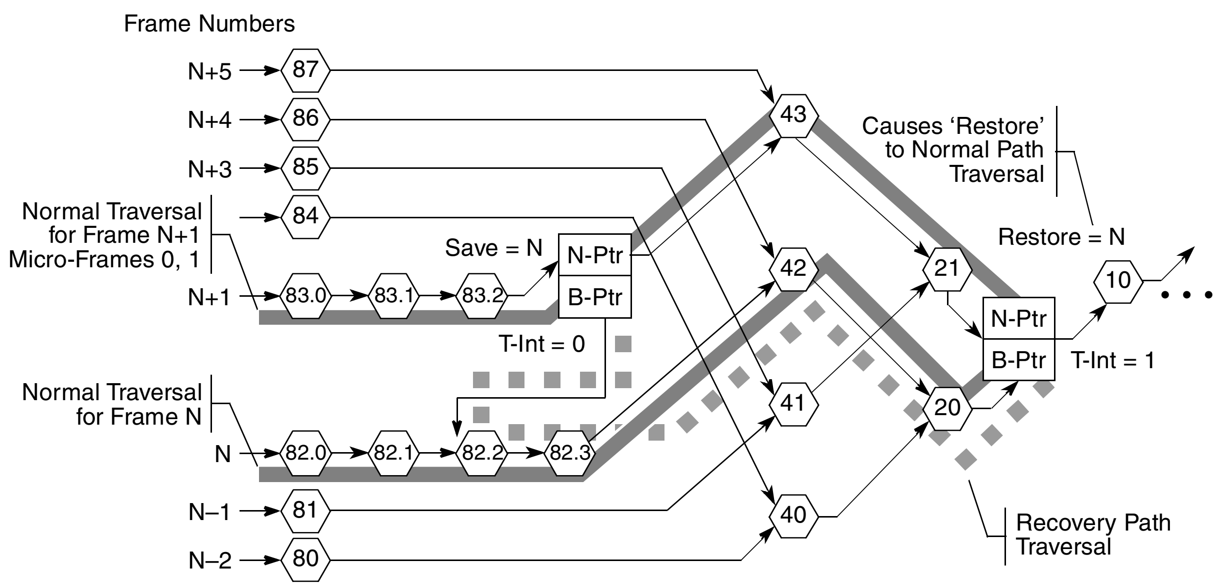


Figure 20-55. Example Host Controller Traversal of Recovery Path via FSTNs

In frame N (micro-frames 0-7), for this example, the host controller traverses all of the schedule data structures utilizing the Normal Path Link Pointers in any FSTNs it encounters. This is because the host controller has not yet encountered a Save-Place FSTN so it is not executing in Recovery Path mode. When it encounters the Restore FSTN, (Restore-N), during micro-frames 0 and 1, it uses Restore-N. Normal Path Link Pointer to traverse to the next data structure (that is, normal schedule traversal). This is because the host controller must use a Restore FSTN's Normal Path Link Pointer when not executing in a Recovery-Path mode. The nodes traversed during frame N include: {$8_{2.0}$, $8_{2.1}$, $8_{2.2}$, $8_{2.3}$, 4_2, 2_0, Restore-N, 1_0 ...}.

In frame N+1 (micro-frames 0 and 1), when the host controller encounters Save-Path FSTN (Save-N), it observes that Save-N.Back Path Link Pointer.T-bit is zero (definition of a Save-Path indicator). The host controller saves the value of Save-N. Normal Path Link Pointer and follows Save-N.Back Path Link Pointer. At the same time, it sets an internal flag indicating that it is now in Recovery Path mode (the recovery path is annotated in Figure 20-55 with a large dashed line). The host controller continues traversing data structures on the recovery path and executing only those bus transactions as noted above, on the recovery path until it reaches Restore FSTN (Restore-N). Restore-N.Back Path Link Pointer.T-bit is set (definition of a Restore indicator), so the host controller exits Recovery Path mode by clearing the internal Recovery Path mode flag and commences (restores) schedule traversal using the saved value of the Save-Place FSTN's Normal Path Link Pointer (for example, Save-N.Normal Path Link Pointer). The nodes traversed during these micro-frames include: {$8_{3.0}$, $8_{3.1}$, $8_{3.2}$, Save-A, $8_{2.2}$, $8_{2.3}$, 4_2, 2_0, Restore-N, 4_3, 2_1, Restore-N, 10 ...}.

In frame N+1 (micro-frames 2-7), when the host controller encounters Save-Path FSTN Save-N, it unconditionally follows Save-N.Normal Path Link Pointer. The nodes traversed during these micro-frames include: {$8_{3.0}$, $8_{3.1}$, $8_{3.2}$, Save-A, 4_3, 2_1, Restore-N, 1_0 ...}.

20.6.12.2.3 Software Operational Model for FSTNs

Software must create a consistent, coherent schedule for the host controller to traverse. When using FSTNs, system software must adhere to the following rules:

- Each Save-Place indicator requires a matching Restore indicator.

  The Save-Place indicator is an FSTN with a valid Back Path Link Pointer and T-bit equal to zero. Note that Back Path Link Pointer[Typ] field must be set to indicate the referenced data structure is a queue head. The Restore indicator is an FSTN with its Back Path Link Pointer[T] bit set.

  A Restore FSTN may be matched to one or more Save-Place FSTNs. For example, if the schedule includes a poll-rate 1 level, then system software only needs to place a Restore FSTN at the beginning of this list in order to match all possible Save-Place FSTNs.
- If the schedule does not have elements linked at a poll-rate level of one, and one or more Save-Place FSTNs are used, then System Software must ensure the Restore FSTN's Normal Path Link Pointer's T-bit is set, as this will be use to mark the end of the periodic list.
- When the schedule does have elements linked at a poll rate level of one, a Restore FSTN must be the first data structure on the poll rate one list. All traversal paths from the frame list converge on the poll-rate one list. System software must ensure that Recovery Path mode is exited before the host controller is allowed to traverse the poll rate level one list.
- A Save-Place FSTN's Back Path Link Pointer must reference a queue head data structure. The referenced queue head must be reachable from the previous frame list location. In other words, if the Save-Place FSTN is reachable from frame list offset N, then the FSTN's Back Path Link Pointer must reference a queue head that is reachable from frame list offset N-1.

Software should make the schedule as efficient as possible. What this means in this context is that software should have no more than one Save-Place FSTN reachable in any single frame. Note there will be times when two (or more, depending on the implementation) could exist as full-/low-speed footprints change with bandwidth adjustments. This could occur, for example when a bandwidth rebalance causes system software to move the Save-Place FSTN from one poll rate level to another. During the transition, software must preserve the integrity of the previous schedule until the new schedule is in place.

20.6.12.2.4 Tracking Split Transaction Progress for Interrupt Transfers

To correctly maintain the data stream, the host controller must be able to detect and report errors where data is lost. For interrupt-IN transfers, data is lost when it makes it into the USB 2.0 hub, but the USB 2.0 host system is unable to get it from the USB 2.0 hub and into the system before it expires from the transaction translator pipeline. When a lost data condition is detected, the queue is halted, thus signaling system software to recover from the error. A data-loss condition exists whenever a start-split is issued, accepted and successfully executed by the USB 2.0 hub, but the complete-splits get unrecoverable errors on the high-speed link, or the complete-splits do not occur at the correct times. One reason complete-splits might not occur at the right time would be due to host-induced system hold-offs that cause the host controller to miss bus transactions because it cannot get timely access to the schedule in system memory.

The same condition can occur for an interrupt-OUT, but the result is not an endpoint halt condition, but rather effects only the progress of the transfer. The queue head has the following fields to track the progress of each split transaction. These fields are used to keep incremental state about which (and when) portions have been executed.

- C-prog-mask. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the transaction translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets one of the C-prog-mask bits for each complete-split executed. The bit position is determined by the micro-frame number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed then it means one (or more) have been skipped and data has potentially been lost.
- FrameTag. This field is used by the host controller during the complete-split portion of the split transaction to tag the queue head with the frame number (H-Frame number) when the next complete split must be executed.
- S-bytes. This field can be used to store the number of data payload bytes sent during the start-split (if the transaction was an OUT). The S-bytes field must be used to accumulate the data payload bytes received during the complete-splits (for an IN).

20.6.12.2.5 Split Transaction Execution State Machine for Interrupt

In the following section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

As with asynchronous Full- and Low-speed endpoints, a split-transaction state machine is used to manage the split transaction sequence. Aside from the fields defined in the queue head for scheduling and tracking the split transaction, the host controller calculates one internal mechanism that is also used to manage the split transaction. The internal calculated mechanism is:

- cMicroFrameBit. This is a single-bit encoding of the current micro-frame number. It is an eight-bit value calculated by the host controller at the beginning of every micro-frame. It is calculated from the three least significant bits of the FRINDEX register (that is, cMicroFrameBit = (1 shifted-left(FRINDEX[2–0]))). The cMicroFrameBit has at most one bit asserted, which always corresponds to the current micro-frame number. For example, if the current micro-frame is 0, then cMicroFrameBit will equal 0b0000_0001.

  The variable cMicroFrameBit is used to compare against the S-mask and C-mask fields to determine whether the queue head is marked for a start- or complete-split transaction for the current micro-frame.

Figure 20-56 illustrates how a complete interrupt split transaction is managed. There are two phases to each split transaction. The first is a single start-split transaction, which occurs when the SplitXState is at Do_Start and the single bit in cMicroFrameBit has a corresponding bit active in QH[S-mask]. The transaction translator does not acknowledge the receipt of the periodic start-split, so the host controller unconditionally transitions the state to Do_Complete. Due to the available jitter in the transaction translator pipeline, there will be more than one complete-split transaction scheduled by software for the Do_Complete state. This translates simply to the fact that there are multiple bits set in the QH[C-mask] field.

The host controller keeps the queue head in the Do_Complete state until the split transaction is complete (see definition below), or an error condition triggers the three-strikes-rule (for example, after the host tries the same transaction three times, and each encounters an error, the host controller stops retrying the bus transaction and halts the endpoint, thus requiring system software to detect the condition and perform system-dependent recovery).

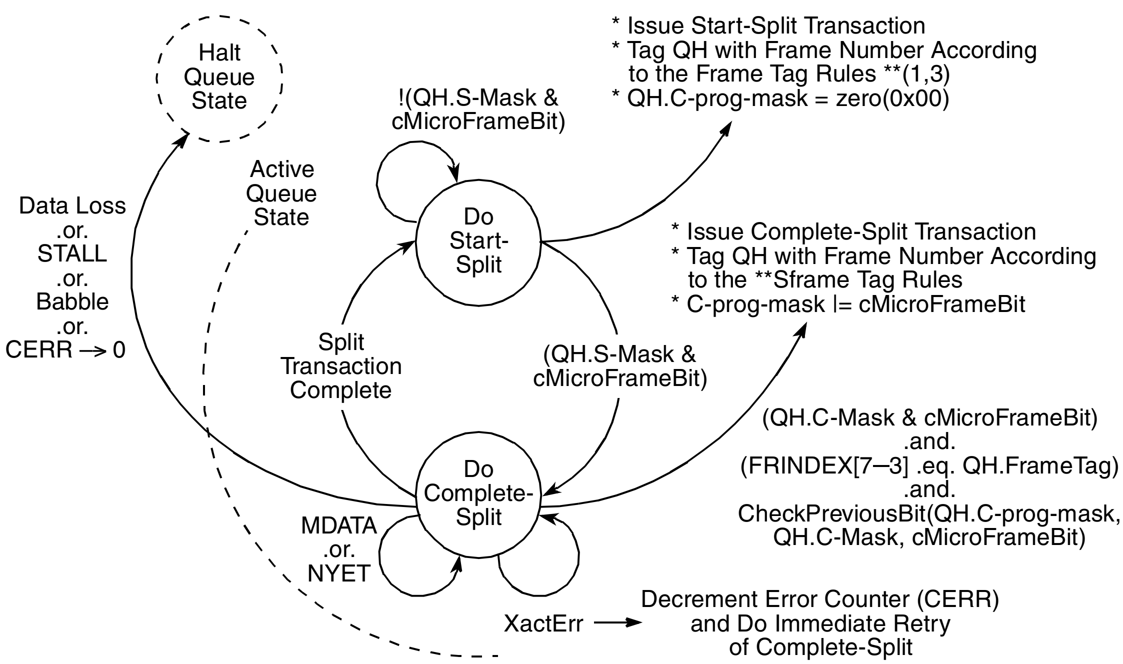


Figure 20-56. Split Transaction State Machine for Interrupt

20.6.12.2.6 Periodic Interrupt—Do-Start-Split

This is the state software must initialize a full- or low-speed interrupt queue head StartXState bit. This state is entered from the Do_Complete Split state only after the split transaction is complete. This occurs when one of the following events occur: The transaction translator responds to a complete-split transaction with one of the following:

- NAK. A NAK response is a propagation of the full- or low-speed endpoint's NAK response.
- ACK. An ACK response is a propagation of the full- or low-speed endpoint's ACK response. Only occurs on an OUT endpoint.
- DATA 0/1. Only occurs for INs. Indicates that this is the last of the data from the endpoint for this split transaction.
- ERR. The transaction on the low-/full-speed link below the transaction translator had a failure (for example, timeout, bad CRC, etc.).
- NYET (and Last). The host controller issued the last complete-split and the transaction translator responded with a NYET handshake. This means that the start-split was not correctly received by the transaction translator, so it never executed a transaction to the full- or low-speed endpoint, see Section Periodic Interrupt - Do Complete Split for the definition of 'Last'.

Each time the host controller visits a queue head in this state (once within the Execute Transaction state), bit-wise ANDs QH[S-mask] with cMicroFrameBit to determine whether to execute a start-split. If the result is non-zero, then the host controller issues a start-split transaction. If the PID Code field indicates an IN transaction, the host controller must zero-out the QH[S-bytes] field. After the split-transaction has been executed, the host controller sets up state in the queue head to track the progress of the complete-split phase of the split transaction. Specifically, it records the expected frame number into QH[FrameTag] field, sets C-prog-mask to zero (0x00), and exits this state. Note that the host controller must not adjust the value of Cerr as a result of completion of a start-split transaction.

20.6.12.2.7 Periodic Interrupt—Do-Complete-Split

This state is entered unconditionally from the Do Start Split state after a start-split transaction is executed on the bus. Each time the host controller visits a queue head in this state (once within the Execute Transaction state), it checks to determine whether a complete-split transaction should be executed now.

There are four tests to determine whether a complete-split transaction should be executed.

- Test A. cMicroFrameBit is bit-wise ANDed with QH[C-mask] field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame.
- Test B. QH[FrameTag] is compared with the current contents of FRINDEX[7–3]. An equal indicates a match.
- Test C. The complete-split progress bit vector is checked to determine whether the previous bit is set, indicating that the previous complete-split was appropriately executed. An example algorithm for this test is provided below:

```
Algorithm Boolean CheckPreviousBit(QH.C-prog-mask, QH.C-mask, cMicroFrameBit)
Begin
-- Return values:
-- TRUE - no error
-- FALSE - error
```

```
--
Boolean rvalue = TRUE;
previousBit = cMicroframeBit logical-rotate-right(1)
-- Bit-wise anding previousBit with C-mask indicates
-- whether there was an intent
-- to send a complete split in the previous micro-frame. So,
-- if the
-- 'previous bit' is set in C-mask, check C-prog-mask to
-- make sure it
-- happened.
If (previousBit bitAND QH.C-mask)then
                                If not(previousBit bitAND QH.C-prog-mask) then
         rvalue = FALSE;
     End if
End If
-- If the C-prog-mask already has a one in this bit position,
-- then an aliasing
-- error has occurred. It will probably get caught by the
-- FrameTag Test, but
-- at any rate it is an error condition that as detectable here
-- should not allow
-- a transaction to be executed.
If (cMicroFrameBit bitAND QH.C-prog-mask) then
rvalue = FALSE;
End if
return (rvalue)
End Algorithm
```

- Test D. Check to see if a start-split should be executed in this micro-frame. Note this is the same test performed in the Do Start Split state. Whenever it evaluates to TRUE and the controller is NOT processing in the context of a Recovery Path mode, it means a start-split should occur in this micro-frame. Test D and Test A evaluating to TRUE at the same time is a system software error. Behavior is undefined.

If (A .and. B .and. C .and. not(D)) then the host controller will execute a complete-split transaction. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. On completion of the complete-split transaction, the host controller records the result of the transaction in the queue head and sets QH[FrameTag] to the expected H-Frame number. The effect to the state of the queue head and thus the state of the transfer depends on the response by the transaction translator to the complete-split transaction. The following responses have the effects (note that any responses that result in decrementing of the Cerr will result in the queue head being halted by the host controller if the result of the decrement is zero):

- NYET (and Last). On each NYET response, the host controller checks to determine whether this is the last complete-split for this split transaction. Last is defined in this context as the condition where all of the scheduled complete-splits have been executed. If it is the last complete-split (with a NYET response), then the transfer state of the queue head is not advanced (never received any data) and this state exited. The transaction translator must have responded to all the complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. The start-split should be retried at the next poll period.
- The test for whether this is the Last complete split can be performed by XOR QH[C-mask] with QH[C-prog-mask]. If the result is all zeros then all complete-splits have been executed. When this condition occurs, the XactErr status bit is set and the Cerr field is decremented.

- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask and FrameTag) and stay in this state. The host controller must not adjust Cerr on this response.
- Transaction Error (XactErr). Timeout, data CRC failure, etc. The Cerr field is decremented and the XactErr bit in the Status field is set. The complete split transaction is immediately retried (if Cerr is non-zero).If there is not enough time in the micro-frame to complete the retry and the endpoint is an IN, or Cerr is decremented to a zero from a one, the queue is halted. If there is not enough time in the micro-frame to complete the retry and the endpoint is an OUT and Cerr is not zero, then this state is exited (that is, return to Do Start Split). This results in a retry of the entire OUT split transaction, at the next poll period. Refer to Chapter 11 Hubs (specifically the section on full- and low-speed interrupts) in the *USB Specification Revision 2.0* for detailed requirements on why these errors must be immediately retried.
- ACK. This can only occur if the target endpoint is an OUT. The target endpoint ACK'd the data and this response is a propagation of the endpoint ACK up to the host controller. The host controller must advance the state of the transfer. The Current Offset field is incremented by Maximum Packet Length or Bytes to Transfer, whichever is less. The field Bytes To Transfer is decremented by the same amount. And the data toggle bit (dt) is toggled. The host controller will then exit this state for this queue head. The host controller must reload Cerr with maximum value on this response. Advancing the transfer state may cause other process events such as retirement of the qTD and advancement of the queue.
- MDATA. This response will only occur for an IN endpoint. The transaction translator responded with zero or more bytes of data and an MDATA PID. The incremental number of bytes received is accumulated in QH[S-bytes]. The host controller must not adjust Cerr on this response.
- DATA0/1. This response may only occur for an IN endpoint. The number of bytes received is added to the accumulated byte count in QH[S-bytes]. The state of the transfer is advanced by the result and the host controller exits this state for this queue head.
- Advancing the transfer state may cause other processing events such as retirement of the qTD and advancement of the queue.
- If the data sequence PID does not match the expected, the entirety of the data received in this split transaction is ignored, the transfer state is not advanced and this state is exited.
- NAK. The target endpoint Nak'd the full- or low-speed transaction. The state of the transfer is not advanced, and this state is exited. The host controller must reload Cerr with maximum value on this response.
- ERR. There was an error during the full- or low-speed transaction. The ERR status bit is set, Cerr is decremented, the state of the transfer is not advanced, and this state is exited.
- STALL. The queue is halted (an exit condition of the Execute Transaction state). The status field bits: Active bit is cleared and the Halted bit is set and the qTD is retired. Responses which are not enumerated in the list or which are received out of sequence are illegal and may result in undefined

host controller behavior. The other possible combinations of tests A, B, C, and D may indicate that data or response was lost. Table 20-69 lists the possible combinations and the appropriate action.

Table 20-69. Interrupt IN/OUT Do Complete Split State Execution Criteria

Condition	Action	Description
not(A) not(D)	Ignore QHD	Neither a start nor complete-split is scheduled for the current micro-frame.Host controller should continue walking the schedule.
A not(C)	If PIDCode = IN Halt QHDIf PIDCode = OUT Retry start-split	Progress bit check failed. These means a complete-split has been missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted.If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one.
A not(B) C	If PIDCode = IN Halt QHD If PIDCode = OUT Retry start-split	QH.FrameTag test failed. This means that exactly one or more H-Frames have been skipped. This means complete-splits and have missed. There is the possibility of lost data. If PID Code is an IN, then the Queue head must be halted.If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one.
A B C not(D)	Execute complete-split	This is the non-error case where the host controller executes a complete-split transaction.
D	If PIDCode = IN Halt QHDIf PIDCode = OUT Retry start-split	This is a degenerate case where the start-split was issued, but all of the complete-splits were skipped and all possible intervening opportunities to detect the missed data failed to fire. If PID Code is an IN, then the Queue head must be halted. If PID Code is an OUT, then the transfer state is not advanced and the state exited (for example, start-split is retried). This is a host-induced error and does not effect Cerr. In either case, set the Missed Micro-frame bit in the status field to a one. Note that when executing in the context of a Recovery Path mode, the host controller is allowed to process the queue head and take the actions indicated above, or it may wait until the queue head is visited in the normal processing mode. Regardless, the host controller must not execute a start-split in the context of a executing in a Recovery Path mode.

20.6.12.2.8 Managing the QH[FrameTag] Field

The QH[FrameTag] field in a queue head is completely managed by the host controller. The rules for setting QH[FrameTag] are simple:

- Rule 1: If transitioning from Do Start Split to Do Complete Split and the current value of FRINDEX[2–0] is 6, QH[FrameTag] is set to FRINDEX[7–3] + 1. This accommodates split transactions whose start-split and complete-splits are in different H-Frames (case 2a, see Figure 20-53).
- Rule 2: If the current value of FRINDEX[2–0] is 7, QH[FrameTag] is set to FRINDEX[7–3] + 1. This accommodates staying in Do Complete Split for cases 2a, 2b, and 2c in Figure 20-53.
- Rule 3: If transitioning from Do_Start Split to Do Complete Split and the current value of FRINDEX[2–0] is not 6, or currently in Do Complete Split and the current value of (FRINDEX[2–0]) is not 7, FrameTag is set to FRINDEX[7–3]. This accommodates all other cases in Figure 20-53.

20.6.12.2.9 Rebalancing the Periodic Schedule

System software must occasionally adjust a periodic queue head's S-mask and C-mask fields during operation. This need occurs when adjustments to the periodic schedule create a new bandwidth budget and one or more queue head's are assigned new execution footprints (that is, new S-mask and C-mask values).

It is imperative that system software must not update these masks to new values in the midst of a split transaction. In order to avoid any race conditions with the update, the host controller provides a simple assist to system software. System software sets the Inactivate-on-next-Transaction (I) bit to signal the host controller that it intends to update the S-mask and C-mask on this queue head. System software then waits for the host controller to observe the I-bit is set and transitions the Active bit to a zero. The rules for how and when the host controller clears the Active bit are:

- If the Active bit is cleared, no action is taken. The host controller does not attempt to advance the queue when the I-bit is set.
- If the Active bit is set and the SplitXState is DoStart (regardless of the value of S-mask), the host controller simply clears the Active bit. The host controller is not required to write the transfer state back to the current qTD. Note that if the S-mask indicates that a start-split is scheduled for the current micro-frame, the host controller must not issue the start-split bus transaction; it must clear the Active bit.

System software must save transfer state before setting the I-bit. This is required so that it can correctly determine what transfer progress (if any) occurred after the I-bit was set and the host controller executed it's final bus-transaction and cleared the Active bit.

After system software has updated the S-mask and C-mask, it must then reactivate the queue head. Since the Active bit and the I-bit cannot be updated with the same write, system software needs to use the following algorithm to coherently re-activate a queue head that has been stopped using the I-bit.

1. Set the Halted bit, then
2. Clear the I-bit, then
3. Set the Active bit and clear the Halted bit in the same write.

Setting the Halted bit inhibits the host controller from attempting to advance the queue between the time the I-bit is cleared and the Active bit is set.

20.6.12.3 Split Transaction Isochronous

Full-speed isochronous transfers are managed using the split-transaction protocol through a USB 2.0 transaction translator in a USB 2.0 hub. The host controller utilizes siTD data structure to support the special requirements of isochronous split-transactions. This data structure uses the scheduling model of isochronous TDs (see Section 20.6.8, “Managing Isochronous Transfers Using iTDs,” for the operational model of iTDs) with the contiguous data feature provided by queue heads. This simple arrangement allows a single isochronous scheduling model and adds the additional feature that all data received from the endpoint (per split transaction) must land into a contiguous buffer.

20.6.12.3.1 Split Transaction Scheduling Mechanisms for Isochronous

Full-speed isochronous transactions are managed through a transaction translator's periodic pipeline. As with full- and low-speed interrupt, system software manages each transaction translator's periodic pipeline by budgeting and scheduling exactly during which micro-frames the start-splits and complete-splits for each full-speed isochronous endpoint occur. The requirements described in Section Split Transaction Scheduling Mechanisms for Interrupt apply. Figure 20-57 illustrates the general scheduling boundary conditions that are supported by the EHCI periodic schedule. The S_n and C_n labels indicate micro-frames where software can schedule start- and complete-splits (respectively). The H-Frame boundaries are marked with a large, solid bold vertical line. The B-Frame boundaries are marked with a large, bold, dashed line. The bottom of Figure 20-57 illustrates the relationship of an siTD to the H-Frame.

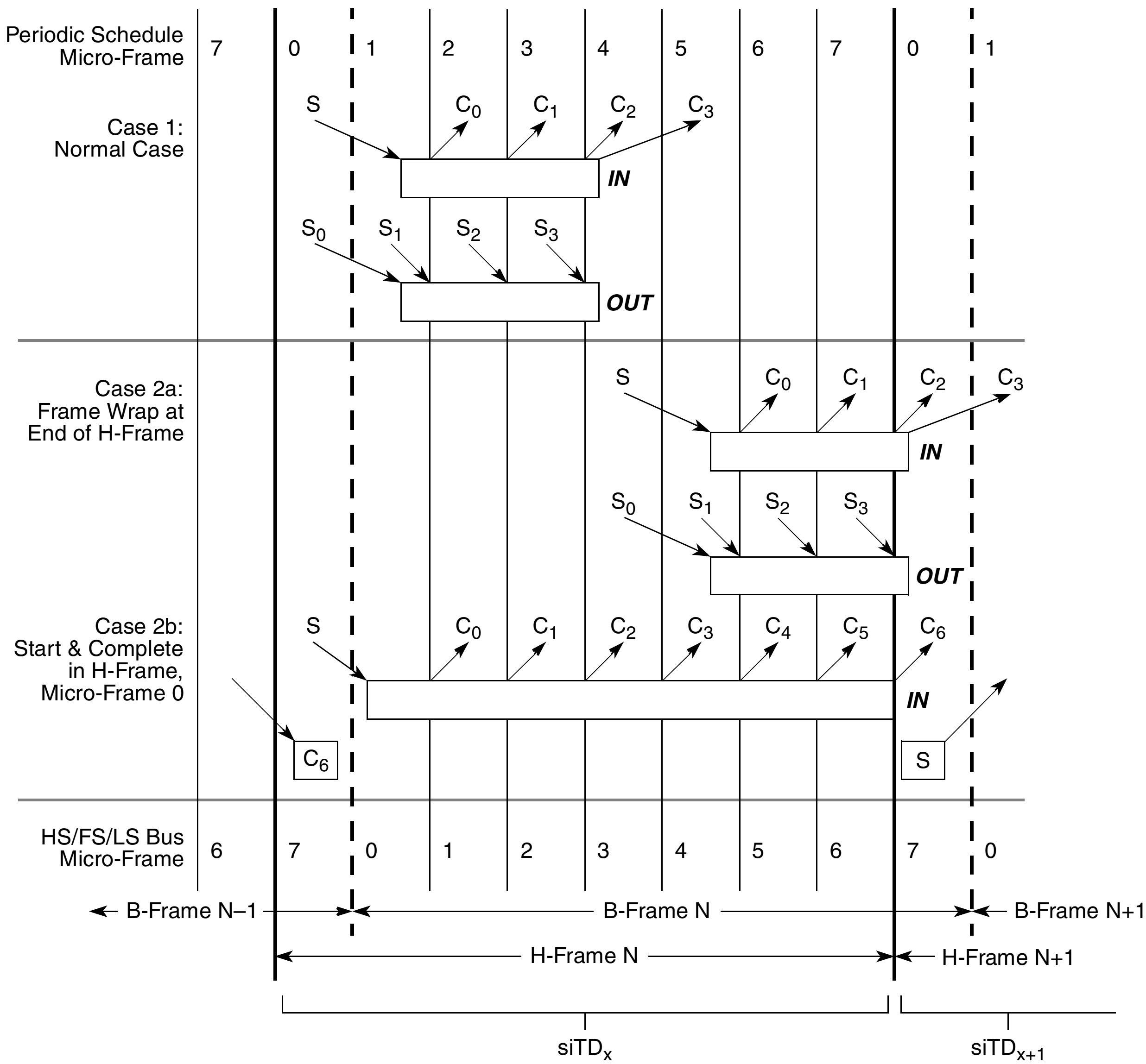


Figure 20-57. Split Transaction, Isochronous Scheduling Boundary Conditions

When the endpoint is an isochronous OUT, there are only start-splits, and no complete-splits. When the endpoint is an isochronous IN, there is at most one start-split and one to N complete-splits. The scheduling boundary cases are:

- Case 1: The entire split transaction is completely bounded by an H-Frame. For example, the start-splits and complete-splits are all scheduled to occur in the same H-Frame.
- Case 2a: This boundary case is where one or more (at most two) complete-splits of a split transaction IN are scheduled across an H-Frame boundary. This can only occur when the split transaction has the possibility of moving data in B-Frame, micro-frames 6 or 7 (H-Frame micro-frame 7 or 0). When an H-Frame boundary wrap condition occurs, the scheduling of the split transaction spans more than one location in the periodic list.(for example, it takes two siTDs in adjacent periodic frame list locations to fully describe the scheduling for the split transaction).

  Although the scheduling of the split transaction may take two data structures, all of the complete-splits for each full-speed IN isochronous transaction must use only one data pointer. For this reason, siTDs contain a back pointer.

  Software must never schedule full-speed isochronous OUTs across an H-Frame boundary.
- Case 2b: This case can only occur for a very large isochronous IN. It is the only allowed scenario where a start-split and complete-split for the same endpoint can occur in the same micro-frame. Software must enforce this rule by scheduling the large transaction first. Large is defined to be anything larger than 579 byte maximum packet size.

A subset of the same mechanisms employed by full- and low-speed interrupt queue heads are employed in siTDs to schedule and track the portions of isochronous split transactions. The following fields are initialized by system software to instruct the host controller when to execute portions of the split transaction protocol:

- SplitXState. This is a single bit residing in the Status field of an siTD (see Table 20-50). This bit is used to track the current state of the split transaction. The rules for managing this bit are described in Section 20.6.12.3.3, “Split Transaction Execution State Machine for Isochronous.”
- Frame S-mask. This is a bit-field wherein system software sets a bit corresponding to the micro-frame (within an H-Frame) that the host controller should execute a start-split transaction. This is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 20-57, case 1, the S-mask would have a value of 0b0000_0001 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Start Split, and the current micro-frame as indicated by FRINDEX[2–0] is 0, then execute a start-split transaction.
- Frame C-mask. This is a bit-field where system software sets one or more bits corresponding to the micro-frames (within an H-Frame) that the host controller should execute complete-split transactions. The interpretation of this field is always qualified by the value of the SplitXState bit. For example, referring to the IN example in Figure 20-57, case 1, the C-mask would have a value of 0b 0011_1100 indicating that if the siTD is traversed by the host controller, and the SplitXState indicates Do Complete Split, and the current micro-frame as indicated by FRINDEX[2–0] is 2, 3, 4, or 5, then execute a complete-split transaction.
- Back Pointer. This field in a siTD is used to complete an IN split-transaction using the previous H-Frame's siTD. This is only used when the scheduling of the complete-splits span an H-Frame boundary.

There exists a one-to-one relationship between a high-speed isochronous split transaction (including all start- and complete-splits) and one full-speed isochronous transaction. An siTD contains (amongst other things) buffer state and split transaction scheduling information. An siTD's buffer state always maps to one full-speed isochronous data payload. This means that for any full-speed transaction payload, a single siTD's data buffer must be used. This rule applies to both IN an OUTs. An siTD's scheduling information usually also maps to one high-speed isochronous split transaction. The exception to this rule is the H-Frame boundary wrap cases mentioned above.

The siTD data structure describes at most, one frame's worth of high-speed transactions and that description is strictly bounded within a frame boundary. Figure 20-58 illustrates some examples. On the top are examples of the full-speed transaction footprints for the boundary scheduling cases described above. In the middle are time-frame references for both the B-Frames (HS/FS/LS Bus) and the H-Frames. On the bottom is illustrated the relationship between the scope of an siTD description and the time references. Each H-Frame corresponds to a single location in the periodic frame list. The implication is that each siTD is reachable from a single periodic frame list location at a time.

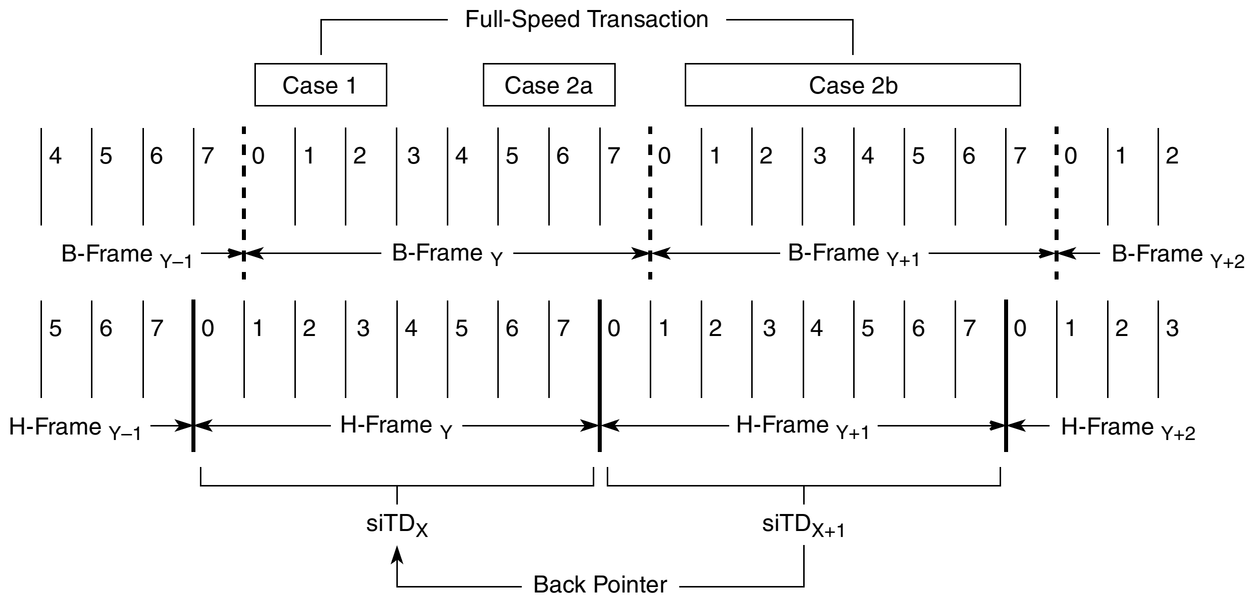


Figure 20-58. siTD Scheduling Boundary Examples

Each case is described below:

- Case 1: One siTD is sufficient to describe and complete the isochronous split transaction because the whole isochronous split transaction is tightly contained within a single H-Frame.
- Case 2a, 2b: Although both INs and OUTs can have these footprints, OUTs always take only one siTD to schedule. However, INs (for these boundary cases) require two siTDs to complete the scheduling of the isochronous split transaction. siTDX is used to always issue the start-split and the first N complete-splits. The full-speed transaction (for these cases) can deliver data on the full-speed bus segment during micro-frame 7 of H-Frame$_{Y+1}$, or micro-frame 0 of H-Frame$_{Y+2}$. The complete splits are scheduled using siTD$_{X+2}$ (not shown). The complete-splits to extract this data must use the buffer pointer from siTD$_{X+1}$. The only way for the host controller to reach siTD$_{X+1}$ from H-Frame$_{Y+2}$ is to use siTD$_{X+2}$'s back pointer.

Software must apply the following rules when calculating the schedule and linking the schedule data structures into the periodic schedule:

- Software must ensure that an isochronous split-transaction is started so that it will complete before the end of the B-Frame.
- Software must ensure that for a single full-speed isochronous endpoint, there is never a start-split and complete-split in H-Frame, micro-frame 1. This is mandated as a rule so that case 2a and case 2b can be discriminated. According to the core USB specification, the long isochronous transaction illustrated in Case 2b, could be scheduled so that the start-split was in micro-frame 1 of H-Frame N and the last complete-split would need to occur in micro-frame 1 of H-Frame N+1. However, it is impossible to discriminate between cases 2a and case 2b, which has significant impact on the complexity of the host controller.

20.6.12.3.2 Tracking Split Transaction Progress for Isochronous Transfers

Isochronous endpoints do not employ the concept of a halt on error, however the host controller does identify and report per-packet errors observed in the data stream. This includes schedule traversal problems (skipped micro-frames), timeouts and corrupted data received.

In similar kind to interrupt split-transactions, the portions of the split transaction protocol must execute in the micro-frames they are scheduled. The queue head data structure used to manage full- and low-speed interrupt has several mechanisms for tracking when portions of a transaction have occurred. Isochronous transfers use siTDs for their transfers and the data structures are only reachable using the schedule in the exact micro-frame in which they are required (so all the mechanism employed for tracking in queue heads is not required for siTDs). Software has the option of reusing siTD several times in the complete periodic schedule. However, it must ensure that the results of split transaction N are consumed and the siTD re-initialized (activated) before the host controller gets back to the siTD (in a future micro-frame).

Split-transaction isochronous OUTs utilize a low-level protocol to indicate which portions of the split transaction data have arrived. Control over the low-level protocol is exposed in an siTD using the fields Transaction Position (TP) and Transaction Count (T-count). If the entire data payload for the OUT split transaction is larger than 188 bytes, there will be more than one start-split transaction, each of which require proper annotation. If host hold-offs occur, then the sequence of annotations received from the host will not be complete, which is detected and handled by the transaction translator. See Section 20.6.12.3.1, “Split Transaction Scheduling Mechanisms for Isochronous,” for a description on how these fields are used during a sequence of start-split transactions.

The fields siTD[T-Count] and siTD[TP] are used by the host controller to drive and sequence the transaction position annotations. It is the responsibility of system software to properly initialize these fields in each siTD. Once the budget for a split-transaction isochronous endpoint is established, S-mask, T-Count, and TP initialization values for all the siTD associated with the endpoint are constant. They remain constant until the budget for the endpoint is recalculated by software and the periodic schedule adjusted.

For IN-endpoints, the transaction translator simply annotates the response data packets with enough information to allow the host controller to identify the last data. As with split transaction Interrupt, it is the host controller's responsibility to detect when it has missed an opportunity to execute a complete-split. The following field in the siTD is used to track and detect errors in the execution of a split transaction for an IN isochronous endpoint.

- C-prog-mask. This is an eight-bit bit-vector where the host controller keeps track of which complete-splits have been executed. Due to the nature of the transaction translator periodic pipeline, the complete-splits need to be executed in-order. The host controller needs to detect when the complete-splits have not been executed in order. This can only occur due to system hold-offs where the host controller cannot get to the memory-based schedule. C-prog-mask is a simple bit-vector that the host controller sets a bit for each complete-split executed. The bit position is determined by the micro-frame (FRINDEX[2–0]) number in which the complete-split was executed. The host controller always checks C-prog-mask before executing a complete-split transaction. If the previous complete-splits have not been executed, then it means one (or more) have been skipped and data has potentially been lost. System software is required to initialize this field to zero before setting an siTD's Active bit to a one.

If a transaction translator returns with the final data before all of the complete-splits have been executed, the state of the transfer is advanced so that the remaining complete-splits are not executed. It is important to note that an IN siTD is retired based solely on the responses from the transaction translator to the complete-split transactions. This means, for example, that it is possible for a transaction translator to respond to a complete-split with an MDATA PID. The number of bytes in the MDATA's data payload could cause the siTD[Total Bytes to Transfer] field to decrement to zero. This response can occur, before all of the scheduled complete-splits have been executed. In other interface, data structures (for example, high-speed data streams through queue heads), the transition of Total Bytes to Transfer to zero signals the end of the transfer and results in clearing the Active bit. However, in this case, the result has not been delivered by the transaction translator and the host must continue with the next complete-split transaction to extract the residual transaction state. This scenario occurs because of the pipeline rules for a transaction translator. In summary, the periodic pipeline rules require that on a micro-frame boundary, the transaction translator holds the final two bytes received (if it has not seen an End Of Packet (EOP)) in the full-speed bus pipe stage and gives the remaining bytes to the high-speed pipeline stage. At the micro-frame boundary, the transaction translator could have received the entire packet (including both CRC bytes) but not received the packet EOP. In the next micro-frame, the transaction translator responds with an MDATA and sends all of the data bytes (with the two CRC bytes being held in the full-speed pipeline stage). This could cause the siTD to decrement it's Total Bytes to Transfer field to zero, indicating it has received all expected data. The host must still execute one more (scheduled) complete-split transaction in order to extract the results of the full-speed transaction from the transaction translator (for example, the transaction translator may have detected a CRC failure, and this result must be forwarded to the host).

If the host experiences hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous OUT, then the protocol to the transaction translator is not consistent and the transaction translator detects and reacts to the problem. Likewise, for host hold-offs that cause the host controller to skip one or more (but not all) scheduled split transactions for an isochronous IN, the C-prog-mask is used by the host controller to detect errors. However, if the host experiences a hold-off that causes it to skip all of an siTD, or an siTD expires during a host hold off (for example, a hold-off occurs and the siTD is no longer reachable by the host controller in order for it to report the hold-off event), then system software must detect that the siTDs have not been processed by the host controller (for example, state not advanced) and report the appropriate error to the client driver.

20.6.12.3.3 Split Transaction Execution State Machine for Isochronous

In this section, all references to micro-frame are in the context of a micro-frame within an H-Frame.

If the Active bit in the Status byte is a zero, the host controller ignores the siTD and continues traversing the periodic schedule. Otherwise the host controller processes the siTD as specified below. A split transaction state machine is used to manage the split-transaction protocol sequence. The host controller uses the fields defined in Section 20.6.12.3.2, “Tracking Split Transaction Progress for Isochronous Transfers,” plus the variable cMicroFrameBit defined in Section 20.6.12.2.5, “Split Transaction Execution State Machine for Interrupt,” to track the progress of an isochronous split transaction. Figure 20-59 illustrates the state machine for managing an siTD through an isochronous split transaction. Bold, dotted circles denote the state of the Active bit in the Status field of a siTD. The Bold, dotted arcs denote the transitions between these states. Solid circles denote the states of the split transaction state machine and the solid arcs denote the transitions between these states. Dotted arcs and boxes reference actions that take place either as a result of a transition or from being in a state.

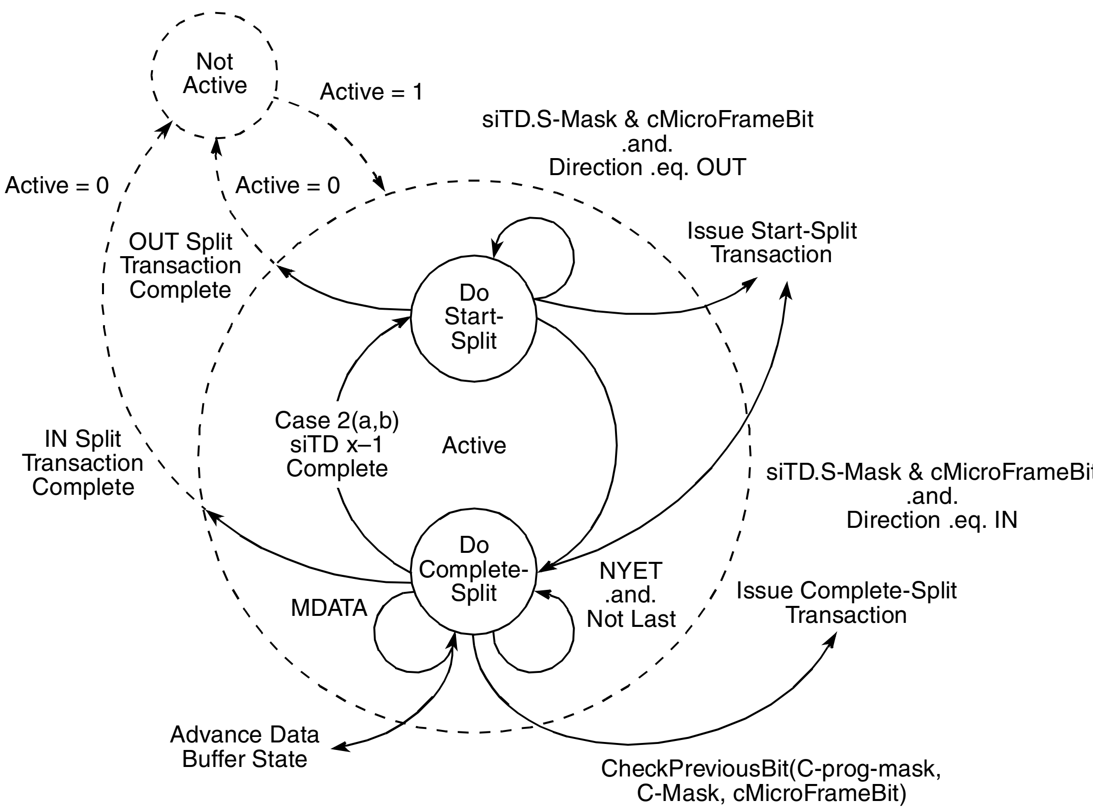


Figure 20-59. Split Transaction State Machine for Isochronous

20.6.12.3.4 Periodic Isochronous—Do-Start-Split

Isochronous split transaction OUTs use only this state. An siTD for a split-transaction isochronous IN is either initialized to this state, or the siTD transitions to this state from Do Complete Split when a case 2a (IN) or 2b scheduling boundary isochronous split-transaction completes.

Each time the host controller reaches an active siTD in this state, it checks the siTD[S-mask] against cMicroFrameBit. If there is a one in the appropriate position, the siTD executes a start-split transaction.

By definition, the host controller cannot reach an siTD at the wrong time. If the I/O field indicates an IN, then the start-split transaction includes only the extended token plus the full-speed token. Software must initialize the siTD[Total Bytes To Transfer] field to the number of bytes expected. This is usually the maximum packet size for the full-speed endpoint. The host controller exits this state when the start-split transaction is complete.

The remainder of this section is specific to an isochronous OUT endpoint (that is, the I/O field indicates an OUT). When the host controller executes a start-split transaction for an isochronous OUT it includes a data payload in the start-split transaction. The memory buffer address for the data payload is constructed by concatenating siTD[Current Offset] with the page pointer indicated by the page select field (siTD[P]). A zero in this field selects Page 0 and a 1 selects Page 1. During the start-split for an OUT, if the data transfer crosses a page boundary during the transaction, the host controller must detect the page cross, update the siTD[P] bit from a zero to a one, and begin using the siTD Page 1 with siTD[Current Offset] as the memory address pointer. The field siTD[TP] is used to annotate each start-split transaction with the indication of which part of the split-transaction data the current payload represents (ALL, BEGIN, MID, END). In all cases, the host controller simply uses the value in siTD[TP] to mark the start-split with the correct transaction position code.

T-count is always initialized to the number of start-splits for the current frame. TP is always initialized to the first required transaction position identifier. The scheduling boundary case (see Figure 20-58) is used to determine the initial value of TP. The initial cases are summarized in Table 20-70.

Table 20-70. Initial Conditions for OUT siTD TP and T-Count Fields

Case	T-Count	TP	Description
1, 2a	=1	ALL	When the OUT data payload is less than (or equal to) 188 bytes, only one start-split is required to move the data. The one start-split must be marked with an ALL.
1, 2a	!=1	BEGIN	When the OUT data payload is greater than 188 bytes more than one start-split must be used to move the data. The initial start-split must be marked with a BEGIN.

After each start-split transaction is complete, the host controller updates T-count and TP appropriately so that the next start-split is correctly annotated. Table 20-71 illustrates all of the TP and T-count transitions, which must be accomplished by the host controller.

Table 20-71. Transaction Position (TP)/Transaction Count (T-Count) Transition Table

TP	T-Count Next	TP Next	Description
ALL	0	N/A	Transition from ALL, to done.
BEGIN	1	END	Transition from BEGIN to END. Occurs when T-count starts at 2.
BEGIN	!=1	MID	Transition from BEGIN to MID. Occurs when T-count starts at greater than 2.
MID	!=1	MID	TP stays at MID while T-count is not equal to 1 (for example, greater than 1). This case can occur for any of the scheduling boundary cases where the T-count starts greater than 3.
MID	1	END	Transition from MID to END. This case can occur for any of the scheduling boundary cases where the T-count starts greater than 2.

The start-split transactions do not receive a handshake from the transaction translator, so the host controller always advances the transfer state in the siTD after the bus transaction is complete. To advance the transfer state the following operations take place:

- The siTD[Total Bytes To Transfer] and the siTD[Current Offset] fields are adjusted to reflect the number of bytes transferred.
- The siTD[P] (page select) bit is updated appropriately.
- The siTD[TP] and siTD[T-count] fields are updated appropriately as defined in Table 20-71.

These fields are then written back to the memory based siTD. The S-mask is fixed for the life of the current budget. As mentioned above, TP and T-count are set specifically in each siTD to reflect the data to be sent from this siTD. Therefore, regardless of the value of S-mask, the actual number of start-split transactions depends on T-count (or equivalently, Total Bytes to Transfer). The host controller must clear the Active bit when it detects that all of the schedule data has been sent to the bus. The preferred method is to detect when T-Count decrements to zero as a result of a start-split bus transaction. Equivalently, the host controller can detect when Total Bytes to Transfer decrements to zero. Either implementation must ensure that if the initial condition is Total Bytes to Transfer is equal to zero and T-count is equal to a one, then the host controller will issue a single start-split, with a zero-length data payload. Software must ensure that TP, T-count and Total Bytes to Transfer are set to deliver the appropriate number of bus transactions from each siTD. An inconsistent combination will yield undefined behavior.

If the host experiences hold-offs that cause the host controller to skip start-split transactions for an OUT transfer, the state of the transfer will not progress appropriately. The transaction translator observes protocol violations in the arrival of the start-splits for the OUT endpoint (that is, the transaction position annotation is incorrect as received by the transaction translator).

Example scenarios are described in Section 20.6.12.3.7, “Split Transaction for Isochronous—Processing Example.”

The host controller can optionally track the progress of an OUT split transaction by setting appropriate bits in the siTD[C-prog-mask] as it executes each scheduled start-split. The checkPreviousBit() algorithm defined in Section 20.6.12.3.5, “Periodic Isochronous—Do Complete Split,” can be used prior to executing each start-split to determine whether start-splits were skipped. The host controller can use this mechanism to detect missed micro-frames. It can then clear the siTD's Active bit and stop execution of this siTD. This saves on both memory and high-speed bus bandwidth.

20.6.12.3.5 Periodic Isochronous—Do Complete Split

This state is only used by a split-transaction isochronous IN endpoint. This state is entered unconditionally from the Do Start State after a start-split transaction is executed for an IN endpoint. Each time the host controller visits an siTD in this state, it conducts a number of tests to determine whether it should execute a complete-split transaction. The individual tests are listed below. The sequence they are applied depends on which micro-frame the host controller is currently executing which means that the tests might not be applied until after the siTD referenced from the back pointer has been fetched.

- Test A. cMicroFrameBit is bit-wise ANDed with the siTD[C-mask] field. A non-zero result indicates that software scheduled a complete-split for this endpoint, during this micro-frame. This test is always applied to a newly fetched siTD that is in this state.

- Test B. The siTD[C-prog-mask] bit vector is checked to determine whether the previous complete splits have been executed. An example algorithm is given below (this is slightly different than the algorithm used in Section 20.6.12.2.7, “Periodic Interrupt—Do-Complete-Split”). The sequence in which this test is applied depends on the current value of FRINDEX[2–0]. If FRINDEX[2–0] is 0 or 1, it is not applied until the back pointer has been used. Otherwise it is applied immediately.

```
Algorithm Boolean CheckPreviousBit(siTD.C-prog-mask, siTD.C-mask, cMicroFrameBit)
Begin
     Boolean rvalue = TRUE;
     previousBit = cMicroFrameBit rotate-right(1)
     -- Bit-wise anding previousBit with C-mask indicates whether there
     -- was an intent to send a complete split in the previous micro-
     -- frame. So, if the 'previous bit' is set in C-mask, check
     -- C-prog-mask to make sure it happened.
     if previousBit bitAND siTD.C-mask then
          if not (previousBit bitAND siTD.C-prog-mask) then
                                                                      rvalue = FALSE
          End if
     End if
     Return rvalue
End Algorithm
```

If Test A is true and FRINDEX[2–0] is zero or one, then this is a case 2a or 2b scheduling boundary (see Figure 20-57). See Section 20.6.12.3.6, “Complete-Split for Scheduling Boundary Cases 2a, 2b,” for details in handling this condition.

If Test A and Test B evaluate to true, then the host controller executes a complete-split transaction using the transfer state of the current siTD. When the host controller commits to executing the complete-split transaction, it updates QH[C-prog-mask] by bit-ORing with cMicroFrameBit. The transfer state is advanced based on the completion status of the complete-split transaction. To advance the transfer state of an IN siTD, the host controller must:

- Decrement the number of bytes received from siTD[Total Bytes To Transfer]
- Adjust siTD[Current Offset] by the number of bytes received
- Adjust the siTD[P] (page select) field if the transfer caused the host controller to use the next page pointer
- Set any appropriate bits in the siTD[Status] field, depending on the results of the transaction.

Note that if the host controller encounters a condition where siTD[Total Bytes To Transfer] is zero, and it receives more data, the host controller must not write the additional data to memory. The siTD[Status-Active] bit must be cleared and the siTD[Status-Babble Detected] bit must be set. The fields siTD[Total Bytes To Transfer], siTD[Current Offset], and siTD[P] are not required to be updated as a result of this transaction attempt.

The host controller accepts (assuming good data packet CRC and sufficient room in the buffer as indicated by the value of siTD[Total Bytes To Transfer]) MDATA and DATA0/1 data payloads up to and including 192 bytes. The host controller may optionally clear siTD[Status-Active] and set siTD[Status-Babble

Detected] when it receives MDATA or DATA0/1 with a data payload of more than 192 bytes. The following responses have the noted effects:

- ERR. The full-speed transaction completed with a time-out or bad CRC and this is a reflection of that error to the host. The host controller sets the ERR bit in the siTD[Status] field and clears the Active bit.
- Transaction Error (XactErr). The complete-split transaction encounters a Timeout, CRC16 failure, etc. The siTD[Status] field XactErr field is set and the complete-split transaction must be retried immediately. The host controller must use an internal error counter to count the number of retries as a counter field is not provided in the siTD data structure. The host controller will not retry more than two times. If the host controller exhausts the retries or the end of the micro-frame occurs, the Active bit is cleared.
- DATAx (0 or 1). This response signals that the final data for the split transaction has arrived. The transfer state of the siTD is advanced and the Active bit is cleared. If the Bytes To Transfer field has not decremented to zero (including the reception of the data payload in the DATAx response), then less data than was expected, or allowed for was actually received. This short packet event does not set the USB interrupt status bit (USBSTS[UI]) to a one. The host controller will not detect this condition.
- NYET (and Last). On each NYET response, the host controller also checks to determine whether this is the last complete-split for this split transaction. Last was defined in Section Periodic Interrupt - Do Complete Split. If it is the last complete-split (with a NYET response), then the transfer state of the siTD is not advanced (never received any data) and the Active bit is cleared. No bits are set in the Status field because this is essentially a skipped transaction. The transaction translator must have responded to all the scheduled complete-splits with NYETs, meaning that the start-split issued by the host controller was not received. This result should be interpreted by system software as if the transaction was completely skipped. The test for whether this is the last complete split can be performed by XORing C-mask with C-prog-mask. A zero result indicates that all complete-splits have been executed.
- MDATA (and Last). See above description for testing for Last. This can only occur when there is an error condition. Either there has been a babble condition on the full-speed link, which delayed the completion of the full-speed transaction, or software set up the S-mask and/or C-masks incorrectly. The host controller must set the XactErr bit and clear the Active bit.
- NYET (and not Last). See above description for testing for Last. The complete-split transaction received a NYET response from the transaction translator. Do not update any transfer state (except for C-prog-mask) and stay in this state.
- MDATA (and not Last). The transaction translator responds with an MDATA when it has partial data for the split transaction. For example, the full-speed transaction data payload spans from micro-frame X to X+1 and during micro-frame X, the transaction translator responds with an MDATA and the data accumulated up to the end of micro-frame X. The host controller advances the transfer state to reflect the number of bytes received.

If Test A succeeds, but Test B fails, it means that one or more of the complete-splits have been skipped. The host controller sets the Missed Micro-Frame status bit and clears the Active bit.

20.6.12.3.6 Complete-Split for Scheduling Boundary Cases 2a, 2b

Boundary cases 2a and 2b (INs only) (see Figure 20-57) require that the host controller use the transaction state context of the previous siTD to finish the split transaction. Table 20-72 enumerates the transaction state fields.

Table 20-72. Summary siTD Split Transaction State

Buffer State	Status	Execution Progress
Total Bytes To Transfer P (page select) Current Offset TP (transaction position) T-count (transaction count)	All bits in the status field	C-prog-mask

NOTE

TP and T-count are used only for Host to Device (OUT) endpoints.

If software has budgeted the schedule of this data stream with a frame wrap case, then it must initialize the siTD[Back Pointer] field to reference a valid siTD and have the T bit in the siTD[Back Pointer] field cleared. Otherwise, software must set the T bit in siTD[Back Pointer]. The host controller's rules for interpreting when to use the siTD[Back Pointer] field are listed below. These rules apply only when the siTD's Active bit is a one and the SplitXState is Do Complete Split.

- When cMicroFrameBit is a 0x1 and the $siTD_X$[Back Pointer] T-bit is zero, or
- If cMicroFrameBit is a 0x2 and siTDX[S-mask[0]] is zero

When either of these conditions apply, then the host controller must use the transaction state from $siTD_{X-1}$.

In order to access $siTD_{X-1}$, the host controller reads on-chip the siTD referenced from $siTD_X$[Back Pointer].

The host controller must save the entire state from $siTD_X$ while processing $siTD_{X-1}$. This is to accommodate for case 2b processing. The host controller must not recursively walk the list of siTD[Back Pointers].

If $siTD_{X-1}$ is active (Active bit is set and SplitXStat is Do Complete Split), then both Test A and Test B are applied as described above. If these criteria to execute a complete-split are met, the host controller executes the complete split and evaluates the results as described above. The transaction state (see Table 20-72) of $siTD_{X-1}$ is appropriately advanced based on the results and written back to memory. If the resultant state of $siTD_{X-1}$'s Active bit is a one, then the host controller returns to the context of $siTD_X$, and follows its next pointer to the next schedule item. No updates to $siTD_X$ are necessary.

If $siTD_{X-1}$ is active (Active bit is set and SplitXStat is Do Start Split), then the host controller must clear the Active bit and set the Missed Micro-Frame status bit and the resultant status is written back to memory.

If $siTD_{X-1}$'s Active bit is cleared, (because it was cleared when the host controller first visited $siTD_{X-1}$ via $siTD_X$'s back pointer, it transitioned to zero as a result of a detected error, or the results of $siTD_{X-1}$'s complete-split transaction cleared it), then the host controller returns to the context of $siTD_X$ and transitions its SplitXState to Do Start Split. The host controller then determines whether the case 2b start split boundary condition exists (that is, if cMicroframeBit is 1 and $siTD_X$[S-mask[0]] is 1). If this criterion is met the host controller immediately executes a start-split transaction and appropriately advances the transaction state of $siTD_X$, then follows $siTD_X$[Next Pointer] to the next schedule item. If the criterion is not met, the host controller simply follows $siTD_X$[Next Pointer] to the next schedule item. Note that in the case of a 2b boundary case, the split-transaction of $siTD_{X-1}$ will have its Active bit cleared when the host controller returns to the context of $siTD_X$. Also, note that software should not initialize an siTD with C-mask bits 0 and 1 set and an S-mask with bit 0 set. This scheduling combination is not supported and the behavior of the host controller is undefined.

20.6.12.3.7 Split Transaction for Isochronous—Processing Example

There is an important difference between how the hardware/software manages the isochronous split transaction state machine and how it manages the asynchronous and interrupt split transaction state machines. The asynchronous and interrupt split transaction state machines are encapsulated within a single queue head. The progress of the data stream depends on the progress of each split transaction. In some respects, the split-transaction state machine is sequenced using the Execute Transaction queue head traversal state machine.

Isochronous is a pure time-oriented transaction/data stream. The interface data structures are optimized to efficiently describe transactions that need to occur at specific times. The isochronous split-transaction state machine must be managed across these time-oriented data structures. This means that system software must correctly describe the scheduling of split-transactions across more than one data structure.

Then the host controller must make the appropriate state transitions at the appropriate times, in the correct data structures.

For example, Table 20-73 illustrates a few frames worth of scheduling required to schedule a case 2a full-speed isochronous data stream.

Table 20-73. Example Case 2a—Software Scheduling siTDs for an IN Endpoint

siTDX		Micro-Frames								InitialSplitXState
#	Masks	0	1	2	3	4	5	6	7	
X	S-Mask					1				Do Start Split
	C-Mask	1	1					1	1	
X+1	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+2	S-Mask					1				Do Complete Split
	C-Mask	1	1					1	1	
X+3	S-Mask	Repeats previous pattern								Do Complete Split
	C-Mask									

This example shows the first three siTDs for the transaction stream. Since this is the case-2a frame-wrap case, S-masks of all siTDs for this endpoint have a value of 0x10 (a one bit in micro-frame 4) and C-mask value of 0xC3 (one-bits in micro-frames 0,1, 6 and 7). Additionally, software ensures that the Back Pointer field of each siTD references the appropriate siTD data structure (and the Back Pointer T-bits are cleared).

The initial SplitXState of the first siTD is Do Start Split. The host controller will visit the first siTD eight times during frame X. The C-mask bits in micro-frames 0 and 1 are ignored because the state is Do Start Split. During micro-frame 4, the host controller determines that it can run a start-split (and does) and changes SplitXState to Do Complete Split. During micro-frames 6 and 7, the host controller executes complete-splits. Notice the siTD for frame X+1 has it's SplitXState initialized to Do Complete Split. As the host controller continues to traverse the schedule during H-Frame X+1, it will visit the second siTD eight times. During micro-frames 0 and 1 it will detect that it must execute complete-splits.

During H-Frame X+1, micro-frame 0, the host controller detects that $siTD_{X+1}$'s Back Pointer[T] bit is a zero, saves the state of $siTD_{X+1}$ and fetches $siTD_X$. It executes the complete split transaction using the transaction state of $siTD_X$. If the $siTD_X$ split transaction is complete, siTD's Active bit is cleared and results written back to $siTD_X$. The host controller retains the fact that $siTD_X$ is retired and transitions the SplitXState in $siTD_{X+1}$ to Do Start Split. At this point, the host controller is prepared to execute the start-split for $siTD_{X+1}$ when it reaches micro-frame 4. If the split-transaction completes early (transaction-complete is defined in Section 20.6.12.3.5, “Periodic Isochronous—Do Complete Split”), that is, before all the scheduled complete-splits have been executed, the host controller changes $siTD_X$[SplitXState] to Do Start Split early and naturally skips the remaining scheduled complete-split transactions. For this example, $siTD_{X+1}$ does not receive a DATA0 response until H-Frame X+2, micro-frame 1.

During H-Frame X+2, micro-frame 0, the host controller detects that $siTD_{X+2}$'s Back Pointer[T] bit is zero, saves the state of $siTD_{X+2}$ and fetches $siTD_{X+1}$. As described above, it executes another split transaction, receives an MDATA response, updates the transfer state, but does not modify the Active bit. The host controller returns to the context of $siTD_{X+2}$, and traverses it's next pointer without any state change updates to $siTD_{X+2}$.

During H-Frame X+2, micro-frame 1, the host controller detects $siTD_{X+2}$'s S-mask[0] bit is zero, saves the state of $siTD_{X+2}$ and fetches $siTD_{X+1}$. It executes another complete-split transaction, receives a DATA0 response, updates the transfer state and clears the Active bit. It returns to the state of $siTD_{X+2}$ and changes its SplitXState to Do Start Split. At this point, the host controller is prepared to execute start-splits for $siTD_{X+2}$ when it reaches micro-frame 4.

20.6.13 Port Test Modes

EHCI host controllers implement the port test modes Test J_State, Test K_State, Test_Packet, Test Force_Enable, and Test SE0_NAK as described in the *USB Specification Revision 2.0*. The required, port test sequence is (assuming the CF-bit in the CONFIGFLAG register is set):

- Disable the periodic and asynchronous schedules by clearing the USBCMD[ASE] and USBCMD[PSE].
- Place all enabled root ports into the suspended state by setting the Suspend bit in the PORTSC register (PORTSC[SUSP]).
- Clear USBCMD[RS] (run/stop) and wait for USBSTS[HCH] to transition to a one. Note that an EHCI host controller implementation may optionally allow port testing with RS set. However, all host controllers must support port testing with RS cleared and HCH set.

- Set the Port Test Control field in the port under test PORTSC register to the value corresponding to the desired test mode. If the selected test is Test_Force_Enable, then USBCMD[RS] must then be transitioned back to one, in order to enable transmission of SOFs out of the port under test.
- When the test is complete, system software must ensure the host controller is halted (HCH bit is a one) then it terminates and exits test mode by setting USBCMD[RST].

20.6.14 Interrupts

The EHCI host controller hardware provides interrupt capability based on a number of sources. There are several general groups of interrupt sources:

- Interrupts as a result of executing transactions from the schedule (success and error conditions),
- Host controller events (such as Port change events)
- Host controller error events

All transaction-based sources are maskable through the host controller's interrupt enable register (usbintr). Additionally, individual transfer descriptors can be marked to generate an interrupt on completion. This section describes each interrupt source and the processing that occurs in response to the interrupt.

During normal operation, interrupts may be immediate or deferred until the next interrupt threshold occurs. The interrupt threshold is a tunable parameter by means of the interrupt threshold control field in the USBCMD register. The value of this register controls when the host controller generates an interrupt on behalf of normal transaction execution. When a transaction completes during an interrupt interval period, the interrupt signaling the completion of the transfer will not occur until the interrupt threshold occurs. For example, the default value is eight micro-frames. This means that the host controller will not generate interrupts any more frequently than once every eight micro-frames.

Section 20.6.14.2.4, "Host System Error," details the effects of a host system error.

If an interrupt is scheduled to be generated for the current interrupt threshold interval, the interrupt is not signaled until after the status for the last complete transaction in the interval is written back to system memory. This sometimes results in the interrupt not being signaled until the next interrupt threshold.

Initial interrupt processing is the same, regardless of the reason for the interrupt. When an interrupt is signaled by the hardware, CPU control is transferred to host controller's USB interrupt handler. The precise mechanism to accomplish the transfer is OS specific. For this discussion it is just assumed that control is received. When the interrupt handler receives control, its first action is to reads the USBSTS. It then acknowledges the interrupt by clearing all of the interrupt status bits by writing ones to these bit positions. The handler then determines whether the interrupt is due to schedule processing or some other event. After acknowledging the interrupt, the handler (by means of an OS-specific mechanism), schedules a deferred procedure call (DPC) which will execute later. The DPC routine processes the results of the schedule execution. The precise mechanisms used are beyond the scope of this document.

NOTE

The only method software should use for acknowledging an interrupt is by transitioning the appropriate status bits in the USBSTS register from a one to a zero.

20.6.14.1 Transfer/Transaction Based Interrupts

These interrupt sources are associated with transfer and transaction progress. They are all dependent on the next interrupt threshold.

20.6.14.1.1 Transaction Error

A transaction error is any error that caused the host controller to think that the transfer did not complete successfully. Table 20-74 lists the events/responses that the host can observe as a result of a transaction. The effects of the error counter and interrupt status are summarized in the following paragraphs. Most of these errors set the XactErr status bit in the appropriate interface data structure.

Table 20-74. Summary of Transaction Errors

Event/ Result	Queue Head/qTD/iTD/siTD Side Effects		USBSTS[USBERRINT]
	Cerr	Status Field	
CRC	–1	XactErr set	1[1]
Timeout	–1	XactErr set	1[1]
Bad PID[2]	–1	XactErr set	1[1]
Babble	N/A	See Section 20.6.14.1.2, “Serial Bus Babble”	1
Buffer Error	N/A	See Section 20.6.14.1.3, “Data Buffer Error”	

1 If occurs in a queue head, then USBERRINT is asserted only when Cerr counts down from a one to a zero. In addition the queue is halted.

2 The host controller received a response from the device, but it could not recognize the PID as a valid PID.

There is a small set of protocol errors that relate only when executing a queue head and fit under the umbrella of a WRONG PID error that are significant to explicitly identify. When these errors occur, the XactErr status bit in the queue head is set and the Cerr field is decremented. When the PID Code indicates a SETUP, the following responses are protocol errors and result in XactErr bit being set and the Cerr field being decremented.

- EPS field indicates a high-speed device, and it returns a Nak handshake to a SETUP.
- EPS field indicates a high-speed device, and it returns a Nyet handshake to a SETUP.
- EPS field indicates a low- or full-speed device, and the complete-split receives a Nak handshake.

20.6.14.1.2 Serial Bus Babble

When a device transmits more data on the USB than the host controller is expecting for this transaction, it is defined to be babbling. In general, this is called a packet babble. When a device sends more data than the maximum length number of bytes, the host controller sets the babble detected bit to a one and halts the endpoint if it is using a queue head. Maximum length is defined as the minimum of total bytes to transfer and maximum packet size. The Cerr field is not decremented for a packet babble condition (only applies to queue heads).

A babble condition called a frame babble also exists if IN transaction is in progress at High-speed EOF2 point. A frame babble condition is recorded into the appropriate schedule data structure. In addition, the host controller must disable the port to which the frame babble is detected.

If USBSTS[UEI] (USB error interrupt) and the USBINTR[UEE] (USB error interrupt enable) are set, a hardware interrupt is signaled to the system at the next interrupt threshold. The host controller must never start an OUT transaction that babbles across a micro-frame EOF.

NOTE

When a host controller detects a data PID mismatch, it must either disable the packet babble checking for the duration of the bus transaction or do packet babble checking based solely on Maximum Packet Size. The USB core specification defines the requirements on a data receiver when it receives a data PID mismatch (for example, expects a DATA0 and gets a DATA1 or visa-versa). In summary, it must ignore the received data and respond with an ACK handshake, in order to advance the transmitter's data sequence.The EHCI interface allows system software to provide buffers for a control, bulk, or interrupt IN endpoint that are not an even multiple of the maximum packet size specified by the device. Whenever a device misses an ACK for an IN endpoint, the host and device are out of synchronization with respect to the progress of the data transfer. The host controller may have advanced the transfer to a buffer that is less than maximum packet size. The device re-sends its maximum packet size data packet, with the original data PID, in response to the next IN token. In order to properly manage the bus protocol, the host controller must disable the packet babble check when it observes the data PID mismatch.

20.6.14.1.3 Data Buffer Error

This event indicates that an overrun of incoming data or a underrun of outgoing data has occurred for this transaction. This would generally be caused by the host controller not being able to access required data buffers in memory within necessary latency requirements. These conditions are not considered transaction errors, and do not effect the error count in the queue head. When these errors do occur, the host controller records the fact the error occurred by setting the data buffer error bit in the queue head, iTD or siTD.

If the data buffer error occurs on a non-isochronous IN, the host controller will not issue a handshake to the endpoint. This forces the endpoint to resend the same data (and data toggle) in response to the next IN to the endpoint.

If the data buffer error occurs on an OUT, the host controller must corrupt the end of the packet so that it cannot be interpreted by the device as a good data packet. Simply truncating the packet is not considered acceptable. An acceptable implementation option is to 1's complement the CRC bytes and send them. There are other options suggested in the transaction translator section of the *USB Specification Revision 2.0*.

20.6.14.1.4 USB Interrupt (Interrupt on Completion (IOC))

Transfer Descriptors (iTDs, siTDs, and queue heads (qTDs)) contain a bit that can be set to cause an interrupt on their completion. The completion of the transfer associated with that schedule item causes USBSTS[UI] (USB interrupt) to be set. In addition, if a short packet is encountered on an IN transaction associated with a queue head, then this event also causes USBINT to be set. If USBINTR[UE] (USB interrupt enable) is set, a hardware interrupt is signaled to the system at the next interrupt threshold. If the completion is because of errors, USBSTS[UEI] (USB error interrupt) is also set.

20.6.14.1.5 Short Packet

Reception of a data packet that is less than the endpoint's Max Packet size during Control, Bulk or Interrupt transfers signals the completion of the transfer. Whenever a short packet completion occurs during a queue head execution, USBSTS[UI] (USB interrupt bit) is set. If the USB interrupt enable bit is set (USBINTR[UE]), a hardware interrupt is signaled to the system at the next interrupt threshold.

20.6.14.2 Host Controller Event Interrupts

These interrupt sources are independent of the interrupt threshold (with the one exception being the Interrupt on Async Advance.

20.6.14.2.1 Port Change Events

Port registers contain status and status change bits. When the status change bits are set, the host controller sets the USBSTS[PCI]. If the port change interrupt enable bit (PCE) in the USBINTR register is set, the host controller issues a hardware interrupt. The port status change bits in PORTSC include:

- Connect change status (CSC)
- Port enable/disable change (PEC)
- Over-current change (OCC)
- Force port resume (FPR)

20.6.14.2.2 Frame List Rollover

This event indicates that the host controller has wrapped the frame list. The current programmed size of the frame list effects how often this interrupt occurs. If the frame list size is 1024, then the interrupt occurs every 1024 milliseconds, if it is 512, then it occurs every 512 milliseconds, etc. When a frame list rollover is detected, the host controller sets the frame list rollover bit, USBSTS[FRI]. If USBINTR[FRE] is set (frame list rollover enable), the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

20.6.14.2.3 Interrupt on Async Advance

This event is used for deterministic removal of queue heads from the asynchronous schedule. Whenever the host controller advances the on-chip context of the asynchronous schedule, it evaluates the value of USBCMD[IAA]. If it is set, it sets USBSTS[AAI]. If USBINTR[AAE] is set, the host controller issues a hardware interrupt at the next interrupt threshold. A detailed explanation of this feature is described in Section 20.6.9.2, “Removing Queue Heads from Asynchronous Schedule.”

20.6.14.2.4 Host System Error

The host controller is a bus master and any interaction between the host controller and the system may experience errors. The type of host error may be catastrophic to the host controller making it impossible for the host controller to continue in a coherent fashion. Behavior for these types of errors is to halt the host controller. Host-based error must result in the following actions:

- USBCMD[RS] is cleared.
- USBSTS[SEI] and USBSTS[HCH] register are set
- If the host system error enable bit, USBINTR[SEE] is set, the host controller issues a hardware interrupt. This interrupt is not delayed to the next interrupt threshold.

Table 20-75 summarizes the required actions taken on the various host errors.

Table 20-75. Summary Behavior on Host System Errors

Cycle Type	Master Abort	Target Abort	Data Phase Parity
Frame list pointer fetch (read)	Fatal	Fatal	Fatal
siTD fetch (read)	Fatal	Fatal	Fatal
siTD status write-back (write)	Fatal	Fatal	Fatal
iTD fetch (read)	Fatal	Fatal	Fatal
iTD status write-back (write)	Fatal	Fatal	Fatal
qTD fetch (read)	Fatal	Fatal	Fatal
qHD status write-back (write)	Fatal	Fatal	Fatal
Data write	Fatal	Fatal	Fatal
Data read	Fatal	Fatal	Fatal

NOTE

After a host system error, software must reset the host controller using USBCMD[RST] before re-initializing and restarting the host controller.

20.7 Device Data Structures

This section defines the interface data structures used to communicate control, status, and data between device controller driver (DCD) software and the device controller. The data structure definitions in this chapter support a 32-bit memory buffer address space. The interface consists of device queue heads and transfer descriptors.

NOTE

Software must ensure that no interface data structure reachable by the device controller spans a 4K-page boundary.

The data structures defined in the section are (from the device controller's perspective) a mix of read-only and read/writable fields. The device controller must preserve the read-only fields on all data structure writes.

The USB DR module includes DCD software called the USB 2.0 Device API. The device API provides an easy to use Application Program Interface for developing device (peripheral) applications. The device API incorporates and abstracts for the application developer all of the elements of the program interface.

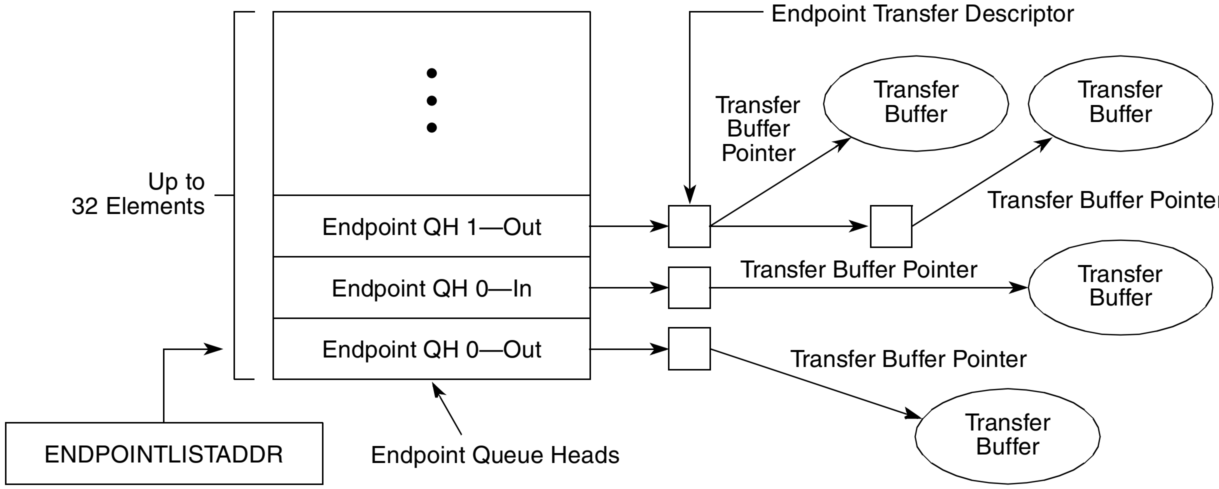


Figure 20-60. Endpoint Queue Head Organization

20.7.1 Endpoint Queue Head

The device Endpoint Queue Head (dQH) is where all transfers are managed. The dQH is a 48-byte data structure, but must be aligned on 64-byte boundaries. During priming of an endpoint, the dTD (device transfer descriptor) is copied into the overlay area of the dQH, which starts at the nextTD pointer DWord and continues through the end of the buffer pointers DWords. After a transfer is complete, the dTD status DWord is updated in the dTD pointed to by the currentTD pointer. While a packet is in progress, the overlay area of the dQH is used as a staging area for the dTD so that the Device Controller can access needed information with little minimal latency.

Figure 20-61 shows the Endpoint Queue Head structure.

Bits 31–0	offset
Mult (31–30) | zlt (29) | 00 (28–27) | Maximum Packet Length (26–16) | ios (15) | 000_0000_0000_0000 (14–0)	0x00
Current dTD Pointer[1] | 0_0000	0x04
Next dTD Pointer[1] | 0000 | T[1]	0x08[2]
0 | Total Bytes[1] | ioc[1] | 000 | MultO[1] | 00 | Status[1]	0x0C[2]
Buffer Pointer (Page 0)[1] | Current Offset[1]	0x10[2]
Buffer Pointer (Page 1)[1] | Reserved	0x14[2]
Buffer Pointer (Page 2)[1] | Reserved	0x18[2]
Buffer Pointer (Page 3)[1] | Reserved	0x1C[2]
Buffer Pointer (Page 4)[1] | Reserved	0x20[2]
Reserved	0x24
Setup Buffer Bytes 3–0[1]	0x28
Setup Buffer Bytes 7–4[1]	0x2C

Figure 20-61. Endpoint Queue Head Layout

[1] Device controller read/write; all others read-only.

[2] Offsets 0x08 through 0x20 contain the transfer overlay.

20.7.1.1 Endpoint Capabilities/Characteristics

This DWord specifies static information about the endpoint, in other words, this information does not change over the lifetime of the endpoint. Device Controller software should not attempt to modify this information while the corresponding endpoint is enabled.

Table 20-76. Endpoint Capabilities/Characteristics

Bits	Name	Description
31–30	Mult	Mult. This field is used to indicate the number of packets executed per transaction description as given by the following: 00 - Execute N Transactions as demonstrated by the USB variable length packet protocol where N is computed using the Maximum Packet Length (dQH) and the Total Bytes field (dTD) 01 Execute 1 Transaction. 10 Execute 2 Transactions. 11 Execute 3 Transactions. **Note:** Non-ISO endpoints must set Mult = 00. **Note:** ISO endpoints must set Mult = 01, 10, or 11 as needed.
29	zlt	Zero length termination select. This bit is used to indicate when a zero length packet is used to terminate transfers where to total transfer length is a multiple. This bit is not relevant for Isochronous transfers. 0 Enable zero length packet to terminate transfers equal to a multiple of the Maximum Packet Length. (default). 1 Disable the zero length packet on transfers that are equal in length to a multiple Maximum Packet Length.
28–27	—	Reserved, should be cleared. These bit reserved for future use and should be cleared.

Table 20-76. Endpoint Capabilities/Characteristics (continued)

Bits	Name	Description
26–16	Maximum Packet Length	Maximum packet length. This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
15	ios	Interrupt on setup (IOS). This bit is used on control type endpoints to indicate if USBINT is set in response to a setup being received.
14–0		Reserved, should be cleared. Bits reserved for future use and should be cleared.

20.7.1.2 Transfer Overlay

The seven DWords in the overlay area represent a transaction working space for the device controller. The general operational model is that the device controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, then it will not read the associated endpoint.

After an endpoint is readied, the dTD will be copied into this queue head overlay area by the device controller. Until a transfer is expired, software must not write the queue head overlay area or the associated transfer descriptor. When the transfer is complete, the device controller will write the results back to the original transfer descriptor and advance the queue.

See dTD for a description of the overlay fields.

20.7.1.3 Current dTD Pointer

The current dTD pointer is used by the device controller to locate the transfer in progress. This word is for USB_DR (hardware) use only and should not be modified by DCD software.

Table 20-77. Current dTD Pointer

Bits	Description
31–5	Current dtd. This field is a pointer to the dTD that is represented in the transfer overlay area. This field will be modified by the Device Controller to next dTD pointer during endpoint priming or queue advance.
4–0	Reserved, should be cleared. Bit reserved for future use and should be cleared.

20.7.1.4 Setup Buffer

The setup buffer is dedicated storage for the 8-byte data that follows a setup PID.

NOTE

Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head is used for receiving setup data packets.

Table 20-78. Multiple Mode Control

DWord	Bits	Description
1	31–0	Setup Buffer 0. This buffer contains bytes 3 to 0 of an incoming setup buffer packet and is written by the device controller to be read by software.
2	31–0	Setup Buffer 1. This buffer contains bytes 7 to 4 of an incoming setup buffer packet and is written by the device controller to be read by software.

20.7.2 Endpoint Transfer Descriptor (dTD)

The dTD describes to the device controller the location and quantity of data to be sent/received for given transfer. The DCD should not attempt to modify any field in an active dTD except the Next Link Pointer, which should only be modified as described in section Managing Transfers with Transfer Descriptors.

<table>
<tr><th>31</th><th>30–16</th><th>15</th><th>14–12</th><th>11–10</th><th>9–8</th><th>7–5</th><th>4–1</th><th>0</th><th>offset</th></tr>
<tr><td colspan="7">Next Link Pointer</td><td>0000</td><td>T</td><td>0x00</td></tr>
<tr><td>0</td><td>Total Bytes[1]</td><td>ioc</td><td>000</td><td>MultO</td><td>00</td><td colspan="3">Status[1]</td><td>0x04</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 0)</td><td colspan="5">Current Offset[1]</td><td>0x08</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 1)</td><td>0</td><td colspan="4">Frame Number[1]</td><td>0x0C</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 2)</td><td colspan="5">0000_0000_0000</td><td>0x10</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 3)</td><td colspan="5">0000_0000_0000</td><td>0x14</td></tr>
<tr><td colspan="4">Buffer Pointer (Page 4)</td><td colspan="5">0000_0000_0000</td><td>0x18</td></tr>
</table>

Figure 20-62. Endpoint Transfer Descriptor (dTD)

[1] Device controller read/write; all others read-only.

Table 20-79. Next dTD Pointer

Bits	Description
31–5	Next transfer element pointer. This field contains the physical memory address of the next dTD to be processed. The field corresponds to memory address signals [31:5], respectively.
4–1	Reserved, should be cleared. Bits reserved for future use and should be cleared.
0	Terminate (T). 1=pointer is invalid. 0=Pointer is valid (points to a valid Transfer Element Descriptor). This bit indicates to the Device Controller that there are no more valid entries in the queue.

Table 20-80. dTD Token

Bits	Description
31	Reserved, should be cleared. Bit reserved for future use and should be cleared.
30–16	Total Bytes. This field specifies the total number of bytes to be moved with this transfer descriptor. This field is decremented by the number of bytes actually moved during the transaction and only on the successful completion of the transaction. The maximum value software may store in the field is 5*4K(5000H). This is the maximum number of bytes 5 page pointers can access. Although it is possible to create a transfer up to 20K this assumes the 1st offset into the first page is 0. When the offset cannot be predetermined, crossing past the 5th page can be guaranteed by limiting the total bytes to 16K**. Therefore, the maximum recommended transfer is 16K(4000H). If the value of the field is zero when the host controller fetches this transfer descriptor (and the active bit is set), the device controller executes a zero-length transaction and retires the transfer descriptor. It is not a requirement for IN transfers that Total Bytes To Transfer be an even multiple of Maximum Packet Length. If software builds such a transfer descriptor for an IN transfer, the last transaction will always be less that Maximum Packet Length.
15	Interrupt On Complete (IOC). This bit is used to indicate if USBINT is to be set in response to device controller being finished with this dTD.
14–12	Reserved, should be cleared. Bits reserved for future use and should be cleared.
11–10	Multiplier Override (MultO). This field can be used for transmit ISO's (that is, ISO-IN) to override the multiplier in the QH. This field must be zero for all packet types that are not transmit-ISO. Example: if QH.multiplier = 3; Maximum packet size = 8; Total bytes = 15; MultiO = 0 [default] Three packets are sent: {Data2(8); Data1(7); Data0(0)} if QH.multiplier = 3; Maximum packet size = 8; Total bytes = 15; MultiO = 2 Two packets are sent: {Data1(8); Data0(7)} For maximal efficiency, software should compute MultO = greatest integer of (Total Bytes/Max. Packet Size) except for the case when Total bytes = 0; then MultO should be 1. **Note:** Non-ISO and non-TX endpoints must set MultO = 00.
9–8	Reserved, should be cleared. Bits reserved for future use and should be cleared.
7–0	Status. This field is used by the Device Controller to communicate individual command execution states back to the Device Controller software. This field contains the status of the last transaction performed on this qTD. The bit encodings are: **Bit Status Field Description** 7 Active 6 Halted 5 Data Buffer Error 3 Transaction Error 4,2,0 Reserved, should be cleared

Table 20-81. Buffer Pointer Page 0

Bits	Description
31–12	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers.
11–0	Current Offset. Offset into the 4kb buffer where the packet is to begin.

Table 20-82. Buffer Pointer Page 1

Bits	Description
31–12	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers.
11	Reserved
10–0	Frame Number. Written by the device controller to indicate the frame number in which a packet finishes. This is typically be used to correlate relative completion times of packets on an ISO endpoint.

Table 20-83. Buffer Pointer Pages 2–4

Bits	Description
31–12	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems will typically set the buffer pointers to a series of incrementing integers.
11–0	Reserved

20.8 Device Operational Model

The function of the device operation is to transfer a request in the memory image to and from the Universal Serial Bus. Using a set of linked list transfer descriptors, pointed to by a queue head, the device controller will perform the data transfers. The following sections explain the use of the device controller from the device controller driver (DCD) point-of-view and further describe how specific USB bus events relate to status changes in the device controller programmer's interface.

20.8.1 Device Controller Initialization

After hardware reset, the USB DR module is disabled until the run/stop bit (USBCMD[RS]) is set to a '1'. In the disabled state, the pull-up on the USB D+ is not active which prevents an attach event from occurring. At a minimum, it is necessary to have the queue heads setup for endpoint zero before the device attach occurs. Shortly after the device is enabled, a USB reset will occur followed by setup packet arriving at endpoint 0. A queue head must be prepared so that the device controller can store the incoming setup packet.

In order to initialize a device, the software should perform the following steps:

1. Set the controller mode to device mode. Optionally set USBMODE[SDIS] (streaming disable).

NOTE

Transitioning from host mode to device mode requires a device controller reset before modifying USBMODE.

2. Optionally modify the BURSTSIZE register.
3. Program PORTSC[PTS] if using a non-ULPI PHY.
4. Set CONTROL[USB_EN]
5. Allocate and initialize device queue heads in system memory Minimum: Initialize device queue heads 0 Tx and 0 Rx.

NOTE

All device queue heads must be initialized for control endpoints before the endpoint is enabled. Device queue heads for non-control endpoints must be initialized before the endpoint can be used.

For information on device queue heads, refer to Section 20.7, "Device Data Structures."

6. Configure the ENDPOINTLISTADDR pointer.

   For additional information on ENDPOINTLISTADDR, refer to the register table.

7. Enable the microprocessor interrupt associated with the USB DR module and optionally change setting of USBCMD[ITC].

   Recommended: enable all device interrupts including: USBINT, USBERRINT, Port Change Detect, USB Reset Received, DCSuspend.

   For a list of available interrupts refer to the USBINTR and the USBSTS register tables.

8. Set USBCMD[RS] to run mode.

   After the run bit is set, a device reset will occur. The DCD must monitor the reset event and set the DEVICEADDR register, set the ENDPTCTRLx registers, and adjust the software state as described in the Bus Reset section of the following Port State and Control section below.

NOTE

Endpoint 0 is designed as a control endpoint only and does not need to be configured using ENDPTCTRL0 register.

It is also not necessary to initially prime Endpoint 0 because the first packet received will always be a setup packet. The contents of the first setup packet will require a response in accordance with USB device framework command set.

20.8.2 Port State and Control

From a chip or system reset, the USB_DR enters the powered state. A transition from the powered state to the attach state occurs when the run/stop bit (USBCMD[RS]) is set to a '1'. After receiving a reset on the bus, the port will enter the defaultFS or defaultHS state in accordance with the protocol reset described in

Appendix C.2 of the *USB Specification Rev. 2.0*. The following state diagram depicts the state of a USB 2.0 device.

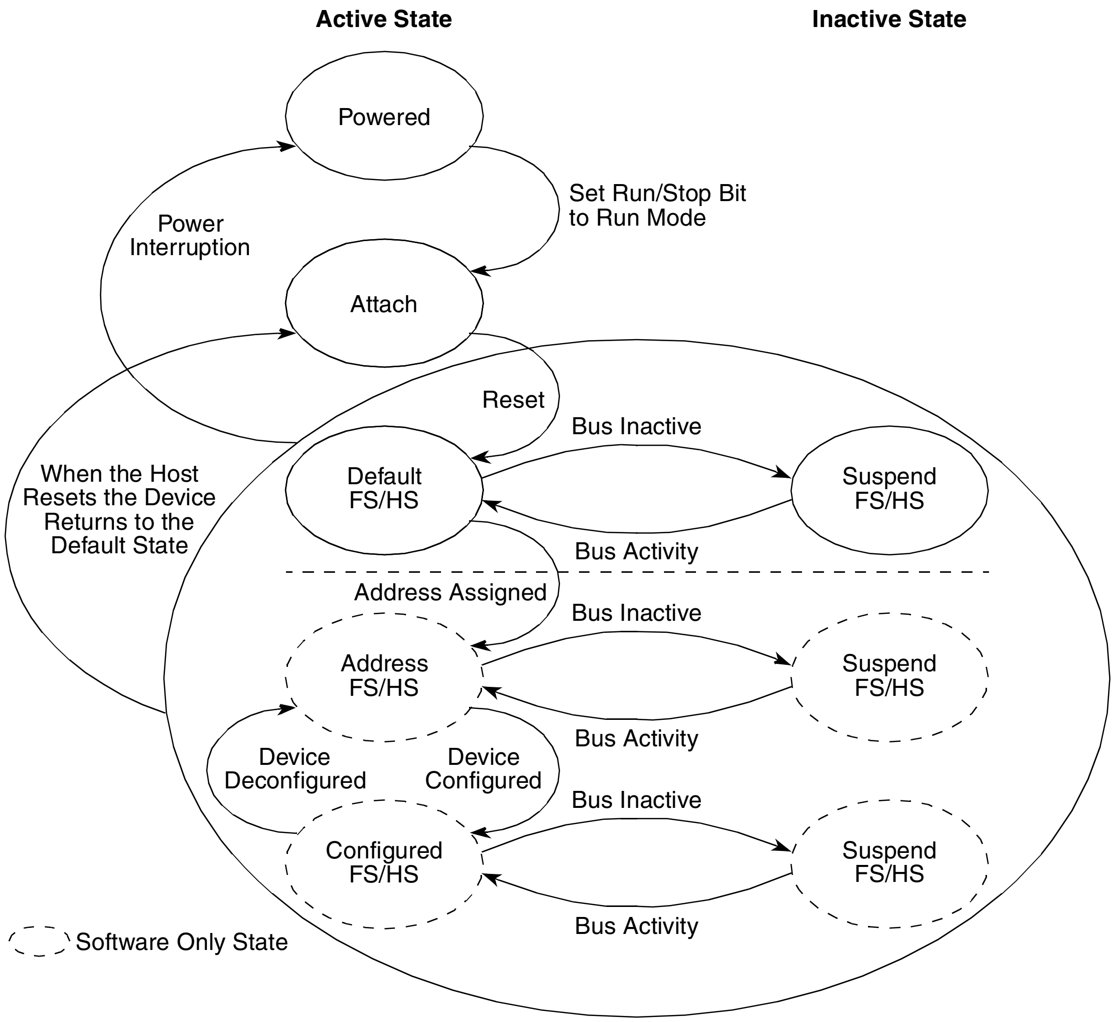


Figure 20-63. USB 2.0 Device States

States powered, attach, defaultFS/HS, suspendFS/HS are implemented in the USB_DR and are communicated to the DCD using the following status bits:

Table 20-84. Device Controller State Information Bits

Bits	Register
DCSuspend (SLI)	USBSTS
USB Reset Received (URI)	USBSTS
Port Change Detect (PCI)	USBSTS
High-Speed Port	PORTSC

It is the responsibility of the DCD to maintain a state variable to differentiate between the DefaultFS/HS state and the Address/Configured states. Change of state from Default to Address and the Configured states is part of the enumeration process described in the device framework section of the USB 2.0 Specification.

As a result of entering the Address state, the device address register (DEVICEADDR) must be programmed by the DCD.

Entry into the Configured indicates that all endpoints to be used in the operation of the device have been properly initialized by programming the ENDPTCTRL*n* registers and initializing the associated queue heads.

20.8.2.1 Bus Reset

A bus reset is used by the host to initialize downstream devices. When a bus reset is detected, the USB_DR controller will renegotiate its attachment speed, reset the device address to 0, and notify the DCD by interrupt (assuming the USB reset interrupt enable bit, USBINTR[URE], is set). After a reset is received, all endpoints (except endpoint 0) are disabled and any primed transactions will be canceled by the device controller. The concept of priming will be clarified below, but the DCD must perform the following tasks when a reset is received:

Clear all setup token semaphores by reading the ENDPTSETUPSTAT register and writing the same value back to the ENDPTSETUPSTAT register.

Clear all the endpoint complete status bits by reading the ENDPTCOMPLETE register and writing the same value back to the ENDPTCOMPLETE register.

Cancel all primed status by waiting until all bits in the ENDPTPRIME are 0 and then writing 0xFFFF_FFFF to ENDPTFLUSH.

Read the reset bit in the PORTSC register (PORTSC[PR]) and make sure that it is still active. A USB reset will occur for a minimum of 3 ms and the DCD must reach this point in the reset cleanup before end of the reset occurs, otherwise a hardware reset of the device controller is recommended (rare.)

- A hardware reset can be performed by writing a one to the USB_DR reset bit in (USBCMD[RST]). Note that a hardware reset will cause the device to detach from the bus by clearing USBCMD[RS] bit. Thus, the DCD must completely re-initialize the USB_DR after a hardware reset.

Free all allocated dTDs because they will no longer be executed by the device controller. If this is the first time the DCD is processing a USB reset event, then it is likely that no dTDs have been allocated.

At this time, the DCD may release control back to the OS because no further changes to the device controller are permitted until a Port Change Detect is indicated.

After a Port Change Detect, the device has reached the default state and the DCD can read the PORTSC to determine if the device is operating in FS or HS mode. At this time, the device controller has reached normal operating mode and DCD can begin enumeration according to the USB Chapter 9, Device Framework.

NOTE

The device DCD may use the FS/HS mode information to determine the bandwidth mode of the device.

In some applications, it may not be possible to enable one or more pipes while in FS mode. Beyond the data rate issue, there is no difference in DCD operation between FS and HS modes.

20.8.2.2 Suspend/Resume

20.8.2.2.1 Suspend Description

In order to conserve power, USB_DR automatically enters the suspended state when no bus traffic has been observed for a specified period. When suspended, the USB_DR maintains any internal status, including its address and configuration. Attached devices must be prepared to suspend at any time they are powered, regardless of if they have been assigned a non-default address, are configured, or neither. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the suspended state when the hub port it is attached to is disabled.

The USB_DR exits suspend mode when there is bus activity. It may also request the host to exit suspend mode or selective suspend by using electrical signaling to indicate remote wake-up. The ability of a device to signal remote wake-up is optional. The USB_DR is capable of remote wake-up signaling. When the USB_DR is reset, remote wake-up signaling must be disabled.

20.8.2.2.2 Suspend Operational Model

The USB_DR moves into the suspend state when suspend signaling is detected or activity is missing on the upstream port for more than a specific period. After the device controller enters the suspend state, the DCD is notified by an interrupt (assuming DC Suspend Interrupt is enabled). When the USBSTS[SLI] (device controller suspend) is set, the device controller is suspended.

DCD response when the device controller is suspended is application specific and may involve switching to low power operation.

Information on the bus power limits in suspend state can be found in USB 2.0 specification.

20.8.2.2.3 Resume

If the USB_DR is suspended, its operation is resumed when any non-idle signaling is received on its upstream facing port. In addition, the USB_DR can signal the system to resume operation by forcing resume signaling to the upstream port. Resume signaling is sent upstream by writing a ‘1’ to the PORTSC[FPR] (resume bit) while the device is in suspend state. Sending resume signal to an upstream port should cause the host to issue resume signaling and bring the suspended bus segment (one more devices) back to the active condition.

NOTE

Before resume signaling can be used, the host must enable it by using the Set Feature command defined in device framework (Chapter 9) of the USB 2.0 Specification.

20.8.3 Managing Endpoints

The USB 2.0 specification defines an endpoint, also called a device endpoint or an address endpoint as a uniquely addressable portion of a USB device that can source or sink data in a communications channel between the host and the device. The endpoint address is specified by the combination of the endpoint number and the endpoint direction.

The channel between the host and an endpoint at a specific device represents a data pipe. Endpoint 0 for a device is always a control type data channel used for device discovery and enumeration. Other types of endpoints support by USB include bulk, interrupt, and isochronous. Each endpoint type has specific behavior related to packet response and error handling. More detail on endpoint operation can be found in the USB 2.0 specification.

The USB_DR supports up to six endpoint specified numbers. The DCD can enable, disable, and configure each endpoint.

Each endpoint direction is essentially independent and can be configured with differing behavior in each direction. For example, the DCD can configure endpoint 1-IN to be a bulk endpoint and endpoint 1-OUT to be an isochronous endpoint. This helps to conserve the total number of endpoints required for device operation. The only exception is that control endpoints must use both directions on a single endpoint number to function as a control endpoint. Endpoint 0 is, for example, is always a control endpoint and uses the pair of directions.

Each endpoint direction requires a queue head allocated in memory. If the maximum of 6 endpoint numbers, one for each endpoint direction are being used by the device controller, then 12 queue heads are required. The operation of an endpoint and use of queue heads are described later in this document.

20.8.3.1 Endpoint Initialization

After hardware reset, all endpoints except endpoint zero are uninitialized and disabled. The DCD must configure and enable each endpoint by writing to configuration bit in the ENDPTCTRL*n* register. Each 32-bit ENDPTCTRL*n* is split into an upper and lower half. The lower half of ENDPTCTRL*n* is used to configure the receive or OUT endpoint and the upper half is likewise used to configure the corresponding transmit or IN endpoint. Control endpoints must be configured the same in both the upper and lower half of the ENDPTCTRL*n* register otherwise the behavior is undefined. The following table shows how to construct a configuration word for endpoint initialization.

Table 20-85. Device Controller Endpoint Initialization

Field	Value
Data Toggle Reset	1
Data Toggle Inhibit	0
Endpoint Type	00 Control 01 Isochronous 10 Bulk 11 Interrupt
Endpoint Stall	0

20.8.3.1.1 Stalling

There are two occasions where the USB_DR may need to return to the host a STALL.

The first occasion is the functional stall, which is a condition set by the DCD as described in the USB 2.0 device framework (Chapter 9). A functional stall is only used on non-control endpoints and can be enabled in the device controller by setting the endpoint stall bit in the ENDPTCTRL*n* register associated with the given endpoint and the given direction. In a functional stall condition, the device controller will continue to return STALL responses to all transactions occurring on the respective endpoint and direction until the endpoint stall bit is cleared by the DCD.

A protocol stall, unlike a function stall, is used on control endpoints is automatically cleared by the device controller at the start of a new control transaction (setup phase). When enabling a protocol stall, the DCD should enable the stall bits (both directions) as a pair. A single write to the ENDPTCTRL*n* register can ensure that both stall bits are set at the same instant.

NOTE

Any write to the ENDPTCTRL*n* register during operational mode must preserve the endpoint type field (that is, perform a read-modify-write).

Table 20-86. Device Controller Stall Response Matrix

USB Packet	Endpoint Stall Bit.	Effect on STALL Bit.	USB Response
SETUP packet received by a non-control endpoint	N/A	None	STALL
IN/OUT/PING packet received by a non-control endpoint	'1	None	STALL
IN/OUT/PING packet received by a non-control endpoint	'0	None	ACK/NAK/NYET
SETUP packet received by a control endpoint	N/A	Cleared	ACK
IN/OUT/PING packet received by a control endpoint	'1	None	STALL
IN/OUT/PING packet received by a control endpoint	'0	None	ACK/NAK/NYET

20.8.3.2 Data Toggle

Data toggle is a mechanism to maintain data coherency between host and device for any given data pipe. For more information on data toggle, refer to the *Universal Serial Bus Revision 2.0 Specification*.

20.8.3.2.1 Data Toggle Reset

The DCD may reset the data toggle state bit and cause the data toggle sequence to reset in the device controller by writing a '1' to the data toggle reset bit in the ENDPTCTRL*n* register. This should only be necessary when configuring/initializing an endpoint or returning from a STALL condition.

20.8.3.2.2 Data Toggle Inhibit

This feature is for test purposes only and should never be used during normal device controller operation.

Setting the data toggle Inhibit bit active ('1') causes the USB_DR to ignore the data toggle pattern that is normally sent and accept all incoming data packets regardless of the data toggle state.

In normal operation, the USB_DR checks the DATA0/DATA1 bit against the data toggle to determine if the packet is valid. If Data PID does not match the data toggle state bit maintained by the device controller for that endpoint, the Data toggle is considered not valid. If the data toggle is not valid, the device controller assumes the packet was already received and discards the packet (not reporting it to the DCD). To prevent the USB_DR from re-sending the same packet, the device controller will respond to the error packet by acknowledging it with either an ACK or NYET response.

20.8.3.3 Device Operational Model For Packet Transfers

All transactions on the USB bus are initiated by the host and in turn, the device must respond to any request from the host within the turnaround time stated in the *Universal Serial Bus Revision 2.0 Specification*.

A USB host will send requests to the USB_DR in an order that can not be precisely predicted as a single pipeline, so it is not possible to prepare a single packet for the device controller to execute. However, the order of packet requests is predictable when the endpoint number and direction is considered. For example, if endpoint 2 (transmit direction) is configured as a bulk pipe, then we can expect the host will send IN requests to that endpoint. This USB_DR prepares packets for each endpoint/direction in anticipation of the host request. The process of preparing the device controller to send or receive data in response to host initiated transaction on the bus is referred to as 'priming' the endpoint. This term will be used throughout the following documentation to describe the USB_DR operation so the DCD can be architected properly use priming. Further, note that the term 'flushing' is used to describe the action of clearing a packet that was queued for execution.

20.8.3.3.1 Priming Transmit Endpoints

Priming a transmit endpoint will cause the device controller to fetch the device transfer descriptor (dTD) for the transaction pointed to by the device queue head (dQH). After the dTD is fetched, it will be stored in the dQH until the device controller completes the transfer described by the dTD. Storing the dTD in the dQH allows the device controller to fetch the operating context needed to handle a request from the host without the need to follow the linked list, starting at the dQH when the host request is received.

After the device has loaded the dTD, the leading data in the packet is stored in a FIFO in the device controller. This FIFO is split into virtual channels so that the leading data can be stored for any endpoint up to the maximum number of endpoints configured at device synthesis time.

After a priming request is complete, an endpoint state of primed is indicated in the ENDPTSTATUS register. For a primed transmit endpoint, the device controller can respond to an IN request from the host and meet the stringent bus turnaround time of High Speed USB.

Since only the leading data is stored in the device controller FIFO, it is necessary for the device controller to begin filling in behind leading data after the transaction starts. The FIFO must be sized to account for the maximum latency that can be incurred by the system memory bus.

20.8.3.3.2 Priming Receive Endpoints

Priming receive endpoints is identical to priming of transmit endpoints from the point of view of the DCD. At the device controller the major difference in the operational model is that there is no data movement of the leading packet data simply because the data is to be received from the host.

Note as part of the architecture, the FIFO for the receive endpoints is not partitioned into multiple channels like the transmit FIFO. Thus, the size of the RX FIFO does not scale with the number of endpoints.

20.8.3.4 Interrupt/Bulk Endpoint Operational Model

The behaviors of the device controller for interrupt and bulk endpoints are identical. All valid IN and OUT transactions to bulk pipes will handshake with a NAK unless the endpoint had been primed. Once the endpoint has been primed, data delivery will commence.

A dTD will be retired by the device controller when the packets described in the transfer descriptor have been completed. Each dTD describes N packets to be transferred according to the USB Variable Length transfer protocol. The formula and table on the following page describe how the device controller computes the number and length of the packets to be sent/received by the USB vary according to the total number of bytes and maximum packet length.

With Zero Length Termination (ZLT) = 0

N = INT(number of bytes/max. packet length) + 1

With Zero Length Termination (ZLT) = 1

N = MAXINT(number of bytes/max. packet length)

Table 20-87. Variable Length Transfer Protocol Example (ZLT = 0)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	
512	256	3	256	256	0
512	512	2	512	0	

Table 20-88. Variable Length Transfer Protocol Example (ZLT = 1)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	
512	256	2	256	256	
512	512	1	512		

NOTE

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

TX-dTD is complete when:

- All packets described dTD were successfully transmitted. *** Total bytes in dTD will equal zero when this occurs.

RX-dTD is complete when:

- All packets described in dTD were successfully received. *** Total bytes in dTD will equal zero when this occurs.
- A short packet (number of bytes < maximum packet length) was received. *** This is a successful transfer completion; DCD must check Total Bytes in dTD to determine the number of bytes that are remaining. From the total bytes remaining in the dTD, the DCD can compute the actual bytes received.
- A long packet was received (number of bytes > maximum packet size) OR (total bytes received > total bytes specified). *** This is an error condition. The device controller will discard the remaining packet, and set the Buffer Error bit in the dTD. In addition, the endpoint will be flushed and the USBERR interrupt will become active.

On the successful completion of the packet(s) described by the dTD, the active bit in the dTD will be cleared and the next pointer will be followed when the Terminate bit is clear. When the Terminate bit is set, the USB_DR will flush the endpoint/direction and cease operations for that endpoint/direction.

On the unsuccessful completion of a packet (see long packet above), the dQH will be left pointing to the dTD that was in error. In order to recover from this error condition, the DCD must properly re-initialize the dQH by clearing the active bit and update the nextTD pointer before attempting to re-prime the endpoint.

NOTE

All packet level errors such as a missing handshake or CRC error will be retried automatically by the device controller.

There is no required interaction with the DCD for handling such errors.

20.8.3.4.1 Interrupt/Bulk Endpoint Bus Response Matrix

Table 20-89. Interrupt/Bulk Endpoint Bus Response Matrix

	Stall	Not Primed	Primed	Underflow	Overflow
Setup	Ignore	Ignore	Ignore	N/A	N/A
In	STALL	NAK	Transmit	BS Error[1]	N/A
Out	STALL	NAK	Receive + NYET/ACK[2]	N/A	NAK
Ping	STALL	NAK	ACK	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

1 Force Bit Stuff Error.

2 NYET/ACK—NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.
SYSERR—System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.

20.8.3.5 Control Endpoint Operation Model

20.8.3.5.1 Setup Phase

All requests to a control endpoint begin with a setup phase followed by an optional data phase and a required status phase. The USB_DR will always accept the setup phase unless the setup lockout is engaged.

The setup lockout will engage so that future setup packets are ignored. Lockout of setup packets ensures that while software is reading the setup packet stored in the queue head, that data is not written as it is being read potentially causing an invalid setup packet.

The setup lockout mechanism can be disabled and a tripwire type semaphore will ensure that the setup packet payload is extracted from the queue head without being corrupted be an incoming setup packet. This is the preferred behavior because ignoring repeated setup packets due to long software interrupt latency would be a compliance issue.

Setup Packet Handling

- Disable Setup Lockout by writing ‘1’ to Setup Lockout Mode (SLOM) in USBMODE (once at initialization). Setup lockout is not necessary when using the tripwire as described below.

NOTE

Leaving the Setup Lockout Mode as ‘0’ will result in a potential compliance issue.

- After receiving an interrupt and inspecting ENDPTSETUPSTAT to determine that a setup packet was received on a particular pipe:
  - — Write ‘1’ to clear corresponding bit ENDPTSETUPSTAT.
  - — Write ‘1’ to Setup Tripwire (SUTW) in USBCMD register.
  - — Duplicate contents of dQH.SetupBuffer into local software byte array.
  - — Read Setup TripWire (SUTW) in USBCMD register. (if set—continue; if cleared—goto 2)
  - — Write ‘0’ to clear Setup Tripwire (SUTW) in USBCMD register.
  - — Process setup packet using local software byte array copy and execute status/handshake phases.

NOTE

After receiving a new setup packet the status and/or handshake phases may still be pending from a previous control sequence. These should be flushed and de-allocated before linking a new status and/or handshake dTD for the most recent setup packet.

20.8.3.5.2 Data Phase

Following the setup phase, the DCD must create a device transfer descriptor for the data phase and prime the transfer.

After priming the packet, the DCD must verify a new setup packet has not been received by reading the ENDPTSETUPSTAT register immediately verifying that the prime had completed. A prime will complete when the associated bit in the ENDPTPRIME register is zero and the associated bit in the ENDPTSTATUS register is a one. If a prime fails, that is, The ENDPTPRIME bit goes to zero and the ENDPTSTATUS bit is not set, then the prime has failed. This can only be due to improper setup of the dQH, dTD or a setup arriving during the prime operation. If a new setup packet is indicated after the ENDPTPRIME bit is cleared, then the transfer descriptor can be freed and the DCD must reinterpret the setup packet.

Should a setup arrive after the data stage is primed, the device controller will automatically clear the prime status (ENDPTSTATUS) to enforce data coherency with the setup packet.

NOTE

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

NOTE

Error handling of data phase packets is the same as bulk packets described previously.

20.8.3.5.3 Status Phase

Similar to the data phase, the DCD must create a transfer descriptor (with byte length equal zero) and prime the endpoint for the status phase. The DCD must also perform the same checks of the ENDPTSETUPSTAT as described above in the data phase.

NOTE

The MULT field in the dQH must be set to ‘00’ for bulk, interrupt, and control endpoints.

NOTE

Error handling of data phase packets is the same as bulk packets described previously.

20.8.3.5.4 Control Endpoint Bus Response Matrix

Shown in the following table is the device controller response to packets on a control endpoint according to the device controller state.

Table 20-90. Control Endpoint Bus Response Matrix

Token Type	Endpoint State					Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	
Setup	ACK	ACK	ACK	N/A	SYSERR[1]	
In	STALL	NAK	Transmit	BS Error[2]	N/A	N/A
Out	STALL	NAK	Receive + NYET/ACK[3]	N/A	NAK	N/A

Table 20-90. Control Endpoint Bus Response Matrix (continued)

Token Type	Endpoint State					Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	
Ping	STALL	NAK	ACK	N/A	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore

[1] SYSERR—System error should never occur when the latency FIFOs are correctly sized and the DCD is responsive.
[2] Force Bit Stuff Error.
[3] NYET/ACK—NYET unless the Transfer Descriptor has packets remaining according to the USB variable length protocol then ACK.

20.8.3.6 Isochronous Endpoint Operational Model

Isochronous endpoints are used for real-time scheduled delivery of data and their operational model is significantly different than the host throttled Bulk, Interrupt, and Control data pipes. Real time delivery by the USB_DR will is accomplished by the following:

- Exactly MULT Packets per (micro)Frame are transmitted/received. Note that MULT is a two-bit field in the device Queue Head. The variable length packet protocol is not used on isochronous endpoints.
- NAK responses are not used. Instead, zero length packets and sent in response to an IN request to an unprimed endpoints. For unprimed RX endpoints, the response to an OUT transaction is to ignore the packet within the device controller.
- Prime requests always schedule the transfer described in the dTD for the next (micro)frame. If the ISO-dTD is still active after that frame, then the ISO-dTD will be held ready until executed or canceled by the DCD.

The USB_DR in host mode uses the periodic frame list to schedule data exchanges to Isochronous endpoints. The operational model for device mode does not use such a data structure. Instead, the same dTD used for Control/Bulk/Interrupt endpoints is also used for isochronous endpoints. The difference is in the handling of the dTD.

The first difference between bulk and ISO-endpoints is that priming an ISO-endpoint is a delayed operation such that an endpoint will become primed only after a SOF is received. After the DCD writes the prime bit, the prime bit will be cleared as usual to indicate to software that the device controller completed a priming the dTD for transfer. Internal to the design, the device controller hardware masks that prime start until the next frame boundary. This behavior is hidden from the DCD but occurs so that the device controller can match the dTD to a specific (micro)frame.

Another difference with isochronous endpoints is that the transaction must wholly complete in a (micro)frame. Once an ISO transaction is started in a (micro)frame it will retire the corresponding dTD when MULT transactions occur or the device controller finds a fulfillment condition.

The transaction error bit set in the status field indicates a fulfillment error condition. When a fulfillment error occurs, the frame after the transfer failed to complete wholly, the device controller will force retire the ISO-dTD and move to the next ISO-dTD.

It is important to note that fulfillment errors are only caused due to partially completed packets. If no activity occurs to a primed ISO-dTD, the transaction will stay primed indefinitely. This means it is up to software discard transmit ISO-dTDs that pile up from a failure of the host to move the data.

Finally, the last difference with ISO packets is in the data level error handling. When a CRC error occurs on a received packet, the packet is not retried similar to bulk and control endpoints. Instead, the CRC is noted by setting the Transaction Error bit and the data is stored as usual for the application software to sort out.

- TX Packet Retired
  — MULT counter reaches zero.
  — Fulfillment Error [Transaction Error bit is set]
  — #Packets Occurred > 0 AND # Packets Occurred < MULT

NOTE

For TX-ISO, MULT Counter can be loaded with a lesser value in the dTD Multiplier Override field. If the Multiplier Override is zero, the MULT Counter is initialized to the Multiplier in the QH.

- RX Packet Retired:
  — MULT counter reaches zero.
  — Non-MDATA Data PID is received
  — Overflow Error:
  — Packet received is > maximum packet length. [Buffer Error bit is set]
  — Packet received exceeds total bytes allocated in dTD. [Buffer Error bit is set]
  — Fulfillment Error [Transaction Error bit is set]
  — # Packets Occurred > 0 AND # Packets Occurred < MULT
  — CRC Error [Transaction Error bit is set]

NOTE

For ISO, when a dTD is retired, the next dTD is primed for the next frame. For continuous (micro)frame to (micro)frame operation the DCD should ensure that the dTD linked-list is out ahead of the device controller by at least two (micro)frames.

20.8.3.6.1 Isochronous Pipe Synchronization

When it is necessary to synchronize an isochronous data pipe to the host, the (micro)frame number (FRINDEX register) can be used as a marker. To cause a packet transfer to occur at a specific (micro)frame number [N], the DCD should interrupt on SOF during frame N-1. When the FRINDEX=N-1, the DCD must write the prime bit. The USB_DR will prime the isochronous endpoint in (micro)frame N-1 so that the device controller will execute delivery during (micro)frame N.

CAUTION

Priming an endpoint towards the end of (micro)frame N-1 will not guarantee delivery in (micro)frame N. The delivery may actually occur in (micro)frame N+1 if device controller does not have enough time to complete the prime before the SOF for packet N is received.

20.8.3.6.2 Isochronous Endpoint Bus Response Matrix

Table 20-91. Isochronous Endpoint Bus Response Matrix

	Stall	Not Primed	Primed	Underflow	Overflow
Setup	STALL	STALL	STALL	N/A	N/A
In	NULL[1] Packet	NULL Packet	Transmit	BS Error[2]	N/A
Out	Ignore	Ignore	Receive	N/A	Drop Packet
Ping	Ignore	Ignore	Ignore	Ignore	Ignore
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

[1] Zero Length Packet.
[2] Force Bit Stuff Error.

20.8.4 Managing Queue Heads

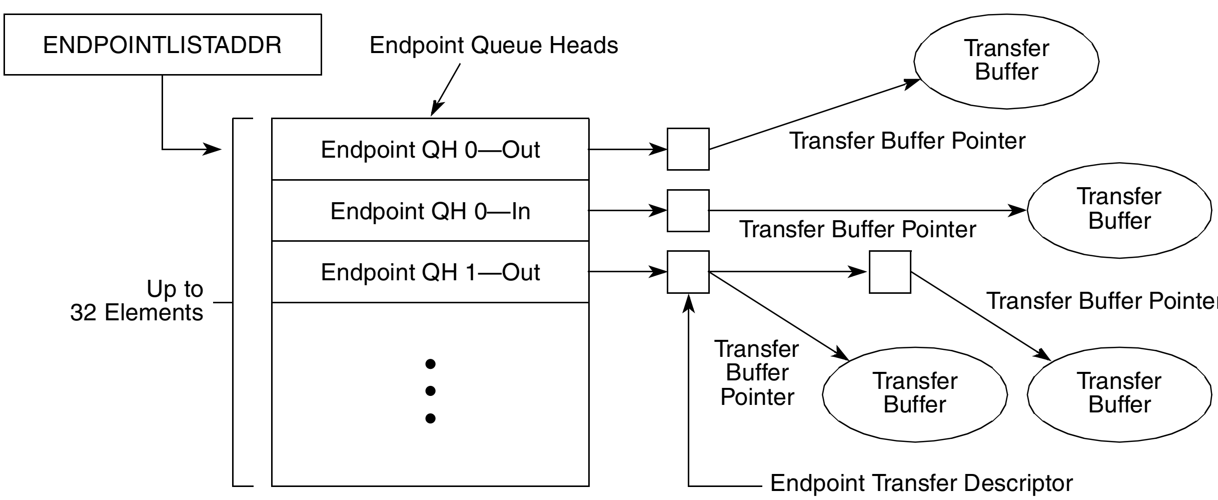


Figure 20-64. Endpoint Queue Head Diagram

The device queue head (dQH) points to the linked list of transfer tasks, each depicted by the device Transfer Descriptor (dTD). An area of memory pointed to by ENDPOINTLISTADDR contains a group of all dQH's in a sequential list as shown in Figure 20-64. The even elements in the list of dQH's are used for receive endpoints (OUT/SETUP) and the odd elements are used for transmit endpoints (IN/INTERRUPT). Device transfer descriptors are linked head to tail starting at the queue head and ending at a terminate bit. Once the dTD has been retired, it will no longer be part of the linked list from the queue head. Therefore, software is required to track all transfer descriptors since pointers will no longer exist within the queue head once the dTD is retired (see section Software Link Pointers).

In addition to the current and next pointers and the dTD overlay examined in section Operational Model For Packet Transfers, the dQH also contains the following parameters for the associated endpoint: Multipler, Maximum Packet Length, Interrupt On Setup. The complete initialization of the dQH including these fields is demonstrated in the next section.

20.8.4.1 Queue Head Initialization

One pair of device queue heads must be initialized for each active endpoint. To initialize a device queue head:

- Write the wMaxPacketSize field as required by the USB Chapter 9 or application specific protocol.
- Write the multiplier field to 0 for control, bulk, and interrupt endpoints. For ISO endpoints, set the multiplier to 1, 2, or 3 as required bandwidth an in conjunction with the USB Chapter 9 protocol. Note that in FS mode, the multiplier field can only be 1 for ISO endpoints.
- Write the next dTD Terminate bit field to ‘1.’
- Write the Active bit in the status field to ‘0.’
- Write the Halt bit in the status field to ‘0.’

NOTE

The DCD must only modify dQH if the associated endpoint is not primed and there are no outstanding dTDs.

20.8.4.2 Operational Model for Setup Transfers

As discussed in Section 20.8.3.5, “Control Endpoint Operation Model,” setup transfer requires special treatment by the DCD. A setup transfer does not use a dTD but instead stores the incoming data from a setup packet in an 8-byte buffer within the dQH.

Upon receiving notification of the setup packet, the DCD should handle the setup transfer as demonstrated here:

1. Copy setup buffer contents from dQH - RX to software buffer.
2. Acknowledge setup backup by writing a ‘1’ to the corresponding bit in ENDPTSETUPSTAT.

NOTE

The acknowledge must occur before continuing to process the setup packet.

NOTE

After the acknowledge has occurred, the DCD must not attempt to access the setup buffer in the dQH - RX. Only the local software copy should be examined.

3. Check for pending data or status dTD’s from previous control transfers and flush if any exist as discussed in section Flushing/De-priming an Endpoint.

NOTE

It is possible for the device controller to receive setup packets before previous control transfers complete. Existing control packets in progress must be flushed and the new control packet completed.

4. Decode setup packet and prepare data phase [optional] and status phase transfer as required by the USB Chapter 9 or application specific protocol.

20.8.5 Managing Transfers with Transfer Descriptors

20.8.5.1 Software Link Pointers

It is necessary for the DCD software to maintain head and tail pointers to the for the linked list of dTDs for each respective queue head. This is necessary because the dQH only maintains pointers to the current working dTD and the next dTD to be executed. The operations described in next section for managing dTD will assume the DCD can use reference the head and tail of the dTD linked list.

NOTE

To conserve memory, the reserved fields at the end of the dQH can be used to store the Head and Tail pointers but it still remains the responsibility of the DCD to maintain the pointers.

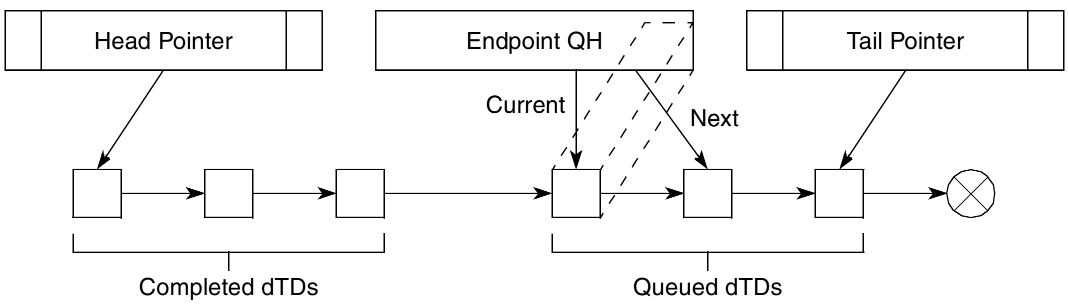


Figure 20-65. Software Link Pointers

20.8.5.2 Building a Transfer Descriptor

Before a transfer can be executed from the linked list, a dTD must be built to describe the transfer. Use the following procedure for building dTDs.

Allocate 8-DWord dTD block of memory aligned to 8-DWord boundaries. Example: bit address 4–0 would be equal to ‘00000’.

Write the following fields:

1. Initialize first 7 DWords to ‘0’.
2. Set the terminate bit to ‘1’.
3. Fill in total bytes with transfer size.
4. Set the interrupt on complete if desired.
5. Initialize the status field with the active bit set to ‘1’ and all remaining status bits set to ‘0’.

6. Fill in buffer pointer page 0 and the current offset to point to the start of the data buffer.
7. Initialize buffer pointer page 1 through page 4 to be one greater than each of the previous buffer pointer.

20.8.5.3 Executing a Transfer Descriptor

To safely add a dTD, the DCD must be follow this procedure which will handle the event where the device controller reaches the end of the dTD list at the same time a new dTD is being added to the end of the list.

Determine whether the link list is empty:

Check DCD driver to see if pipe is empty (internal representation of linked-list should indicate if any packets are outstanding).

Case 1: Link list is empty

1. Write dQH next pointer AND dQH terminate bit to ‘0’ as a single DWord operation.
2. Clear active and halt bit in dQH (in case set from a previous error).
3. Prime endpoint by writing ‘1’ to correct bit position in ENDPTPRIME.

Case 2: Link list is not empty

1. Add dTD to end of linked list.
2. Read correct prime bit in ENDPTPRIME—if ‘1’ DONE.
3. Set ATDTW bit in USBCMD register to ‘1’.
4. Read correct status bit in ENDPTSTATUS. (store in tmp. variable for later).
5. Read ATDTW bit in USBCMD register.
   If ‘0’ goto 3.
   If ‘1’ continue to 6.
6. Write ATDTW bit in USBCMD register to ‘0’.
7. If status bit read in (3) is ‘1’ DONE.
8. If status bit read in (3) is ‘0’ then Goto Case 1: Step 1.

20.8.5.4 Transfer Completion

After a dTD has been initialized and the associated endpoint primed the device controller will execute the transfer upon the host-initiated request. The DCD will be notified with a USB interrupt if the Interrupt On Complete bit was set or alternately, the DCD can poll the endpoint complete register to find when the dTD had been executed. After a dTD has been executed, DCD can check the status bits to determine success or failure.

CAUTION

Multiple dTD can be completed in a single endpoint complete notification. After clearing the notification, DCD must search the dTD linked list and retire all dTDs that have finished (Active bit cleared).

By reading the status fields of the completed dTDs, the DCD can determine if the transfers completed successfully. Success is determined with the following combination of status bits:

- Active = 0
- Halted = 0
- Transaction Error = 0
- Data Buffer Error = 0

Should any combination other than the one shown above exist, the DCD must take proper action. Transfer failure mechanisms are indicated in the Device Error Matrix.

In addition to checking the status bit the DCD must read the Transfer Bytes field to determine the actual bytes transferred. When a transfer is complete, the Total Bytes transferred is by decremented by the actual bytes transferred. For Transmit packets, a packet is only complete after the actual bytes reaches zero, but for receive packets, the host may send fewer bytes in the transfer according the USB variable length packet protocol.

20.8.5.5 Flushing/De-Priming an Endpoint

It is necessary for the DCD to flush to de-prime one more endpoints on a USB device reset or during a broken control transfer. There may also be application specific requirements to stop transfers in progress. The following procedure can be used by the DCD to stop a transfer in progress:

1. Write a ‘1’ to the corresponding bit(s) in ENDPTFLUSH.
2. Wait until all bits in ENDPTFLUSH are ‘0’.
3. Software note: this operation may take a large amount of time depending on the USB bus activity. It is not desirable to have this wait loop within an interrupt service routine.
4. Read ENDPTSTATUS to ensure that for all endpoints commanded to be flushed, that the corresponding bits are now ‘0’’ If the corresponding bits are ‘1’ after step #2 has finished, then the flush failed as described in the following:

   Explanation: In very rare cases, a packet is in progress to the particular endpoint when commanded flush using ENDPTFLUSH. A safeguard is in place to refuse the flush to ensure that the packet in progress completes successfully. The DCD may need to repeatedly flush any endpoints that fail to flush be repeating steps 1–3 until each endpoint is successfully flushed.

20.8.5.6 Device Error Matrix

The following table summarizes packet errors that are not automatically handled by the USB_DR

Table 20-92. Device Error Matrix

Error	Direction	Packet Type	Data Buffer Error Bit	Transaction Error Bit
Overflow **	RX	Any	1	0
ISO Packet Error	RX	ISO	0	1
ISO Fulfillment Error	Both	ISO	0	1

Notice that the device controller handles all errors on Bulk/Control/Interrupt Endpoints except for a data buffer overflow. However, for ISO endpoints, errors packets are not retried and errors are tagged as indicated.

Table 20-93. Error Descriptions

Overflow	Number of bytes received exceeded max. packet size or total buffer length. ** This error will also set the Halt bit in the dQH and if there are dTDs remaining in the linked list for the endpoint, then those will not be executed.
ISO Packet Error	CRC Error on received ISO packet. Contents not guaranteed to be correct.
ISO Fulfillment Error	Host failed to complete the number of packets defined in the dQH mult field within the given (micro)frame. For scheduled data delivery the DCD may need to readjust the data queue because a fulfillment error will cause Device Controller to cease data transfers on the pipe for one (micro)frame. During the 'dead' (micro)frame, the Device Controller reports error on the pipe and primes for the following frame.

20.8.6 Servicing Interrupts

The interrupt service routine must consider that there are high-frequency, low-frequency operations, and error operations and order accordingly.

20.8.6.1 High-Frequency Interrupts

High frequency interrupts in particular should be handed in the order below. The most important of these is listed first because the DCD must acknowledge a setup buffer in the timeliest manner possible.

Table 20-94. Interrupt Handling Order

Execution Order	Interrupt	Action
1a	USB Interrupt[1] ENDPTSETUPSTATUS	Copy contents of setup buffer and acknowledge setup packet (as indicated in section Managing Queue Heads). Process setup packet according to USB 2.0 Chapter 9 or application specific protocol.
1b	USB Interrupt ENDPTCOMPLETE	Handle completion of dTD as indicated in section Managing Queue Heads.
2	SOF Interrupt	Action as deemed necessary by application. This interrupt may not have a use in all applications.

[1] It is likely that multiple interrupts to stack up on any call to the Interrupt Service Routine AND during the Interrupt Service Routine.

20.8.6.2 Low-Frequency Interrupts

The low frequency events include the following interrupts. These interrupt can be handled in any order since they don't occur often in comparison to the high-frequency interrupts.

Table 20-95. Low Frequency Interrupt Events

Interrupt	Action
Port Change	Change software state information.
Sleep Enable (Suspend)	Change software state information. Low power handling as necessary.
Reset Received	Change software state information. Abort pending transfers.

20.8.6.3 Error Interrupts

Error interrupts will be least frequent and should be placed last in the interrupt service routine.

Table 20-96. Error Interrupt Events

Interrupt	Action
USB Error Interrupt	This error is redundant because it combines USB Interrupt and an error status in the dTD. The DCD will more aptly handle packet-level errors by checking dTD status field upon receipt of USB Interrupt (w/ ENDPTCOMPLETE).
System Error	Unrecoverable error. Immediate Reset of core; free transfers buffers in progress and restart the DCD.

20.9 Deviations from the EHCI Specifications

The host mode operation of the USB DR module is nearly EHCI-compatible with few minor differences. For the most part, the module conforms to the data structures and operations described in Section 3, “Data Structures,” and Section 4, “Operational Model,” in the EHCI specification. The particulars of the deviations occur in the following areas:

- Embedded transaction translator—Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation—In host mode, the device operational registers are generally disabled and thus device mode is mostly transparent when in host mode. However, there are a couple exceptions documented in the following sections.
- Embedded design interface—The module does not have a PCI interface and therefore the PCI configuration registers described in the EHCI specification are not applicable.

For the purposes of the DR implementing dual-role host/device controller with support for OTG applications, it is necessary to deviate from the EHCI specification. Device operation and OTG operation are not specified in the EHCI and thus the implementation supported in the DR module is proprietary.

20.9.1 Embedded Transaction Translator Function

The DR module supports directly connected full and low speed devices without requiring a companion controller by including the capabilities of a USB 2.0 high speed hub transaction translator. Although there is no separate transaction translator block in the system, the transaction translator function normally associated with a high speed hub has been implemented within the DMA and Protocol engine blocks. The embedded transaction translator function is an extension to EHCI interface, but makes use of the standard data structures and operational models that exist in the EHCI specification to support full and low speed devices.

20.9.1.1 Capability Registers

The following additions have been added to the capability registers to support the embedded transaction translator Function:

- N_TT added to HSCPARAMS—Host Controller Structural Parameters
- N_PTT added to HSCPARAMS—Host Controller Structural Parameters

See Section 20.3.1.3, "Host Controller Structural Parameters (HCSPARAMS)," for usage information.

20.9.1.2 Operational Registers

The following additions have been added to the operational registers to support the embedded TT:

- ASYNCTTSTS is a new register.
- Addition of two-bit Port Speed (PSPD) to the PORTSC register.

20.9.1.3 Discovery

In a standard EHCI controller design, the EHCI host controller driver detects a Full speed (FS) or Low speed (LS) device by noting if the port enable bit is set after the port reset operation. The port enable will only be set in a standard EHCI controller implementation after the port reset operation and when the host and device negotiate a High-Speed connection (that is, Chirp completes successfully).

The module will always set the port enable after the port reset operation regardless of the result of the host device chirp result and the resulting port speed will be indicated by the PSPD field in PORTSC. Therefore, the standard EHCI host controller driver requires an alteration to handle directly connected Full and Low speed devices or hubs. The change is a fundamental one in that is summarized in Table 20-97.

Table 20-97. Functional Differences Between EHCI and EHCI with Embedded TT

Standard EHCI	EHCI with Embedded Transaction Translator
After port enable bit is set following a connection and reset sequence, the device/hub is assumed to be HS.	After port enable bit is set following a connection and reset sequence, the device/hub speed is noted from PORTSC.
FS and LS devices are assumed to be downstream from a HS hub thus, all port-level control is performed through the Hub Class to the nearest Hub.	FS and LS device can be either downstream from a HS hub or directly attached. When the FS/LS device is downstream from a HS hub, then port-level control is done using the Hub Class through the nearest Hub. When a FS/LS device is directly attached, then port-level control is accomplished using PORTSC.
FS and LS devices are assumed to be downstream from a HS hub with HubAddr=X. [where HubAddr > 0 and HubAddr is the address of the Hub where the bus transitions from HS to FS/LS (that is, Split target hub)]	FS and LS device can be either downstream from a HS hub with HubAddr = X [HubAddr > 0] or directly attached [where HubAddr = 0 and HubAddr is the address of the Root Hub where the bus transitions from HS to FS/LS (that is, Split target hub is the root hub)]

20.9.1.4 Data Structures

The same data structures used for FS/LS transactions though a HS hub are also used for transactions through the Root Hub. Here it is demonstrated how the Hub Address and Endpoint Speed fields should be set for directly attached FS/LS devices and hubs:

1. QH (for direct attach FS/LS)—Async. (Bulk/Control Endpoints) Periodic (Interrupt)

- Hub Address = 0
- Transactions to direct attached device/hub.
  - QH.EPS = Port Speed
- Transactions to a device downstream from direct attached FS hub.
  - QH.EPS = Downstream Device Speed

NOTE

When QH.EPS = 01 (LS) and PORTSC[PSPD] = 00 (FS), a LS-pre-pid will be sent before the transmitting LS traffic.

Maximum Packet Size must be less than or equal 64 or undefined behavior may result.

2. siTD (for direct attach FS)—Periodic (ISO Endpoint)

- All FS ISO transactions:
  - Hub Address = 0
  - siTD.EPS = 00 (full speed)

Maximum Packet Size must less than or equal to 1023 or undefined behavior may result.

20.9.1.5 Operational Model

The operational models are well defined for the behavior of the transaction translator (see *Universal Serial Bus Revision 2.0 Specification*) and for the EHCI controller moving packets between system memory and a USB-HS hub. Since the embedded transaction translator exists within the DR module there is no physical bus between EHCI host controller driver and the USB FS/LS bus. These sections will briefly discuss the operational model for how the EHCI and transaction translator operational models are combined without the physical bus between. The following sections assume the reader is familiar with both the EHCI and USB 2.0 transaction translator operational models.

20.9.1.5.1 Microframe Pipeline

The EHCI operational model uses the concept of H-frames and B-frames to describe the pipeline between the Host (H) and the Bus (B). The embedded transaction translator shall use the same pipeline algorithms specified in the *Universal Serial Bus Revision 2.0 Specification* for a Hub-based transaction translator.

All periodic transfers always begin at B-frame 0 (after SOF) and continue until the stored periodic transfers are complete. As an example of the microframe pipeline implemented in the embedded transaction translator, all periodic transfers that are tagged in EHCI to execute in H-frame 0 will be ready to execute on the bus in B-frame 0.

It is important to note that when programming the S-mask and C-masks in the EHCI data structures to schedule periodic transfers for the embedded transaction translator, the EHCI host controller driver must follow the same rules specified in EHCI for programming the S-mask and C-mask for downstream Hub-based transaction translators.

Once periodic transfers are exhausted, any stored asynchronous transfer will be moved. Asynchronous transfers are opportunistic in that they shall execute whenever possible and their operation is not tied to H-frame and B-frame boundaries with the exception that an asynchronous transfer can not babble through the SOF (start of B-frame 0).

20.9.1.5.2 Split State Machines

The start and complete split operational model differs from EHCI slightly because there is no bus medium between the EHCI controller and the embedded transaction translator. Where a start or complete-split operation would occur by requesting the split to the HS hub, the start/complete split operation is simple an internal operation to the embedded transaction translator. Table 20-98 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 20-98. Emulated Handshakes

Condition	Emulate TT Response
Start-Split: All asynchronous buffers full	NAK
Start-Split: All periodic buffers full	ERR
Start-Split: Success for start of Async. Transaction	ACK
Start-Split: Start Periodic Transaction	No Handshake (Ok)
Complete-Split: Failed to find transaction in queue	Bus Time Out
Complete-Split: Transaction in Queue is Busy	NYET
Complete-Split: Transaction in Queue is Complete	[Actual Handshake from FS/LS device]

20.9.1.5.3 Asynchronous Transaction Scheduling and Buffer Management

The following *Universal Serial Bus Revision 2.0 Specification* items are implemented in the embedded transaction translator:

- USB 2.0–11.17.3
  - Sequencing is provided and a packet length estimator ensures no full-speed/low-speed packet babbles into SOF time.
- USB 2.0–11.17.4
  - Transaction tracking for 2 data pipes.
- USB 2.0–11.17.5
  - Clear_TT_Buffer capability provided

20.9.1.5.4 Periodic Transaction Scheduling and Buffer Management

The following *Universal Serial Bus Revision 2.0 Specification* items are implemented in the embedded transaction translator:

- USB 2.0–11.18.6.[1-2]

- Abort of pending start-splits
  - EOF (and not started in microframes 6)
  - Idle for more than 4 microframes
- Abort of pending complete-splits
  - EOF
  - Idle for more than 4 microframes

NOTE

There is no data schedule mechanism for these transactions other than the microframe pipeline. The embedded TT assumes the number of packets scheduled in a frame does not exceed the frame duration (1 msec) or else undefined behavior may result.

20.9.1.5.5 Multiple Transaction Translators

The maximum number of embedded transaction translators that is currently supported is one as indicated by the N_TT field in the HCSPARAMS register. See Section 20.3.1.3, "Host Controller Structural Parameters (HCSPARAMS)," for more information.

20.9.2 Device Operation

The co-existence of a device operational controller within the DR module has little effect on EHCI compatibility for host operation except as noted in this section.

20.9.3 Non-Zero Fields the Register File

Some of the reserved fields and reserved addresses in the capability registers and operational registers have use in device mode, the following must be adhered to:

- Write operations to all EHCI reserved fields (some of which are device fields in the DR module) in the operation registers should always be written to zero. This is an EHCI requirement of the device controller driver that must be adhered to.
- Read operations by the module must properly mask EHCI reserved fields (some of which are device fields in the DR module registers).

20.9.4 SOF Interrupt

The SOF interrupt is a free running 125 µsec interrupt for host mode. EHCI does not specify this interrupt, but it has been added for convenience and as a potential software time base. Note that the free running interrupt is shared with the device-mode start-of-frame interrupt. See Section 20.3.2.2, "USB Status Register (USBSTS)," and Section 20.3.2.3, "USB Interrupt Enable Register (USBINTR)," for more information.

20.9.5 Embedded Design

This is an Embedded USB Host Controller as defined by the EHCI specification and thus does not implement the PCI configuration registers.

20.9.5.1 Frame Adjust Register

Given that the optional PCI configuration registers are not included in this implementation, there is no corresponding bit level timing adjustments like those provided by the Frame Adjust register in the PCI configuration registers. Starts of microframes are timed precisely to 125 µsec using the transceiver clock as a reference clock. That is, 60 MHz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces or 30 MHz transceiver clock for 16-bit physical interfaces.

20.9.6 Miscellaneous Variations from EHCI

20.9.6.1 Programmable Physical Interface Behavior

The modules support multiple physical interfaces which can operate in different modes when the module is configured with the software programmable Physical Interface Modes. The control bits for selecting the PHY operating mode have been added to the PORTSC register providing a capability that is not defined by the EHCI specification.

20.9.6.2 Discovery

20.9.6.2.1 Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the register for a duration of 10 msec. Due to the complexity required to support the attachment of devices that are not high speed there are counter already present in the design that can count the 10 msec reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is then summarized as the following:

- [Port Change Interrupt] Port connect change occurs to notify the host controller driver that a device has attached.
- Software shall write a ‘1’ to the reset the device.
- Software shall write a ‘0’ to the reset the device after 10 msec.
  — This step, which is necessary in a standard EHCI design, may be omitted with this implementation. Should the EHCI host controller driver attempt to write a ‘0’ to the reset bit while a reset is in progress the write will simple be ignored and the reset will continue until completion.
- [Port Change Interrupt] Port enable change occurs to notify the host controller that the device in now operational and at this point the port speed has been determined.

20.9.6.2.2 Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation which will re-assign the port owner for any device that does not connect at High-Speed, this host controller supports direct attach of non-HS devices. Therefore, the following differences are important regarding port speed detection:

- Port owner is read-only and always reads 0.
- A 2-bit port speed indicator has been added to PORTSC to provide the current operating speed of the port to the host controller driver.
- A 1-bit high-speed indicator has been added to PORTSC to signify that the port is in HS vs. FS/LS

20.10 Timing Diagrams

This section contains diagrams showing the basic operation of the ULPI interface. For a more detailed description refer to the ULPI Specifications.

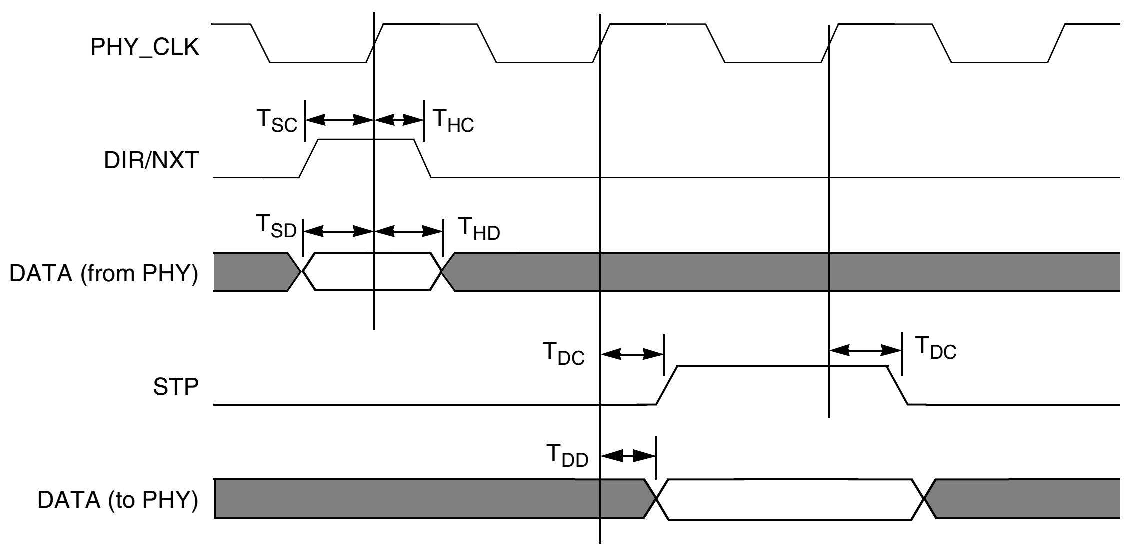


Figure 20-66. ULPI Timing

Table 20-99. ULPI Timing

Parameter	Symbol	Min	Max	Units
Control signal setup time	T_{SC}	—	4	ns
Data setup time	T_{SD}	—	4	ns
Control signal hold time	T_{HC}	0	—	ns
Data hold time	T_{HD}	0	—	ns
Control output delay	T_{DC}	2	7	ns
Data output delay	T_{DD}	2	7	ns

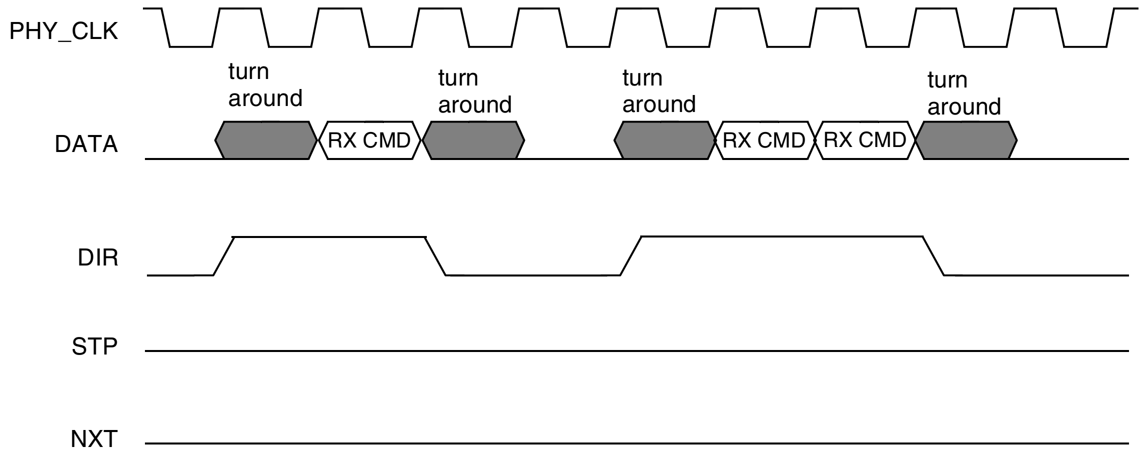


Figure 20-67. Sending of RX CMD

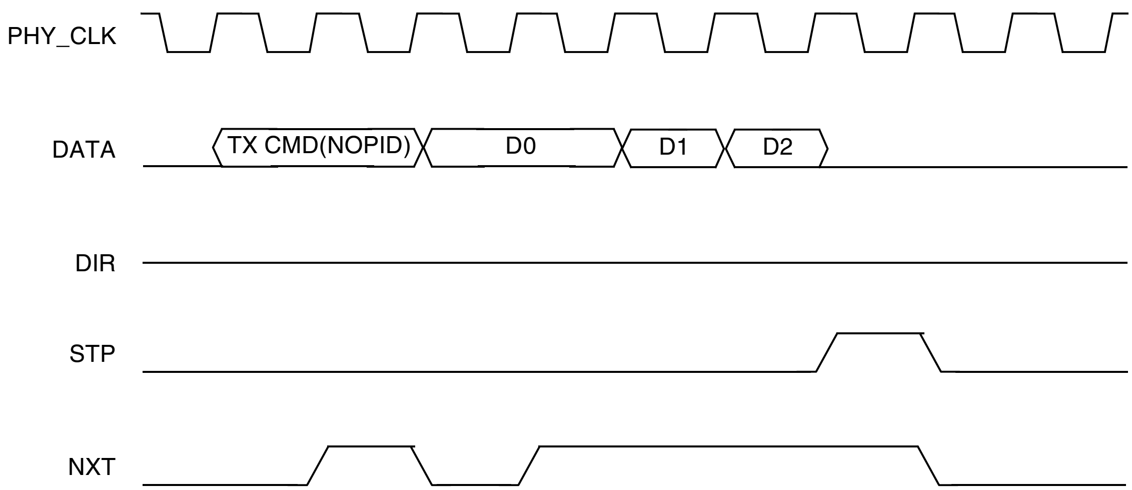


Figure 20-68. ULPI Data Transmit (NOPID)

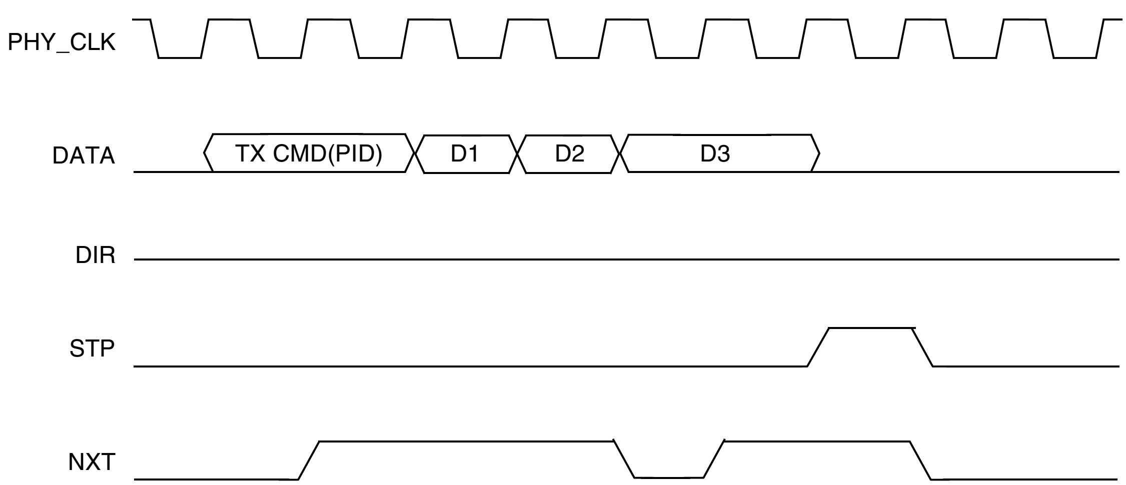


Figure 20-69. ULPI Data Transmit (PID)

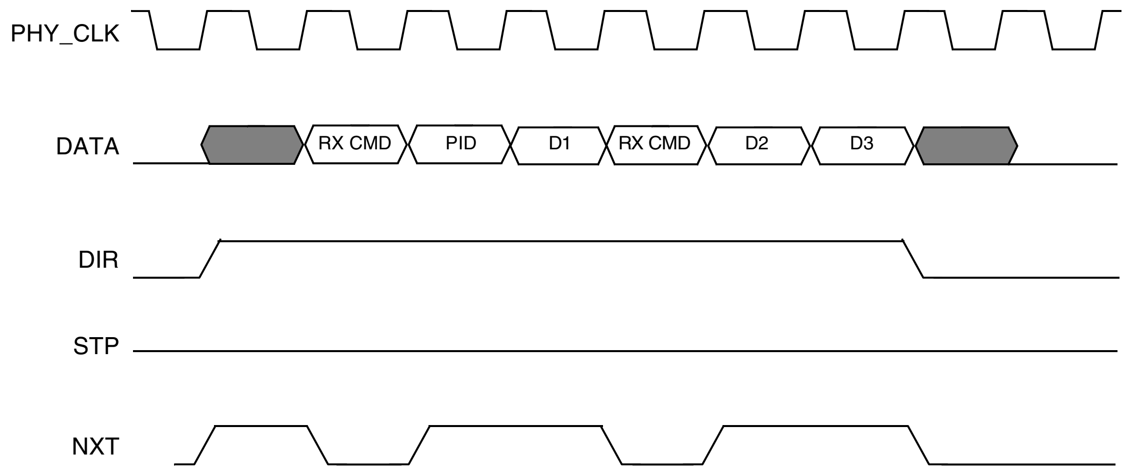


Figure 20-70. ULPI Data Receive

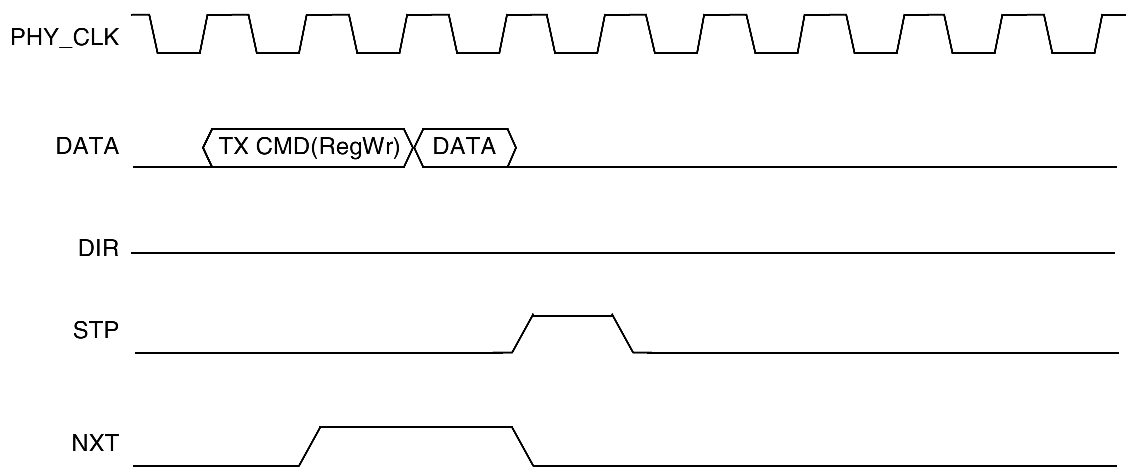


Figure 20-71. ULPI Register Write

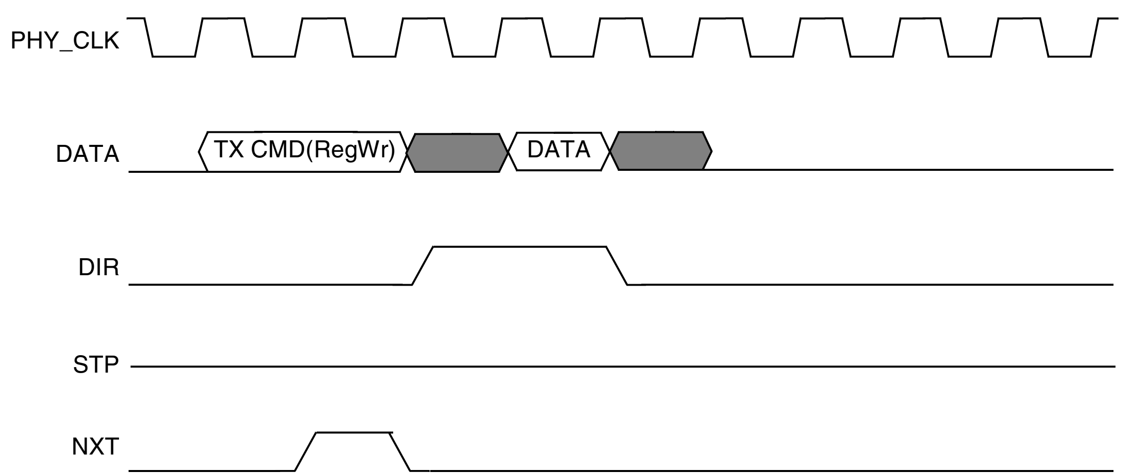


Figure 20-72. ULPI Register Read

Chapter 21
I²C Interfaces

This chapter describes the two inter-IC (IIC or I²C) bus interfaces implemented on this device. Note that for most intents, the I²C interfaces are identical and are described as a single generic controller. Where necessary, differences between the two controllers are noted.

21.1 Introduction

The inter-IC (IIC or I²C) bus is a two-wire—serial data (SDA) and serial clock (SCL)—bidirectional serial bus that provides a simple, efficient method of data exchange between this device and other devices, such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. Figure 21-1 shows a block diagram of the I²C interface.

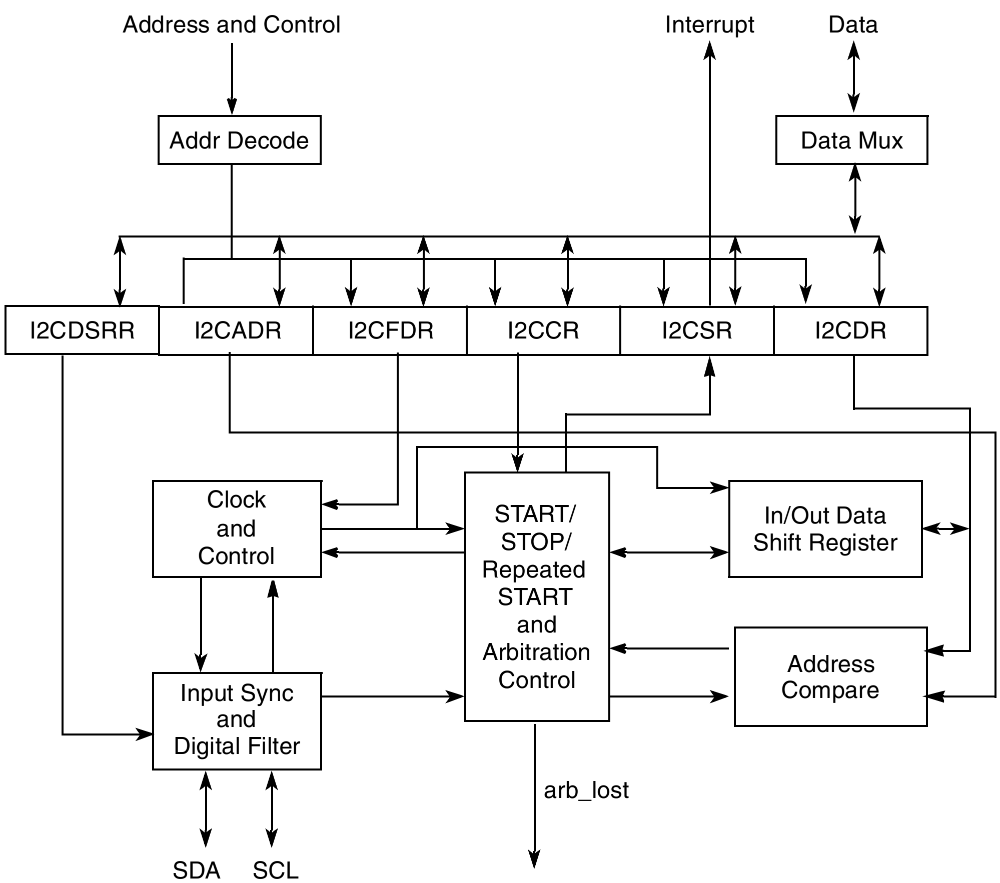


Figure 21-1. I²C Block Diagram

The two-wire I²C bus minimizes interconnections between devices. The synchronous, multiple-master I²C bus allows the connection of additional devices to the bus for expansion and system development. The bus includes collision detection and arbitration that prevent data corruption if two or more masters attempt to control the bus simultaneously.

21.1.1 Features

Each I^2C interface includes the following features:

- Two-wire interface
- Multiple-master operational
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP signal generation/detection
- Acknowledge bit generation/detection
- Bus busy detection
- Software-programmable clock frequency
- Software-selectable acknowledge bit
- On-chip filtering for spikes on the bus

21.1.2 Modes of Operation

The I^2C unit on this device can operate in one of the following modes:

- Master mode. The I^2C initiates a transfer, generates clock signals, and terminates a transfer. It cannot use its own slave address as a calling address. The I^2C cannot be a master and a slave simultaneously.
- Slave mode. The I^2C is addressed by an I^2C master. The module must be enabled before a START condition from an I^2C master is detected.
- Interrupt-driven byte-to-byte data transfer. When successful slave addressing is achieved (and SCL*n* returns to zero), the data transfer can proceed on a byte-to-byte basis in the direction specified by the R/$\overline{W}$ bit sent by the calling master. Each byte of data must be followed by an acknowledge bit, which is signaled from the receiving device. Several bytes can be transferred during a data transfer session.
- Boot sequencer mode. I^2C1 controller supports boot sequencer mode. This mode can be used to initialize the configuration registers in the device after the I^2C module is initialized. Boot sequencer mode is selected using the BOOTSEQ field in the reset configuration word high. Note that the hard-coded reset configuration word high value is boot sequencer mode disabled. This mode is not supported by I^2C2 controller.
- Reset configuration load (I^2C1 only). In this mode, the I^2C1 interface loads the reset configuration words from an EEPROM at a specific calling address while the rest of the device is in the reset state ($\overline{HRESET}$ asserted). Once the reset configuration words are latched inside the device, I^2C1 is reset until $\overline{HRESET}$ is negated. After $\overline{HRESET}$ is negated, the device may be initialized using boot sequence mode according to the BOOTSEQ field in the reset configuration word. See Section 21.4.5, "Boot Sequencer Mode."

Additionally, the following three I^{2}C–specific states are defined for the I^{2}C interface:

- START condition. This condition denotes the beginning of a new data transfer (each data transfer contains several bytes of data) and awakens all slaves.
- Repeated START condition. A START condition that is generated without a STOP condition to terminate the previous transfer.
- STOP condition. The master can terminate the transfer by generating a STOP condition to free the bus.

21.2 External Signal Descriptions

The following sections give an overview of signals and provide detailed signal descriptions.

21.2.1 Signal Overview

The I^{2}C interface uses the SDA*n* and SCL*n* signals, described in Table 21-1, for data transfer. Note that the signal patterns driven on SDA*n* represent address, data, or read/write information at different stages of the protocol.

Table 21-1. I^{2}C Interface Signal Descriptions

Signal Name	Idle State	I/O	State Meaning
Serial Clock (SCL1, SCL2)	High	I	When the I^{2}C module is idle or acts as a slave, SCL*n* defaults as an input. The unit uses SCL*n* to synchronize incoming data on SDA*n*. The bus is assumed to be busy when SCL*n* is detected low.
		O	As a master, the I^{2}C module drives SCL*n* along with SDA*n* when transmitting. As a slave, the I^{2}C module drives SCL*n* negates for data pacing.
Serial Data (SDA1, SDA2)	High	I	When the I^{2}C module is idle or in a receiving mode, SDA*n* defaults as an input. The unit receives data from other I^{2}C devices on SDA*n*. The bus is assumed to be busy when SDA*n* is detected low.
		O	When writing as a master or slave, the I^{2}C module drives data on SDA*n* synchronous to SCL*n*.

21.2.2 Detailed Signal Descriptions

SDA*n* and SCL*n*, described in Table 21-2, serve as a communication interconnect with other devices. All devices connected to these signals must have open-drain or open-collector outputs. The logic AND function is performed on both of these signals with external pull-up resistors. Refer to the hardware specifications for electrical characteristics.

Table 21-2. I^{2}C Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
SCL1, SCL2	I/O	Serial clock. Performs as an input when the device is programmed as an I^{2}C slave. SCL*n* also performs as an output when the device is programmed as an I^{2}C master.	
	O	As outputs for the bidirectional serial clock, these signals operate as described below.	
		State Meaning	Asserted/Negated—Driven along with SDA*n* as the clock for the data.
	I	As inputs for the bi-directional serial clock, these signals operate as described below.	
		State Meaning	Asserted/Negated—The I^{2}C unit uses this signal to synchronize incoming data on SDA*n*. The bus is assumed to be busy when this signal is detected low.
SDA1, SDA2	I/O	Serial data. Performs as an input when the device is in a receiving mode. SDA*n* also performs as an output signal when the device is transmitting (as an I^{2}C master or a slave).	
	O	As outputs for the bi-directional serial data, these signals operate as described below.	
		State Meaning	Asserted/Negated—Data is driven.
	I	As inputs for the bi-directional serial data, these signals operate as described below.	
		State Meaning	Asserted/Negated—Used to receive data from other devices. The bus is assumed to be busy when SDA*n* is detected low.

21.3 Memory Map/Register Definition

Table 21-3 lists the I^{2}C-specific registers and their addresses.

Table 21-3. I^{2}C Memory Map

Address	I^{2}C Register	Access	Reset	Section/Page
0x0_3000	I2C1ADR—I^{2}C1 address register	R/W	0x00	21.3.1.1/21-5
0x0_3004	I2C1FDR—I^{2}C1 frequency divider register	R/W	0x00	21.3.1.2/21-6
0x0_3008	I2C1CR—I^{2}C1 control register	R/W	0x00	21.3.1.3/21-7
0x0_300C	I2C1SR—I^{2}C1 status register	R/W	0x81	21.3.1.4/21-8
0x0_3010	I2C1DR—I^{2}C1 data register	R/W	0x00	21.3.1.5/21-9
0x0_3014	I2C1DFSRR—I^{2}C1 digital filter sampling rate register	R/W	0x10	21.3.1.6/21-9
0x0_301C–0x0_30FF	Reserved, should be cleared	—	—	—
0x0_3100	I2C2ADR—I^{2}C2 address register	R/W	0x00	21.3.1.1/21-5
0x0_3104	I2C2FDR—I^{2}C2 frequency divider register	R/W	0x00	21.3.1.2/21-6
0x0_3108	I2C2CR—I^{2}C2 control register	R/W	0x00	21.3.1.3/21-7
0x0_310C	I2C2SR—I^{2}C2 status register	R/W	0x81	21.3.1.4/21-8

Table 21-3. I^2C Memory Map (continued)

Address	I^2C Register	Access	Reset	Section/Page
0x0_3110	I2C2DR—I^2C2 data register	R/W	0x00	21.3.1.5/21-9
0x0_3114	I2C2DFSRR—I^2C2 digital filter sampling rate register	R/W	0x10	21.3.1.6/21-9
0x0_311C–0x0_31FF	Reserved, should be cleared	—	—	—

21.3.1 Register Descriptions

This section describes the I^2C registers in detail. Note that reserved bits should always be written with the value they return when read. That is, the register should be programmed by reading the value, modifying appropriate fields, and writing back the value. The return value of the reserved fields should not be assumed, even though the reserved fields return zero. This does not apply to the I^2Cn data register (I2C*n*DR).

21.3.1.1 I^2Cn Address Register (I2C*n*ADR)

Figure 21-2 shows the I2C*n*ADR register, which contains the address to which the I^2C interface responds when addressed as a slave. Note that this is not the address that is sent on the bus during the address-calling cycle when the I^2C module is in master mode.

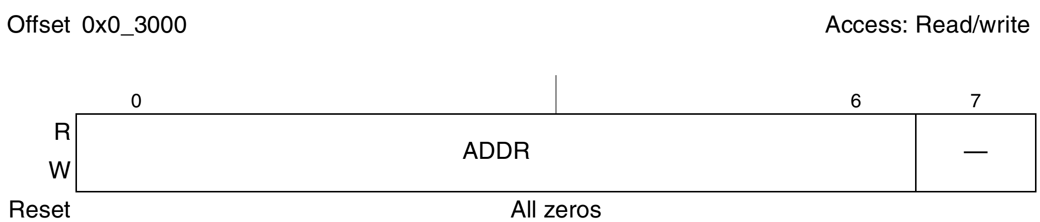


Figure 21-2. I^2Cn Address Register (I2C*n*ADR)

Table 21-4 describes the bit settings of I2C*n*ADR.

Table 21-4. I2C*n*ADR Field Descriptions

Bits	Name	Description
0–6	ADDR	Slave address. Contains the specific slave address that is used by the I^2C interface. Note that the default mode of the I^2C interface is slave mode for an address match. Note that an address match is one of the conditions that can cause I2C*n*SR[MIF] to be set, signaling an interrupt pending condition.
7	—	Reserved, should be cleared

21.3.1.2 I²C*n* Frequency Divider Register (I2C*n*FDR)

Figure 21-3 shows the bits of the I²C*n* frequency divider register.

Offset 0x0_3004 — Access: Read/write

	0–1	2–7
R / W	—	FDR
Reset	All zeros	

Figure 21-3. I²C*n* Frequency Divider Register (I2C*n*FDR)

Table 21-5 describes the bit settings of I2C*n*FDR. It also maps I2C*n*FDR[FDR] to the clock divider values. Although it describes the ratio between the I²C controller internal clock and SCL, the default ratio of I²C controller clock and CSB is 1:3 (I²C controller clock frequency is three times slower than CSB clock frequency). This ratio is set in SCCR[SDHCCM]. Clock ratios of I²C1 are controllable but clock ratio for I²C2 is not and it is always 1:1 with CSB. Consider this factor when selecting an FDR value.

Table 21-5. I2C*n* FDR Field Descriptions

Bits	Name	Description
0–1	—	Reserved, should be cleared
2–7	FDR	Frequency divider ratio. Used to prescale the clock for bit-rate selection. The serial bit clock frequency of SCL*n* is equal to the I²C*n* controller clock divided by the divider. The serial bit clock frequency divider selections are described as follows:

FDR	Divider (Decimal)	FDR	Divider (Decimal)	FDR	Divider (Decimal)
0x00	384	0x16	12288	0x2B	1024
0x01	416	0x17	15360	0x2C	1280
0x02	480	0x18	18432	0x2D	1536
0x03	576	0x19	20480	0x2E	1792
0x04	640	0x1A	24576	0x2F	2048
0x05	704	0x1B	30720	0x30	2560
0x06	832	0x1C	36864	0x31	3072
0x07	1024	0x1D	40960	0x32	3584
0x08	1152	0x1E	49152	0x33	4096
0x09	1280	0x1F	61440	0x34	5120
0x0A	1536	0x20	256	0x35	6144
0x0B	1920	0x21	288	0x36	7168
0x0C	2304	0x22	320	0x37	8192
0x0D	2560	0x23	352	0x38	10240
0x0E	3072	0x24	384	0x39	12288
0x0F	3840	0x25	448	0x3A	14336
0x10	4608	0x26	512	0x3B	16384
0x11	5120	0x27	576	0x3C	20480
0x12	6144	0x28	640	0x3D	24576
0x13	7680	0x29	768	0x3E	28672
0x14	9216	0x2A	896	0x3F	32768
0x15	10240				

Note: I2C controller clock of I2C1 is derived from csb_clk / SCCR[SDHCCM].

Note: The value's shown in the table are applicable only for the default value of DFSRR. Refer to AN2919.

21.3.1.3 I^{2}C*n* Control Register (I2C*n*CR)

Figure 21-4 shows the I^{2}C*n* control register.

Offset 0x0_3008 Access: Mixed

	0	1	2	3	4	5	6	7
R	MEN	MIEN	MSTA	MTX	TXAK		—	BCST
W						RSTA		
Reset	All zeros							

Figure 21-4. I^{2}C*n* Control Register (I2C*n*CR)

Table 21-6 describes the I2C*n*CR bit settings.

Table 21-6. I2C*n*CR Field Descriptions

Bits	Name	Description
0	MEN	Module enable. Controls the software reset of the I^{2}C module. 0 The module is reset and disabled. The interface is held in reset, but the registers can still be accessed. 1 The I^{2}C module is enabled. MEN must be set before any other control register bits have any effect. All I^{2}C registers for slave receive or master START can be initialized before setting this bit.
1	MIEN	Module interrupt enable 0 Interrupts from the I^{2}C module are disabled. This does not clear any pending interrupt conditions. 1 Interrupts from the I^{2}C module are enabled. An interrupt occurs provided I2C*n*SR[MIF] is also set.
2	MSTA	Master/slave mode START 0 On a transition to zero, a STOP condition is generated and the mode changes from master to slave. Cleared without generating a STOP condition when the master loses arbitration. 1 When MSTA changes from zero to one, a START condition is generated on the bus and master mode is selected.
3	MTX	Transmit/receive mode select. Selects the direction of the master and slave transfers. When configured as a slave, this bit should be set by software according to I2C*n*SR[SRW]. In master mode, the bit should be set according to the type of transfer required. Therefore, for address cycles, this bit will always high. MTX is cleared when the master loses arbitration. 0 Receive mode 1 Transmit mode
4	TXAK	Transfer acknowledge. Specifies the value driven onto the SDA*n* line during acknowledge cycles for both master and slave receivers. The value of this bit applies only when the I^{2}C module is configured as a receiver, not a transmitter. It also does not apply to address cycles; when the device is addressed as a slave, an acknowledge is always sent. 0 An acknowledge signal (low value on SDA*n*) is sent out to the bus at the 9th clock bit after receiving one byte of data. 1 No acknowledge signal response (high value on SDA*n*) is sent.
5	RSTA	Repeated START. Note that this bit is not readable, which means if a read is performed to RSTA, a zero value is returned. 0 No START condition is generated 1 Setting this bit always generates a repeated START condition on the bus, provides the device with the current bus master. Attempting a repeated START at the wrong time (or if the bus is owned by another master), results in loss of arbitration.
6	—	Reserved, should be cleared
7	BCST	Broadcast 0 Disables the broadcast accept capability 1 Enables the I^{2}C to accept broadcast messages at address zero

21.3.1.4 I^{2}C*n* Status Register (I2C*n*SR)

I2CSR is shown in Figure 21-5.

Offset 0x0_300C — Access: Mixed

	0	1	2	3	4	5	6	7
R	MCF	MAAS	MBB	MAL	BCSTM	SRW	MIF	RXAK
W				MAL			MIF	
Reset	1	0	0	0	0	0	0	1

Figure 21-5. I^{2}C*n* Status Register (I2C*n*SR)

Table 21-7 describes the bit settings of the I2C*n*SR.

Table 21-7. I2C*n*SR Field Descriptions

Bits	Name	Description
0	MCF	Data transfer. When one byte of data is transferred, the bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. 0 Byte transfer in progress. MCF is cleared under the following conditions: • When I2C*n*DR is read in receive mode or when I2C*n*DR is written in transmit mode. • After a start sequence is recognized by the I^{2}C controller in slave mode. 1 Byte transfer is completed
1	MAAS	Addressed as a slave. When the value in I2C*n*ADR matches the calling address or when the calling address is the broadcast address and broadcast mode is enabled (I2C*n*CR[BCST] is set), this bit is set. The processor is interrupted if I2C*n*CR[MIEN] is set. Next, the processor must check the SRW bit and set I2C*n*CR[MTX] accordingly. Writing to the I2C*n*CR automatically clears this bit. 0 Not addressed as a slave 1 Addressed as a slave
2	MBB	Bus busy. Indicates the status of the bus. When a START condition is detected, MBB is set. If a STOP condition is detected, it is cleared. 0 I^{2}C bus is idle 1 I^{2}C bus is busy
3	MAL	Arbitration lost. Automatically set when the arbitration procedure is lost. Note that the device does not automatically retry a failed transfer attempt. 0 Arbitration is not lost. Can only be cleared by software 1 Arbitration is lost
4	BCSTM	Broadcast match. Writing to the I2C*n*CR automatically clears this bit. 0 There has not been a broadcast match. 1 The calling address matches with the broadcast address and broadcast mode is enabled. This is also set if this I^{2}C drives an address of all 0s.
5	SRW	Slave read/write. When MAAS is set, SRW indicates the value of the R/W command bit of the calling address, which is sent from the master. 0 Slave receive, master writing to slave 1 Slave transmit, master reading from slave. This bit is valid only when both of the following conditions are true: • A complete transfer occurred and no other transfers have been initiated. • The I^{2}C interface is configured as a slave and has an address match. By checking SRW, the processor can select slave transmit/receive mode according to the command of the master.

Table 21-7. I2C*n*SR Field Descriptions (continued)

Bits	Name	Description
6	MIF	Module interrupt. The MIF bit is set when an interrupt is pending, causing a processor interrupt request (provided I2C*n*CR[MIEN] is set). 0 No interrupt is pending. Can be cleared only by software. 1 Interrupt is pending. MIF is set when one of the following events occurs: • One byte of data is transferred (set at the falling edge of the 9th clock). • The value in I2C*n*ADR matches with the calling address in slave-receive mode. • Arbitration is lost.
7	RXAK	Received acknowledge. The value of SDA*n* during the reception of acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates that an acknowledge signal has been received after the completion of eight bits of data transmission on the bus. If RXAK is high, it means no acknowledge signal has been detected at the 9th clock. 0 Acknowledge received 1 No acknowledge received

21.3.1.5 I²C*n* Data Register (I2C*n*DR)

The I2C*n* data register is shown in Figure 21-6.

Offset 0x0_3010 Access: Read/write

	0 – 7
R W	DATA
Reset	All zeros

Figure 21-6. I²C*n* Data Register (I2C*n*DR)

Table 21-8 shows the bit descriptions for I2C*n*DR.

Table 21-8. I2C*n*DR Field Descriptions

Bits	Name	Description
0–7	DATA	Transmission starts when an address and the R/W bit are written to the data register and the I²C interface performs as the master. A data transfer is initiated when data is written to the I2C*n*DR. The most-significant bit is sent first in both cases. In master receive mode, reading the data register allows the read to occur, but also allows the I²C module to receive the next byte of data on the I²C interface. In slave mode, the same function is available after it is addressed. Note that in both master receive and slave receive modes, the very first read is always a dummy read.

21.3.1.6 Digital Filter Sampling Rate Register (I2C*n*DFSRR)

I2C*n*DFSRR is shown in Figure 21-7.

Offset 0x0_3014 Access: Read/write

	0	1	2					7
R W	—		DFSR					
Reset	0	0	0	1	0	0	0	0

Figure 21-7. I²C*n* Digital Filter Sampling Rate Register (I2C*n*DFSRR)

Table 21-9 shows the I2C*n*DFSRR field descriptions.

Table 21-9. I2C*n*DFSRR Field Descriptions

Bits	Name	Description
0–1	—	Reserved, should be cleared
2–7	DFSR	Digital filter sampling rate. To assist in filtering out signal noise, the sample rate is programmed. DFSR is used to prescale the frequency at which the digital filter takes samples from the I^{2}C bus. The resulting sampling rate is calculated by dividing the platform frequency by the non-zero value of DFSR. If I2C*n*DFSRR is cleared, the I^{2}C bus sample points default to the reset divisor 0x10.

21.4 Functional Description

The I^{2}C unit always performs as a slave receiver as a default, unless explicitly programmed to be a master or slave transmitter. If boot sequencer mode is selected, the I^{2}C interface performs as a slave receiver after the boot sequence has completed.

21.4.1 Transaction Protocol

A standard I^{2}C transfer consists of the following:

- START condition
- Slave target address transmission
- Data transfer
- STOP condition

Figure 21-8 shows the interaction of these four parts with the calling address, data byte, and new calling address components of the I^{2}C protocol. The details of the protocol are described in the following subsections.

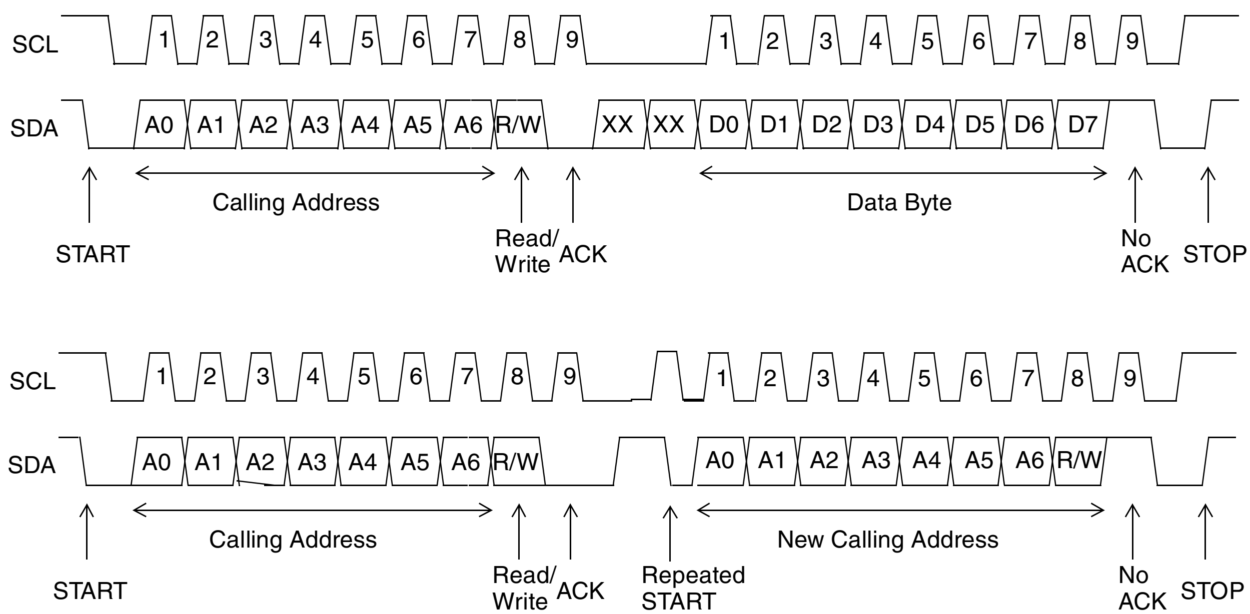


Figure 21-8. I^{2}C Interface Transaction Protocol

21.4.1.1 START Condition

When the I^{2}C bus is not engaged (both SDA*n* and SCL*n* lines are at logic high), a master can initiate a transfer by sending a START condition. As shown in Figure 21-8, a START condition is defined as a high-to-low transition of SDA*n* while SCL*n* is high. This condition denotes the beginning of a new data transfer. Each data transfer can contain several bytes and awakens all slaves. The START condition is initiated by a software write that sets I2C*n*CR[MSTA].

21.4.1.2 Slave Address Transmission

The first byte of data transferred by the master immediately after the START condition is the slave address. This is a seven-bit calling address followed by a $R/\overline{W}$ bit, which indicates the direction of the data transferred to the slave. Each slave in the system has a unique address. When the I^{2}C module is operating as a master, it must not transmit an address that is the same as its slave address. An I^{2}C device cannot be master and slave at the same time.

Only the slave with a calling address that matches the one transmitted by the master responds by returning an acknowledge bit (negating the SDA*n* signal at the 9th clock) as shown in Figure 21-8. If no slave acknowledges the address, the master should generate a STOP condition or a repeated START condition.

When slave addressing is successful (and SCL*n* returns to zero), the data transfer can proceed on a byte-to-byte basis in the direction specified by the $R/\overline{W}$ bit sent by the calling master.

The I^{2}C module responds to a general call (broadcast) command when I2C*n*CR[BCST] is set. A broadcast address is always zero; however the I^{2}C module does not check the R/W bit. The second byte of the broadcast message is the master address. Because the second byte is automatically acknowledged by hardware, the receiver device software must verify that the broadcast message is intended for itself by reading the second byte of the message. If the master address is for another receiver device and the third byte is a write command, software can ignore the third byte during the broadcast. If the master address is for another receiver device and the third byte is a read command, software must write 0xFF to I2C*n*DR with I2C*n*CR[TXAK] = 1, so that it does not interfere with the data written from the addressed device.

Each data byte is 8 bits long. Data bits can be changed only while SCL*n* is low and must be held stable while SCL*n* is high, as shown in Figure 21-8. There is one clock pulse on SCL*n* for each data bit, and the most significant bit (msb) is transmitted first. Each byte of data must be followed by an acknowledge bit, which is signaled from the receiving device by pulling SDA*n* low at the 9th clock. Therefore, one complete data byte transfer takes 9 clock pulses. Several bytes can be transferred during a data transfer session.

If the slave receiver does not acknowledge the master, the SDA*n* line must be left high by the slave. The master can then generate a stop condition to abort the data transfer or a START condition (repeated START) to begin a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte of transmission, the slave interprets that the end-of-data has been reached. Then the slave releases the SDA*n* line for the master to generate a STOP or a START condition.

21.4.1.3 Repeated START Condition

Figure 21-8 shows a repeated START condition, which is generated without a STOP condition that can terminate the previous transfer. The master uses this method to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

21.4.1.4 STOP Condition

The master can terminate the transfer by generating a STOP condition to free the bus. A STOP condition is defined as a low-to-high transition of the SDA*n* signal while SCL*n* is high. For more information, see Figure 21-8. Note that a master can generate a STOP even if the slave has transmitted an acknowledge bit, at which point the slave must release the bus. The STOP condition is initiated by a software write that clears I2C*n*CR[MSTA].

As described in Section 21.4.1.3, "Repeated START Condition," the master can generate a START condition followed by a calling address without generating a STOP condition for the previous transfer. This is called a repeated START condition.

21.4.1.5 Protocol Implementation Details

The following sections give details of how aspects of the protocol are implemented in the I^2C module.

21.4.1.5.1 Transaction Monitoring—Implementation Details

The different conditions of the I^2C data transfers are monitored as follows (see Figure 21-8):

- START conditions are detected when an SDA*n* fall occurs while SCL*n* is high.
- STOP conditions are detected when an SDA*n* rise occurs while SCL*n* is high.
- Data transfers in progress are canceled when a STOP condition is detected or if there is a slave address mismatch. Cancellation of data transactions resets the clock module.
- The bus is detected to be busy upon the detection of a START condition and idle upon the detection of a STOP condition.

21.4.1.5.2 Control Transfer—Implementation Details

The I^2C module contains logic that controls the output to the serial data (SDA) and serial clock (SCL) lines of the I^2C. The SCL*n* output is pulled low as determined by the internal clock generated in the clock module. The SDA*n* output can change only at the midpoint of a low cycle of the SCL*n*, unless it is performing a START, STOP, or repeated START condition. Otherwise, the SDA*n* output is held constant.

SDA*n* is negated when one or more of the following conditions are true:

- Master mode
  - — Data bit (transmit)
  - — ACK bit (receive)
  - — START condition
  - — STOP condition
  - — Repeated START condition

- Slave mode
  - — Acknowledging address match
  - — Data bit (transmit)
  - — ACK bit (receive)

The SCL*n* signal corresponds to the internal SCL*n* signal when one or more of the following conditions are true in either master or slave mode:

- Master mode
  - — Bus owner
  - — Lost arbitration
  - — START condition
  - — STOP condition
  - — Repeated START condition begin
  - — Repeated START condition end
- Slave mode
  - — Address cycle
  - — Transmit cycle
  - — ACK cycle

21.4.1.6 Address Compare—Implementation Details

The address compare block determines whether a slave has been properly addressed, either by its slave address or by the general broadcast address (which addresses all slaves). The following address comparisons are performed:

- Whether a broadcast message has been received, to update I2C*n*SR
- Whether the module has been addressed as a slave, to update I2C*n*SR and to generate an interrupt
- Whether the address transmitted by the current master matches the general broadcast address

21.4.2 Arbitration Procedure

The I^{2}C interface is a true multiple-master bus. If two or more masters simultaneously try to control the bus, each master's clock synchronization procedure (including the I^{2}C module) determines the bus clock—the low period is equal to the longest clock-low period and the high is equal to the shortest one among the masters. A bus master loses arbitration if it transmits a logic 1 on SDA*n* while another master transmits a logic 0. The losing masters immediately switch to slave-receive mode and stop driving the SDA*n* line. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, the I^{2}C unit sets I2C*n*SR[MAL] to indicate the loss of arbitration and, as a slave, services the transaction if it is directed to itself.

If the I^{2}C module is enabled in the middle of an ongoing byte transfer, the interface behaves as follows:

- Slave mode—the I^{2}C module ignores the current transfer on the bus and starts operating whenever a subsequent START condition is detected.

- Master mode—the I^{2}C module cannot tell whether the bus is busy; therefore, if a START condition is initiated, the current bus cycle can be corrupted. This ultimately causes in the current bus master to lose arbitration, after which bus operations return to normal.

21.4.2.1 Arbitration Control

The arbitration control block controls the arbitration procedure of the master mode. A loss of arbitration occurs whenever the master detects a 0 on the external SDA*n* line while attempting to drive a 1, tries to generate a START or repeated START at an inappropriate time, or detects an unexpected STOP request on the line.

In master mode, arbitration by the master is lost (and I2C*n*SR[MAL] is set) under the following conditions:

- SDA*n* samples low when the master drives high during an address or data-transmit cycle (transmit).
- SDA*n* samples low when the master drives high during a data-receive cycle of the acknowledge (ACK) bit (receive).
- A START condition is attempted when the bus is busy.
- A repeated START condition is requested in slave mode.
- A repeated START condition is attempted when the requesting device is not the bus owner
- Unexpected STOP condition detected

Note that the I^{2}C module does not automatically retry a failed transfer attempt.

21.4.3 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices can hold SCL*n* low after completion of a 1-byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait states until the slave releases the SCL*n* line.

21.4.4 Clock Control

The clock control block handles requests from the clock signal for transferring and controlling data for multiple tasks.

A 9-cycle data transfer clock is requested for the following conditions:

- Master mode
  - Transmit slave address after START condition
  - Transmit slave address after repeated START condition
  - Transmit data
  - Receive data
- Slave mode
  - Transmit data
  - Receive data
  - Receive slave address after START or repeated START condition

21.4.4.1 Clock Synchronization

Due to the wire AND logic on the SCL*n* line, a high-to-low transition on the SCL*n* line affects all devices connected on the bus. The devices begin counting their low period when the master negates the SCL*n* line. After a device has negated SCL*n*, it holds the SCL*n* line low until the clock high state is reached. However, the change of low-to-high in a device clock may not change the state of SCL*n* if another device is still within its low period. Therefore, SCL*n* is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time. When all devices concerned have counted off their low periods, SCL*n* is released and asserted. Then there is no difference between the devices' clocks and the state of SCL*n*, and all the devices begin counting their high periods. The first device to complete its high period negates SCL*n* again.

21.4.4.2 Input Synchronization and Digital Filter

The following sections describes synchronization of the input signals and the filtering of SCL*n* and SDA*n* in detail.

21.4.4.2.1 Input Signal Synchronization

The input synchronization block synchronizes the input SCL*n* and SDA*n* signals to the system clock and detects transitions of these signals.

21.4.4.2.2 Filtering of SCL*n* and SDA*n* Lines

The SCL*n* and SDA*n* inputs are filtered to eliminate noise. Three consecutive samples of the SCL*n* and SDA*n* lines are compared to a pre-determined sampling rate. If they are all high, the output of the filter is high. If they are all low, the output is low. If they are any combination of highs and lows, the output is whatever the value of the line was in the previous clock cycle.

The sampling rate is equal to a binary value stored in the frequency register I2CDFSRR. The duration of the sampling cycle is controlled by a down counter. This allows a software write to the I2CDFSRR to control the filtered sampling rate.

21.4.4.3 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven SCL*n* low, the slave can drive SCL*n* low for the required period and then release it. If the slave SCL*n* low period is greater than the master SCL*n* low period, the resulting SCL*n* low period is extended.

21.4.5 Boot Sequencer Mode

Boot sequencer mode is selected at power-on reset by the BOOTSEQ field of the high-order reset configuration word. If boot sequencer mode is selected, the I²C module communicates with one or more EEPROMs through the I²C interface. EEPROMs can be programmed to initialize one or more configuration registers. Note that as described in Section 4.3.2.2.3, "Boot Sequencer Configuration," the default value for BOOTSEQ is 0b00, which corresponds to the I²C boot sequencer being disabled at power-up.

Boot sequencer mode also supports an extension of the standard I^2C interface that uses more address bits to allow for EEPROM devices that have more than 256 bytes. This extended addressing mode is selectable using a different encoding in the BOOTSEQ field of the high-order reset configuration word (see Section 4.3.2.2.3, “Boot Sequencer Configuration.”) In this mode, only one EEPROM device can be used and the maximum number of registers is limited by the size of the EEPROM.

If the standard I^2C interface is used, the I^2C module addresses the first EEPROM, and reads 256 bytes. Then it issues a repeated start and addresses the next EEPROM address. This sequence continues until the CONT bit is cleared. If the last register is not detected before wrapping back to the first address, an error condition is detected. In other words, if the CONT bit for not cleared on the final 7 bytes, an error condition is detected, causing the I^2C controller to hang. The I^2C module continues to read from the EEPROM as long as the continue (CONT) bit is set in the EEPROM. The CONT bit resides in the address/attributes field that is transferred from the EEPROM, as described in Section 21.4.5.2, “EEPROM Calling Address.” There should be no other I^2C traffic when the boot sequencer is active.

21.4.5.1 Using the Boot Sequencer for Reset Configuration

The reset configuration word can be loaded by using the I^2C boot sequencer. See Section 4.3.2.2.3, “Boot Sequencer Configuration.”

Note that this usage does not prevent using the I^2C boot sequencer to initiate the device in the normal functional mode, after reset state has completed. However, an I^2C serial EEPROM of extended addressing type must be used and the first two EEPROM data structures must contain dedicated reset information.

21.4.5.2 EEPROM Calling Address

The EEPROM calling address is 0b101_0000. The first EEPROM to be addressed must be programmed to respond to this address, or an error is generated. Any additional EEPROMs are addressed in sequential order.

21.4.5.3 EEPROM Data Format

The I^{2}C module expects a particular format for data to be programmed in the EEPROM. Figure 21-9 shows an example of the EEPROM contents, including the preamble, data format, and CRC.

0	1	2	3	4	5	6	7	
1	0	1	0	1	0	1	0	Preamble
0	1	0	1	0	1	0	1	
1	0	1	0	1	0	1	0	
ACS	BYTE_EN				1	ADDR[12:13]		First Configuration Preload Command
ADDR[14:21]								
ADDR[22:29]								
DATA[0:7]								
DATA[8:15]								
DATA[16:23]								
DATA[24:31]								

ACS	BYTE_EN				1	ADDR[12:13]		Second Configuration Preload Command
ADDR[14:21]								
ADDR[22:29]								
DATA[0:7]								
DATA[8:15]								
DATA[16:23]								
DATA[24:31]								
....								
ACS	BYTE_EN				1	ADDR[12:13]		Last Configuration Preload Command
ADDR[14:21]								
ADDR[22:29]								
DATA[0:7]								
DATA[8:15]								
DATA[16:23]								
DATA[24:31]								
0	0	0	0	0	0	0	0	End Command
0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	
CRC[0:7]								
CRC[8:15]								
CRC[16:23]								
CRC[24:31]								

Figure 21-9. EEPROM Contents

- A preamble should be the first 3 bytes programmed into the EEPROM. It should have a value of 0xAA55AA. The I²C checks to ensure that this preamble is correctly detected before proceeding.

- Following the preamble, there should be a series of configuration registers (known as register preloads). Each configuration register should be programmed according to a particular format, as shown in Figure 21-10.

0	1	2	3	4	5	6	7
ACS	BYTE_EN				CONT	ADDR[12:13]	
ADDR[14:21]							
ADDR[12:29]							
DATA[0:7]							
DATA[8:15]							
DATA[16:23]							
DATA[24:31]							

Figure 21-10. EEPROM Data Format for One Register Preload Command

— The first byte holds alternate configuration space (ACS), byte enables, and continue (CONT) attributes.

— The 2 least-significant bits of the address are derived from the byte enables. address offset. Therefore, the address offset programmed into the EEPROM preload should be a word offset.

— The most significant 16 bits (assuming 36-bit addressing) of the address are prepended from either IMMRRBAR or alternate configuration space.

— After the first 3 bytes, 4 bytes of data should hold the desired value of the configuration register, regardless of the size of transaction.

Byte enables should be asserted for any byte that will be written, and they should be asserted contiguously, creating a 1, 2, or 4 byte write to a register. The boot sequencer assumes that a big-endian address is stored in the EEPROM. In addition, byte enable bit 0 (bit 1 of the byte) corresponds to the most-significant byte of data (data[0:7]), and byte enable bit 3 (bit 4 of the byte) corresponds to the least-significant byte of data (data[24:31]).

By asserting ACS, an alternate configuration space address is prepended to the write request from the boot sequencer according to the value in the ALTCBAR register. This will allow for external memories to be configured. Otherwise, IMMRBAR is prepended to the EEPROM address.

If the CONT bit is cleared, the first 3 bytes, including ACS, the byte enables, and the address, should be cleared 0. Also, the data contains the final CRC. A CRC-32 algorithm is used to check the integrity of the data. The following polynomial is used:

$$1 + x^1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$$

The CRC should cover all bytes stored in the EEPROM before the CRC. This includes the preamble, all register preloads, and the first 3 bytes of the last 7-byte preload (which should be all zeros).

21.4.5.4 Boot Sequencer Done Indication

Dedicated hardware is not provided to indicate whether the boot sequencer operation completed successfully. It is recommended to use one of the GPIO signals for that purpose. To do this, the last register preload programmed into the EEPROM should contain the address of the appropriate GPIO register and data that causes the setting of the required GPIO signal. The GPIO signal may be used for an external device or for debug purposes.

21.5 Initialization/Application Information

This section describes some programming guidelines recommended for the I^{2}C interface. Figure 21-11 is a recommended flowchart for I^{2}C interrupt service routines.

A **sync** assembly instruction must be executed after each I^{2}C register read/write access to guarantee that register accesses occur in order.

The I^{2}C controller does not guarantee its recovery from all illegal I^{2}C bus activity. In addition, a malfunctioning device may hold the bus captive. A good programming practice is for software to rely on a watchdog timer to help recover from I^{2}C bus hangs. The recovery routine should also handle the case when the illegal I^{2}C bus behavior causes the status bits returned after an interrupt to be inconsistent with what was expected.

21.5.1 Interrupt Service Routine Flowchart

Figure 21-11 shows an example algorithm for an I^{2}C interrupt service routine. Deviation from the flowchart may result in unpredictable I^{2}C bus behavior. However, in the slave receive mode (not shown), the interrupt service routine may need to set I2C*n*CR[TXAK] when the next-to-last byte is to be accepted.

It is recommended that a **sync** instruction follow each I²C register read or write to guarantee that register accesses occur in order.

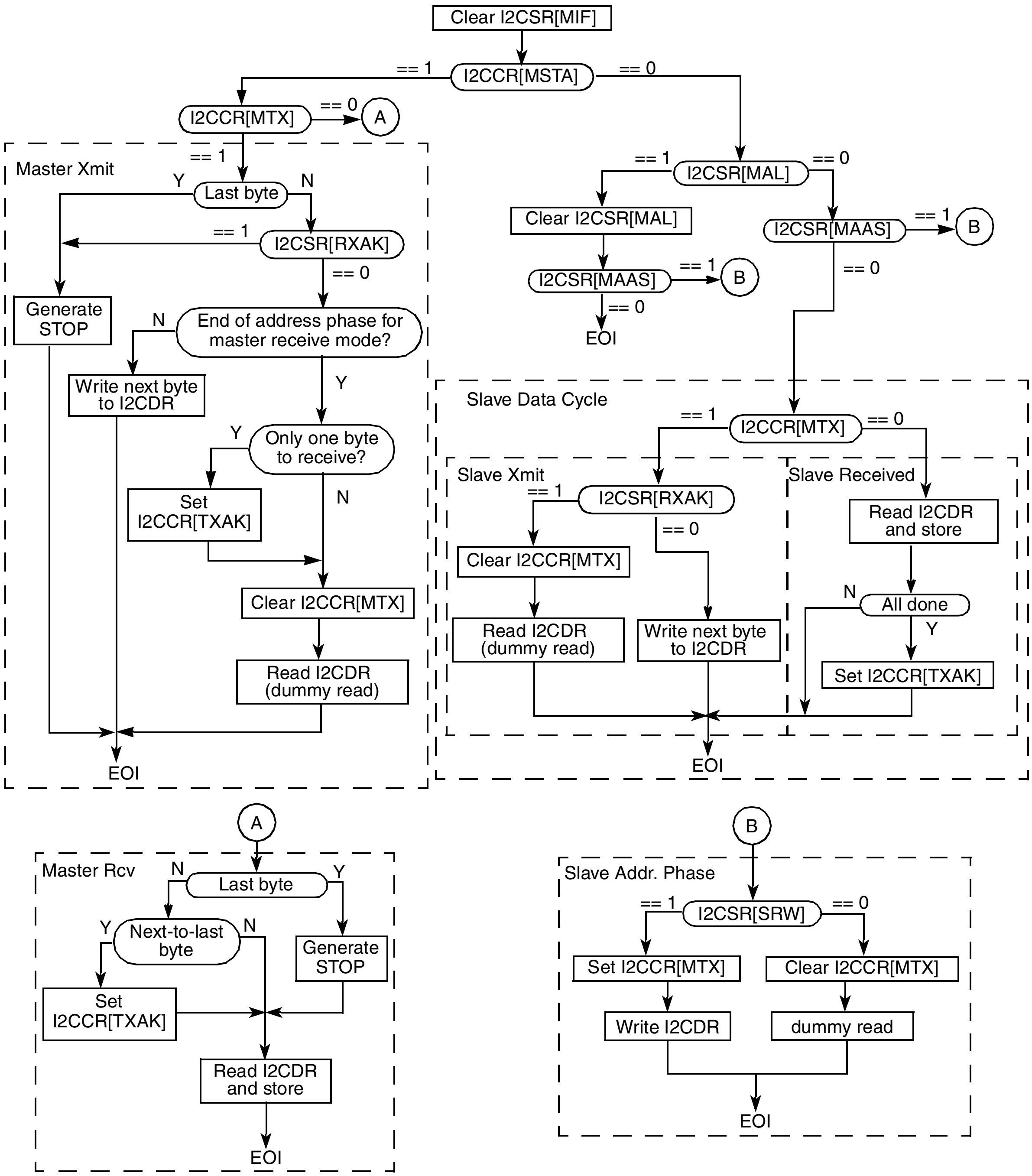


Figure 21-11. Example I²C Interrupt Service Routine Flowchart

21.5.2 Initialization Sequence

A hard reset initializes all of the I²C registers to their default states. The following initialization sequence initializes the I²C unit:

1. All I²C registers must be located in a cache-inhibited page.
2. Update I2C*n*FDR[FDR] and select the required division ratio to obtain the SCL*n* frequency from the CSB (platform) clock.
3. Update I2C*n*ADR to define the slave address for this device.
4. Modify I2C*n*CR to select master/slave mode, transmit/receive mode, and interrupt-enable or disable.
5. Set the I2C*n*CR[MEN] to enable the I²C interface.

21.5.3 Generation of START

After initialization, the following sequence can be used to generate START:

1. If the device is connected to a multimaster I²C system, check whether the serial bus is free (I2C*n*SR[MBB] = 0) before switching to master mode.
2. Select master mode (set I2C*n*CR[MSTA]) to transmit serial data and select transmit mode (set I2C*n*CR[MTX]) for the address cycle.
3. Write the slave address being called into I2C*n*DR. The data written to I2C*n*DR[0–6] comprises the slave calling address. I2C*n*CR[MTX] indicates the direction of transfer (transmit/receive) required from the slave.

The scenario above assumes that the I²C interrupt bit (I2C*n*SR[MIF]) is cleared. If MIF is set at any time, an I²C interrupt is generated (provided interrupt reporting is enabled with I2C*n*CR[MIEN] =1).

21.5.4 Post-Transfer Software Response

Transmission or reception of a byte automatically sets the data transferring bit (I2C*n*SR[MCF]), which indicates that one byte has been transferred. The I²C interrupt bit (I2C*n*SR[MIF]) is also set and an interrupt is generated to the processor if the interrupt function is enabled during the initialization sequence (I2C*n*CR[MIEN] is set). In the interrupt handler, software must take the following steps:

1. Clear I2C*n*SR[MIF]
2. Read the I2C*n*DR in receive mode or write to I2C*n*DR in transmit mode. Note that this causes I2C*n*SR[MCF] to be cleared, as shown in Figure 21-11.
3. When an interrupt occurs at the end of the address cycle, the master remains in transmit mode. If master receive mode is required, I2C*n*CR[MTX] must be toggled at this stage (see Figure 21-11).

If the interrupt function is disabled, software can service the I2C*n*DR in the main program by monitoring I2C*n*SR[MIF]. In this case, I2C*n*SR[MIF] must be polled rather than I2C*n*SR[MCF] because MCF behaves differently when arbitration is lost. Note that interrupt or other bus conditions may be detected before the I²C signals have time to settle. Thus, when polling I2C*n*SR[MIF] (or any other I2C*n*SR bits), software delays may be needed to give the I²C signals sufficient time to settle.

During slave-mode address cycles (I2C*n*SR[MAAS] is set), I2C*n*SR[SRW] should be read to determine the direction of the subsequent transfer and I2C*n*CR[MTX] should be programmed accordingly. For slave-mode data cycles (MAAS is cleared), I2C*n*SR[SRW] is not valid and I2C*n*CR[MTX] must be read to determine the direction of the current transfer (see Figure 21-11).

21.5.5 Generation of STOP

A data transfer ends with a STOP condition generated by the master device. A master transmitter can generate a STOP condition after all the data has been transmitted.

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data (by setting the transmit acknowledge bit (I2C*n*CR[TXAK])) before reading the next-to-last byte of data. At this time, the next-to-last byte of data has been transferred on the I^2C interface, so the last byte does not receive the data acknowledge (because I2C*n*CR[TXAK] is set). Before the interrupt service routine reads the last byte of data, a STOP condition must first be generated.

21.5.6 Generation of Repeated START

At the end of a data transfer, if the master still wants to communicate on the bus, it can generate another START condition followed by another slave address without first generating a STOP condition. This is accomplished by setting I2C*n*CR[RSTA].

21.5.7 Generation of SCL*n* When SDA*n* is Negated

It is sometimes necessary to force the I^2C module to become the I^2C bus master out of reset and drive SCL*n* (even though SDA*n* may already be driven, which indicates that the bus is busy). This can occur when a system reset does not cause all I^2C devices to be reset. Thus, SDA*n* can be negated low by another I^2C device while this I^2C module is coming out of reset and will stay low indefinitely. The following procedure can be used to force this I^2C module to generate SCL*n* so that the device driving SDA*n* can finish its transaction:

1. Disable the I^2C module and set the master bit by setting I2C*n*CR to 0x20.
2. Enable the I^2C module by setting I2C*n*CR to 0xA0.
3. Read I2C*n*DR.
4. Return the I^2C module to slave mode by setting I2C*n*CR to 0x80.

21.5.8 Slave Mode Interrupt Service Routine

In the slave interrupt service routine, the module addressed as a slave should be tested to check if a calling of its own address has been received. If I2C*n*SR[MAAS] is set, software should set the transmit/receive mode select bit (I2C*n*CR[MTX]) according to the R/$\overline{W}$ command bit (I2C*n*SR[SRW]). Writing to I2C*n*CR clears MAAS automatically. MAAS is read as set only in the interrupt handler at the end of that address cycle where an address match occurred; interrupts resulting from subsequent data transfers clear MAAS. A data transfer can then be initiated by writing to I2C*n*DR for slave transmits or dummy reading from I2C*n*DR in slave-receive mode. The slave negates SCL*n* between byte transfers. SCL*n* is released when I2C*n*DR is accessed in the required mode.

21.5.8.1 Slave Transmitter and Received Acknowledge

In the slave transmitter routine, the received acknowledge bit (I2C*n*SR[RXAK]) must be tested before sending the next byte of data. The master signals an end-of-data by not acknowledging the data transfer from the slave. When no acknowledge is received (I2C*n*SR[RXAK] is set), the slave transmitter interrupt routine must clear I2C*n*CR[MTX] to switch the slave from transmitter to receiver mode. A dummy read of I2C*n*DR then releases SCL*n* so that the master can generate a STOP condition. See Figure 21-11.

21.5.8.2 Loss of Arbitration and Forcing of Slave Mode

When a master loses arbitration the following conditions all occur:

- I2C*n*SR[MAL] is set
- I2C*n*CR[MSTA] is cleared (changing the master to slave mode)
- An interrupt occurs (if enabled) at the falling edge of the 9th clock of this transfer

Thus, the slave interrupt service routine should first test I2C*n*SR[MAL] and software should clear it if it is set. See Section 21.4.2.1, “Arbitration Control.”

Chapter 22
DUART

This chapter describes the two (dual) universal asynchronous receiver/transmitters (UARTs) of the device. It describes the functional operation, the DUART initialization sequence, and the programming details for the DUART registers and features.

22.1 Overview

The DUART consists of two (dual) universal asynchronous receiver/transmitters (UARTs). The UARTs act independently; all references to UART refer to one of these receiver/transmitters. Each UART is clocked by the system clock. The DUART programming model is compatible with the PC16552D.

The UART interface is point-to-point, meaning that only two UART devices are attached to the connecting signals. As shown in Figure 22-1, each UART module consists of the following:

- Receive and transmit buffers
- Clear to send ($\overline{CTS}$) input port and request to send ($\overline{RTS}$) output port for data-flow control.
- 16-bit counter for baud rate generation
- Interrupt control logic

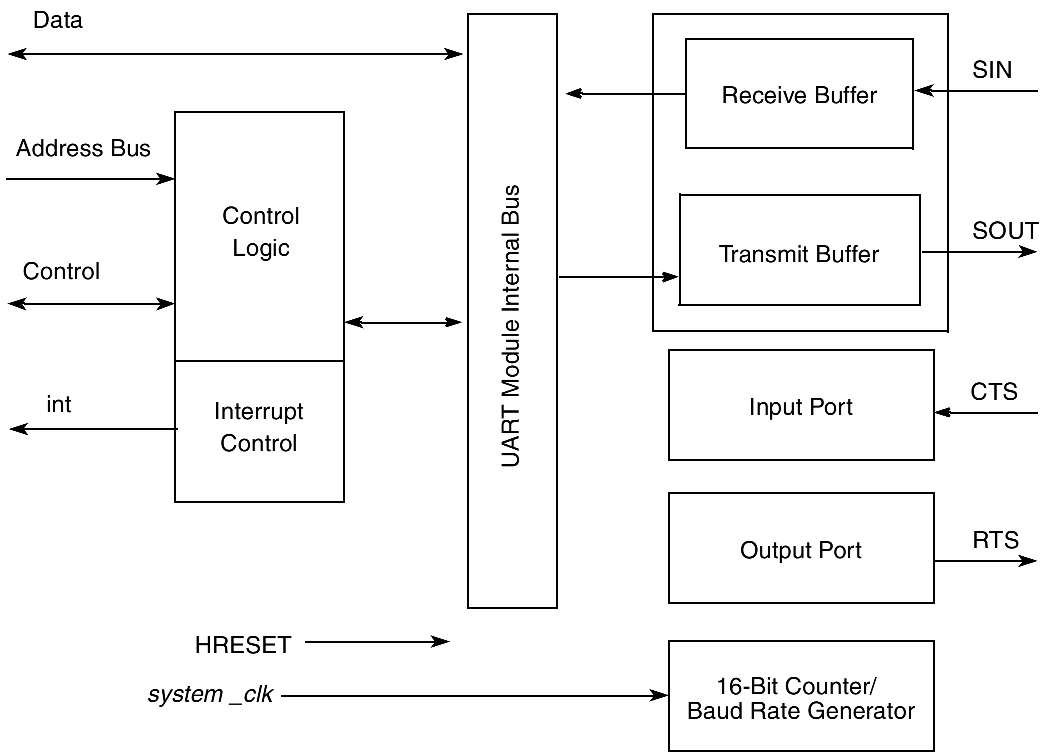


Figure 22-1. UART Block Diagram

22.1.1 Features

The DUART includes these features:

- Full-duplex operation
- Programming model compatible with the original PC16450 UART and the PC16550D (an improved version of the PC16450 that also operates in FIFO mode)
- PC16450 register reset values
- FIFO mode for both transmitter and receiver, providing 16-byte FIFOs
- Serial data encapsulation and decapsulation with standard asynchronous communication bits (START, STOP, and parity)
- Maskable transmit, receive, line status, and MODEM status interrupts
- Software-programmable baud generators that divide the system clock by 1 to (2^{16}–1) and generate a 16x clock for the transmitter and receiver engines
- Clear-to-send ($\overline{\text{CTS}}$) and ready-to-send ($\overline{\text{RTS}}$) MODEM control functions
- Software-selectable serial interface data format (data length, parity, 1/1.5/2 STOP bit, baud rate)
- Line and MODEM status registers
- Line-break detection and generation
- Internal diagnostic support, local loopback, and break functions
- Prioritized interrupt reporting
- Overrun, parity, and framing error detection

22.1.2 Modes of Operation

The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the system clock.

The transmitter accepts parallel data from a write to the transmitter holding register (UTHR). In FIFO mode, the data is placed directly into an internal transmitter shift register of the transmitter FIFO. The transmitter converts the data to a serial bit stream, inserting the appropriate START, STOP, and optional parity bits. Finally, it outputs a composite serial data stream on the channel transmitter serial data output signal (SOUT). The transmitter status may be polled or interrupt driven.

The receiver accepts serial data bits on the channel receiver serial data input signal (SIN), converts it to parallel format, checks for a START bit, parity (if any), STOP bits, and transfers the assembled character (with START, STOP, parity bits removed) from the receiver buffer (or FIFO) in response to a read of the UART's receiver buffer register (URBR). The receiver status may be polled or interrupt driven.

22.2 External Signal Descriptions

This section contains a signal overview and detailed signal descriptions.

22.2.1 Signal Overview

Table 22-1 summarizes the DUART signals. Note that although the actual device signal names are prepended with the 'UART_' prefix as shown in the table, the functional (abbreviated) signal names are often used throughout this chapter.

Table 22-1. DUART Signal Overview

Signal Name	I/O	Pins	Reset Value	State Meaning
UART_SIN[1:2]	I	2	1	Serial in data UART1 and UART2
UART_SOUT[1:2]	O	2	1	Serial out data UART1 and UART2
UART_CTS[1:2]	I	2	1	Clear to send UART1 and UART2
UART_RTS[1:2]	O	2	1	Request to send UART1 and UART2

22.2.2 Detailed Signal Descriptions

The DUART signals are described in detail in Table 22-2.

Table 22-2. DUART Signals—Detailed Signal Descriptions

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">UART_SIN[1:2]/DSP_UART_SIN</td><td rowspan="3">I</td><td colspan="2">Serial data in. Data is received on the receivers of UART1, UART2, or DSP_UART through its respective serial data input signal, with the least significant bit received first.</td></tr>
<tr><td>State Meaning</td><td>Asserted/Negated—Represents the data being received on the UART interface.</td></tr>
<tr><td>Timing</td><td>Assertion/Negation—An internal logic sample signal, rxcnt, uses the frequency of the baud-rate generator to sample the data on SIN.</td></tr>
</table>

Table 22-2. DUART Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
UART_SOUT[1:2]/ DSP_UART_SOUT	O	Serial data out. The serial data output signals for the UART1, UART2, or DSP_UART are set (mark condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on these signals, with the least significant bit transmitted first.	
		State Meaning	Asserted/Negated—Represents the data transmitted on the respective UART interface.
		Timing	Assertion/Negation—An internal logic sample signal, *rxcnt*, uses the frequency of the baud-rate generator to update and drive the data on SOUT.
$\overline{\text{UART\_CTS}}$[1:2]	I	Clear to send. Connected to the respective $\overline{\text{RTS}}$ outputs of the other UART devices on the bus. They can be programmed to generate an interrupt on change-of-state of the signal.	
		State Meaning	Asserted/Negated—Represent the clear to send condition for their respective UART.
		Timing	Assertion/Negation—Sampled at the rising edge of every system clock.
$\overline{\text{UART\_RTS}}$[1:2]	O	Request to send. Can be programmed to be negated and asserted by either the receiver or transmitter. When connected to the $\overline{\text{CTS}}$ input of a transmitter, this signal can be used to control serial data flow.	
		State Meaning	Asserted/Negated—Represents the data being transmitted on the respective UART interface.
		Timing	Assertion/Negation—Updated and driven at the rising edge of every system clock.

22.3 Memory Map/Register Definition

There are two complete sets of DUART registers (one for UART1 and one for UART2). The two UARTs are identical, except that the registers for UART1 are located at offsets 0x0_4500 (local), and the registers for UART2 are located at offsets 0x0_4600 (local). Throughout this chapter, the registers are described by a singular acronym: for example, LCR represents the line control register for either UART1 or UART2.

The registers in each UART interface are used for configuration, control, and status. The divisor latch access bit, ULCR[DLAB], is used to access the divisor latch least- and most-significant bit registers and the alternate function register. Refer to Section 22.3.1.7, "Line Control Registers (ULCR1 and ULCR2)," for more information on ULCR[DLAB].

All DUART registers are one byte wide; reads and writes to these registers must be byte-wide operations. Table 22-3 provides a register summary with references to the section and page that contain detailed information about each register. Undefined byte address spaces within offset 0x4000–0x4FFF are reserved.

Table 22-3. DUART Register Summary

Address	Register	Access	Reset	Section/Page
0x0_4500	URBR—ULCR[DLAB] = 0 UART1 receiver buffer register	R	0x00	22.3.1.1/22-5
	UTHR—ULCR[DLAB] = 0 UART1 transmitter holding register	W	0x00	22.3.1.2/22-6
	UDLB—ULCR[DLAB] = 1 UART1 divisor least significant byte register	R/W	0x00	22.3.1.3/22-6

Table 22-3. DUART Register Summary (continued)

Address	Register	Access	Reset	Section/Page
0x0_4501	UIER—ULCR[DLAB] = 0 UART1 interrupt enable register	R/W	0x00	22.3.1.4/22-8
	UDMB—ULCR[DLAB] = 1 UART1 divisor most significant byte register	R/W	0x00	22.3.1.3/22-6
0x0_4502	UIIR—ULCR[DLAB] = 0 UART1 interrupt ID register	R	0x01	22.3.1.5/22-9
	UFCR—ULCR[DLAB] = 0 UART1 FIFO control register	W	0x00	22.3.1.6/22-10
	UAFR—ULCR[DLAB] = 1 UART1 alternate function register	R/W	0x00	22.3.1.12/22-16
0x0_4503	ULCR—ULCR[DLAB] = x UART1 line control register	R/W	0x00	22.3.1.7/22-11
0x0_4504	UMCR—ULCR[DLAB] = x UART1 MODEM control register	R/W	0x00	22.3.1.8/22-13
0x0_4505	ULSR—ULCR[DLAB] = x UART1 line status register	R	0x60	22.3.1.9/22-14
0x0_4506	UMSR—ULCR[DLAB] = x UART1 MODEM status register	R	0x00	22.3.1.10/22-15
0x0_4507	USCR—ULCR[DLAB] = x UART1 scratch register	R/W	0x00	22.3.1.11/22-16
0x0_4510	UDSR—ULCR[DLAB] = x UART1 DMA status register	R	0x01	22.3.1.13/22-17
0x0_4600	URBR—ULCR[DLAB] = 0 UART2 receiver buffer register	R	0x00	22.3.1.1/22-5
	UTHR—ULCR[DLAB] = 0 UART2 transmitter holding register	W	0x00	22.3.1.2/22-6
	UDLB—ULCR[DLAB] = 1 UART2 divisor least significant byte register	R/W	0x00	22.3.1.3/22-6
0x0_4601	UIER—ULCR[DLAB] = 0 UART2 interrupt enable register	R/W	0x00	22.3.1.4/22-8
	UDMB—ULCR[DLAB] = 1 UART2 divisor most significant byte register	R/W	0x00	22.3.1.3/22-6
0x0_4602	UIIR—ULCR[DLAB] = 0 UART2 interrupt ID register	R	0x01	22.3.1.5/22-9
	UFCR—ULCR[DLAB] = 0 UART2 FIFO control register	W	0x00	22.3.1.6/22-10
	UAFR—ULCR[DLAB] = 1 UART2 alternate function register	R/W	0x00	22.3.1.12/22-16
0x0_4603	ULCR—ULCR[DLAB] = x UART2 line control register	R/W	0x00	22.3.1.7/22-11
0x0_4604	UMCR—ULCR[DLAB] = x UART2 MODEM control register	R/W	0x00	22.3.1.8/22-13
0x0_4605	ULSR—ULCR[DLAB] = x UART2 line status register	R	0x60	22.3.1.9/22-14
0x0_4606	UMSR—ULCR[DLAB] = x UART2 MODEM status register	R	0x00	22.3.1.10/22-15
0x0_4607	USCR—ULCR[DLAB] = x UART2 scratch register	R/W	0x00	22.3.1.11/22-16
0x0_4610	UDSR—ULCR[DLAB] = x UART2 DMA status register	R	0x01	22.3.1.13/22-17

22.3.1 Register Descriptions

The following sections describe the UART1 and UART2 registers.

22.3.1.1 Receiver Buffer Registers (URBR1 and URBR2)

These registers contain the data received from the transmitter on the UART buses. In FIFO mode, when read, they return the first byte received. For FIFO status information, refer to the UDSR[RXDY] description.

Except for the case when there is an overrun, URBR returns the data in the order it was received from the transmitter. Refer to the ULSR[OE] description, Section 22.3.1.9, “Line Status Registers (ULSR1 and ULSR2).” Figure 22-2 shows the receiver buffer registers. Note that these registers have same offset as the UTHRs.

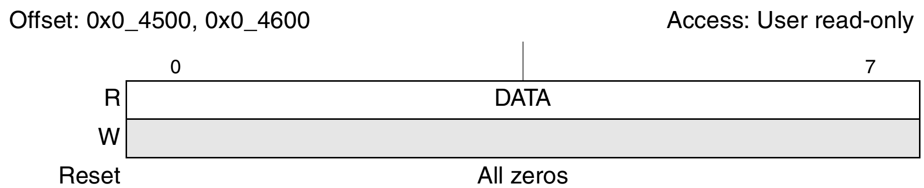


Figure 22-2. Receiver Buffer Registers (URBR1 and URBR2)

Table 22-4 describes URBR.

Table 22-4. URBR Field Descriptions

Bits	Name	Description
0–7	DATA	Data received from the transmitter on the UART bus [read only]

22.3.1.2 Transmitter Holding Registers (UTHR1 and UTHR2)

A write to these 8-bit registers causes the UART devices to transfer 5 to 8 data bits on the UART bus in the format set up in the ULCR (line control register). In FIFO mode, data written to UTHR is placed into the FIFO. The data written to UTHR is the data sent onto the UART bus, and the first byte written to UTHR is the first byte onto the bus. UDSR[TXRDY] indicates when the FIFO is full. Refer to Table 22-21 and Table 22-22.

Figure 22-3 shows the bits in the UTHRs.

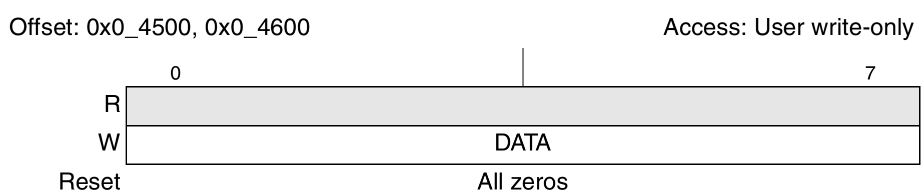


Figure 22-3. Transmitter Holding Registers (UTHR1 and UTHR2)

Table 22-5 describes the UTHR.

Table 22-5. UTHR Field Descriptions

Bits	Name	Description
0–7	DATA	Data that is written to UTHR [Write only]

22.3.1.3 Divisor Most and Least Significant Byte Registers (UDMB and UDLB)

UDLB is concatenated with the divisor most significant byte register (UDMB) to create the divisor used to divide the input clock into the DUART. The output frequency of the baud generator is 16 times the baud rate; therefore, the desired baud rate = platform clock frequency ÷ (16 × [UDMB||UDLB]). Equivalently, [UDMB||UDLB:0b0000] = platform clock frequency/desired baud rate. Baud rates that can be generated by specific input clock frequencies are shown in Table 22-8.

Figure 22-4 shows the bits in the UDMBs.

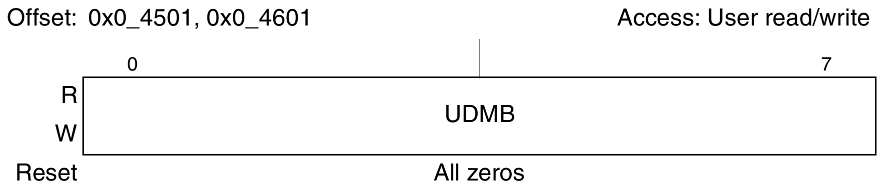


Figure 22-4. Divisor Most Significant Byte Registers (UDMB1 and UDMB2)

Table 22-6 describes the UDMB.

Table 22-6. UDMB Field Descriptions

Bits	Name	Description
0–7	UDMB	Divisor most significant byte

Figure 22-5 shows the bits in the UDLBs.

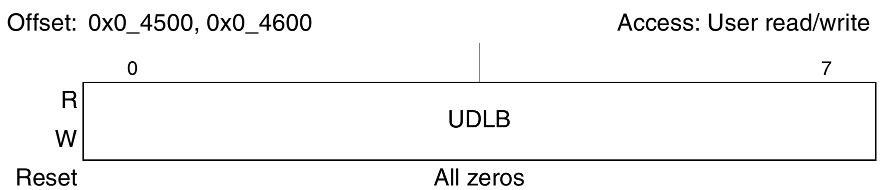


Figure 22-5. Divisor Least Significant Byte Registers (UDLB1 and UDLB2)

Table 22-7 describes the UDLB.

Table 22-7. UDLB Field Descriptions

Bits	Name	Description
0–7	UDLB	Divisor least significant byte. This is concatenated with UDMB.

Table 22-8 shows baud rate for a variety of input clock frequencies.

Table 22-8. Baud Rate Examples

Baud Rate (Decimal)	Divisor		Input Clock (System Clock) Frequency (MHz)	Percent Error (Decimal)
	Decimal	Hex		
9,600	1736	6C8	266	0.0064
19,200	868	364	266	0.0064
38,400	434	1B2	266	0.0064
56,000	298	12A	266	0.1280
128,000	130	82	266	0.1600

Table 22-8. Baud Rate Examples (continued)

Baud Rate (Decimal)	Divisor		Input Clock (System Clock) Frequency (MHz)	Percent Error (Decimal)
	Decimal	Hex		
256,000	65	41	266	0.1600
9,600	2170	87A	333	0.0064
19,200	1085	43D	333	0.0064
38,400	543	21F	333	0.0858
56,000	372	174	333	0.0064
128,000	163	A3	333	0.1472
256,000	81	51	333	0.4672

To get the percent error value, the following three steps are taken:

1. The input clock frequency (ICF) is divided by the actual frequency input (AFI) to get the correct divisor value (ICF/AFI, where AFI = baud rate $\times$ 16 $\times$ divisor).
2. The divisor value is subtracted from 1.
3. The result from the step two is multiplied by 100 to calculate the final percent error. The result is calculated in absolute value (no negative numbers).

   These steps can be described with the following equation:

   Percent error value = $(1 - AFI/ICF) \times 100$

22.3.1.4 Interrupt Enable Registers (UIER1 and UIER2)

The UIER gives the user the ability to mask specific UART interrupts to the programmable interrupt controller (PIC).

Figure 22-6 shows the bits in the UIER.

Offset: 0x0_4501, 0x0_4601 Access: User read/write

	0–3	4	5	6	7
R/W	—	EMSI	ERLSI	ETHREI	ERDAI
Reset	All zeros				

Figure 22-6. Interrupt Enable Registers (UIER1 and UIER2)

Table 22-9 describes the UIER fields.

Table 22-9. UIER Field Descriptions

Bits	Name	Description
0–3	—	Reserved
4	EMSI	Enable MODEM status interrupt 0 Mask interrupts caused by UMSR[DCTS] being set. 1 Enable and assert interrupts when UMSR[CTS] changes state.
5	ERLSI	Enable receiver line status interrupt 0 Mask interrupts when ULSR's overrun, parity error, framing error, or break interrupt bits are set. 1 Enable and assert interrupts when ULSR's overrun, parity error, framing error or break interrupt bits are set.
6	ETHREI	Enable transmitter holding register empty interrupt 0 Mask interrupt when ULSR[THRE] is set. 1 Enable and assert interrupts when ULSR[THRE] is set.
7	ERDAI	Enable received data available interrupt 0 Mask interrupt when new receive data is available or receive data time-out has occurred. 1 Enable and assert interrupts when a new data character is received from the external device and/or a time-out interrupt occurs in FIFO mode.

22.3.1.5 Interrupt ID Registers (UIIR1 and UIIR2)

The UIIRs indicate when an interrupt is pending from the corresponding UART and what type of interrupt is active. They also indicate if the FIFOs are enabled.

The DUART prioritizes interrupts into four levels and records these in the corresponding UIIR. The four levels of interrupt conditions in order of priority are as follows:

1. Receiver line status
2. Received data ready/character time-out
3. Transmitter holding register empty
4. MODEM status

See Table 22-11 for more details.

When the UIIR is read, the associated DUART serial channel freezes all interrupts and indicates the highest priority pending interrupt. While this read transaction is occurring, the associated DUART serial channel records new interrupts, but does not change the contents of UIIR until the read access is complete.

Figure 22-7 shows the bits in the UIIR.

Offset: 0x0_4502, 0x0_4602 Access: User read-only

	0	1	2	3	4	5	6	7
R	FE		—		IID3	IID2	IID1	IID0
W								
Reset	0	0	0	0	0	0	0	1

Figure 22-7. Interrupt ID Registers (UIIR1 and UIIR2)

Table 22-10 describes the fields of the UIIR.

Table 22-10. UIIR Field Descriptions

Bits	Name	Description
0–1	FE	FIFOs enabled. Reflects the setting of UFCR[FEN].
2–3	—	Reserved
4	IID3	Interrupt ID bits identify the highest priority interrupt that is pending as indicated in Table 22-11. IID3 is set along with IID2 only when a time out interrupt is pending for FIFO mode.
5–6	IID2–IID1	Interrupt ID bits identify the highest priority pending interrupt as indicated in Table 22-11.
7	IID0	IID0 indicates when an interrupt is pending. 0 The UART has an active interrupt ready to be serviced. 1 No interrupt is pending.

The bits contained in the UIIR registers are described in Table 22-11.

Table 22-11. UIIR IID Bits Summary

IID3–IID0	Priority Level	Interrupt Type	Interrupt Description	How To Reset Interrupt
0001	—	—	—	—
0110	Highest	Receiver line status	Overrun error, parity error, framing error, or break interrupt	Reading the line status register
0100	Second	Received data available	Receiver data available or trigger level reached in FIFO mode.	Reading the receiver buffer register or if the number of bytes in the receiver FIFO drops below the trigger level.
1100	Second	Character time-out	No characters were removed from or input to the receiver FIFO during the last four character times and at least one character is in the receiver FIFO.	Reading the receiver buffer register
0010	Third	UTHR empty	Transmitter holding register is empty.	Reading UIIR or writing to UTHR
0000	Fourth	MODEM status	$\overline{\text{CTS}}$ input value changed since last read of UMSR.	Reading UMSR

22.3.1.6 FIFO Control Registers (UFCR1 and UFCR2)

UFCR is used to enable and clear the receiver and transmitter FIFOs, set a receiver FIFO trigger level to control the received data available interrupt, and select the type of DMA signaling.

UFCR bits cannot be programmed unless FIFO enable bits are set. When changing from FIFO mode to 16450 mode (non-FIFO mode) and vice versa, data is automatically cleared from the FIFOs.

After all of the bytes in the receiver FIFO are cleared, the receiver internal shift register is not cleared. Similarly, the bytes are cleared in the transmitter FIFO, but the transmitter internal shift register is not cleared. Both TFR and RFR are self clearing.

Figure 22-8 shows the bits in the UFCRs.

Offset: 0x0_4502, 0x0_4602 — Access: User write-only

	0	1	2	3	4	5	6	7
R								
W	RTL		—		DMS	TFR	RFR	FEN
Reset	All zeros							

Figure 22-8. FIFO Control Registers (UFCR1 and UFCR2)

Table 22-12 describes the fields of the UFCRs.

Table 22-12. UFCR Field Descriptions

Bits	Name	Description
0–1	RTL	Receiver trigger level. A received data available interrupt occurs when UIER[ERDAI] is set and the number of bytes in the receiver FIFO equals RTL value. 00 1 byte 01 4 bytes 10 8 bytes 11 14 bytes
2–3	—	Reserved
4	DMS	DMA mode select. See Section 22.4.5.2, “DMA Mode Select” 0 UDSR[RXRDY] and UDSR[TXRDY] bits are in mode 0. 1 UDSR[RXRDY] and UDSR[TXRDY] bits are in mode 1 if UFCR[FEN] = 1.
5	TFR	Transmitter FIFO reset 0 No action 1 Clears all bytes in the transmitter FIFO and resets the FIFO counter/pointer to 0
6	RFR	Receiver FIFO reset 0 No action 1 Clears all bytes in the receiver FIFO and resets the FIFO counter/pointer to 0
7	FEN	FIFO enable 0 FIFOs are disabled and cleared 1 Transmitter and receiver FIFOs are enabled.

22.3.1.7 Line Control Registers (ULCR1 and ULCR2)

The ULCRs specify the data format for the UART bus and set the divisor latch access bit ULCR[DLAB], which controls the ability to access the divisor latch least and most significant bit registers and the alternate function register.

After initializing ULCR, the software should not rewrite the ULCR while valid transfers on the UART bus are active. The software should not rewrite the ULCR until the last STOP bit is received and no new characters are being transferred on the bus.

The stick parity bit, ULCR[SP], assigns a set parity value for the parity bit time slot sent on the UART bus. The set value is defined as mark parity (logic 1) or space parity (logic 0). ULCR[PEN] and ULCR[EPS] help determine the set parity value. See Table 22-14. ULCR[NSTB] defines the number of STOP bits to be sent at the end of the data transfer. The receiver checks only the first STOP bit, regardless of the number of STOP bits selected. The word length select bits (1 and 0) define the number of data bits transmitted or received as a serial character. The word length does not include START, parity, and STOP bits.

Figure 22-9 shows the bits in the ULCRs.

Offset: 0x0_4503, 0x0_4603 — Access: User read/write

	0	1	2	3	4	5	6–7
R/W	DLAB	SB	SP	EPS	PEN	NSTB	WLS
Reset	All zeros						

Figure 22-9. Line Control Register (ULCR1 and ULCR2)

Table 22-13 describes the ULCR fields.

Table 22-13. ULCR Field Descriptions

Bits	Name	Description
0	DLAB	Divisor latch access bit 0 Access to all registers except UDLB, UAFR, and UDMB. 1 Ability to access UDMB, UDLB, and UAFR.
1	SB	Set break 0 Send normal UTHR data onto the SOUT signal. 1 Force logic 0 to be on SOUT. Data in the UTHR is not affected.
2	SP	Stick parity 0 Stick parity is disabled. 1 If PEN = 1 and EPS = 1, space parity is selected; if PEN = 1 and EPS = 0, mark parity is selected.
3	EPS	Even parity select. See Table 22-14. 0 If PEN = 1 and SP = 0 then odd parity is selected. 1 If PEN = 1 and SP = 0 then even parity is selected.
4	PEN	Parity enable 0 No parity generation and checking. 1 Generate parity bit as a transmitter, and check parity as a receiver.
5	NTSB	Number of STOP bits 0 One STOP bit is generated in the transmitted data. 1 When a 5-bit data length is selected, 1 1/2 STOP bits are generated. When either a 6-, 7-, or 8-bit word length is selected, two STOP bits are generated.
6–7	WLS	Word length select. Number of bits that comprise the character length. 00 5 bits 01 6 bits 10 7 bits 11 8 bits

Table 22-14. Parity Selection Using ULCR[PEN], ULCR[SP], and ULCR[EPS]

PEN	SP	EPS	Parity Selected
0	0	0	No parity
0	0	1	No parity
0	1	0	No parity

Table 22-14. Parity Selection Using ULCR[PEN], ULCR[SP], and ULCR[EPS] (continued)

PEN	SP	EPS	Parity Selected
0	1	1	No parity
1	0	0	Odd parity
1	0	1	Even parity
1	1	0	Mark parity
1	1	1	Space parity

22.3.1.8 MODEM Control Registers (UMCR1 and UMCR2)

The UMCRs, shown in Figure 22-10, control the interface with the external peripheral device on the UART bus.

Offset: 0x0_4504, 0x0_4604 Access: User read/write

	0	1	2	3	4	5	6	7
R W	—			LOOP	—		RTS	—
Reset	All zeros							

Figure 22-10. Modem Control Register (UMCR1 and UMCR2)

Table 22-15 describes the UMCR fields.

Table 22-15. UMCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared
3	LOOP	Local loopback mode 0 Normal operation. 1 Functionally, the data written to UTHR can be read from URBR of the same UART, and UMCR[RTS] is tied to UMSR[CTS].
4–5	—	Reserved
6	RTS	Ready to send 0 Negates corresponding $\overline{\text{UART\_RTS}}$ output. 1 Assert corresponding $\overline{\text{UART\_RTS}}$ output. Informs external MODEM or peripheral that the UART is ready for sending/receiving data.
7	—	Reserved

22.3.1.9 Line Status Registers (ULSR1 and ULSR2)

The ULSRs, shown in Figure 22-11, monitor the status of the data transfer on the UART buses. To isolate the status bits from the proper character received through the UART bus, software should read the ULSR and then the URBR.

Offset: 0x0_4505, 0x0_4605 — Access: User read-only

	0	1	2	3	4	5	6	7
R	RFE	TEMT	THRE	BI	FE	PE	OE	DR
W								
Reset	0	1	1	0	0	0	0	0

Figure 22-11. Line Status Register (ULSR1 and ULSR2)

Table 22-16 describes the ULSR fields.

Table 22-16. ULSR Field Descriptions

Bits	Name	Description
0	RFE	Receiver FIFO error. 0 Cleared when there are no errors in the receiver FIFO or on a read of the ULSR with no remaining receiver FIFO errors. 1 Set when one of the characters in the receiver FIFO encounters an error (framing, parity, or break interrupt).
1	TEMT	Transmitter empty 0 Either or both the UTHR or the internal transmitter shift register has a data character. In FIFO mode, a data character is in the transmitter FIFO or the internal transmitter shift register. 1 Both the UTHR and the internal transmitter shift register are empty. In FIFO mode, both the transmitter FIFO and the internal transmitter shift register are empty.
2	THRE	Transmitter holding register empty 0 UTHR is not empty. 1 A data character has transferred from the UTHR into the internal transmitter shift register. In FIFO mode, the transmitter FIFO contains no data character.
3	BI	Break interrupt 0 Cleared when the ULSR is read or when a valid data transfer is detected (that is, STOP bit is received). 1 Received data of logic 0 for more than START bit + Data bits + Parity bit + one STOP bits length of time. A new character is not loaded until SIN returns to the mark state (logic 1) and a valid START is detected. In FIFO mode, a zero character is encountered in the FIFO (the zero character is at the top of the FIFO). In FIFO mode, only one zero character is stored.
4	FE	Framing error 0 Cleared when ULSR is read or when a new character is loaded into the URBR from the receiver shift register. 1 Invalid STOP bit for receive data (only the first STOP bit is checked). In FIFO mode, FE is set when the character that detected a framing error is encountered in the FIFO (that is the character at the top of the FIFO). An attempt to resynchronize occurs after a framing error. The UART assumes that the framing error (due to a logic 0 being read when a logic 1 (STOP) was expected) was due to a STOP bit overlapping with the next START bit, so it assumes this logic 0 sample is a true START bit and then will receive the following new data.
5	PE	Parity error 0 Cleared when ULSR is read or when a new character is loaded into URBR. 1 Unexpected parity value encountered when receiving data. In FIFO mode, the character with the error is at the top of the FIFO.

Table 22-16. ULSR Field Descriptions (continued)

Bits	Name	Description
6	OE	Overrun error 0 Cleared when ULSR is read 1 Before URBR was read, it was overwritten with a new character. The old character is lost. In FIFO mode, the receiver FIFO is full (regardless of the receiver FIFO trigger level setting) and a new character has been received into the internal receiver shift register. The old character was overwritten by the new character. Data in the receiver FIFO was not overwritten.
7	DR	Data ready 0 Cleared when URBR is read or when all of the data in the receiver FIFO is read. 1 A character was received in the URBR or the receiver FIFO.

22.3.1.10 MODEM Status Registers (UMSR1 and UMSR2)

The UMSRs, shown in Figure 22-12, track the status of the MODEM (or external peripheral device) $\overline{\text{CTS}}$, set for the corresponding UART.

Offset: 0x0_4506, 0x0_4606 Access: User read-only

	0–2	3	4–6	7
R	—	CTS	—	DCTS
W				
Reset	All zeros			

Figure 22-12. Modem Status Register (UMSR1 and UMSR2)

Table 22-17 describes UMSR fields.

Table 22-17. UMSR Field Descriptions

Bits	Name	Description
0–2	—	Reserved, should be cleared
3	CTS	Clear to send. Represents the inverted value of the $\overline{\text{CTS}}$ input pin from the external peripheral device. 0 Corresponding $\overline{\text{CTS}}n$ is negated. 1 Corresponding $\overline{\text{CTS}}n$ is asserted. The MODEM or peripheral device is ready for data transfers.
4–6	—	Reserved, should be cleared
7	DCTS	Delta clear to send 0 No change on the corresponding $\overline{\text{CTS}}n$ signal since the last read of UMSR[CTS]. 1 $\overline{\text{CTS}}n$ changed since the last read of UMSR[CTS]. Causes an interrupt if UIER[EMSI] is set to detect this condition.

22.3.1.11 Scratch Registers (USCR1 and USCR2)

USCR, shown in Figure 22-13, are for debugging software or the DUART hardware. The USCRs do not affect the operation of the DUART.

Offset: 0x0_4507, 0x0_4607 — Access: User read/write

	0–3	4–7
R W	DATA	
Reset	All zeros	

Figure 22-13. Scratch Register (USCR)

Table 22-18 describes USCR fields.

Table 22-18. USCR Field Descriptions

Bits	Name	Description
0–7	DATA	Data

22.3.1.12 Alternate Function Registers (UAFR1 and UAFR2)

The UAFRs, shown in Figure 22-14, allow software to write to both UART1 and UART2 registers simultaneously with the same write operation. The UAFRs also provide a means for the device's performance monitor to track the baud clock.

Offset: 0x0_4502, 0x0_4602 — Access: User read/write

	0–5	6	7
R W	—	BO	CW
Reset	All zeros		

Figure 22-14. Alternate Function Register (UAFR)

Table 22-19 describes UAFR fields.

Table 22-19. UAFR Field Descriptions

Bits	Name	Description
0–5	—	Reserved
6	BO	Baud clock select 0 The baud clock is not gated off. 1 The baud clock is gated off.
7	CW	Concurrent write enable 0 Disables writing to both UART1 and UART2. 1 Enables concurrent writes to corresponding UART registers. A write to a register in UART1 is also a write to the corresponding register in UART2 and vice versa.

22.3.1.13 DMA Status Registers (UDSR1 and UDSR2)

The DMA status registers (UDSRs), shown in Figure 22-15, return transmitter and receiver FIFO status and provide the ability to assist DMA data operations to and from the FIFOs.

Offset: 0x0_4510, 0x0_4610 Access: User read-only

	0	1	2	3	4	5	6	7
R	—						TXRDY	RXRDY
W								
Reset	0	0	0	0	0	0	0	1

Figure 22-15. DMA Status Register (UDSR)

Table 22-20 describes the fields of the UDSRs.

Table 22-20. UDSR Field Descriptions

Bits	Name	Description
0–5	—	Reserved
6	TXRDY	Transmitter ready. Reflects the status of the transmitter FIFO or the UTHR. The status depends on the DMA mode selected, which is determined by UFCR[DMS] and UFCR [FEN]. 0 The bit is cleared, as shown inTable 22-22. 1 This bit is set, as shown in Table 22-21.
7	RXRDY	Receiver ready. This read-only bit reflects the status of the receiver FIFO or URBR. The status depends on the DMA mode selected, which is determined by UFCR[DMS] and UFCR [FEN]. 0 The bit is cleared, as shown in Table 22-24. 1 This bit is set, as shown in Table 22-23.

Table 22-21. UDSR[TXRDY] Set Conditions

DMS	FEN	DMA Mode	Meaning
0	0	0	TXRDY is set after the first character is loaded into the transmitter FIFO or UTHR.
0	1	0	
1	0	0	
1	1	1	TXRDY is set when the transmitter FIFO is full.

Table 22-22. UDSR[TXRDY] Cleared Conditions

DMS	FEN	DMA Mode	Meaning
0	0	0	TXRDY is cleared when there are no characters in the transmitter FIFO or UTHR.
0	1	0	
1	0	0	
1	1	1	TXRDY is cleared when there are no characters in the transmitter FIFO or UTHR. TXRDY remains clear while the transmitter FIFO is not yet full.

Table 22-23. UDSR[RXRDY] Set Conditions

DMS	FEN	DMA Mode	Meaning
0	0	0	RXRDY is set when there are no characters in the receiver FIFO or URBR.
0	1	0	
1	0	0	
1	1	1	RXRDY is set when the trigger level has not been reached and there has been no time out.

Table 22-24. UDSR[RXRDY] Cleared

DMS	FEN	DMA Mode	Meaning
0	0	0	RXRDY is cleared when there is at least one character in the receiver FIFO or URBR.
0	1	0	
1	0	0	
1	1	1	RXRDY is cleared when the trigger level or a time-out has been reached. RXRDY remains cleared until the receiver FIFO is empty.

22.4 Functional Description

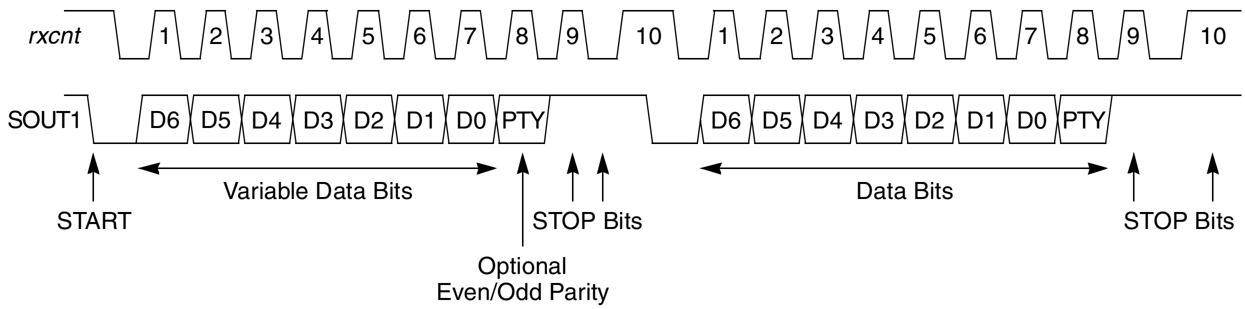
The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the system clock signal.

The transmitter accepts parallel data with a write access to UTHR. In FIFO mode, the data is placed directly into an internal transmitter shift register, or into the transmitter FIFO, as explained in Section 22.4.5, “FIFO Mode.” The transmitting registers convert the data to a serial bit stream by inserting the appropriate START, STOP, and optional parity bits. Finally, the registers output a composite serial data stream on the channel transmitter serial data output (SOUT). The transmitter status may be polled or interrupt driven.

The receiver accepts serial data on the channel receiver serial data input (SIN), converts the data into parallel format, and checks for START, STOP, and parity bits. In FIFO mode, the receiver removes the START, STOP, and parity bits and then transfers the assembled character from the receiver buffer, or receiver FIFO. This transfer occurs in response to a read of the UART receiver buffer register (URBR). The receiver status may be polled or interrupt-driven.

22.4.1 Serial Interface

The UART bus is a serial, full-duplex, point-to-point bus as shown in Figure 22-16. Therefore, only two devices are attached to the same signals and there is no need for address or arbitration bus cycles.



Two 7-Bit Data Transmissions with Parity and 2-Bit STOP Transactions

Figure 22-16. UART Bus Interface Transaction Protocol Example

A standard UART bus transfer is composed of either three or four parts:

- START bit
- Data transfer (least significant bit is first data bit on the bus)
- Parity bit (optional)
- STOP bits

An internal logic sample signal, *rxcnt*, uses the frequency of the baud-rate generator to drive the bits on SOUT.

The following sections describe the four components of the serial interface, the baud-rate generator, local loopback mode, different errors, and FIFO mode.

22.4.1.1 START Bit

A write to UTHR generates a START bit on the SOUT signal. Figure 22-16 shows that the START bit is defined as a logic 0. The START bit denotes the beginning of a new data transfer which is limited to the bit length programmed in ULCR.When the bus is idle, SOUT is high.

22.4.1.2 Data Transfer

Each data transfer contains 5, 6, 7, or 8 bits of data. The ULCR data bit length for the transmitter and receiver UART devices must agree before a transfer begins; otherwise, a parity or framing error may occur. A transfer begins when UTHR is written. At that time, a START bit is generated followed by 5 to 8 of the data bits previously written to the UTHR. The data bits are driven from the least- to the most-significant bits. After the parity and STOP bits, a new data transfer can begin if new data is written to UTHR.

22.4.1.3 Parity Bit

The user has the option of using even, odd, no parity, or stick parity (see Section 22.3.1.7, “Line Control Registers (ULCR1 and ULCR2)”). Both the receiver and transmitter parity definitions must agree before transferring data. When receiving data, a parity error can occur if an unexpected parity value is detected (see Section 22.3.1.9, “Line Status Registers (ULSR1 and ULSR2)”).

22.4.1.4 STOP Bit

The transmitter device ends the write transfer by generating a STOP bit. The STOP bit is always high. The user can program the length of the STOP bit(s) in the ULCR. Both the receiver and transmitter STOP bit length must agree before attempting to transfer data. A framing error can occur if an invalid STOP bit is detected.

22.4.2 Baud-Rate Generator Logic

Each UART contains an independent programmable baud-rate generator, that is capable of taking the system clock input and dividing the input by any divisor from 1 to $2^{16} – 1$.

The baud rate is defined as the number of bits per second that can be sent over the UART bus. The formula for calculating baud rate is as follows:

$$\text{Baud rate} = (1/16) \times (\text{system clock frequency/divisor value})$$

Therefore, the output frequency of the baud-rate generator is 16 times the baud rate.

The divisor value is determined by the following two 8-bit registers to form a 16-bit binary number:

- UART divisor most significant byte register (UDMB)
- UART divisor least significant byte register (UDLB)

Upon loading either of the divisor latches, a 16-bit baud-rate counter is loaded.

The divisor latches must be loaded during initialization to ensure proper operation of the baud-rate generator. Both UART devices on the same bus must be programmed for the same baud rate before starting a transfer.

The baud clock can be passed to the performance monitor by enabling UAFR[BO]. This can be used to determine baud-rate errors.

22.4.3 Local Loopback Mode

Local loopback mode is provided for diagnostic testing. The data written to UTHR can be read from the receiver buffer register (URBR) of the same UART. In this mode, the MODEM control register UMCR[RTS] is internally tied to the MODEM status register UMSR[CTS]. The transmitter SOUT is set to a logic 1 and the receiver SIN is disconnected. The output of the transmitter shift register is looped back into the receiver shift register input. The $\overline{\text{CTS}}$ (input signal) is disconnected, $\overline{\text{RTS}}$ is internally connected to $\overline{\text{CTS}}$, and the $\overline{\text{RTS}}$ (output signal) becomes inactive. In this diagnostic mode, data that is transmitted is immediately received. In local loopback mode the transmit and receive data paths of the DUART can be verified. Note that in local loopback mode, the transmit/receive interrupts are fully operational and can be controlled by the interrupt enable register (UIER).

22.4.4 Errors

The following sections describe framing, parity, and overrun errors which may occur while data is transferred on the UART bus. Each of the error bits are usually cleared, as described below, when the line status register (ULSR) is read.

22.4.4.1 Framing Error

When an invalid STOP bit is detected, a framing error occurs and ULSR[FE] is set. Note that only the first STOP bit is checked. In FIFO mode, ULSR[FE] is set when the character at the top of the FIFO detects a framing error. An attempt to resynchronize occurs after a framing error. The UART assumes that the framing error (due to a logic 0 being read when a logic 1 (STOP) was expected) was due to a STOP bit overlapping with the next START bit. ULSR[FE] is cleared when ULSR is read or when a new character is loaded into the URBR from the receiver shift register.

22.4.4.2 Parity Error

When unexpected parity values are encountered while receiving data, a parity error occurs and ULSR[PE] is set. In FIFO mode, ULSR[PE] is set when the character with the error is at the top of the FIFO. ULSR[PE] is cleared when ULSR is read or when a new character is loaded into the URBR.

22.4.4.3 Overrun Error

When a new (overwriting character) STOP bit is detected and the old character is lost, an overrun error occurs and ULSR[OE] is set. In FIFO mode, ULSR[OE] is set after the receiver FIFO is full (despite the receiver FIFO trigger level setting) and a new character has been received into the internal receiver shift register. Data in the FIFO is not overwritten; only the shift register data is overwritten. Therefore, the interrupt occurs immediately. ULSR[OE] is cleared when ULSR is read.

22.4.5 FIFO Mode

The UARTs use an alternate mode (FIFO mode) to relieve the processor core from excessive software overhead. The FIFO control register (UFCR) is used to enable and clear the receiver and transmitter FIFOs and set the FIFO receiver trigger level UFCR[RTL] to control the received data available interrupt UIER[ERDAI].

The UFCR also selects the type of DMA signaling. The UDSR[RXRDY] indicates the status of the receiver FIFO. UDSR[TXRDY] indicate when the transmitter FIFO is full. When in FIFO mode, data written to UTHR is placed into the transmitter FIFO. The first byte written to UTHR is the first byte onto the UART bus.

22.4.5.1 FIFO Interrupts

In FIFO mode, the UIER[ERDAI] is set when a time-out interrupt occurs. A receive data time-out generates a maskable interrupt condition (through UIER[ERDAI]). See Section 22.3.1.4, “Interrupt Enable Registers (UIER1 and UIER2).”

UIIR indicates whether the FIFOs are enabled. UIIR[IID3] is set only for FIFO mode interrupts. The character time-out interrupt occurs when no characters have been removed from or input to the receiver FIFO during the last four character times and at least one character is in the receiver FIFO. The character time-out interrupt (controlled by UIIR[IID*n*]) is cleared when URBR is read. See Section 22.3.1.5, "Interrupt ID Registers (UIIR1 and UIIR2)."

UIIR[FE] indicates whether FIFO mode is enabled.

22.4.5.2 DMA Mode Select

UDSR[RXRDY] reflects the status of the receiver FIFO or URBR. In mode 0 (UFCR[DMS] is cleared), UDSR[RXRDY] is cleared when at least one character is in the receiver FIFO or URBR; it is set when there are no more characters in the receiver FIFO or URBR. This occurs regardless of the UFCR[FEN] setting. In mode 1 (UFCR[DMS] and UFCR[FEN] are set), UDSR[RXRDY] is cleared when the trigger level or a time-out has been reached; it is set when there are no more characters in the receiver FIFO.

UDSR[TXRDY] reflects the status of the transmitter FIFO or UTHR. In mode 0 (UFCR[DMS] is cleared), UDSR[TXRDY] is cleared when there are no characters in the transmitter FIFO or UTHR; it is set after the first character is loaded into the transmitter FIFO or UTHR. This occurs regardless of the UFCR[FEN] setting. In mode 1 (UFCR[DMS] and UFCR[FEN] are set), UDSR[TXRDY] is cleared when there are no characters in the transmitter FIFO or UTHR; it is set when the transmitter FIFO is full.

See Section 22.3.1.13, "DMA Status Registers (UDSR1 and UDSR2)," for a complete description of the USDR[RXRDY] and USDR[TXRDY] bits.

22.4.5.3 Interrupt Control Logic

An interrupt is active when DUART interrupt ID register bit 7 (UIIR[IID0]), is cleared. UIER is used to mask specific interrupt types. See Section 22.3.1.4, "Interrupt Enable Registers (UIER1 and UIER2)."

When the interrupts are disabled in UIER, polling software can not use UIIR[IID0] to determine whether the UART is ready for service. Software must monitor the appropriate ULSR and UMSR bits. UIIR[IID0] can be used for polling if the interrupts are enabled in UIER.

22.5 DUART Initialization/Application Information

The following requirements must be met for DUART accesses:

- All DUART registers must be mapped to a cache-inhibited and guarded area. (That is, the WIMG setting in the MMU needs to be 0b01x1.)
- All DUART registers are 1 byte wide. Reads and writes to these registers must be byte-length operations.

A system reset puts the DUART registers to a default state. Before the interface can transfer serial data, the following initialization steps are recommended:

1. Update the programmable interrupt controller (PIC) DUART channel interrupt vector source registers.
2. Set data attributes and control bits in the ULCR, UFCR, UAFR, UMCR, UDLB, and UDMB.

3. Set the data attributes and control bits of the external MODEM or peripheral device.
4. Set the interrupt enable register (UIER).
5. To start a write transfer, write to the UTHR.
6. Poll UIIR if the interrupts generated by the DUART are masked.

Chapter 23
Serial Peripheral Interface

23.1 Overview

The serial peripheral interface (SPI) allows the device to exchange data between other PowerQUICC® family chips, the MC68360, MC68302, M68HC11, and M68HC05 microcontroller families, and other family devices. The SPI can be used to communicate with peripheral devices such as EEPROMs, real-time clocks, A/D converters, and ISDN devices.

The SPI is a full-duplex, synchronous, character-oriented channel that supports a four-wire interface (receive, transmit, clock, and slave select). The SPI block consists of transmitter and receiver sections, an independent baud-rate generator, and a control unit. The transmitter and receiver sections use the same clock, which is derived from the SPI baud rate generator in master mode or externally in slave mode. During an SPI transfer, data is sent and received simultaneously.

The SPI receiver and transmitter are double-buffered, as shown in Figure 23-1, giving an effective FIFO size (latency) of two characters. The SPI's MSB/LSB is shifted out first. When the SPI is disabled in the SPI mode register (SPMODE[EN] = 0), it consumes little power.

23.2 Introduction

The SPI block diagram is shown in Figure 23-1.

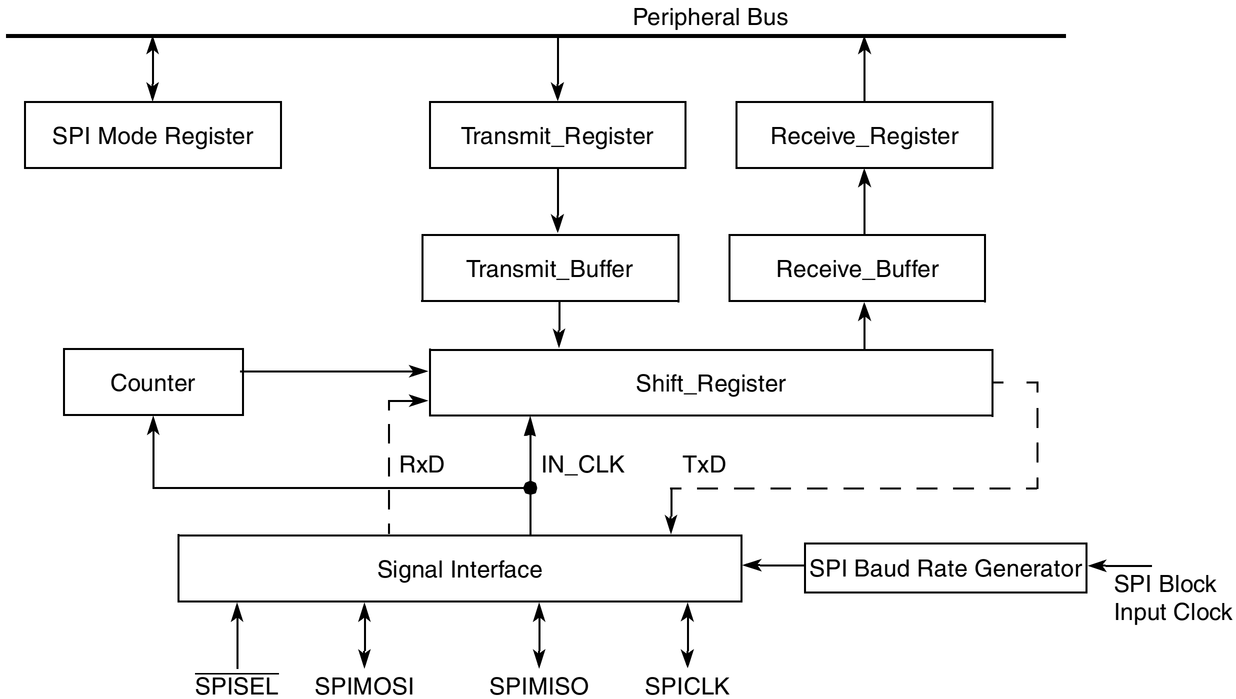


Figure 23-1. SPI Block Diagram

23.2.1 Features

The major features of the SPI are listed as follows:

- Four-signal interface (SPIMOSI, SPIMISO, SPICLK, and $\overline{\text{SPISEL}}$)
- Full-duplex operation
- Works with 32-bit data characters or with a range from 4-bit to 16-bit data characters
- Supports back-to-back character transmission and reception
- Supports reverse data mode for 8/16/32 character length
- Supports master SPI mode
- Supports multiple-master environment
- Maximum clock rate is (input clock rate/4) in master mode; (input clock rate/2) in slave mode
- Independent programmable baud rate generator
- Programmable clock phase and polarity
- Local loopback capability for testing
- Open-drain outputs support multiple-master configuration

23.2.2 SPI Transmission and Reception Process

Because the SPI is a character-oriented communication unit, the core is responsible for packing and unpacking the receive and transmit frames. A frame consists of all of the characters transmitted or received during a completed SPI transmission session, from the first character written to the SPITD register to the last character transmitted following the setting of SPCOM[LST]. See Section 23.4.1.4, "SPI Command Register (SPCOM)," for more information.

The core receives data by reading the SPI receive data hold register (SPIRD). The SPI then clears the not empty SPIE[NE] to free up the SPIRD register for the next receive operation. The core transmits data by writing it into the SPI transmit data hold register (SPITD). The SPI then clears the not full (NF) bit in the SPI event register (SPIE) to indicate that the SPITD register contains a character for transmission. When the next character to be transmitted is going to be the final one in the current frame, the core sets SPCOM[LST], and then writes the final character to SPITD.

The SPI core handshake protocol can be implemented by either using polling or interrupts.When using a polling, the core reads the SPIE in a predefined frequency and acts according to the value of the SPIE bits.The polling frequency depends on the SPI serial channel frequency.When using the interrupt mechanism, setting either the not full (NF) or not empty (NE) bits of SPIE causes an interrupt to the processor core.The core then reads SPIE and acts accordingly. The three basic modes of operation for transmitting and receiving are master, slave, and multiple-master.

NOTE

When both NE and NF bits are set, the processor core should read the received data before transmitting new data.

The SPMODE[LEN] determines the character length sent by the hardware. The core is responsible for any bit manipulation to pack/unpack data into the appropriate character length. See the SPMODE[LEN] description in Table 23-4 for more information.

23.2.3 Modes of Operation

The SPI can be programmed to work in a single- or multiple-master environment. This section describes SPI master and slave operations in a single-master configuration. It also discusses the multiple master environment.

The following sections summarize the main modes of operation that the SPI supports.

23.2.3.1 SPI as a Master Device

In master mode, the SPI sends a message to the slave peripheral, which sends back a simultaneous reply. A single master device with multiple slaves can use general-purpose parallel I/O signals to selectively enable slaves, as shown in Figure 23-2. To eliminate the multi-master error in a single-master environment, the master's SPISEL input should be forced inactive by an external pull up.

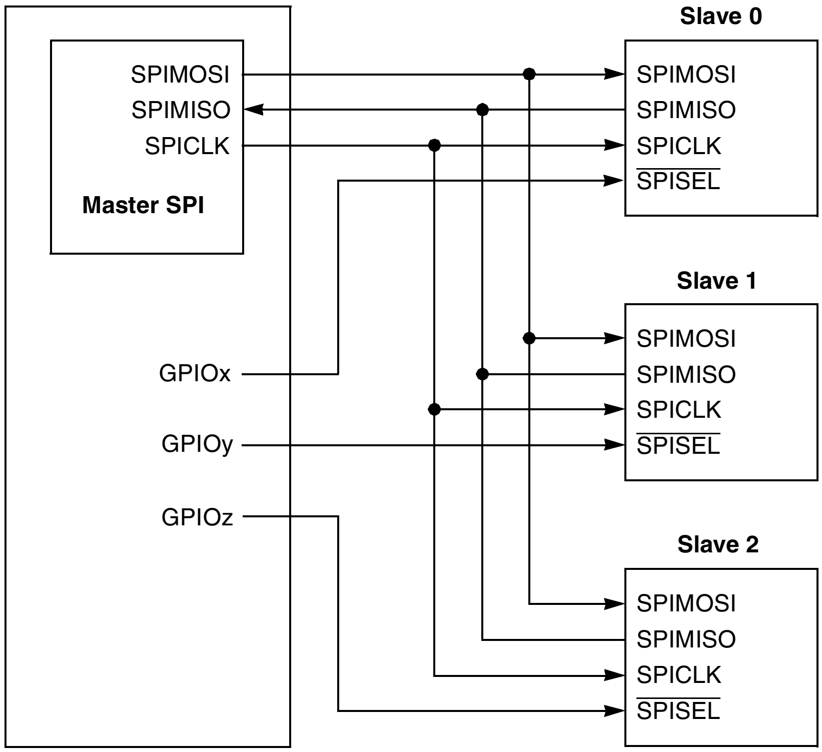


Figure 23-2. Single-Master/Multi-Slave Configuration

To start exchanging data, the processor core writes the data to be sent into the SPITD register. The SPI then generates programmable clock pulses on SPICLK for each character. It shifts Tx data out on the SPI master-out slave-in (SPIMOSI) and Rx data in on the SPI master-in slave-out (SPIMISO) simultaneously. During transmission, the core is responsible for supplying the data whenever the SPI requests it to ensure smooth operation. After the last data (LST command and data afterwards), the first character written to SPITD acts as a start command for the SPI.

The SPI continues transmitting and receiving characters until SPCOM[LST] is set or an error occurs.

The SPI sets SPIE[NF] to issue a maskable interrupt to the interrupt controller whenever its transmit buffer is not full. It also sets the NF bit after sending the last word. In response, the core should read the exception flags that relate to the last word. The SPI sets SPIE[NE] to issue a maskable interrupt to the interrupt controller whenever the receiver buffer has been filled with data.

23.2.3.2 SPI as a Slave Device

In slave mode, the SPI receives messages from an SPI master and sends a simultaneous reply. The slave's SPISEL must be asserted before Rx clocks are recognized. Once SPISEL is asserted, SPICLK becomes an input from the master to the slave. SPICLK can be any frequency from DC to input clock/2.

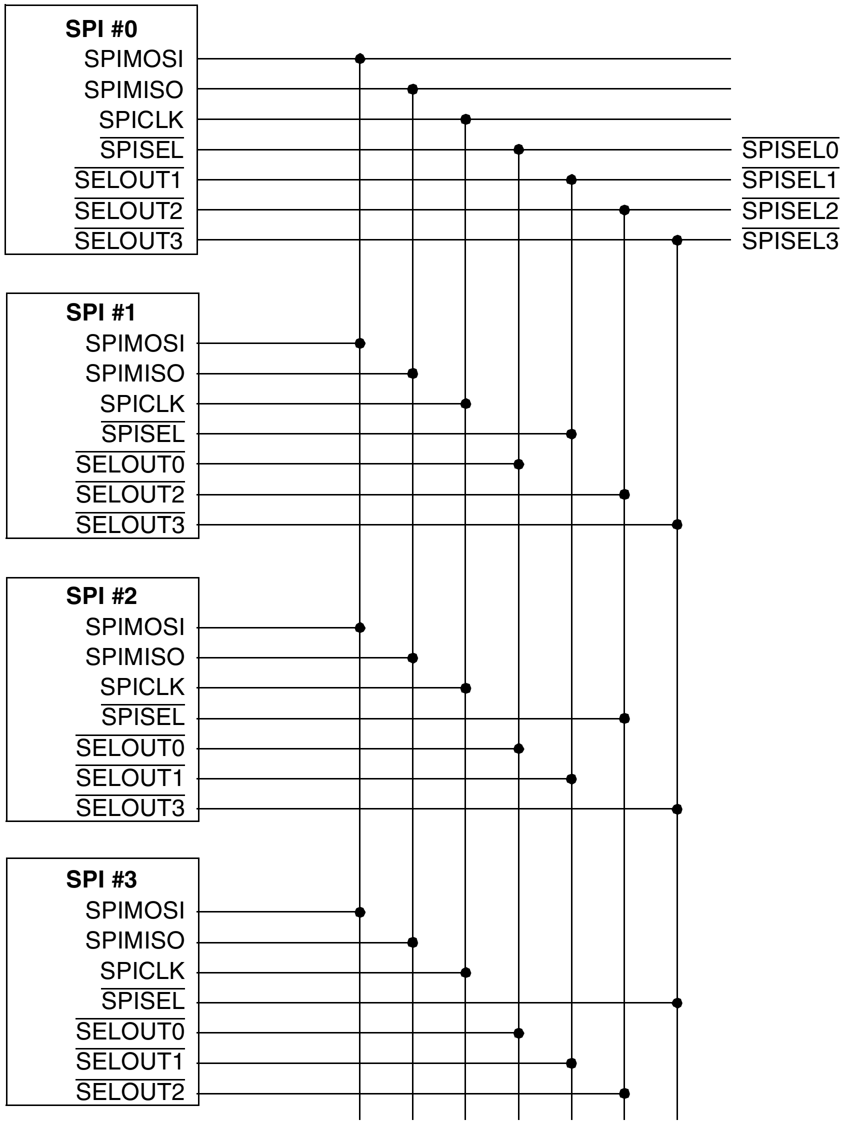
To prepare for data transfers, the core writes data to be sent into the SPITD register. Once SPISEL is asserted, the slave shifts data out from SPIMISO and in through SPIMOSI. The SPI sets the NF bit of the

SPIE register and a maskable interrupt is issued when a full buffer finishes receiving and sending or after an error. The SPI continues reception until $\overline{\text{SPISEL}}$ is negated.

Transmission continues until no more data is available or $\overline{\text{SPISEL}}$ is negated. Transmission continues once $\overline{\text{SPISEL}}$ is reasserted and SPICLK begins toggling. After the characters in the buffer are sent, the SPI sends one as long as $\overline{\text{SPISEL}}$ remains asserted.

23.2.3.3 SPI in Multiple-Master Operation

The SPI can operate in a multiple-master environment in which all SPI devices are connected to the same bus. In this configuration, the SPIMOSI, SPIMISO, and SPICLK signals of all SPIs are shared; but the $\overline{\text{SPISEL}}$ inputs are connected separately, as shown in Figure 23-3. Only one SPI device can act as master at a time—all others must be slaves. When a SPI is configured as a master, if its $\overline{\text{SPISEL}}$ input is asserted, a multiple-master error occurs because more than one SPI device is a bus master. The SPI sets SPIE[MME] in the SPI event register and a maskable interrupt is issued to the core. It also disables SPI operation and the output drivers of the SPI signals. The core must clear SPMODE[EN], correct the problems, and clear SPIE[MME] before the SPI can be used again.



Notes:
1. All signals are open-drain.
2. For a multiple-master configuration with more than two masters, SPISEL and SPIE[MME] do not detect all possible conflicts.
3. It is the responsibility of software to arbitrate for the SPI bus (with token passing, for example).
4. SELOUT*x* signals are implemented in software with general-purpose I/O signals.

Figure 23-3. Multiple-Master Configuration

The maximum sustained data rate that the SPI supports is input clock/50. However, the SPI can transfer a single character at much higher rates—input clock/4 in master mode and input clock/2 in slave mode, and subjected to the timing parameters of the interconnected devices, and board trace delays. Gaps should be inserted between multiple characters to keep from exceeding the maximum sustained data rate.

23.3 External Signal Descriptions

The SPI's four wire interface consists of transmit, receive, clock, and slave select.

23.3.1 Overview

Table 23-1 lists signal properties.

Table 23-1. Signal Properties

Name	Function	Reset	Pull Up
SPIMISO	Master input slave output	—	Required in open drain mode
SPIMOSI	Master output slave input	—	Required in open drain mode
SPICLK	Input/output serial clock connected to the other SPICLK	—	Required in open drain mode
$\overline{\text{SPISEL}}$	SPI slave select	—	Required in open drain mode

23.3.2 Detailed Signal Descriptions

Table 23-2 describes the signals in detail.

Table 23-2. Detailed Signal Descriptions

Signal	I/O	Description	
SPIMISO	I/O	Master input slave output	
		State Meaning	Asserted—The data that has been transmitted/received from/to the SPI (depends if master or slave mode) is high Negated—The data that has ben transmitted/received from/to the SPI (depends if master or slave mode) is low
		Timing	Assertion—According to the SPICLK assertion/negation/in the middle of phase (depends on SPMODE) Negation—According to the SPICLK assertion/negation/in the middle of phase (depends on SPMODE)
SPIMOSI	I/O	Master output slave input	
		State Meaning	Asserted—The data that has been transmitted/received from/to the SPI (depends if master or slave mode) is high Negated—The data that has ben transmitted/received from/to the SPI (depends if master or slave mode) is low
		Timing	Assertion—According to the SPICLK assertion/negation/in the middle of phase (depends on SPMODE) Negation—According to the SPICLK assertion/negation/in the middle of phase (depends on SPMODE)

Table 23-2. Detailed Signal Descriptions (continued)

Signal	I/O	Description	
SPICLK	I/O	Serial clock in or serial clock out for slave or master mode respectively	
		State Meaning	Assertion/Negation according to SPMODE[PM,DIV16] register rate configuration
		Timing	Assertion/Negation—during frame reception/transmission
$\overline{\text{SPISEL}}$	I	SPI slave select	
		State Meaning	Asserted—In slave mode declares the slave has been selected for the coming frame. In master mode assertion causes MME multiple-master error. Negated—In slave mode means the specific SPI has not been selected. In master mode needs to be negated for regular operation.
		Timing	Assertion—In slave mode along with the data from the slave Negation—In slave mode with the end of the frame (according to SPMODE[LEN]). In master mode before data is first written to SPITD and remains constant.

The SPI can be configured as a slave or a master in single- or multiple-master environments mode. The master SPI generates the transfer clock SPICLK using the SPI baud rate generator (BRG).The SPI BRG takes its input from input clock, which is generated in the device clock synthesizer.

SPICLK is a gated clock, active only during data transfers. Four combinations of SPICLK phase and polarity can be configured with the clock invert (SPMODE[CI]) and clock phase (SPMODE[CP]) register bits. SPI signals can also be configured as open-drain to support a multiple-master configuration in which a shared SPI signal is driven by the device or an external SPI device.

The SPI master-in slave-out SPIMISO signal acts as an input for master devices and as an output for slave devices. Conversely, the master-out slave-in SPIMOSI signal is an output for master devices and an input for slave devices. The dual functionality of these signals allows the SPIs in a multiple-master environment to communicate with one another using a common hardware configuration.

- When the SPI is a master, SPICLK is the clock output signal that shifts received data in from SPIMISO and transmitted data out to SPIMOSI. SPI masters must output a slave select signal to enable SPI slave devices by using a separate general-purpose I/O signal. Assertion of the $\overline{\text{SPISEL}}$ while the SPI is configured as a master causes an error.
- When the SPI is a slave, SPICLK is the clock input that shifts received data in from SPIMOSI and transmitted data out through SPIMISO. $\overline{\text{SPISEL}}$ is the enable input to the SPI slave. In a multiple-master environment, $\overline{\text{SPISEL}}$ (always an input) is also used to detect an error when more than one master is operating.

23.4 Memory Map/Register Definition

Table 23-3 shows the memory mapped registers of the SPI and their offsets. It lists the offset, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of IMMRBAR together with the SPI block base address and offset listed in Table 23-3. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 23-3. SPI Register Summary

Offset	Register	Access	Reset Value	Section/Page
0x000–0x01F	Reserved	—	—	—
0x020	SPI mode register (SPMODE)	R/W	0x0000_0000	23.4.1.1/2323-9
0x024	SPI event register (SPIE)	Mixed	0x0000_0000	23.4.1.2/2323-12
0x028	SPI mask register (SPIM)	R/W	0x0000_0000	23.4.1.3/2323-13
0x02C	SPI command register (SPCOM)	W	0x0000_0000	23.4.1.4/2323-14
0x030	SPI transmit register (SPITD)	W	0x0000_0000	23.4.1.5/2323-14
0x034	SPI receive register (SPIRD)	R	0xFFFF_FFFF	23.4.1.6/2323-15
0x038–0xFFF	Reserved	—	—	—

23.4.1 Register Descriptions

23.4.1.1 SPI Mode Register (SPMODE)

SPMODE, shown in Figure 23-4, controls both the SPI operation mode and clock source.

Offset 0x020 | Access: Read/write

	0	1	2	3	4	5	6	7	8–11	12–15	16–18	19	20–31
R/W	—	LOOP	CI	CP	DIV16	REV	M/S	EN	LEN	PM	—	OD	—
Reset	All zeros												

Figure 23-4. SPMODE-SPI Mode Register Definition

Table 23-4 describes the SPMODE fields.

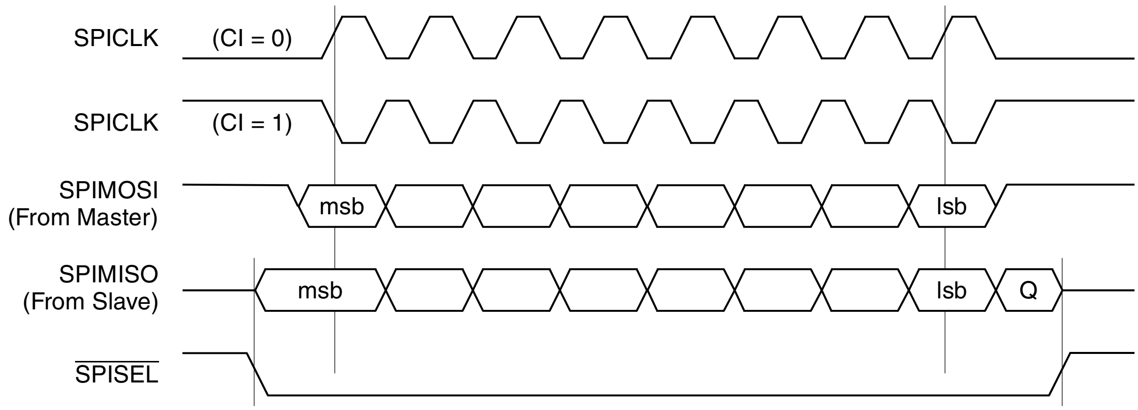
Table 23-4. SPMODE Field Descriptions

Bits	Name	Description
0	—	Reserved. Should be cleared.
1	LOOP	Loop mode. Enables local loopback operation. 0 Normal operation. 1 Loopback mode. Used to test the SPI controller internal functionality, the transmitter output is internally connected to the receiver input. The receiver and transmitter operate normally, except that received data is ignored. The SPI acts normally in loop back mode; therefore, negating SPISEL in slave mode stops transmission, negating it in master mode and causing and MME error.
2	CI	Clock invert. Inverts SPI clock polarity. See Figure 23-5 and Figure 23-6 for more information 0 The inactive state of SPICLK is low. 1 The inactive state of SPICLK is high.

Table 23-4. SPMODE Field Descriptions (continued)

Bits	Name	Description
3	CP	Clock phase. Selects the transfer format. See Figure 23-5 and Figure 23-6 for more information. 0 SPICLK starts toggling at the middle of the data transfer. 1 SPICLK starts toggling at the beginning of the data transfer.
4	DIV16	Divide by 16. Selects the clock source for the SPI baud rate generator (SPI BRG) when configured as an SPI master. In slave mode, SPICLK is the clock source. 0 The SPI block input clock is the input to the SPI BRG. 1 The SPI block input clock/16 is the input to the SPI BRG. In slave mode this bit must be cleared.
5	REV	Reverse data mode for 8-/16-/32-bit character length only (see Section 23.4.1.6.1, “Reverse Mode SPMODE[REV] Examples.”) 0 LSB sent/received first (for data LEN < 32 the data is located at the lower half-word LSB) 1 MSB sent/received first
6	M/S	Master/slave. Selects master or slave mode. 0 The SPI is a slave. 1 The SPI is a master.
7	EN	Enable SPI. Any other bits in SPMODE must not change when EN is set. 0 The SPI is disabled. The SPI is in a idle state and consumes minimal power. The SPI BRG is not functioning and the input clock is disabled. 1 The SPI is enabled. **Note:** The SPI controller requires a minimal gap of at least 10 input clocks between disabling the SPI and re-enabling. This minimal gap is sufficient provided that SPMODE[PM] and SPMODE[DIV16] are cleared during the time in which SPMODE[EN] is cleared.
8–11	LEN	Character length in bits per character. LEN can be either 32-bits, or 4- to 16-bits that are shown as follows: 0000 32-bit characters 0001–0010 Reserved, causes erratic behavior. 0011 4-bit characters … 1111 16-bit characters The TX and RX registers (SPITD, SPIRD) hold 32 bits at a time. A character length of 32 bits fills the TX and RX registers; therefore, all of the bits in these registers are valid. However, if the character length selected by LEN is equal or less than 16 bits, then the valid bits will reside in the lower half-word of the transmit and receive registers. For example, if the character length is set to 16 bits than the valid bits will be 16–31, if the character length is set to 5 bits that the valid bits will be16–20. Note that the transmit and receive registers each can hold only one character regardless of the character length.
12–15	PM	Prescale modulus select. Specifies the divide ratio of the prescale divider in the SPI clock generator. The SPI baud rate generator clock source (either input clock or input clock divided by 16, depending on DIV16 bit) is divided by 4 * ([PM] + 1), a range from 4 to 64. The clock has a 50% duty cycle. For example, if the prescale modulus is set to PM = 0011 and DIV16 is set, the system/SPICLK clock ratio will be 16 * (4 * (0011 + 1)) = 256. In slave mode this field must be cleared.
16–18	—	Reserved. Should be cleared.
19	OD	Open drain mode. 0 All output pins are configured to normal mode. 1 All output pins are configured to open drain mode.
20–31	—	**Note:** Reserved. Should be cleared.

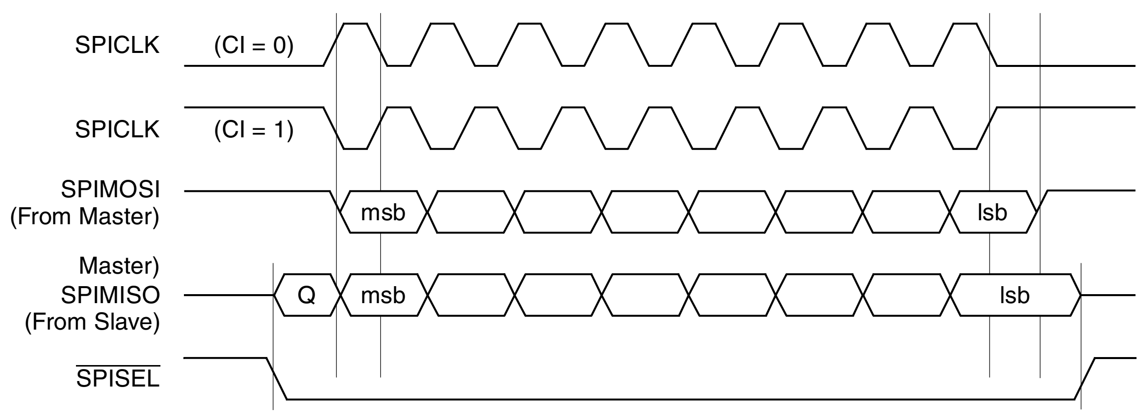
Figure 23-5 shows the SPI transfer format in which SPICLK starts toggling in the middle of the transfer (SPMODE[CP] = 0).



NOTE: Q = Undefined signal.

Figure 23-5. SPI Transfer Format with SPMODE[CP] = 0

Figure 23-6 shows the SPI transfer format in which SPICLK starts toggling at the beginning of the transfer (SPMODE[CP] = 1).



NOTE: Q = Undefined signal.

Figure 23-6. SPI Transfer Format with SPMODE[CP] = 1

23.4.1.2 SPI Event Register (SPIE)

The SPI event register (SPIE) generates interrupts and reports events recognized by the SPI. When an event is recognized, the SPI sets the corresponding SPIE bit. Most SPIE bits can be cleared by writing a ‘1’. Writing ‘0’ has no effect. Setting a bit in the SPI mask register (SPIM) enables, and clearing a bit masks the corresponding interrupt. Unmasked SPIE bits must be cleared before the core clears internal interrupt requests. Figure 23-7 shows SPI event register.

Offset 0x024 Access: Mixed

	0–16	17	18	19	20	21	22	23	24–31
R	—	LT	DNR	OV	UN	MME	NE	NF	—
W		w1c	w1c	w1c	w1c	w1c			
Reset	All zeros								

Figure 23-7. SPIE—SPI Event Register Definition

Table 23-5 describes the SPIE fields.

Table 23-5. SPIE Field Descriptions

Bits	Name	Description
0–16	—	Reserved, should be cleared.
17	LT	Last character was transmitted. The last character is transmitted and new data can be written to SPID for further transmission.
18	DNR	**Note:** Data not ready. In slave mode only when $\overline{\text{SPISEL}}$ is asserted before data is ready in the SPI, IDLE is sent on the line and UN bit is also asserted the SPI should be disable to restart its operation.
19	OV	Slave/master overrun. Indicates whether an overrun has occurred during reception. In case of overrun the SPI continues transmission/reception process while reporting overrun for the missing characters.
20	UN	Slave underrun. Indicates whether the SPI transmitter did not have data to transmit on time, and therefore, whether IDLE was sent on the line. Valid only in slave mode (SPMODE[M/S]) = 0. In master mode (SPMODE[M/S]) = 1) if the SPI's transmitter has no valid data to transmit the SPICLK stop toggling and transmission/reception is frozen (no underrun is reported), when data is written to the SPITD the transmission resumes.
21	MME	Multiple-master error. Set when $\overline{\text{SPISEL}}$ is asserted externally while the SPI is in master mode. Note that the MME error can occur in loopback mode.
22	NE	Not empty. When set Indicates that SPIRD contains a received character. 0 The receiver is empty 1 The receiver has valid received data and indications about LST (command register) and OV (SPIE).The core is free to read the content of the receiver. Reading the receiver SPIRD clears NE if no more data is available.
23	NF	Not full. Indicates whether SPITD is not in use and a new character can be written to it by the core. 0 The transmitter is full. 1 The transmitter is not full. The core is free to write to the transmitter. NF must be clear to enable the transmission of another character (writing to the transmitter clears NF)
24–31	—	Reserved. Should be cleared.

23.4.1.3 SPI Mask Register (SPIM)

The SPI mask register (SPIM), shown in Figure 23-8, enables/masks interrupts for events that are recognized by the SPI. When an event is recognized, the SPI sets the corresponding SPIE bit. Setting a

SPIM bit enables and clearing a SPIM bit masks the corresponding interrupt. Unmasked SPIE bits must be cleared before the core clears its internal interrupt requests.

Offset 0x028 Access: Read/write

	0–16	17	18	19	20	21	22	23	24–31
R W	—	LT	DNR	OV	UN	MME	NE	NF	—
Reset	All zeros								

Figure 23-8. SPIM—SPI Mask Register Definition

Table 23-6 describes the SPIM fields.

Table 23-6. SPIM Field Descriptions

Bits	Name	Description
0–16	—	Reserved, should be cleared.
17	LT	Last character transmitted 0 LT event will not cause an SPI interrupt 1 LT event causes an SPI interrupt
18	DNR	In slave mode data not ready 0 Slave DNR event will not cause an SPI interrupt 1 Slave DNR event causes an SPI interrupt
19	OV	Slave/Master Overrun interrupt mask 0 Slave/Master Overrun event will not cause an SPI interrupt 1 Slave/Master Overrun event causes an SPI interrupt
20	UN	Slave Underrun interrupt mask 0 Slave Underrun event will not cause an SPI interrupt 1 Slave Underrun event causes an SPI interrupt
21	MME	Multimaster error interrupt mask 0 Multimaster error event will not cause an SPI interrupt 1 Multimaster error event causes an SPI interrupt
22	NE	Not Empty interrupt mask 0 Not Empty event will not cause an SPI interrupt 1 Not Empty event causes an SPI interrupt
23	NF	Not Full interrupt mask 0 Not Full event will not cause an SPI interrupt 1 Not Full event causes an SPI interrupt
24–31	—	Reserved, should be cleared.

23.4.1.4 SPI Command Register (SPCOM)

The SPI command register (SPCOM), shown in Figure 23-9, is used to end SPI operation.

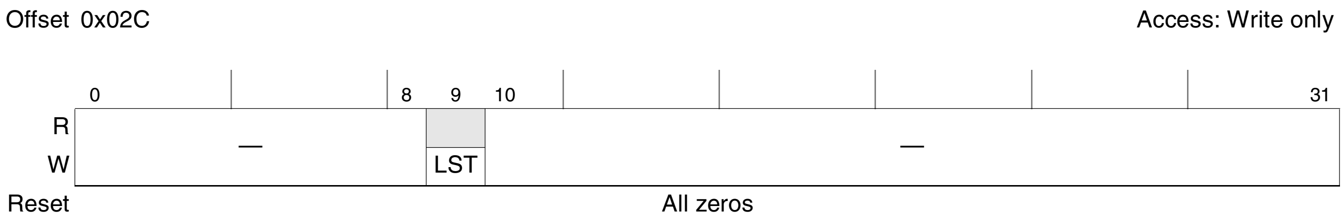


Figure 23-9. SPI Command Register Definition

Table 23-7 describes the SPCOM fields.

Table 23-7. SPCOM Field Descriptions

Bits	Name	Description
0–8	—	Reserved, should be cleared.
9	LST	This bit represents the last character. Should be set before the last character is written to the SPITD. This results in SPIE[LT] being set when the character is fully transmitted and by that gives indication about the frame being fully transmitted. 0 This character is not the last character of the frame 1 This character is the last character of the frame
10–31	—	Reserved, should be cleared.

23.4.1.5 SPI Transmit Data Hold Register (SPITD)

SPITD holds the character to be transmitted. The number of bits in each character is specified by SPMODE[LEN]. Each time SPIE[NF] is set, the core can write another character of data to SPITD, if there is no error indication in the SPIE. At the end of the frame the core should set SPCOM[LST] and prepare the last character of data. Figure 23-10 shows the SPI transmit data hold register.

Figure 23-10. SPI Transmit Data Hold Register Definition

Table 23-8 shows the field descriptions of the SPI transmit data hold register.

Table 23-8. SPI Transmit Data Hold Field Descriptions

Bits	Name	Description
0–31	DATA	These bits are the data to be sent.

23.4.1.6 SPI Receive Data Hold Register (SPIRD)

SPIRD, shown in Figure 23-11, is used to receive a character of data from the SPI channel. Each time SPIE[NE] is set, the core can read SPIRD.

Offset 0x034 Access: Read-only

	0–31
R	DATA
W	
Reset	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 23-11. SPI Receive Data Hold Register Definition

Table 23-9 shows the field descriptions of the SPI receive data hold register.

Table 23-9. SPI Receive Data Hold Field Descriptions

Bits	Name	Description
0–31	DATA	Received data. These bits are the received data from the SPI bus.

23.4.1.6.1 Reverse Mode SPMODE[REV] Examples

In reverse data mode (SPMODE[REV] = 1) and regular data mode (SPMODE[REV] = 0) the data is placed in the SPIRD after reception is completed as described below for character length of 8 bits (SPMODE[LEN] = 7).

Offset 0x036 Access: Read-only

	0–15	16	17–22	23	24–31
R	—	MSB	DATA	LSB	—
W					
Reset	All zeros				

Figure 23-12. Example SPMODE[REV] = 0 SPMODE[LEN] = 7 LSB Sent First

Offset 0x036 Access: Read-only

	0–23	24	25–30	31
R	—	MSB	DATA	LSB
W				
Reset	All zeros			

Figure 23-13. Example SPMODE[REV] = 1 SPMODE[LEN] = 7 MSB Sent First

Offset 0x036 | Access: Read-only

Bits	0–15	16	17–30	31
R	—	MSB	DATA	LSB
W				
Reset	All zeros			

Figure 23-14. Example SPMODE[REV] = 1 SPMODE[LEN] = 15 MSB Sent First

Offset 0x036 | Access: Read-only

Bits	0–15	16	17–30	31
R	—	MSB	DATA	LSB
W				
Reset	All zeros			

Figure 23-15. Example SPMODE[REV] = 0 SPMODE[LEN] = 15 LSB Sent First

23.5 SPI Boot ROM

This section contains:

- Section 23.5.1, “Overview”
- Section 23.5.2, “Features”
- Section 23.5.3, “EEPROM Data Structure”
- Section 23.5.4, “ROM Data Structure”
- Section 23.5.5, “SPI Controller Configuration”

23.5.1 Overview

The MPC8379E is capable of loading initialization code from a memory device that is connected to the SPI controller interface. This device can be either an EEPROM or a serial Flash with an SPI-compatible interface. The term EEPROM will be used when referring to the memory device.

Boot from SPI is supported using an on-chip ROM which contains the basic SPI device driver and the code to perform block copy from SPI EPROM to any target memory. Selecting on-chip ROM in boot ROM location, see Table 4-16, causes the e300 CPU to fetch instructions from the on-chip ROM. The on-chip ROM is selected by the first six hard-coded default reset configuration words options. See Section 4.3.3.3, “Default Reset Configuration Words,” for more details. Note that if neither of the six hard-coded RCW options meet the application requirements, the application can choose to load RCW from I^2C EPROM. In this case, the RCW, which is loaded from the I^2C EPROM, should set ROMLOC to on-chip ROM. See Section 4.3.2.2.4, “Boot ROM Location,” for more details.

After the device has completed the reset sequence, if the ROM location selects the on-chip ROM, the e300 core starts to execute code from the internal on-chip ROM. The e300 core configures the SPI controller, enabling it to communicate with the external EEPROM. The EEPROM should contain a specific data structure with control words, device configuration information and initialization code. The on-chip ROM boot code uses the information from the EEPROM content to configure the device, and to copy the initialization code to a target memory device (for example, the DDR) through the SPI interface. After all the code has been copied, the e300 core starts to execute the code from the target memory device.

23.5.2 Features

The key features are as follows:

- Provides mechanism to load initialization code from external SPI EEPROM
- Simple data structure in SPI EEPROM
- BOOT signature will be checked to validate that the EEPROM contains valid code
- Supports variable code length in EEPROM
- Flexible target memory device
- Supports target memory configuration controlled by the user
- Supports standard SPI interface EEPROMs with read instruction code 0x03 followed by a 3-byte address
- Initial setting will generate a serial clock below 5 MHz; the control word will allow for user modification

23.5.3 EEPROM Data Structure

The EEPROM should contain the initialization code length in bytes, source address in the SPI EEPROM, destination address in the target memory device, execution starting address, and multiple configuration words with pairs of target address and its respective data.

Figure 23-16 shows the required SPI EEPROM data structure.

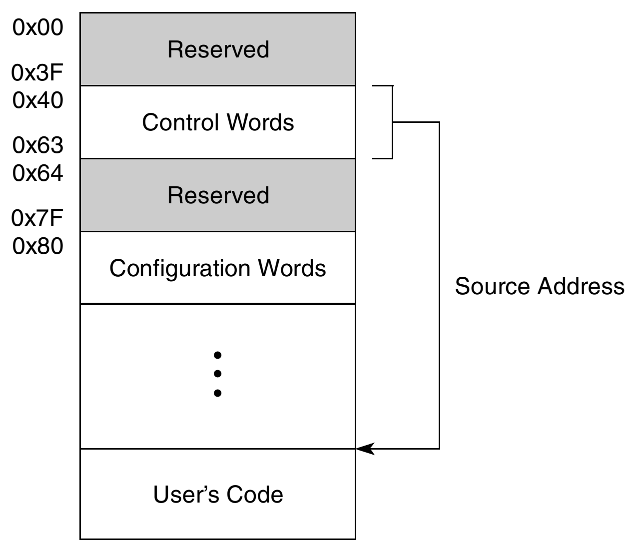


Figure 23-16. SPI EEPROM Data Structure

Table 23-10 shows the required data structure in details.

Table 23-10. EEPROM Data Structure Fields

Address	Data Bits [0:31]
0x00–0x3F	Reserved.
0x40–0x43	BOOT signature. This location should contain the value 0x424f_4f54, which is the ascii code for BOOT. The SPI loader code will search for this signature. If the value in this location doesn't match the BOOT signature, it means that the SPI device doesn't contain a valid user code. In such case the SPI loader code will negate the Select signal (GPIO2[20]) and will halt the e300 CPU by setting ACR[COREDIS].
0x44–0x47	Reserved
0x48–0x4B	User's code length. Number of bytes in the user's code to be copied.
0x4C–0x4F	Reserved
0x50–0x53	Source Address. Contains the starting address of the user's code as an offset from the EEPROM starting address.
0x54–0x57	Reserved
0x58–0x5B	Target Address. Contains the target address in a CSB memory address space in which the user's code will be copied to.
0x5C–0x5F	Reserved
0x60–0x63	Execution Starting Address. Contains the jump address in a CSB memory address space into the user's code first instruction to be executed.
0x64–0x7F	Reserved.
0x80–0x83	Config Address 1
0x84–0x87	Config Data 1
0x88–0x8B	Config Address 2
0x8C–0x8F	Config Data 2
0x80 + 8 × (N–1)	Config Address N
0x80 + 8 × (N–1) + 4	Config Data N
...	...
—	User's Code

23.5.3.1 Configuration Words Section

The configuration words section is comprised of Config Address and Config Data pairs of adjacent 32-bit fields that can be generally used to configure the device system and the target memory controller's registers.

The Config Address field has two modes that are selected by the least significant bit in the field (CNT). If the CNT bit is clear, the 30 most significant bits are used as the address pointer and the Config Data contains the data to be written to this address. If the CNT bit is set, the 30 most significant bits are used for control instruction. This flexible structure allows the user to configure any 4-byte aligned memory mapped register, perform control instructions, and specify the end of the configuration stage.

Note that it is illegal to change the content of the IMMRBAR by using this mechanism. Any attempt to do so will cause the boot process to hang.

The Config Address structure is shown in Figure 23-17.

	0	1	2	3 … 29	30	31
CNT = 0	Address				—	CNT
CNT = 1	EC	DLY	CF	—		CNT

Figure 23-17. Config Address Fields

Table 23-11 defines the Config Address bits when CNT = 0 (address mode).

Table 23-11. Config Address Field Descriptions, CNT = 0

Bits	Name	Description
0-29	Address	Address bits 0-29. The data in the Config Data field is copied by the e300 core to this address. The two least significant bits of the address (30:31) are always considered to be zero.
30	-	Reserved. Must be zero.
31	CNT	Control. Select between Address mode and Control mode. 0 Address mode 1 Control mode

Table 23-12 defines the Config Address bits when CNT = 1 (control mode).

Table 23-12. Config Address Field Descriptions, CNT = 1

Bits	Name	Description
0	EC	End Configuration. Indicates the end of the configuration stage. Valid only if bit CNT is set. 0 Not the last Config Address field. 1 The Last Config Address field. The e300 core will stop the configuration stage and start to copy the user's code.
1	DLY	Delay. Instruct the e300 core to perform delay (Loop cycles) according to the number that is specified in the adjacent Config Data field. Valid only if bit CNT is set. 0 No delay. **Note:** 1 Delay. Every loop cycle takes approximately 30 csb_clk cycles.
2	CF	Change frequency. Instruct the e300 core to perform sequence of operations to setup the SPI mode register with the frequency related (PM and DIV16) bits as defined by the user. The adjacent Config Data field will be written to the SPI mode register. Software will use DIV16 and PM bits and mask all other bits such that they will not change. Software will perform the necessary steps which are required by the SPI controller before and after changing the SPI mode register.
3-29	-	Reserved. Must be zero.

Table 23-12. Config Address Field Descriptions, CNT = 1 (continued)

Bits	Name	Description
30	-	Reserved. Must be zero.
31	CNT	Control. Select between Address mode and Control mode. (0 Address mode) 1 Control mode **Note:** When CNT=1, bits 0-29 select the control instruction. Only one bit in the range of bits 0-29 can be set at any specific control instruction. A control instruction with bits 0-29 all cleared is also illegal.

Note that the user must specify at least one control word with bits CNT and EC set to indicate the end of the configuration stage.

The Config Data structure is shown in Figure 23-18.

	0 — 31
CNT = 0	Data
CNT = 1	Control Argument (optional)

Figure 23-18. Config Data Fields

23.5.4 ROM Data Structure

The internal ROM contains the code to configure the SPI controller, decode and execute the configuration data structure of the EEPROM, copy the EEPROM user's code content to the target memory device, and ends with a jump instruction to the execution starting address. The ROM contains 64 Kbytes located at address IMMRBAR + 0xF0000 to IMMRBAR + 0xFFFFF.

Table 23-13 shows the ROM data structure.

Table 23-13. ROM Data Structure

Offset	Data Bytes
0xF0000-0xF0001	ROM code major Rev ID
0xF0002-0xF0003	ROM code minor Rev ID
0xF0004-0xF001F	Reserved. All Zero
0xF0020-0xFFFFF	ROM Code

23.5.5 SPI Controller Configuration

The SPI controller configuration is used by the SPI boot ROM software. After the boot from SPI has finished, the user can change this configuration for other uses of the SPI interface.

The SPI controller is configured to operate in master mode. The $\overline{\text{SPISEL}}$ input should be forced inactive by an external pull up. The general-purpose I/O signal, TSEC1_COL/GPIO2[20], is assumed to be connected to the EEPROM $\overline{CS}$ and selectively enables the EEPROM. This signal is not being used by the TSEC because it is configured to RGMII mode as a result of using the default reset configuration words.

Figure 23-19 shows the external signal connection.

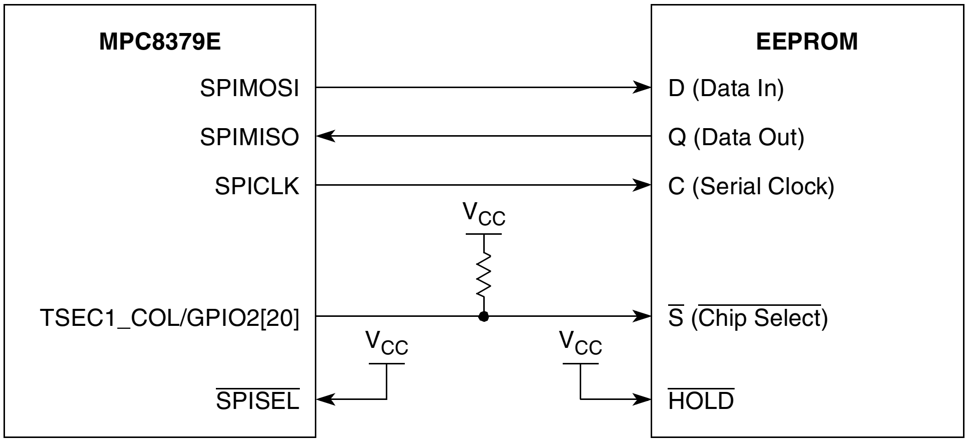


Figure 23-19. SPI External Signal Connection

The SPI controller is configured by the on-chip ROM code. The controller is configured to shift out data on SPIMOSI during the falling edge of SPICLK. It samples data in from SPIMISO during the rising edge of SPICLK. It uses 32-bit length characters and divides the system clock by 128. For example, when the *csb_clk* is configured to 333 MHz, the SPICLK will run at 2.6 MHz. (Note that frequency setting can be changed by using the CF control word, as explained in Section 23.5.3.1, “Configuration Words Section.”) When the CI bit is clear, the clock is low when the line is idle. When the REV bit is set, the MSB is sent and received first.

Figure 23-20 shows for SPI Mode Register (SPMODE) configuration.

Bits	0	1	2	3	4	5	6	7	8–11	12–15	16–18	19	20–31
Field	—	LOOP	CI	CP	DIV16	REV	M/S	EN	LEN	PM	—	OD	—
Value	0	0	0	0	1	1	1	1	0 0 0 0	0 0 0 1	0 0 0	0	0 0 0 0 0 0 0 0 0 0 0 0

Figure 23-20. SPI Mode Register (SPMODE) Configuration

The ROM code will use the SPI controller to generate standard read instruction code 0x03 followed by a 3-byte address for every non-sequential read operation (reading from a location which is not sequential to the last byte read). For sequential read operation, toggling the SPI clock will cause the SPI EEPROM to present the content of the next address location. The serial EEPROM must have an SPI compatible interface with read instruction code 0x03 followed by a 3-byte address.

Figure 23-21 shows the read instruction timing diagram.

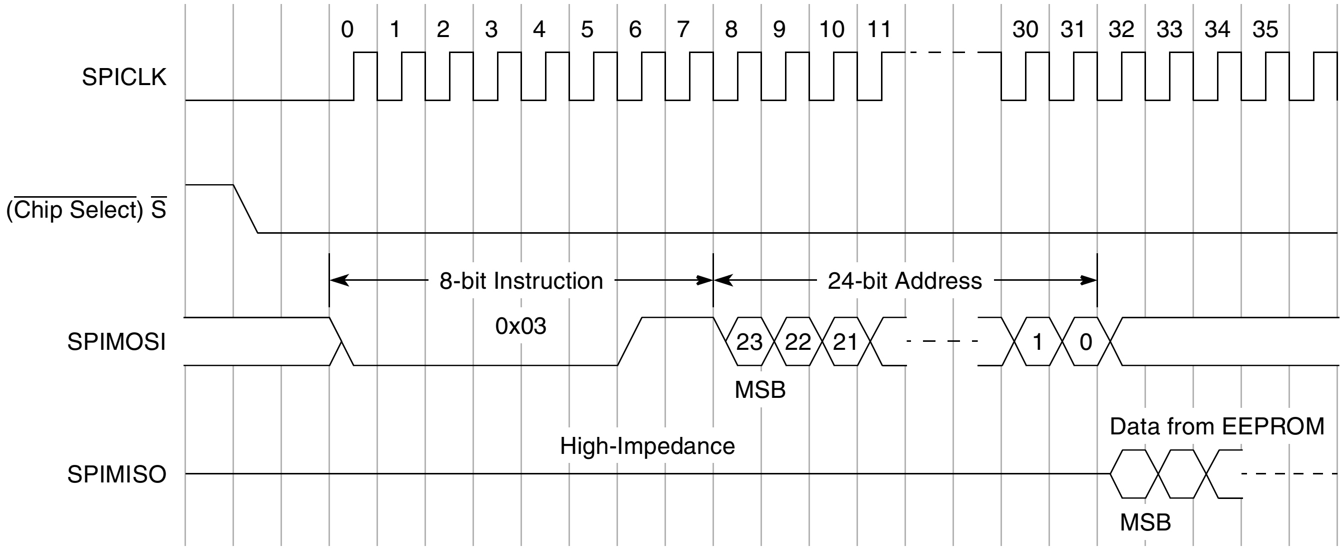


Figure 23-21. Read Instruction Timing Diagram

23.6 Initialization/Application Information

The following sections describe programming examples of the SPI master and slave.

23.6.1 SPI Master Programming Example

The following sequence initialize the SPI to run at a high speed in master mode:

1. Configure a parallel I/O signal to operate as the SPI select output signal if needed.
2. Write 0xFFFF_FFFF to SPIE to clear any previous events. Configure SPIM to enable all desired SPI interrupts.
3. Configure SPMODE to enable normal operation (not loopback), master mode, SPI enabled, character length, and the fastest speed possible.
4. Write the first character to be sent to SPITD.

23.6.2 SPI Slave Programming Example

The following is an example initialization sequence to follow when the SPI is in slave mode. It is very similar to the SPI master example, except that $\overline{\text{SPISEL}}$ is used instead of a general-purpose I/O signal.

1. Write 0xFFFF_FFFF to SPIE to clear any previous events.
2. Configure SPIM to enable all desired SPI interrupts.
3. Configure SPMODE to enable normal operation (not loopback), slave mode, SPI enabled, and characters length.
4. Write the first data to be sent to SPITD to enable the SPI to be ready once the master begins to transfer.

Chapter 24
JTAG/Testing Support

24.1 Overview

The device provides a JTAG (joint test action group) interface to facilitate boundary-scan testing. The JTAG interface is compliant with the IEEE 1149.1 boundary-scan specification. For additional information about JTAG operations, refer to the IEEE 1149.1 specification.

The JTAG interface consists of a set of five signals, three JTAG registers (see Section 24.3, “JTAG Registers and Scan Chains,”) and a test access port (TAP) controller, described in the following sections. A block diagram of the JTAG interface is shown in Figure 24-1.

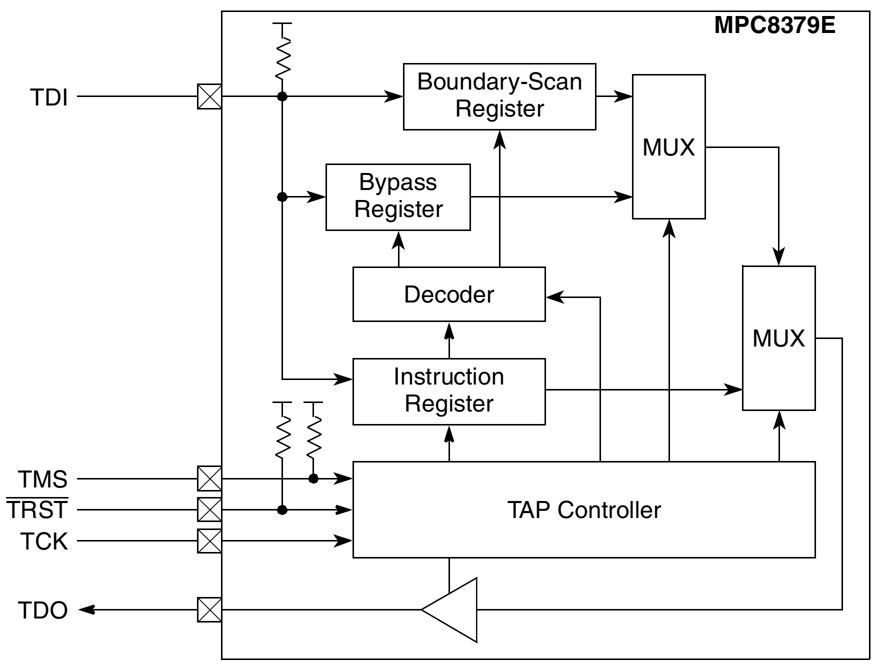


Figure 24-1. JTAG Interface Block Diagram

24.2 JTAG Signals

The device provides the following five dedicated JTAG signals:

- Test data input (TDI)
- Test data output (TDO)
- Test mode select (TMS)

- Test reset ($\overline{\text{TRST}}$)
- Test clock (TCK)

The TDI and TDO signals input and output all instructions and data to the JTAG scan registers. JTAG operations are controlled by the TAP controller through the TMS and TCK signals. Boundary-scan data is latched by the TAP controller on the rising edge of the TCK signal. The $\overline{\text{TRST}}$ signal is specified as optional by the IEEE 1149.1 specification, and is used to reset the TAP controller asynchronously. The assertion of the $\overline{\text{TRST}}$ signal at power-on reset ensures that the JTAG logic does not interfere with the normal operation of the device.

24.2.1 External Signal Descriptions

The JTAG signals are summarized in Table 24-1.

Table 24-1. JTAG Test Signals Summary

Name	Description	Functional Block	Function	Reset Value	I/O
TCK	Test clock	Debug	Clock for JTAG testing.	—	I
TDI	Test data input		Serial input for instructions and data to the JTAG test subsystem. Internally pulled up.	—	I
TDO	Test data output		Serial data output for the JTAG test subsystem. High impedance except when scanning out data.	High impedance	O
TMS	Test mode select		Carries commands to the TAP controller for boundary scan operations. Internally pulled up.	—	I
$\overline{\text{TRST}}$	Test reset		Resets the TAP controller asynchronously. Internally pulled up.	—	I

Table 24-2 shows detailed descriptions of the JTAG test signals.

Table 24-2. JTAG Test—Detailed Signal Descriptions

Signal	I/O	Description	
TCK	I	JTAG test clock.	
		State Meaning	Asserted/Negated—Should be driven by a free-running clock signal with a 30–70% duty cycle. Input signals to the TAP are clocked in on the rising edge. Changes to the TAP output signals occur on the falling edge. The test logic allows TCK to be stopped.
		Timing	See IEEE 1149.1 specification for more details.
TDI	I	JTAG test data input.	
		State Meaning	Asserted/Negated—The value present on the rising edge of TCK is clocked into the selected JTAG test instruction or data register. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor.
		Timing	See IEEE 1149.1 specification for more details.

Table 24-2. JTAG Test—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
TDO	O	JTAG test data output.	
		State Meaning	Asserted/Negated—The contents of the selected internal instruction or data register are shifted out on this signal on the falling edge of TCK. Remains in a high-impedance state except when scanning data.
		Timing	See IEEE 1149.1 specification for more details.
TMS	I	JTAG test mode select.	
		State Meaning	Asserted/Negated—Decoded by the internal JTAG TAP controller to distinguish the primary operation of the test support circuitry. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor.
		Timing	See IEEE 1149.1 specification for more details.
$\overline{\text{TRST}}$	I	JTAG test reset.	
		State Meaning	Asserted—Causes asynchronous initialization of the internal JTAG TAP controller. Must be asserted during power-on reset in order to properly initialize the JTAG TAP and for normal operation of the device. An unterminated input appears as a high signal level to the test logic due to an internal pull-up resistor. Negated— Normal operation.
		Timing	See IEEE 1149.1 specification for more details.

24.3 JTAG Registers and Scan Chains

The bypass, boundary-scan, and instruction JTAG registers along with their associated scan chains are mandatory for compliance with the IEEE 1149.1 specification.

- Bypass register. The bypass register is a single-stage register used to bypass the boundary-scan latches of the device during board-level boundary-scan operations involving components other than the device. The use of the bypass register reduces the total scan string size of the boundary-scan test.
- Boundary-scan registers. The JTAG interface provides a chain of registers dedicated to boundary-scan operations. To be JTAG-compliant, these registers cannot be shared with any functional registers of the device. The boundary-scan register chain includes registers controlling the direction of the input/output drivers, in addition to the registers reflecting the signal value received or driven.

  The boundary-scan registers capture the input or output state of the device's signals during a Capture_DR TAP controller state. When a data scan is initiated following the Capture_DR state, the sampled values are shifted out through the TDO output while new boundary-scan register values are shifted in through the TDI input. At the end of the data scan operation, the boundary-scan registers are updated with the new values during an update_DR TAP controller state.
- Instruction register. The 8-bit JTAG instruction register serves as an instruction and status register. As TAP controller instructions are scanned in through the TDI input, the TAP controller status bits are scanned out through the TDO output.

- TAP controller. The device provides a standard JTAG TAP controller that controls instruction and data scan operations. The TMS signal controls the state transitions of the TAP controller.

Chapter 25
General Purpose I/O (GPIO)

25.1 Introduction

This chapter describes the general-purpose I/O module, including pin descriptions, register settings, and interrupt capabilities. The device includes two identical GPIO modules. Figure 25-1 shows the block diagram of one GPIO module.

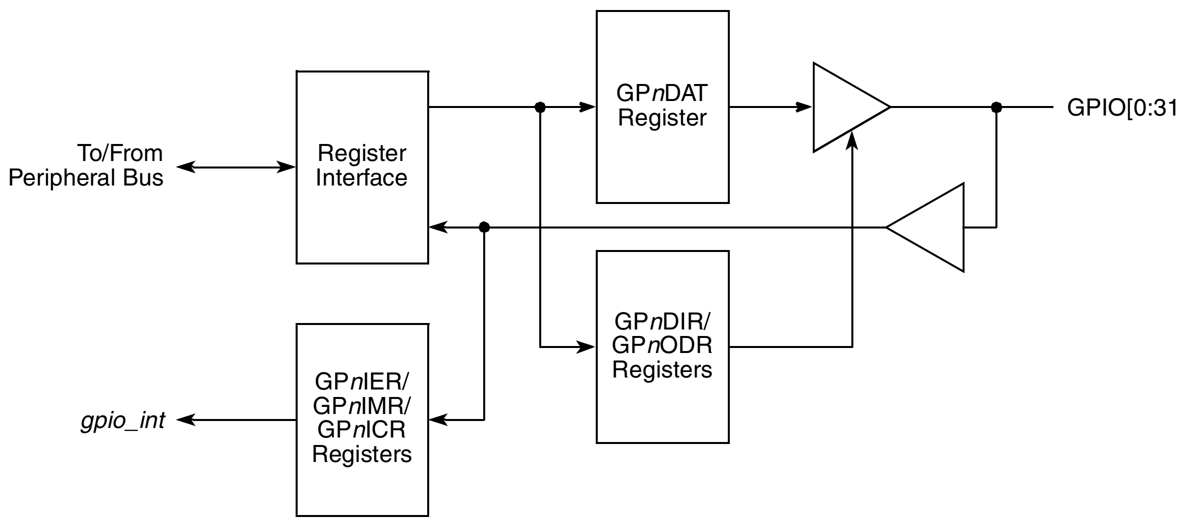


Figure 25-1. GPIO*n* Module Block Diagram

25.1.1 Overview

Each GPIO module supports 32 general-purpose I/O ports. In 8379 only 26 (0 to 25) I/O ports are pinned out for each module. Each port can be configured as an input or as an output. If a port is configured as an input, it can optionally generate an interrupt on detection of a change. If a port is configured as an output, it can be individually configured as an open-drain or a fully active output.

25.1.2 Features

The GPIO unit implements the following features:

- 32 input/output ports
- Some ports have dedicated processor signals. Others are multiplexed together with other functional signals. See Chapter 3, "Signal Descriptions."

- All signals are configured as inputs when the device comes out of reset and also when $\overline{HRESET}$ is asserted.
- Open-drain capability on all ports
- All ports can optionally generate an interrupt upon changing their state.

25.2 External Signal Description

The following section provides information about GPIO signals.

25.2.1 Signals Overview

Table 25-1 provides detailed descriptions of the external GPIO signals.

Table 25-1. IPIC External Signals—Detailed Signal Descriptions

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">GPIO1[0:31]
GPIO2[0:31]</td><td rowspan="3">I/O</td><td colspan="2">General purpose I/O. Each signal can be set individually to act as input or output, according to application needs.</td></tr>
<tr><td>State Meaning</td><td>Asserted/Negated—Defined per application.</td></tr>
<tr><td>Timing</td><td>Assertion/Negation—Inputs can be asserted completely asynchronously.
Outputs are asynchronous to any externally visible clock</td></tr>
</table>

25.3 Memory Map/Register Definition

Each GPIO has programmable registers that occupy 24 bytes of memory-mapped space. Note that reading undefined portions of the memory map returns all zeros and writing has no effect.

All GPIO registers are 32 bits wide and are located on 32-bit address boundaries. All addresses used in this chapter are offsets from the address held in IMMRBAR as defined in Chapter 2, “Memory Map.”

Table 25-2 shows the memory map of GPIO.

Table 25-2. GPIO Register Address Map

Offset	Register	Access	Reset Value	Section/Page
0xC00	GPIO1 direction register (GP1DIR)	R/W	0x0000_0000	25.3.1/25-3
0xC04	GPIO1 open drain register (GP1ODR)	R/W	0x0000_0000	25.3.2/25-3
0xC08	GPIO1 data register (GP1DAT)	R/W	0x0000_0000	25.3.3/25-4
0xC0C	GPIO1 interrupt event register (GP1IER)	w1c	Undefined	25.3.4/25-4
0xC10	GPIO1 interrupt mask register (GP1IMR)	R/W	0x0000_0000	25.3.5/25-5
0xC14	GPIO1 external interrupt control register (GP1ICR)	R/W	0x0000_0000	25.3.6/25-5
0xC1C–0xCFF	Reserved	—	—	—
0xD00	GPIO2 direction register (GP2DIR)	R/W	0x0000_0000	25.3.1/25-3
0xD04	GPIO2 open drain register (GP2ODR)	R/W	0x0000_0000	25.3.2/25-3

Table 25-2. GPIO Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/Page
0xD08	GPIO2 data register (GP2DAT)	R/W	0x0000_0000	25.3.3/25-4
0xD0C	GPIO2 interrupt event register (GP2IER)	w1c	Undefined	25.3.4/25-4
0xD10	GPIO2 interrupt mask register (GP2IMR)	R/W	0x0000_0000	25.3.5/25-5
0xD14	GPIO2 external interrupt control register (GP2ICR)	R/W	0x0000_0000	25.3.6/25-5
0xD1C–0xDFF	Reserved	—	—	—

25.3.1 GPIO*n* Direction Register (GP1DIR–GP2DIR)

The GPIO*n* direction registers (GP1DIR–GP2DIR), shown in Figure 25-2, defines the direction of the individual ports.

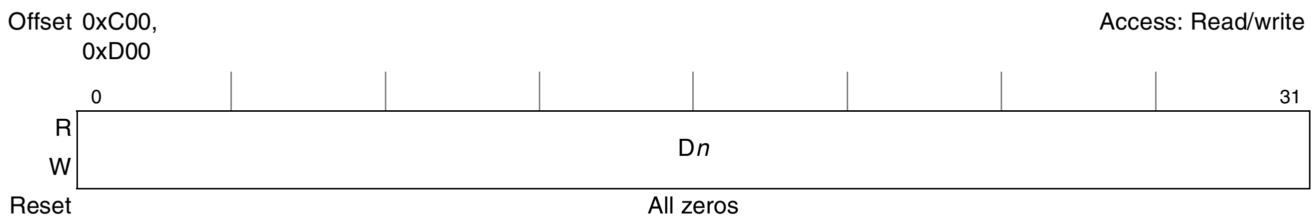


Figure 25-2. GPIO*n* Direction Register (GP*n*DIR)

Table 25-3 defines the bit fields of GP*n*DIR.

Table 25-3. GP*n*DIR Bit Settings

Bits	Name	Description
0–31	D*n*	Direction. Indicates whether a signal is used as an input or an output. 0 The corresponding signal is an input. 1 The corresponding signal is an output.

25.3.2 GPIO*n* Open Drain Register (GP1ODR–GP2ODR)

The GPIO*n* open drain register (GP1ODR–GP2ODR), shown in Figure 25-3, defines the way individual ports drive their output.

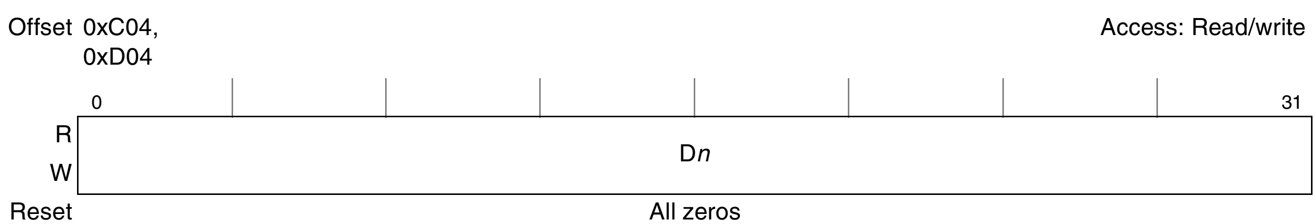


Figure 25-3. GPIO*n* Open Drain Register (GP1ODR–GP2ODR)

Table 25-4 defines the bit fields of GP*n*ODR.

Table 25-4. GP*n*ODR Bit Settings

Bits	Name	Description
0–31	D*n*	Open-drain configuration. Indicates whether a signal is actively driven as an output or an open-drain driver. This register has no effect on signals programmed as inputs in the corresponding GP*n*DIR. 0 The I/O signal is actively driven as an output. 1 The I/O signal is an open-drain driver. As an output, the signal is driven active-low, otherwise it is three-stated.

25.3.3 GPIO*n* Data Register (GP1DAT–GP2DAT)

The GPIO*n* data register (GP1DAT–GP2DAT), shown in Figure 25-4, carries the data in/out for the individual ports.

Offset 0xC08, 0xD08 — Access: Read/write

	0–31
R	D*n*
W	
Reset	All zeros

Figure 25-4. GPIO*n* Data Register (GP1DAT–GP2DAT)

Table 25-5 defines the bit fields of GP*n*DAT.

Table 25-5. GP*n*DAT Bit Settings

Bits	Name	Description
0–31	D*n*	Data. Write data is latched and presented on external signals if GP*n*DIR has configured the port as an output. Read operation always returns the data at the signal.

25.3.4 GPIO*n* Interrupt Event Register (GP1IER–GP2IER)

The GPIO*n* interrupt event register (GP1IER–GP2IER), shown in Figure 25-5, carries information of the events that caused an interrupt. Each bit in GP*n*IER, corresponds to an interrupt source. GP*n*IER bits are cleared by writing ones. However, writing zero has no effect.

Offset 0xC0C, 0xD0C — Access: w1c

	0–31
R	D*n*
W	
Reset	Undefined (the user should write 1s to clear before using)

Figure 25-5. GPIO*n* Interrupt Event Register (GP1IER–GP2IER)

Table 25-6 defines the bit fields of GP*n*IER.

Table 25-6. GP*n*IER Bit Settings

Bits	Name	Description
0–31	D*n*	Interrupt events. Indicates whether an interrupt event occurred on the corresponding GPIO signal. 0 No interrupt event occurred on the corresponding GPIO signal. 1 Interrupt event occurred on the corresponding GPIO signal.

25.3.5 GPIO*n* Interrupt Mask Register (GP1IMR–GP2IMR)

The GPIO*n* interrupt mask register (GP1IMR–GP2IMR), shown in Figure 25-6, defines the interrupt masking for the individual ports. When a masked interrupt request occurs, the corresponding GP*n*IER bit is set, regardless of the GP*n*IMR state. When one or more non-masked interrupt events occur, the GPIO module issues an interrupt to the on-chip interrupt controller.

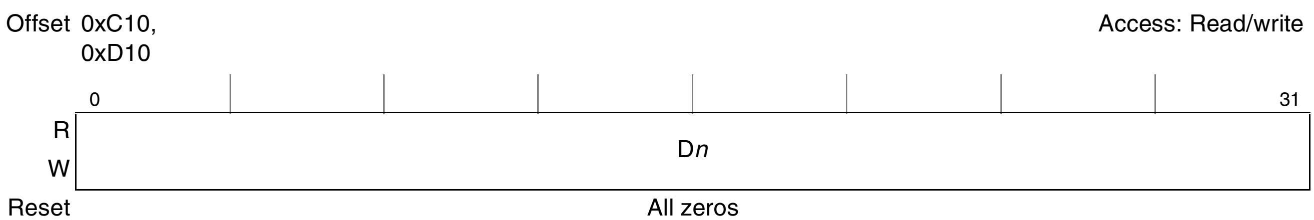


Figure 25-6. GPIO*n* Interrupt Mask Register (GP1IMR–GP2IMR)

Table 25-7 defines the bit fields of GP*n*IMR.

Table 25-7. GP*n*IMR Bit Settings

Bits	Name	Description
0–31	D*n*	Interrupt mask. Indicates whether an interrupt event is masked or not masked. 0 The input interrupt signal is masked (disabled). 1 The input interrupt signal is not masked (enabled).

25.3.6 GPIO*n* Interrupt Control Register (GP1ICR–GP2ICR)

The GPIO*n* interrupt control register (GP1ICR–GP2ICR), shown in Figure 25-7, determines whether the corresponding port line asserts an interrupt request on either a high-to-low change or any change on the state of the signal.

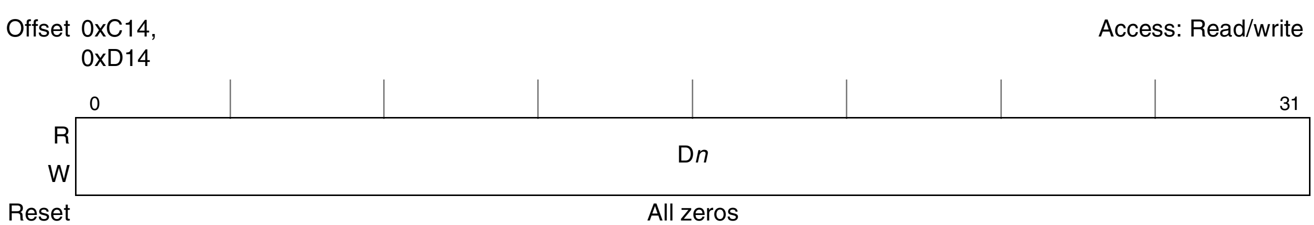


Figure 25-7. GPIO*n* Interrupt Control Register (GP1ICR–GP2ICR)

Table 25-8 defines the bit fields of GP*n*ICR.

Table 25-8. GP*n*ICR Bit Settings

Bits	Name	Description
0–31	D*n*	Edge detection mode. The corresponding port line asserts an interrupt request according to the following: 0 Any change on the state of the port generates an interrupt request. 1 High-to-low change on the port generates an interrupt request.

Appendix A
Complete List of Configuration, Control, and Status Registers

A.1 Local Access Windows

Table A-1. Local Access Window Registers

Local Access Window—Block Base Address 0x0_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	Internal memory map base address register (IMMRBAR)	R/W	0xFF40_0000[1]	5.2.4.1/5-6
0x004	Reserved	—	—	—
0x008	Alternate configuration base address register (ALTCBAR)	R/W	0x0000_0000	5.2.4.2/5-8
0x00C–0x01C	Reserved	—	—	—
0x020	eLBC local access window 0 base address register (LBLAWBAR0)	R/W	0x0000_0000[2]	5.2.4.3/5-9
0x024	eLBC local access window 0 attribute register (LBLAWAR0)	R/W	0x0000_0000[3]	5.2.4.4/5-10
0x028	eLBC local access window 1 base address register (LBLAWBAR1)	R/W	0x0000_0000	5.2.4.3/5-9
0x02C	eLBC local access window 1 attribute register (LBLAWAR1)	R/W	0x0000_0000	5.2.4.4/5-10
0x030	eLBC local access window 2 base address register (LBLAWBAR2)	R/W	0x0000_0000	5.2.4.3/5-9
0x034	eLBC local access window 2 attribute register (LBLAWAR2)	R/W	0x0000_0000	5.2.4.4/5-10
0x038	eLBC local access window 3 base address register (LBLAWBAR3)	R/W	0x0000_0000	5.2.4.3/5-9
0x03C	eLBC local access window 3 attribute register (LBLAWAR3)	R/W	0x0000_0000	5.2.4.4/5-10
0x040–0x05C	Reserved	—	—	—
0x060	PCI local access window 0 base address register (PCILAWBAR0)	R/W	0x0000_0000[4]	5.2.4.5/5-11
0x064	PCI local access window 0 attribute register (PCILAWAR0)	R/W	0x0000_0000[5]	5.2.4.6/5-12
0x068	PCI local access window 1 base address register (PCILAWBAR1)	R/W	0x0000_0000[6]	5.2.4.5/5-11
0x06C	PCI local access window 1 attribute register (PCILAWAR1)	R/W	0x0000_0000	5.2.4.6/5-12

Table A-1. Local Access Window Registers (continued)

Local Access Window—Block Base Address 0x0_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x070–0x07C	Reserved	—	—	—
0x080	PCI Express 1 local access window base address register (PCIEXP1LAWBAR)	R/W	0x0000_0000	5.2.4.7/5-13
0x084	PCI Express 1 local access window attribute register (PCIEXP1LAWAR)	R/W	0x0000_0000	5.2.4.8/5-13
0x088	PCI Express 2 local access window base address register (PCIEXP2LAWBAR)	R/W	0x0000_0000	5.2.4.9/5-14
0x08C	PCI Express 2 local access window attribute register (PCIEXP2LAWAR)	R/W	0x0000_0000	5.2.4.10/5-15
0x090–0x09C	Reserved	—	—	—
0x0A0	DDR local access window 0 base address register (DDRLAWBAR0)	R/W	0x0000_0000 [7]	5.2.4.11/5-15
0x0A4	DDR local access window 0 attribute register (DDRLAWAR0)	R/W	0x0000_0000 [8]	5.2.4.12/5-16
0x0A8	DDR local access window 1 base address register (DDRLAWBAR1)	R/W	0x0000_0000	5.2.4.11/5-15
0x0AC	DDR local access window 1 attribute register (DDRLAWAR1)	R/W	0x0000_0000	5.2.4.12/5-16
0x0B0–0x0FC	Reserved	—	—	—

1 Depends on reset configuration word high values. See Section 5.2.4.1.1, "Updating IMMRBAR," for details.

2 Depends on reset configuration word high values. See Section 5.2.4.3.1, "LBLAWBAR0[BASE_ADDR] Reset Value," for details.

3 Depends on reset configuration word high values. See Section 5.2.4.4.1, "LBLAWAR0[EN] and LBLAWAR0[SIZE] Reset Value," for details.

4 Depends on reset configuration word high values. See Section 5.2.4.5.1, "PCILAWBAR0[BASE_ADDR] Reset Value," for details.

5 Depends on reset configuration word high values. See Section 5.2.4.11.1, "DDRLAWBAR0[BASE_ADDR] Reset Value," for details.

6 Depends on reset configuration word high values. See Section 5.2.4.6.1, "PCILAWAR0[EN] and PCILAWAR0[SIZE] Reset Value," for details.

7 Depends on reset configuration word high values. See Section 5.2.4.11.1, "DDRLAWBAR0[BASE_ADDR] Reset Value," for details.

8 Depends on reset configuration word high values. See Section 5.2.4.12.1, "DDRLAWAR0[EN] and DDRLAWAR0[SIZE] Reset Value," for details.

A.2 System Configuration Registers

Table A-2. System Configuration Registers

System Configuration—Block Base Address 0x0_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x100	System general purpose register low (SGPRL)	R/W	0x480E_FF20	5.3.2.1/5-20
0x104	System general purpose register high (SGPRH)	R/W	0x0000_0000	5.3.2.2/5-21
0x108	System part and revision ID register (SPRIDR)	R	0x80C0_0010	5.3.2.3/5-21
0x10C	Reserved	—	—	—
0x110	System priority configuration register (SPCR)	R/W	0x0000_0000	5.3.2.4/5-22
0x114	System I/O configuration register low (SICRL)	R/W	0x0000_0000[1]	5.3.2.5/5-24
0x118	System I/O configuration register high (SICRH)	R/W	0x1FFC_0000	5.3.2.6/5-28
0x11C–0x124	Reserved	—	—	—
0x128	DDR control driver register (DDRCDR)	R/W	0x7304_0001	5.3.2.8/5-31
0x12C	DDR debug status register (DDRDSR)	R	0x3300_0000	5.3.2.9/5-33
0x130	Output buffer impedance register (OBIR)	R/W	0x3111_0000	5.3.2.10/55-34
0x134–0x13C	Reserved	—	—	—
0x140	PCI Express control register 1 (PECR1)	R/W	0x0000_0000	5.3.2.11/55-34
0x144	PCI Express control register 2 (PECR2)	R/W	0x0000_0000	5.3.2.11/55-34
0x148–0x1FC	Reserved	—	—	—

[1] Depends on the reset configuration word high configuration values.

A.3 Watchdog Timer (WDT)

Table A-3. Watchdog Timer (WDT) Registers

Watchdog Timer (WDT)—Block Base Address 0x0_0200				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0–0x3	Reserved	—	—	—
0x4	System watchdog control register (SWCRR)	R/W	0xFFFF_0003 or 0xFFFF_0007[1]	5.4.4.1/5-38
0x8	System watchdog count register (SWCNR)	R	0x0000_FFFF	5.4.4.2/5-39
0xC–0xD	Reserved	—	—	—
0xE	System watchdog service register (SWSRR)	R/W	0x0000	5.4.4.3/5-39

[1] SWCRR[SWEN] reset value directly depends on RCWHR[SWEN] (reset configuration word high).

A.4 Real Time Clock (RTC)

Table A-4. Real Time Clock (RTC) Registers

Real Time Clock (RTC)—Block Base Address 0x0_0300				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	Real time counter control register (RTCNR)	R/W	0x0000_0000	5.5.5.1/5-45
0x04	Real time counter load register (RTLDR)	R/W	0x0000_0000	5.5.5.2/5-46
0x08	Real time counter prescale register (RTPSR)	R/W	0x0000_0000	5.5.5.3/5-46
0x0C	Real time counter register (RTCTR)	R	0x0000_0000	5.5.5.4/5-47
0x10	Real time counter event register (RTEVR)	w1c	0x0000_0000	5.5.5.5/5-47
0x14	Real time counter alarm register (RTALR)	R/W	0xFFFF_FFFF	5.5.5.6/5-48
0x18–0x1F	Reserved	—	—	

A.5 Periodic Interval Timer (PIT)

Table A-5. Periodic Interval Timer (PIT) Registers

Periodic Interval Timer (PIT)—Block Base Address 0x0_0400				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	Periodic interval timer control register (PTCNR)	R/W	0x0000_0000	5.6.5.1/5-52
0x04	Periodic interval timer load register (PTLDR)	R/W	0x0000_0000	5.6.5.2/5-53
0x08	Periodic interval timer prescale register (PTPSR)	R/W	0x0000_0000	5.6.5.3/5-54
0x0C	Periodic interval timer counter register (PTCTR)	R	0x0000_0000	5.6.5.4/5-54
0x10	Periodic interval timer event register (PTEVR)	w1c	0x0000_0000	5.6.5.5/5-55
0x14–0x1F	Reserved	—	—	

A.6 General Purpose (Global) Timers (GTMs)

Table A-6. General Purpose (Global) Timers (GTMs) Registers

General Purpose (Global) Timer Module 1—Block Base Address 0x0_0500 General Purpose (Global) Timer Module 2—Block Base Address 0x0_0600				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x00	5.7.5.1/5-62
0x01–0x03	Reserved	—	—	—
0x04	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x00	5.7.5.1/5-62
0x05–0x0F	Reserved	—	—	—
0x10	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	5.7.5.2/5-66
0x12	Timer 2 global timers mode register (GTMDR2)			

Table A-6. General Purpose (Global) Timers (GTMs) Registers (continued)

General Purpose (Global) Timer Module 1—Block Base Address 0x0_0500 General Purpose (Global) Timer Module 2—Block Base Address 0x0_0600				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x14	Timer 1 global timers reference register (GTRFR1)	R/W	0xFFFF	5.7.5.3/5-67
0x16	Timer 2 global timers reference register (GTRFR2)			
0x18	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	5.7.5.4/5-67
0x1A	Timer 2 global timers capture register (GTCPR2)			
0x1C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	5.7.5.5/5-68
0x1E	Timer 2 global timers counter register (GTCNR2)			
0x20	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	5.7.5.2/5-66
0x22	Timer 4 global timers mode register (GTMDR4)			
0x24	Timer 3 global timers reference register (GTRFR3)	R/W	0xFFFF	5.7.5.3/5-67
0x26	Timer 4 global timers reference register (GTRFR4)			
0x28	Timer 3 global timers capture register (GTCPR3)	R	0x0000	5.7.5.4/5-67
0x2A	Timer 4 global timers capture register (GTCPR4)			
0x2C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	5.7.5.5/5-68
0x2E	Timer 4 global timers counter register (GTCNR4)			
0x30	Timer 1 global timers event register (GTEVR1)	w1c	0x0000	5.7.5.6/5-68
0x32	Timer 2 global timers event register (GTEVR2)			
0x34	Timer 3 global timers event register (GTEVR3)			
0x36	Timer 4 global timers event register (GTEVR4)			
0x38	Timer 1 global timers prescale register (GTPSR1)	R/W	0x0003	5.7.5.7/5-69
0x3A	Timer 2 global timers prescale register (GTPSR2)			
0x3C	Timer 3 global timers prescale register (GTPSR3)			
0x3E	Timer 4 global timers prescale register (GTPSR4)			

A.7 Integrated Programmable Interrupt Controller (IPIC)

Table A-7. Integrated Programmable Interrupt Controller (IPIC) Registers

Integrated Programmable Interrupt Controller—Block Base Address 0x0_0700				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	System global interrupt configuration register (SICFR)	R/W	0x0000_0000	8.5.1/8-7
0x04	System regular interrupt vector register (SIVCR)	R	0x0000_0000	8.5.2/8-9
0x08	System internal interrupt pending register (SIPNR_H)	R	0x0000_0000	8.5.3/8-11

Table A-7. Integrated Programmable Interrupt Controller (IPIC) Registers (continued)

Integrated Programmable Interrupt Controller—Block Base Address 0x0_0700				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0C	System internal interrupt pending register (SIPNR_L)	R	0x0000_0000	8.5.3/8-11
0x10	System internal interrupt group A priority register (SIPRR_A)	R/W	0x0530_9770	8.5.4/8-14
0x14	System internal interrupt group B priority register (SIPRR_B)	R/W	0x0530_9770	8.5.5/8-15
0x18	System internal interrupt group C priority register (SIPRR_C)	R/W	0x0530_9770	8.5.6/8-16
0x1C	System internal interrupt group D priority register (SIPRR_D)	R/W	0x0530_9770	8.5.7/8-17
0x20	System internal interrupt mask register (SIMSR_H)	R/W	0x0000_0000	8.5.8/8-18
0x24	System internal interrupt mask register (SIMSR_L)	R/W	0x0000_0000	8.5.8/8-18
0x28	System internal interrupt control register (SICNR)	R/W	0x0000_0000	8.5.9/8-19
0x2C	System external interrupt pending register (SEPNR)	R/W	Special	8.5.10/8-21
0x30	System mixed interrupt group A priority register (SMPRR_A)	R/W	0x0530_9770	8.5.11/8-21
0x34	System mixed interrupt group B priority register (SMPRR_B)	R/W	0x0530_9770	8.5.12/8-22
0x38	System external interrupt mask register (SEMSR)	R/W	0x0000_0000	8.5.13/8-23
0x3C	System external interrupt control register (SECNR)	R/W	0x0000_0000	8.5.14/8-24
0x40	System error status register (SERSR)	R/W	0x0000_0000	8.5.15/8-25
0x44	System error mask register (SERMR)	R/W	0xFFFE0000	8.5.16/8-26
0x48	System error control register (SERCR)	R/W	0x0000_0000	8.5.17/8-27
0x4C	System external interrupt polarity control register (SEPCR)	R/W	0x0000_0000	8.5.18/8-28
0x4F	Reserved	—	—	—
0x50	System internal interrupt force register (SIFCR_H)	R/W	0x0000_0000	8.5.19/8-29
0x54	System internal interrupt force register (SIFCR_L)	R/W	0x0000_0000	8.5.19/8-29
0x58	System external interrupt force register (SEFCR)	R/W	0x0000_0000	8.5.20/8-30
0x5C	System error force register (SERFR)	R/W	0x0000_0000	8.5.21/8-30
0x60	System critical interrupt vector register (SCVCR)	R	0x0000_0000	8.5.22/8-31
0x64	System management interrupt vector register (SMVCR)	R	0x0000_0000	8.5.23/8-31
0x68–0xBF	Reserved	—	—	—

A.8 System Arbiter

Table A-8. System Arbiter Registers

System Arbiter—Block Base Address 0x0_0800				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	Arbiter configuration register (ACR)	R/W	0x0000_0000/ 0x0010_0000[1]	6.2.1/6-2
0x04	Arbiter timers register (ATR)	R/W	0xFFFF_FFFF	6.2.2/6-4
0x0C	Arbiter event register (AER)	w1c	0x0000_0000	6.2.3/6-5
0x10	Arbiter interrupt definition register (AIDR)	R/W	0x0000_0000	6.2.4/6-6
0x14	Arbiter mask register (AMR)	R/W	0x0000_0000	6.2.5/6-7
0x18	Arbiter event attributes register (AEATR)	R	0x0000_0000[2]	6.2.6/6-8
0x1C	Arbiter event address register (AEADR)	R	0x0000_0000[2]	6.2.7/6-9
0x20	Arbiter event response register (AERR)	R/W	0x0000_0000	6.2.8/6-10

1 Reset value is determined from the core PLL configuration of the reset configuration word. See Chapter 4, "Reset, Clocking, and Initialization," for details.

2 The registers AEATR and AEADR are affected only by the assertion of $\overline{\text{PORESET}}$

A.9 Reset Configuration

Table A-9. Reset Configuration Registers

Reset Configuration—Block Base Address 0x0_0900				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	Reset configuration word low register (RCWLR)	R	0x0000_0000	4.5.1.1/4-35
0x04	Reset configuration word high register (RCWHR)	R	0x0000_0000	4.5.1.2/4-35
0x08	Reserved, should be cleared	—	—	—
0x0C	Reserved, should be cleared	—	—	—
0x10	Reset status register (RSR)	R/W	0x0000_0000	4.5.1.3/4-35
0x14	Reset mode register (RMR)	R/W	0x0000_0000	4.5.1.4/4-37
0x18	Reset protection register (RPR)	R/W	0x0000_0000	4.5.1.5/4-37
0x1C	Reset control register (RCR)	R/W	0x0000_0000	4.5.1.6/4-38
0x20	Reset control enable register (RCER)	R/W	0x0000_0000	4.5.1.7/4-39
0x24–0xFC	Reserved, should be cleared.	—	—	—

A.10 Clock Configuration

Table A-10. Clock Configuration Registers

Clock Configuration—Block Base Address 0x0_0A00				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	System PLL mode register (SPMR)	R	0x*nnnn_nnnn*	4.5.2.1/4-39
0x04	Output clock control register (OCCR)	R/W	0x0000_FFF8	4.5.2.2/4-41
0x08	System clock control register (SCCR)	R/W	0xFFFF_FFFF	4.5.2.3/4-42
0x0C–0xFC	Reserved, should be cleared	—	—	—

A.11 Power Management Controller (PMC)

Table A-11. Power Management Controller (PMC) Registers

Power Management Controller—Block Base Address 0x0_0B00				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	Power management controller configuration register (PMCCR)	R/W	0x0000_0000	5.8.2.1/5-74
0x04	Power management controller event register (PMCER)	R/W	0x0000_0000	5.8.2.2/5-75
0x08	Power management controller mask register (PMCMR)	R/W	0x0000_0000	5.8.2.3/5-77
0x0C	Power management controller configuration register 1 (PMCCR1)	R/W	0x0000_0000	5.8.2.4/5-78
0x10–0xFC	Reserved	—	—	—

A.12 General Purpose I/O (GPIO)

Table A-12. General Purpose I/O (GPIO) Registers

General Purpose I/O (GPIO) 1—Block Base Address 0x0_0C00 General Purpose I/O (GPIO) 2—Block Base Address 0x0_0D00				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	GPIO1 direction register (GP1DIR)	R/W	0x0000_0000	25.3.1/25-3
0x04	GPIO1 open drain register (GP1ODR)	R/W	0x0000_0000	25.3.2/25-3
0x08	GPIO1 data register (GP1DAT)	R/W	0x0000_0000	25.3.3/25-4
0x0C	GPIO1 interrupt event register (GP1IER)	w1c	Undefined	25.3.4/25-4
0x10	GPIO1 interrupt mask register (GP1IMR)	R/W	0x0000_0000	25.3.5/25-5
0x14	GPIO1 external interrupt control register (GP1ICR)	R/W	0x0000_0000	25.3.6/25-5
0x1C–0xFF	Reserved	—	—	—

A.13 DDR Memory Controller

Table A-13. DDR Memory Controller Registers

DDR Memory Controller—Block Base Address 0x0_2000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	CS0_BNDS—Chip select memory bounds	R/W	0x0000_0000	9.4.1.1/9-10
0x080	CS0_CONFIG—Chip select configuration	R/W	0x0000_0000	9.4.1.2/9-10
0x100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	0x0000_0000	9.4.1.3/9-12
0x104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	9.4.1.4/9-13
0x108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	0x0000_0000	9.4.1.5/9-15
0x10C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	0x0000_0000	9.4.1.6/9-17
0x110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	9.4.1.7/9-19
0x114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	0x0000_0000	9.4.1.8/9-22
0x118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	0x0000_0000	9.4.1.9/9-24
0x11C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	0x0000_0000	9.4.1.10/9-25
0x120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	0x0000_0000	9.4.1.11/9-25
0x124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	0x0000_0000	9.4.1.12/9-28
0x128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	0x0000_0000	9.4.1.13/9-28
0x130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	9.4.1.14/9-29
0x140–0x144	Reserved	—	—	—
0x148	DDR_INIT_ADDR—DDR training initialization address	R/W	0x0000_0000	9.4.1.15/9-29
0x150–0xBF4	Reserved	—	—	—
0xBF8	DDR_IP_REV1—DDR IP block revision 1	R	0x*nnnn_nnnn*[1]	9.4.1.16/9-30
0xBFC	DDR_IP_REV2—DDR IP block revision 2	R	0x00*nn*_00*nn*[1]	9.4.1.17/9-30
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	0x0000_0000	9.4.1.18/9-31
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	0x0000_0000	9.4.1.19/9-31
0xE08	ERR_INJECT—Memory data path error injection mask ECC	R/W	0x0000_0000	9.4.1.20/9-32
0xE20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	0x0000_0000	9.4.1.21/9-32
0xE24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	0x0000_0000	9.4.1.22/9-33
0xE28	CAPTURE_ECC—Memory data path read capture ECC	R/W	0x0000_0000	9.4.1.23/9-33
0xE40	ERR_DETECT—Memory error detect	w1c	0x0000_0000	9.4.1.24/9-33
0xE44	ERR_DISABLE—Memory error disable	R/W	0x0000_0000	9.4.1.25/9-34

Table A-13. DDR Memory Controller Registers (continued)

DDR Memory Controller—Block Base Address 0x0_2000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xE48	ERR_INT_EN—Memory error interrupt enable	R/W	0x0000_0000	9.4.1.26/9-35
0xE4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	0x0000_0000	9.4.1.27/9-36
0xE50	CAPTURE_ADDRESS—Memory error address capture	R/W	0x0000_0000	9.4.1.28/9-37
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	0x0000_0000	9.4.1.29/9-37

1 Implementation-dependent reset values are listed in specified section/page.

A.14 I^{2}C Controller

Table A-14. I^{2}C Controller Registers

I^{2}C Controller—Block Base Address 0x0_3000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	I2C1ADR—I^{2}C1 address register	R/W	0x00	21.3.1.1/21-5
0x004	I2C1FDR—I^{2}C1 frequency divider register	R/W	0x00	21.3.1.2/21-6
0x008	I2C1CR—I^{2}C1 control register	R/W	0x00	21.3.1.3/21-7
0x00C	I2C1SR—I^{2}C1 status register	R/W	0x81	21.3.1.4/21-8
0x010	I2C1DR—I^{2}C1 data register	R/W	0x00	21.3.1.5/21-9
0x014	I2C1DFSRR—I^{2}C1 digital filter sampling rate register	R/W	0x10	21.3.1.6/21-9
0x01C–0x0FF	Reserved, should be cleared	—	—	—

A.15 DUART

Table A-15. DUART Registers

UART 1—Block Base Address 0x0_4500 UART 2—Block Base Address 0x0_4600				
Address	**Register**	**Access**	**Reset**	**Section/Page**
0x0_4500	URBR—ULCR[DLAB] = 0 UART1 receiver buffer register	R	0x00	22.3.1.1/22-5
	UTHR—ULCR[DLAB] = 0 UART1 transmitter holding register	W	0x00	22.3.1.2/22-6
	UDLB—ULCR[DLAB] = 1 UART1 divisor least significant byte register	R/W	0x00	22.3.1.3/22-6
0x0_4501	UIER—ULCR[DLAB] = 0 UART1 interrupt enable register	R/W	0x00	22.3.1.4/22-8
	UDMB—ULCR[DLAB] = 1 UART1 divisor most significant byte register	R/W	0x00	22.3.1.3/22-6

Table A-15. DUART Registers (continued)

UART 1—Block Base Address 0x0_4500 UART 2—Block Base Address 0x0_4600				
Address	**Register**	**Access**	**Reset**	**Section/Page**
0x02	UIIR—ULCR[DLAB] = 0 UART1 interrupt ID register	R	0x01	22.3.1.5/22-9
	UFCR—ULCR[DLAB] = 0 UART1 FIFO control register	W	0x00	22.3.1.6/22-10
	UAFR—ULCR[DLAB] = 1 UART1 alternate function register	R/W	0x00	22.3.1.12/22-16
0x03	ULCR—ULCR[DLAB] = x UART1 line control register	R/W	0x00	22.3.1.7/22-11
0x04	UMCR—ULCR[DLAB] = x UART1 MODEM control register	R/W	0x00	22.3.1.8/22-13
0x05	ULSR—ULCR[DLAB] = x UART1 line status register	R	0x60	22.3.1.9/22-14
0x06	UMSR—ULCR[DLAB] = x UART1 MODEM status register	R	0x00	22.3.1.10/22-15
0x07	USCR—ULCR[DLAB] = x UART1 scratch register	R/W	0x00	22.3.1.11/22-16
0x10	UDSR—ULCR[DLAB] = x UART1 DMA status register	R	0x01	22.3.1.13/22-17

A.16 Enhanced Local Bus Controller (eLBC)

Table A-16. Enhanced Local Bus Controller (eLBC) Registers

Enhanced Local Bus Controller (eLBC)—Block Base Address 0x0_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	BR0—Base register 0	R/W	0x0000_*nnnn*	10.3.1.1/10-11
0x008	BR1—Base register 1	R/W	0x0000_0000	10.3.1.1/10-11
0x010	BR2—Base register 2	R/W	0x0000_0000	10.3.1.1/10-11
0x018	BR3—Base register 3	R/W	0x0000_0000	10.3.1.1/10-11
0x020	BR4—Base register 4	R/W	0x0000_0000	10.3.1.1/10-11
0x028	BR5—Base register 5	R/W	0x0000_0000	10.3.1.1/10-11
0x030	BR6—Base register 6	R/W	0x0000_0000	10.3.1.1/10-11
0x038	BR7—Base register 7	R/W	0x0000_0000	10.3.1.1/10-11
0x004	OR0—Options register 0	R/W	0x0000_0FF7	10.3.1.2/10-12
0x00C	OR1—Options register 1	R/W	0x0000_0000	10.3.1.2/10-12
0x014	OR2—Options register 2	R/W	0x0000_0000	10.3.1.2/10-12
0x01C	OR3—Options register 3	R/W	0x0000_0000	10.3.1.2/10-12
0x024	OR4—Options register 4	R/W	0x0000_0000	10.3.1.2/10-12
0x02C	OR5—Options register 5	R/W	0x0000_0000	10.3.1.2/10-12
0x034	OR6—Options register 6	R/W	0x0000_0000	10.3.1.2/10-12
0x03C	OR7—Options register 7	R/W	0x0000_0000	10.3.1.2/10-12

Table A-16. Enhanced Local Bus Controller (eLBC) Registers (continued)

Enhanced Local Bus Controller (eLBC)—Block Base Address 0x0_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x040–0x064	Reserved	—	—	—
0x068	MAR—UPM address register	R/W	0x0000_0000	10.3.1.3/10-20
0x06C	Reserved	—	—	—
0x070	MAMR—UPMA mode register	R/W	0x0000_0000	10.3.1.4/10-21
0x074	MBMR—UPMB mode register	R/W	0x0000_0000	10.3.1.4/10-21
0x078	MCMR—UPMC mode register	R/W	0x0000_0000	10.3.1.4/10-21
0x07C–0x080	Reserved	—	—	—
0x084	MRTPR—Memory refresh timer prescaler register	R/W	0x0000_0000	10.3.1.5/10-23
0x088	MDR—UPM/FCM data register	R/W	0x0000_0000	10.3.1.6/10-23
0x08C	Reserved	—	—	—
0x090	LSOR—Special operation initiation register	R/W	0x0000_0000	10.3.1.7/10-24
0x094–0x09C	Reserved	—	—	—
0x0A0	LURT—UPM refresh timer	R/W	0x0000_0000	10.3.1.4/10-21
0x0A4–0x0AC	Reserved	—	—	—
0x0B0	LTESR—Transfer error status register	w1c	0x0000_0000	10.3.1.9/10-26
0x0B4	LTEDR—Transfer error disable register	R/W	0x0000_0000	10.3.1.10/10-28
0x0B8	LTEIR—Transfer error interrupt register	R/W	0x0000_0000	10.3.1.11/10-29
0x0BC	LTEATR—Transfer error attributes register	R/W	0x0000_0000	10.3.1.12/10-30
0x0C0	LTEAR—Transfer error address register	R/W	0x0000_0000	10.3.1.13/10-31
0x0C8–0x0CC	Reserved	—	—	—
0x0D0	LBCR—Configuration register	R/W	0x0000_0000	10.3.1.14/10-31
0x0D4	LCRR—Clock ratio register	R/W	0x8000_0008	10.3.1.15/10-33
0x0D8–0x0DC	Reserved	—	—	—
0x0E0	FMR—Flash mode register	R/W	0x0000_0*n*00	10.3.1.16/10-34
0x0E4	FIR—Flash instruction register	R/W	0x0000_0000	10.3.1.17/10-36
0x0E8	FCR—Flash command register	R/W	0x0000_0000	10.3.1.18/10-37
0x0EC	FBAR—Flash block address register	R/W	0x0000_0000	10.3.1.19/10-38
0x0F0	FPAR—Flash page address register	R/W	0x0000_0000	10.3.1.20/10-38

Table A-16. Enhanced Local Bus Controller (eLBC) Registers (continued)

Enhanced Local Bus Controller (eLBC)—Block Base Address 0x0_5000				
Offset	Register	Access	Reset	Section/Page
0x0F4	FBCR—Flash byte count register	R/W	0x0000_0000	10.3.1.21/10-40
0x0F8–0x0FC	Reserved	—	—	—

A.17 Serial Peripheral Interface (SPI)

Table A-17. Serial Peripheral Interface (SPI) Registers

Serial Peripheral Interface (SPI)—Block Base Address 0x0_7000				
Offset	Register	Access	Reset	Section/Page
0x000–0x01F	Reserved	—	—	—
0x020	SPI mode register (SPMODE)	R/W	0x0000_0000	23.4.1.1/2323-9
0x024	SPI event register (SPIE)	Mixed	0x0000_0000	23.4.1.2/2323-12
0x028	SPI mask register (SPIM)	R/W	0x0000_0000	23.4.1.3/2323-13
0x02C	SPI command register (SPCOM)	W	0x0000_0000	23.4.1.4/2323-14
0x030	SPI transmit register (SPITD)	W	0x0000_0000	23.4.1.5/2323-14
0x034	SPI receive register (SPIRD)	R	0xFFFF_FFFF	23.4.1.6/2323-15
0x038–0xFFF	Reserved	—	—	—

A.18 DMA Controller

Table A-18. DMA Controller Registers

DMA—Block Base Address 0x0_8000				
Offset	Register	Access	Reset	Section/Page
0x030	OMISR—Outbound message interrupt status register	Mixed	0x0000_0000	13.4.1/13-4
0x034	OMIMR—Outbound message interrupt mask register	R/W	0x0000_0000	13.4.2/13-5
0x050	IMR0—Inbound message register 0	R/W	0x0000_0000	13.4.3/13-6
0x054	IMR1—Inbound message register 1	R/W	0x0000_0000	13.4.3/13-6
0x058	OMR0—Outbound message register 0	R/W	0x0000_0000	13.4.4/13-6
0x05C	OMR1—Outbound message register 1	R/W	0x0000_0000	13.4.4/13-6
0x060	ODR—Outbound doorbell register	R/W	0x0000_0000	13.4.5/13-7
0x068	IDR—Inbound doorbell register	R/W	0x0000_0000	13.4.5/13-7
0x080	IMISR—Inbound message interrupt status register	Mixed	0x0000_0000	13.4.6/13-8
0x084	IMIMR—Inbound message interrupt mask register	R/W	0x0000_0000	13.4.7/13-9

Table A-18. DMA Controller Registers (continued)

DMA—Block Base Address 0x0_8000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x100	DMAMR0—DMA 0 mode register	R/W	0x0000_0000	13.4.8.1/13-10
0x104	DMASR0—DMA 0 status register	R/W	0x0000_0000	13.4.8.2/13-12
0x108	DMACDAR0—DMA 0 current descriptor address register	R/W	0x0000_0000	13.4.8.3/13-13
0x110	DMASAR0—DMA 0 source address register	R/W	0x0000_0000	13.4.8.4/13-14
0x118	DMADAR0—DMA 0 destination address register	R/W	0x0000_0000	13.4.8.5/13-14
0x120	DMABCR0—DMA 0 byte count register	R/W	0x0000_0000	13.4.8.6/13-15
0x124	DMANDAR0—DMA 0 next descriptor address register	R/W	0x0000_0000	13.4.8.7/13-15
0x180	DMAMR1—DMA 1 mode register	R/W	0x0000_0000	13.4.8.1/13-10
0x184	DMASR1—DMA 1 status register	R/W	0x0000_0000	13.4.8.2/13-12
0x188	DMACDAR1—DMA 1 current descriptor address register	R/W	0x0000_0000	13.4.8.3/13-13
0x190	DMASAR1—DMA 1 source address register	R/W	0x0000_0000	13.4.8.4/13-14
0x198	DMADAR1—DMA 1 destination address register	R/W	0x0000_0000	13.4.8.5/13-14
0x1A0	DMABCR1—DMA 1 byte count register	R/W	0x0000_0000	13.4.8.6/13-15
0x1A4	DMANDAR1—DMA 1 next descriptor address register	R/W	0x0000_0000	13.4.8.7/13-15
0x200	DMAMR2—DMA 2 mode register	R/W	0x0000_0000	13.4.8.1/13-10
0x204	DMASR2—DMA 2 status register	R/W	0x0000_0000	13.4.8.2/13-12
0x208	DMACDAR2—DMA 2 current descriptor address register	R/W	0x0000_0000	13.4.8.3/13-13
0x210	DMASAR2—DMA 2 source address register	R/W	0x0000_0000	13.4.8.4/13-14
0x218	DMADAR2—DMA 2 destination address register	R/W	0x0000_0000	13.4.8.5/13-14
0x220	DMABCR2—DMA 2 byte count register	R/W	0x0000_0000	13.4.8.6/13-15
0x224	DMANDAR2—DMA 2 next descriptor address register	R/W	0x0000_0000	13.4.8.7/13-15
0x280	DMAMR3—DMA 3 mode register	R/W	0x0000_0000	13.4.8.1/13-10
0x284	DMASR3—DMA 3 status register	R/W	0x0000_0000	13.4.8.2/13-12
0x288	DMACDAR3—DMA 3 current descriptor address register	R/W	0x0000_0000	13.4.8.3/13-13
0x290	DMASAR3—DMA 3 source address register	R/W	0x0000_0000	13.4.8.4/13-14
0x298	DMADAR3—DMA 3 destination address register	R/W	0x0000_0000	13.4.8.5/13-14
0x2A0	DMABCR3—DMA 3 byte count register	R/W	0x0000_0000	13.4.8.6/13-15
0x2A4	DMANDAR3—DMA 3 next descriptor address register	R/W	0x0000_0000	13.4.8.7/13-15
0x2A8	DMAGSR—DMA general status register	R	0x0000_0000	13.4.8.8/13-16
0x2B0–0x2FF	Reserved	—	—	—

A.19 PCI Configuration Access

Table A-19. PCI Configuration Access Registers

PCI Configuration Access—Block Base Address 0x0_8300				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	PCI_CONFIG_ADDRESS	W	0x0000_0000	14.3.1.1/14-13
0x04	PCI_CONFIG_DATA	R/W	0x0000_0000	14.3.1.2/14-14
0x08	PCI_INT_ACK	R	N/A	14.3.1.3/14-15
0x80	PCIPMR0—PCI power management register 0	R	0x7E4B_0001	14.3.3.27/14-42
0x84	PCIPMR1—PCI power management register 1	R/W	0x00*n*0_0000	14.3.3.28/14-43
0x88–0xFF	Reserved	—	—	—

A.20 I/O Sequencer (IOS)

Table A-20. I/O Sequencer (IOS) Registers

I/O Sequencer (IOS)—Block Base Address 0x0_8400				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	POTAR0—PCI outbound translation address register 0	R/W	0x0000_0000	12.4.1/12-3
0x08	POBAR0—PCI outbound base address register 0	R/W	0x0000_0000	12.4.2/12-3
0x10	POCMR0—PCI outbound comparison mask register 0	R/W	0x0000_0000	12.4.3/12-4
0x18	POTAR1—PCI outbound translation address register 1	R/W	0x0000_0000	12.4.1/12-3
0x20	POBAR1—PCI outbound base address register 1	R/W	0x0000_0000	12.4.2/12-3
0x28	POCMR1—PCI outbound comparison mask register 1	R/W	0x0000_0000	12.4.3/12-4
0x30	POTAR2—PCI outbound translation address register 2	R/W	0x0000_0000	12.4.1/12-3
0x38	POBAR2—PCI outbound base address register 2	R/W	0x0000_0000	12.4.2/12-3
0x40	POCMR2—PCI outbound comparison mask register 2	R/W	0x0000_0000	12.4.3/12-4
0x48	POTAR3—PCI outbound translation address register 3	R/W	0x0000_0000	12.4.1/12-3
0x50	POBAR3—PCI outbound base address register 3	R/W	0x0000_0000	12.4.2/12-3
0x58	POCMR3—PCI outbound comparison mask register 3	R/W	0x0000_0000	12.4.3/12-4
0x60	POTAR4—PCI outbound translation address register 4	R/W	0x0000_0000	12.4.1/12-3
0x68	POBAR4—PCI outbound base address register 4	R/W	0x0000_0000	12.4.2/12-3
0x70	POCMR4—PCI outbound comparison mask register 4	R/W	0x0000_0000	12.4.3/12-4
0x78	POTAR5—PCI outbound translation address register 5	R/W	0x0000_0000	12.4.1/12-3
0x80	POBAR5—PCI outbound base address register 5	R/W	0x0000_0000	12.4.2/12-3
0x88	POCMR5—PCI outbound comparison mask register 5	R/W	0x0000_0000	12.4.3/12-4

Table A-20. I/O Sequencer (IOS) Registers (continued)

I/O Sequencer (IOS)—Block Base Address 0x0_8400				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xF0	PMCR—Power management control register	R/W	0x0000_0000	12.4.4/12-5
0xF8	DTCR—Discard timer control register	R/W	0x0000_0000	12.4.5/12-6

A.21 PCI Controller

Table A-21. PCI Controller Registers

PCI Controller—Block Base Address 0x0_8500				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
PCI Error Management Registers				
0x00	PCI error status register (PCI_ESR)	w1c	0x0000_0000	14.3.2.1/14-15
0x04	PCI error capture disable register (PCI_ECDR)	R/W	0x0000_0000	14.3.2.2/14-16
0x08	PCI error enable register (PCI_EER)	R/W	0x0000_0000	14.3.2.3/14-17
0x0C	PCI error attributes capture register (PCI_EATCR)	R/W	0x0000_0000	14.3.2.4/14-18
0x10	PCI error address capture register (PCI_EACR)	R	0x0000_0000	14.3.2.5/14-19
0x14	PCI error extended address capture register (PCI_EEACR)	R	0x0000_0000	14.3.2.6/14-20
0x18	PCI error data capture register (PCI_EDCR)	R/W	0x0000_0000	14.3.2.7/14-20
PCI Control and Status Registers				
0x20	PCI general control register (PCI_GCR)	R/W	0x0000_0000	14.3.2.8/14-21
0x24	PCI error control register (PCI_ECR)	R/W	0x0000_0000	14.3.2.9/14-21
0x28	PCI general status register (PCI_GSR)	R	0x0000_0000	14.3.2.10/14-22
PCI Inbound ATU Registers				
0x38	PCI inbound translation address register 2 (PITAR2)	R/W	0x0000_0000	14.3.2.11/14-23
0x3C	Reserved	—	—	—
0x40	PCI inbound base address register 2 (PIBAR2)	R/W	0x0000_0000	14.3.2.12/14-24
0x44	PCI inbound extended base address register 2 (PIEBAR2)	R/W	0x0000_0000	14.3.2.13/14-24
0x48	PCI inbound window attributes register 2 (PIWAR2)	R/W	0x0000_0000	14.3.2.14/14-25
0x50	PCI inbound translation address register 1 (PITAR1)	R/W	0x0000_0000	14.3.2.11/14-23
0x54	Reserved	—	—	—
0x58	PCI inbound base address register 1 (PIBAR1)	R/W	0x0000_0000	14.3.2.12/14-24
0x5C	PCI inbound extended base address register 1 (PIEBAR1)	R/W	0x0000_0000	14.3.2.13/14-24
0x60	PCI inbound window attributes register 1 (PIWAR1)	R/W	0x0000_0000	14.3.2.14/14-25
0x68	PCI inbound translation address register 0 (PITAR0)	R/W	0x0000_0000	14.3.2.11/14-23

Table A-21. PCI Controller Registers (continued)

PCI Controller—Block Base Address 0x0_8500				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x6C	Reserved	—	—	—
0x70	PCI inbound base address register 0 (PIBAR0)	R/W	0x0000_0000	14.3.2.12/14-24
0x78	PCI inbound window attributes register 0 (PIWAR0)	R/W	0x0000_0000	14.3.2.13/14-24
0x7C–0xFF	Reserved	—	—	—

A.22 PCI Express Controller

Table A-22. PCI Express Controller Registers

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
PCI Express Controller 1 Registers				
PCI Express1 Core Configuration Header Registers				
0x000	PCI Express Vendor ID Register	R	0x1957	15.4.1.1/15-15
0x002	PCI Express Device ID Register	R	Device-specific	15.4.1.2/15-16
0x004	PCI Express Command Register	Mixed	0x0000	15.4.1.3/15-16
0x006	PCI Express Status Register	Mixed	0x0010	15.4.1.4/15-17
0x008	PCI Express Revision ID Register	R	Revision-specific	15.4.1.5/15-18
0x009	PCI Express Class Code Register	Mixed	0x0B20	15.4.1.7/15-20
0x00C	PCI Express Cache Line Size Register	R/W	0x00	15.4.1.7/15-20
0x00D	PCI Express Latency Timer Register	R	0x00	15.4.1.8/15-20
0x00E	PCI Express Header Type Register	R	0x00 (EP mode) 0x01 (RC mode)	15.4.1.10/15-22
0x00F	PCI Express BIST Register	R	0x00	15.4.1.10/15-22
0x010–0x014	Base Address Registers 0 and 1 (BAR0/BAR1) (EP mode only)	Mixed	0x0008	15.4.2.1.1/15-23
0x018–0x020	Base Address Registers 2 and 4 (BAR2/BAR4) (EP mode only)	Mixed	0x0000_000C	15.4.2.1.2/15-23
0x01C–0x024	Base Address Registers 3 and 5 (BAR3/BAR5) (EP mode only)	R/W	0x0000_0000	15.4.2.1.3/15-24
0x02C	PCI Express Subsystem Vendor ID Register (EP mode only)	Special	0x0000	15.4.2.2/15-24

Table A-22. PCI Express Controller Registers (continued)

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x02E	PCI Express Subsystem ID Register (EP mode only)	Special	0x0000	15.4.2.3/15-25
0x034	PCI Express Capabilities Pointer Register	R	0x0044	15.4.2.4/15-25
0x03C	PCI Express Interrupt Line Register (EP mode only)	R/W	0x0000	15.4.2.5/15-26
0x03D	PCI Express Interrupt Pin Register	R	0x0001	15.4.2.6/15-26
0x03E	PCI Express Minimum Grant Register (EP mode only)	R	0x0000	15.4.2.7/15-27
0x03F	PCI Express Maximum Latency Register (EP mode only)	R	0x0000	15.4.2.8/15-27
0x018	PCI Express Primary Bus Number Register (RC mode only)	R/W	0x0000	15.4.3.1/15-28
0x019	PCI Express Secondary Bus Number Register (RC mode only)	R/W	0x0000	15.4.3.2/15-29
0x01A	PCI Express Subordinate Bus Number Register (RC mode only)	R/W	0x0000	15.4.3.3/15-29
0x01B	Secondary Latency Timer Register 2 (RC mode only)		0x0000	
0x01C	PCI Express I/O Base Register (RC mode only)	R	0x0000	15.4.3.5/15-30
0x01D	PCI Express I/O Limit Register (RC mode only)	R	0x0000	15.4.3.6/15-30
0x01E	PCI Express Secondary Status Register (RC mode only)	Mixed	0x0000	15.4.3.7/15-31
0x020	PCI Express Memory Base Register (RC mode only)	R/W	0x0000	15.4.3.8/15-31
0x022	PCI Express Memory Limit Register (RC mode only)	R/W	0x0000	15.4.3.9/15-32
0x024	PCI Express Prefetchable Memory Base Register (RC mode only)	R/W	0x0000	15.4.3.10/15-32
0x026	PCI Express Prefetchable Memory Limit Register (RC mode only)	R/W	0x0000	15.4.3.11/15-33
0x028	PCI Express Prefetchable Base Upper 32-Bit Register (RC mode only)	R/W	0x0000	15.4.3.12/15-33
0x02C	PCI Express Prefetchable Limit Upper 32-Bit Register (RC mode only)	R/W	0x0000	15.4.3.13/15-34
0x030	PCI Express I/O Base Upper 16-Bit Register (RC mode only)	R	0x0000	15.4.3.14/15-34
0x032	PCI Express I/O Limit Upper 16-Bit Register (RC mode only)	R‘	0x0000	15.4.3.15/15-35
0x034	PCI Express Capabilities Pointer Register	R	0x044	15.4.3.16/15-35
0x03C	PCI Express Interrupt Line Register	R/W	0x0000	15.4.3.17/15-36
0x03D	PCI Express Interrupt Pin Register	R	0x0001	15.4.3.18/15-36
0x03E	PCI Express Bridge Control Register (RC mode only)	R/W	0x0000	15.4.3.19/15-36
0x044	PCI Express Power Management Capability ID Register	R	0x01	15.4.4.1/15-39
0x045	PCI Express Power Management Next Capabilities Pointer Register	R	0x4C	15.4.4.2/15-39

Table A-22. PCI Express Controller Registers (continued)

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x046	PCI Express Power Management Capabilities Register	R	0x7E02	15.4.4.3/15-39
0x048	PCI Express Power Management Status and Control Register	Mixed	0x0000	15.4.4.4/15-40
0x04B	PCI Express Power Management Data Register	R	0x0000	15.4.4.5/15-41
0x04C	PCI Express Capability ID Register	R	0x10	15.4.4.6/15-41
0x04D	PCI Express Next Capabilities Pointer Register	R	0x70	15.4.4.7/15-41
0x04E	PCI Express Capabilities Register	R	0x00*n*1	15.4.4.8/15-42
0x050	PCI Express Device Capabilities Register	R	0x0000_0000	15.4.4.9/15-42
0x054	PCI Express Device Control Register	R/W	0x2810	15.4.4.10/15-43
0x056	PCI Express Device Status Register	Mixed	0x0000	15.4.4.11/15-44
0x058	PCI Express Link Capabilities Register	R	0x0003_D421	15.4.4.12/15-45
0x05C	PCI Express Link Control Register	R/W	0x0000	15.4.4.13/15-45
0x05E	PCI Express Link Status Register	R	0x0011	15.4.4.14/15-46
0x060	PCI Express Slot Capabilities Register	R	0x000007c0	15.4.4.15/15-46
0x064	PCI Express Slot Control Register	R/W	0x0000	15.4.4.16/15-47
0x066	PCI Express Slot Status Register	Mixed	0x0040	15.4.4.17/15-48
0x068	PCI Express Root Control Register (RC mode only)	R/W	0x0000	15.4.4.18/15-49
0x06C	PCI Express Root Status Register (RC mode only)	Mixed	0x0000_0000	15.4.4.19/15-49
0x070	PCI Express MSI Message Capability ID Register (EP mode only)	R	0x05	15.4.4.20/15-50
0x072	PCI Express MSI Message Control Register (EP mode only)	Mixed	0x0088	15.4.4.21/15-50
0x074	PCI Express MSI Message Address Register (EP mode only)	R/W	0x0000_0000	15.4.4.22/15-51
0x078	PCI Express MSI Message Upper Address Register (EP mode only)	R/W	0x0000_0000	15.4.4.23/15-51
0x07C	PCI Express MSI Message Data Register (EP mode only)	R/W	0x0000	15.4.4.24/15-51
0x100	PCI Express Advanced Error Reporting Capability ID Register	R	0x1381_0001	15.4.5.1/15-53
0x104	PCI Express Uncorrectable Error Status Register	R/W	0x0000_0000	15.4.5.2/15-53
0x108	PCI Express Uncorrectable Error Mask Register	R/W	0x0000_0000	15.4.5.3/15-54
0x10C	PCI Express Uncorrectable Error Severity Register	R/W	0x0006_2010	15.4.5.4/15-55
0x110	PCI Express Correctable Error Status Register	w1c	0x0000_0000	15.4.5.5/15-56
0x114	PCI Express Correctable Error Mask Register	R/W	0x0000_0000	15.4.5.6/15-57
0x118	PCI Express Advanced Error Capabilities and Control Register	R/W	0x0000_00A0	15.4.5.7/15-57

Table A-22. PCI Express Controller Registers (continued)

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x11C	PCI Express Header Log Register	R	0x0000_0000	15.4.5.8/15-59
0x120	PCI Express Header Log Register	R	0x0000_0000	
0x124	PCI Express Header Log Register	R	0x0000_0000	
0x128	PCI Express Header Log Register	R	0x0000_0000	
0x12C	PCI Express Root Error Command Register	R/W	0x0000_0000	15.4.5.9/15-60
0x130	PCI Express Root Error Status Register	Mixed	0x0000_0000	15.4.5.10/15-60
0x134	PCI Express Error Source Identification Register	R	0x0000_0000	15.4.5.11/15-61
PCI Express Core Control and Status Registers (CSRs)				
0x404	PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)	R	0x0000_0000	15.4.6.1/15-62
0x41C	PCI Express N_FTS Control Register (PEX_NFTS_CTRL)	R/W	0x0000_4040	15.4.6.2/15-63
0x438	PCI Express ACK Replay Timeout Register (PEX_ACKRPLY_TO)	R/W	0x00C2_415C	15.4.6.3/15-64
0x440	PCI Express Core Clock Ratio Register (PEX_GCLK_RATIO)	Mixed	0x0000_0010	15.4.6.4/15-65
0x450	PCI Express Power Management Timer Register (PEX_PM_TIMER)	Mixed	0x0019_0960	15.4.6.5/15-66
0x454	PCI Express PME Time-Out Register (PEX_PME_TIMEOUT)	Mixed	0x0262_5A00	15.4.6.6/15-67
0x45C	PCI Express ASPM Request Timer Register (PEX_ASPM_REQTMR) (RC mode only)	R/W	0x0000_0ED8	15.4.6.7/15-68
0x478	PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE)	R/W	0x0000_0000	15.4.6.8/15-68
0x47C	PCI Express Device Capabilities Update Register (PEX_DEVCAP_UPDATE)	R/W	0x0000_0000	15.4.6.9/15-69
0x480	PCI Express Link Capabilities Update Register (PEX_LINKCAP_UPDATE)	R/W	0x0000_3D42	/15-69
0x490	PCI Express Slot Capabilities Update Register (PEX_SLCAP_UPDATE)	R/W	0x000007c0	15.4.6.11/15-71
0x4B0	PCI Express Configuration Ready Register (PEX_CFG_READY)	Mixed	0x0000_0000	15.4.6.12/15-72
PCI Express BAR Configuration Registers (EP Mode)				
0x4D8	PCI Express BAR Size Low Configuration Register (PEX_BAR_SIZEL)	R/W	0xFC00_0000	15.4.7.1/15-73
0x4DC	Reserved	R/W	0xFFFF_FFFF	15.4.7.2/15-74
0x4E0	PCI Express BAR Select Configuration Register (PEX_BAR_SEL)	R/W	0x0000_0400	15.4.7.2/15-74

Table A-22. PCI Express Controller Registers (continued)

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x504	PCI Express BAR Prefetch Configuration Register (PEX_BAR_PF)	R/W	0x0000_0400	15.4.7.3/15-74
PCI Express Extended Status and Control Register				
0x590	PCI Express PME_To_Ack Timeout Register (PEX_PME_TO_ACK_TOR)	Mixed	0x0262_5A00	15.4.8.1/15-75
0x594	PCI Express PME_To_Ack Status Register (PEX_PME_TO_ACK_SR)	w1c	0x0000_0000	15.4.8.2/15-76
0x5A0	PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK)	Mixed	0x0000_003F	15.4.8.3/15-77
PCI Express CSB Bridge Registers				
Global Registers				
0x800	Reserved	RO	0x0110_1010	—
0x804	Reserved	RO	0x0003_249F	—
0x808	PCI Express CSB Bridge Control register (PEX_CSB_CTRL)	R/W	0x0000_0130	15.5.2.1/15-78
0x80C	Reserved	RO	0x0000_0000	—
0x814	PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR)	R/W	0x0000_0000	15.5.2.2/15-79
0x818	Reserved	RO	0x0000_0000	—
0x81C	PCI Express CSB Bridge Status register (PEX_CSB_STAT)	RO	0x0000_0000	15.5.2.3/15-80
0x820	Reserved	RO	0x0000_0000	—
PCI Express Outbound PIO Registers				
0x840	PCI Express Outbound PIO Control Register (PEX_CSB_OBCTRL)	R/W	0x0000_0000	15.5.3.1/15-81
0x844	PCI Express Outbound PIO Status Register (PEX_CSB_OBSTAT)	w1c	0x0000_0000	15.5.3.2/15-82
0x848	Reserved	RO	0x0000_0000	—
PCI Express Inbound PIO Registers				
0x8E0	PCI Express Inbound PIO Control Register (PEX_CSB_IBCTRL)	R/W	0x0000_0000	15.5.4.1/15-83
0x8E4	PCI Express Inbound PIO Status Register (PEX_CSB_IBSTAT)	w1c	0x0000_0000	15.5.4.2/15-83
0x8E8	Reserved	RO	0x0000_0000	—
PCI Express DMA Registers				
0x990	Reserved	RO	0x0000_0000	—

Table A-22. PCI Express Controller Registers (continued)

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x9A0	PCI Express Write DMA Control Register (PEX_WDMA_CTRL)	R/W	0x0000_0000	15.5.5.1/15-84
0x9A4	PCI Express Write DMA first Address Register (PEX_WDMA_ADDR)	R/W	0x0000_0000	15.5.5.2/15-85
0x9A8	PCI Express Write DMA Status Register (PEX_WDMA_STAT)	w1c	0x0000_0000	15.5.5.3/15-85
0x9AC	Reserved	RO	0x0000_0000	—
0xA40	PCI Express Read DMA Control Register (PEX_RDMA_CTRL)	R/W	0x0000_0000	15.5.5.4/15-86
0xA44	PCI Express Read DMA first Address Register (PEX_RDMA_ADDR)	R/W	0x0000_0000	15.5.5.5/15-87
0xA48	PCI Express Read DMA Status Register (PEX_RDMA_STAT)	w1c	0x0000_0000	15.5.5.6/15-87
Mailbox Registers				
0xB20	PCI Express Outbound Mailbox Control Register (PEX_OMBCR)	R/W	0x0000_0000	15.5.6.1/15-88
0xB24	PCI Express Outbound Mailbox Data Register (PEX_OMBDR)	R/W	0x0000_0000	15.5.6.2/15-89
0xB60	PCI Express Inbound Mailbox Control Register (PEX_IMBCR)	R/W	0x0000_0000	15.5.6.3/15-89
0xB64	PCI Express Inbound Mailbox Data Register (PEX_IMBDR)	R/W	0x0000_0000	15.5.6.4/15-90
PCI Express Host Interrupts Registers				
0xBA0	PCI Express Host Interrupt Enable Register (PEX_HIER)	R/W	0x0000_0000	15.5.7.1/15-90
0xBA4	PCI Express Host Interrupt Status Register (PEX_HISR)	w1c	0x0000_0000	15.5.7.2/15-91
0xBA8	PCI Express Host Outbound PIO Interrupt Vector Register (PEX_HOPIVR)	R/W	0x0000_0000	15.5.7.3/15-92
0xBC0	PCI Express Host Inbound PIO Interrupt Vector Register (PEX_HIPIVR)	R/W	0x0000_0000	15.5.7.4/15-93
0xBC8	PCI Express Host Write DMA Interrupt Vector Register (PEX_HWDIVR)	R/W	0x0000_0000	15.5.7.5/15-93
0xBD0	PCI Express Host Read DMA Interrupt Vector Register (PEX_HRDIVR)	R/W	0x0000_0000	15.5.7.6/15-94
0xBD8	PCI Express Host Miscellaneous Interrupt Vector Register (PEX_HMIVR)	R/W	0x0000_0000	15.5.7.7/15-94
CSB System Interrupts Registers				
0xBE0	CSB System PIO Interrupt Enable Register (PEX_CSPIER)	R/W	0x0000_0000	15.5.8.1/15-95
0xBE4	CSB System Write DMA Interrupt Enable Register (PEX_CSWDIER)	R/W	0x0000_0000	15.5.8.2/15-96
0xBE8	CSB System Read DMA Interrupt Enable Register (PEX_CSRDIER)	R/W	0x0000_0000	15.5.8.3/15-97

Table A-22. PCI Express Controller Registers (continued)

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xBEC	CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER)	R/W	0x0000_0002	15.5.8.4/15-98
0xBF0	CSB System PIO Interrupt Status Register (PEX_CSPISR)	w1c	0x0000_0000	15.5.8.5/15-99
0xBF4	CSB System Write DMA Interrupt Status Register (PEX_CSWDISR)	w1c	0x0000_0000	15.5.8.6/15-100
0xBF8	CSB System Read DMA Interrupt Status Register (PEX_CSRDISR)	w1c	0x0000_0000	15.5.8.7/15-101
0xBFC	CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR)	w1c	0x0000_0000	15.5.8.8/15-101
Power Management Registers				
0xC80	PCI Express PM Control Register (PEX_PM_CTRL)	R/W	0x0000_0000	15.5.9.1/15-103
PCI Express Outbound Address Mapping Registers				
0xCA0	PCI Express Outbound Window Attributes Register 0 (PEX_OWAR0)	R/W	0x0000_0000	15.5.10.1/15-104
0xCA4	PCI Express Outbound Window Base Address Register 0 (PEX_OWBAR0)	R/W	0x0000_0000	15.5.10.2/15-105
0xCA8	PCI Express Outbound Window Translation Address Register Low 0 (PEX_OWTARL0)	R/W	0x0000_0000	15.5.10.3/15-106
0xCAC	PCI Express Outbound Window Translation Address Register High 0 (PEX_OWTARH0)	R/W	0x0000_0000	15.5.10.4/15-106
0xCB0	PCI Express Outbound Window Attributes Register 1 (PEX_OWAR1)	R/W	0x0000_0000	15.5.10.1/15-104
0xCB4	PCI Express Outbound Window Base Address Register 1 (PEX_OWBAR1)	R/W	0x0000_0000	15.5.10.2/15-105
0xCB8	PCI Express Outbound Window Translation Address Register Low 1 (PEX_OWTARL1)	R/W	0x0000_0000	15.5.10.3/15-106
0xCBC	PCI Express Outbound Window Translation Address Register High 1 (PEX_OWTARH1)	R/W	0x0000_0000	15.5.10.4/15-106
0xCC0	PCI Express Outbound Window Attributes Register 2 (PEX_OWAR2)	R/W	0x0000_0000	15.5.10.1/15-104
0xCC4	PCI Express Outbound Window Base Address Register 2 (PEX_OWBAR2)	R/W	0x0000_0000	15.5.10.2/15-105
0xCC8	PCI Express Outbound Window Translation Address Register Low 2 (PEX_OWTARL2)	R/W	0x0000_0000	15.5.10.3/15-106
0xCCC	PCI Express Outbound Window Translation Address Register High 2 (PEX_OWTARH2)	R/W	0x0000_0000	15.5.10.4/15-106

Table A-22. PCI Express Controller Registers (continued)

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xCD0	PCI Express Outbound Window Attributes Register 3 (PEX_OWAR3)	R/W	0x0000_0000	15.5.10.1/15-104
0xCD4	PCI Express Outbound Window Base Address Register 3 (PEX_OWBAR3)	R/W	0x0000_0000	15.5.10.2/15-105
0xCD8	PCI Express Outbound Window Translation Address Register Low 3 (PEX_OWTARL3)	R/W	0x0000_0000	15.5.10.3/15-106
0xCDC	PCI Express Outbound Window Translation Address Register High 3 (PEX_OWTARH3)	R/W	0x0000_0000	15.5.10.4/15-106
PCI Express EP Inbound Address Translation Registers				
0xDE0	PCI Express EP Inbound Window Translation Address Register 0 (PEX_EPIWTAR0)	R/W	0x0000_0000	15.5.11.1/15-107
0xDE4	PCI Express EP Inbound Window Translation Address Register 1 (PEX_EPIWTAR1)	R/W	0x0000_0000	15.5.11.1/15-107
0xDE8	PCI Express EP Inbound Window Translation Address Register 2 (PEX_EPIWTAR2)	R/W	0x0000_0000	15.5.11.1/15-107
0xDEC	PCI Express EP Inbound Window Translation Address Register 3 (PEX_EPIWTAR3)	R/W	0x0000_0000	15.5.11.1/15-107
PCI Express RC Inbound Address Mapping Registers				
0xE60	PCI Express RC Inbound Window Attributes Register 0 (PEX_RCIWAR0)	R/W	0x0000_0000	15.5.12.1/15-108
0xE64	PCI Express RC Inbound Window Translation Address Register 0 (PEX_RCIWTAR0)	R/W	0x0000_0000	15.5.12.2/15-109
0xE68	PCI Express RC Inbound Window Base Address Register Low 0 (PEX_RCIWBARL0)	R/W	0x0000_0000	15.5.12.3/15-110
0xE6C	PCI Express RC Inbound Window Base Address Register High 0 (PEX_RCIWBARH0)	R/W	0x0000_0000	15.5.12.4/15-110
0xE70	PCI Express RC Inbound Window Attributes Register 1 (PEX_RCIWAR1)	R/W	0x0000_0000	15.5.12.1/15-108
0xE74	PCI Express RC Inbound Window Translation Address Register 1 (PEX_RCIWTAR1)	R/W	0x0000_0000	15.5.12.2/15-109
0xE78	PCI Express RC Inbound Window Base Address Register Low 1 (PEX_RCIWBARL1)	R/W	0x0000_0000	15.5.12.3/15-110
0xE7C	PCI Express RC Inbound Window Base Address Register High 1 (PEX_RCIWBARH1)	R/W	0x0000_0000	15.5.12.4/15-110
0xE80	PCI Express RC Inbound Window Attributes Register 2 (PEX_RCIWAR2)	R/W	0x0000_0000	15.5.12.1/15-108
0xE84	PCI Express RC Inbound Window Translation Address Register 2 (PEX_RCIWTAR2)	R/W	0x0000_0000	15.5.12.2/15-109

Table A-22. PCI Express Controller Registers (continued)

PCI Express 1—Block Base Address 0x0_9000 PCI Express 2—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xE88	PCI Express RC Inbound Window Base Address Register Low 2 (PEX_RCIWBARL2)	R/W	0x0000_0000	15.5.12.3/15-110
0xE8C	PCI Express RC Inbound Window Base Address Register High 2 (PEX_RCIWBARH2)	R/W	0x0000_0000	15.5.12.4/15-110
0xE90	PCI Express RC Inbound Window Attributes Register 3 (PEX_RCIWAR3)	R/W	0x0000_0000	15.5.12.1/15-108
0xE94	PCI Express RC Inbound Window Translation Address Register 3 (PEX_RCIWTAR3)	R/W	0x0000_0000	15.5.12.2/15-109
0xE98	PCI Express RC Inbound Window Base Address Register Low 3 (PEX_RCIWBARL3)	R/W	0x0000_0000	15.5.12.3/15-110
0xE9C	PCI Express RC Inbound Window Base Address Register High 3 (PEX_RCIWBARH3)	R/W	0x0000_0000	15.5.12.4/15-110
PCI Express Controller 2 Memory-Mapped Registers				
0x000–0xFFC	PCI Express Controller 2 registers **Note:** All registers defined for PCI Express controller 1 are also defined for PCI Express controller 2; the offsets of the PCI Express controller 2 registers are the same except they have a different block base address of 0x0_A000.			

A.23 Serial ATA (SATA) Controller

Table A-23. Serial ATA (SATA) Controller Registers

SATA 1 Controller—Block Base Address 0x1_8000 SATA 2 Controller—Block Base Address 0x1_9000 SATA 3 Controller—Block Base Address 0x1_A000 SATA 4 Controller—Block Base Address 0x1_B000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	CQR—Command queue register	R/W	0x0000_0000	16.3.2.1/16-5
0x008	CAR—Command active register	R	0x0000_0000	16.3.2.2/16-6
0x010	CCR—Command completed register	w1c	0x0000_0000	16.3.2.3/16-7
0x018	CER—Command error register	w1c	0x0000_0000	16.3.2.4/16-7
0x020	DE—Device error register	w1c	0x0000_0000	16.3.2.5/16-8
0x024	CHBA—Command header base address	R/W	0x0000_0000	16.3.2.6/16-9
0x028	HStatus—Host status register	w1c	0x2000_0000	16.3.2.7/16-9
0x02C	HControl—Host control register	Mixed	0x0000_0100	16.3.2.8/16-12
0x030	CQPMP—Port number queue register	R/W	0x0000_0000	16.3.2.9/16-13
0x034	SIG—Signature register	R	0xFFFF_FFFF	16.3.2.10/16-14

Table A-23. Serial ATA (SATA) Controller Registers (continued)

SATA 1 Controller—Block Base Address 0x1_8000 SATA 2 Controller—Block Base Address 0x1_9000 SATA 3 Controller—Block Base Address 0x1_A000 SATA 4 Controller—Block Base Address 0x1_B000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x038	ICC—Interrupt coalescing control register	R/W	0x0100_0000	16.3.2.11/16-14
SATA1 Superset Registers				
0x100	SStatus—SATA interface status register	R	0x0000_0000	16.3.3.1/16-15
0x104	SError—SATA interface error register	w1c	0x0000_0000	16.3.3.2/16-16
0x108	SControl—SATA interface control register	R/W	0x0000_0300	16.3.3.3/16-18
0x10C	SNotification—SATA interface notification register	w1c	0x0000_0000	16.3.3.4/16-19
SATA1 Control Status Registers				
0x140	TransCfg—Transport layer configuration	R/W	0x0800_0016	16.3.4.1/16-20
0x144	TransStatus—Transport layer status	R	0x0000_0000	16.3.4.2/16-21
0x148	LinkCfg—Link layer configuration	R/W	0x0000_FF34	16.3.4.3/16-21
0x14C	LinkCfg1—Link layer configuration1	R/W	0x0000_0000	16.3.4.4/16-22
0x150	LinkCfg2—Link layer configuration2	R/W	0x0000_0000	16.3.4.5/16-23
0x154	LinkStatus—Link layer status	R	0x0000_0000	16.3.4.6/16-23
0x158	LinkStatus1—Link layer status1	R	0x0000_0000	16.3.4.7/16-24
0x15C	PhyCtrlCfg1—PHY control configuration1	R/W	0x0000_3800	16.3.4.8/16-26
0x160	CommandStatus—Link layer command status	R	0x0000_0000	16.3.4.9/16-27
0x164–0x17C	Reserved	—	—	—
SATA1 System Control Registers				
0x410	SYSPR—System priority register	R/W	0x0000_0000	16.3.5.1/16-28
0x40C–0xFFF	Reserved	—	—	—
SATA2—Block Base Address: 0x1_9000				
SATA2 has the same memory-mapped registers that are described for SATA1 from 0x1_8000 to 0x1_8FFF except the offsets are from 0x1_9000 to 0x1_9FFF.				
SATA3—Block Base Address: 0x1_A000				
SATA3 has the same memory-mapped registers that are described for SATA1 from 0x1_8000 to 0x1_8FFF except the offsets are from 0x1_A000 to 0x1_AFFF.				
SATA4—Block Base Address: 0x1_B000				
SATA4 has the same memory-mapped registers that are described for SATA1 from 0x1_8000 to 0x1_8FFF except the offsets are from 0x1_B000 to 0x1_BFFF.				

A.24 USB DR Controller

Table A-24. USB DR Controller Registers

USB DR Controller—Block Base Address 0x2_3000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
USB DR Controller Registers				
0x000–0x0FF	Reserved, should be cleared	—	—	—
0x100	CAPLENGTH—Capability register length	R	0x40	20.3.1.1/2020-10
0x102	HCIVERSION—Host interface version number	R	0x0100	20.3.1.2/2020-11
0x104	HCSPARAMS—Host crtl. structural parameters	R	0x0001_0011	20.3.1.3/2020-11
0x108	HCCPARAMS—Host crtl. capability parameters	R	0x0000_0006	20.3.1.4/2020-12
0x120	DCIVERSION—Device interface version number	R	0x0001	20.3.1.5/2020-13
0x124	DCCPARAMS—Device controller parameters	R	0x0000_0186	20.3.1.6/2020-13
0x140	USBCMD—USB command	Mixed	0x0008_0000	20.3.2.1/2020-14
0x144	USBSTS—USB status	Mixed	0x0000_0000	20.3.2.2/2020-17
0x148	USBINTR—USB interrupt enable	R/W	0x0000_0000	20.3.2.3/2020-19
0x14C	FRINDEX—USB frame index	R/W	0x0000_*nnnn*	20.3.2.4/2020-20
0x154	PERIODICLISTBASE—Frame list base address[1]	R/W	0x*nnnn*_0000	20.3.2.6/2020-22
	DEVICEADDR—USB device address	R/W	0x0000_0000	20.3.2.7/2020-22
0x158	ASYNCLISTADDR—Next asynchronous list addr (host mode)[1]	R/W	0x0000_0000	20.3.2.8/2020-23
	ENDPOINTLISTADDR—Address at endpoint list (device mode)	R/W	0x0000_0000	20.3.2.9/2020-23
0x160	BURSTSIZE—Programmable burst size	R/W	0x0000_1010	20.3.2.10/2020-24
0x164	TXFILLTUNING—Host TT transmit pre-buffer packet tuning	R/W	0x0000_0000	20.3.2.11/2020-24
0x170	ULPI VIEWPORT—ULPI Register Access	Mixed	0x0000_0000	20.3.2.12/2020-26
0x180	CONFIGFLAG—Configured flag register	R	0x0000‘_0001	20.3.2.13/2020-28
0x184	PORTSC—Port status/control	Mixed	0x8000_0010	20.3.2.14/2020-28
0x1A4	Reserved	—	—	—
0x1A8	USBMODE—USB device mode	R/W	0x0000_0000	20.3.2.16/2020-36
0x1AC	ENDPTSETUPSTAT—Endpoint setup status	R/W	0x0000_0000	20.3.2.17/2020-37
0x1B0	ENDPOINTPRIME—Endpoint initialization	R/W	0x0000_0000	20.3.2.18/2020-37
0x1B4	ENDPTFLUSH—Endpoint de-initialize	R/W	0x0000_0000	20.3.2.19/2020-38
0x1B8	ENDPTSTATUS—Endpoint status	R	0x0000_0000	20.3.2.20/2020-39
0x1BC	ENDPTCOMPLETE—Endpoint complete	w1c	0x0000_0000	20.3.2.21/2020-39

Table A-24. USB DR Controller Registers (continued)

USB DR Controller—Block Base Address 0x2_3000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x1C0	ENDPTCTRL0—Endpoint control 0	Mixed	0x0080_0080	20.3.2.22/2020-40
0x1C4	ENDPTCTRL1—Endpoint control 1	R/W	0x0000_0000	20.3.2.23/2020-41
0x1C8	ENDPTCTRL2—Endpoint control 2	R/W	0x0000_0000	20.3.2.23/2020-41
0x1CA	ENDPTCTRL3—Endpoint control 3	R/W	0x0000_0000	20.3.2.23/2020-41
0x1D0	ENDPTCTRL4—Endpoint control 4	R/W	0x0000_0000	20.3.2.23/2020-41
0x1D4	ENDPTCTRL5—Endpoint control 5	R/W	0x0000_0000	20.3.2.23/2020-41
0x400	SNOOP1—Snoop 1	R/W	0x0000_0000	20.3.2.24/2020-42
0x404	SNOOP2—Snoop 2	R/W	0x0000_0000	20.3.2.24/2020-42
0x408	AGE_CNT_THRESH—Age count threshold	R/W	0x0000_0000	20.3.2.25/2020-43
0x40C	PRI_CTRL—Priority control	R/W	0x0000_0000	20.3.2.26/2020-45
0x410	SI_CTRL—System interface control	R/W	0x0000_0000	20.3.2.27/2020-45
0x500	CONTROL—Control		0x0000_0000	20.3.2.28/2020-46
0x504–0xFFF	Reserved, should be cleared	—	—	—

[1] This register has separate functions for the host and device operation; the host function is listed first in the table.

A.25 Enhanced Three-Speed Ethernet Controllers (eTSECs)

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
eTSEC General Control and Status Registers				
0x000	TSEC_ID*—Controller ID register	R	0x0124_0005	18.5.3.1.1/18-22
0x004	TSEC_ID2*—Controller ID register	R	0x00EC_00F0	18.5.3.1.2/18-22
0x008–0x00C	Reserved	—	—	—
0x010	IEVENT—Interrupt event register	w1c	0x0000_0000	18.5.3.1.3/18-23
0x014	IMASK—Interrupt mask register	R/W	0x0000_0000	18.5.3.1.4/18-27
0x018	EDIS—Error disabled register	R/W	0x0000_0000	18.5.3.1.5/18-29
0x01C	Reserved	—	—	—
0x020	ECNTRL—Ethernet control register	R/W	0x0000_RR00[1]	18.5.3.1.6/18-31
0x024	Reserved	—	—	—

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x028	PTV—Pause time value register	R/W	0x0000_0000	18.5.3.1.7/18-33
0x02C	DMACTRL—DMA control register	R/W	0x0000_0000	18.5.3.1.8/18-34
0x030	Reserved	—	—	—
0x034–0x0FC	Reserved	—	—	—
eTSEC Transmit Control and Status Registers				
0x100	TCTRL—Transmit control register	R/W	0x0000_0000	18.5.3.2.1/18-36
0x104	TSTAT—Transmit status register	w1c	0x0000_0000	18.5.3.2.2/18-38
0x108	DFVLAN*—Default VLAN control word	R/W	0x8100_0000	18.5.3.2.3/18-42
0x10C	Reserved	—	—	—
0x110	TXIC—Transmit interrupt coalescing register	R/W	0x0000_0000	18.5.3.2.4/18-43
0x114	TQUEUE*—Transmit queue control register	R/W	0x0000_8000	18.5.3.2.5/18-44
0x118–0x13C	Reserved	—	—	—
0x140	TR03WT*—TxBD Rings 0–3 round-robin weightings	R/W	0x0000_0000	18.5.3.2.6/18-44
0x144	TR47WT*—TxBD Rings 4–7 round-robin weightings	R/W	0x0000_0000	18.5.3.2.7/18-45
0x148–0x180	Reserved	—	—	—
0x184	TBPTR0—TxBD pointer for ring 0	R/W	0x0000_0000	18.5.3.2.8/18-46
0x188	Reserved	—	—	—
0x18C	TBPTR1*—TxBD pointer for ring 1	R/W	0x0000_0000	18.5.3.2.8/18-46
0x190	Reserved	—	—	—
0x194	TBPTR2*—TxBD pointer for ring 2	R/W	0x0000_0000	18.5.3.2.8/18-46
0x198	Reserved	—	—	—
0x19C	TBPTR3*—TxBD pointer for ring 3	R/W	0x0000_0000	18.5.3.2.8/18-46
0x1A0	Reserved	—	—	—
0x1A4	TBPTR4*—TxBD pointer for ring 4	R/W	0x0000_0000	18.5.3.2.8/18-46
0x1A8	Reserved	—	—	—
0x1AC	TBPTR5*—TxBD pointer for ring 5	R/W	0x0000_0000	18.5.3.2.8/18-46
0x1B0	Reserved	—	—	—
0x1B4	TBPTR6*—TxBD pointer for ring 6	R/W	0x0000_0000	18.5.3.2.8/18-46
0x1B8	Reserved	—	—	—

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x1BC	TBPTR7*—TxBD pointer for ring 7	R/W	0x0000_0000	18.5.3.2.8/18-46
0x1C0–0x4200	Reserved	—	—	—
0x204	TBASE0—TxBD base address of ring 0	R/W	0x0000_0000	18.5.3.2.9/18-47
0x208	Reserved	—	—	—
0x20C	TBASE1*—TxBD base address of ring 1	R/W	0x0000_0000	18.5.3.2.9/18-47
0x210	Reserved	—	—	—
0x214	TBASE2*—TxBD base address of ring 2	R/W	0x0000_0000	18.5.3.2.9/18-47
0x218	Reserved	—	—	—
0x21C	TBASE3*—TxBD base address of ring 3	R/W	0x0000_0000	18.5.3.2.9/18-47
0x220	Reserved	—	—	—
0x224	TBASE4*—TxBD base address of ring 4	R/W	0x0000_0000	18.5.3.2.9/18-47
0x228	Reserved	—	—	—
0x22C	TBASE5*—TxBD base address of ring 5	R/W	0x0000_0000	18.5.3.2.9/18-47
0x230	Reserved	—	—	—
0x234	TBASE6*—TxBD base address of ring 6	R/W	0x0000_0000	18.5.3.2.9/18-47
0x238	Reserved	—	—	—
0x23C	TBASE7*—TxBD base address of ring 7	R/W	0x0000_0000	18.5.3.2.9/18-47
0x240–0x27C	Reserved	—	—	—
0x280	TMR_TXTS1_ID*—Tx time stamp identification tag (set 1)	R/W	0x0000_0000	18.5.3.2.10/18-47
0x284	TMR_TXTS2_ID*—Tx time stamp identification tag (set 2)	R/W	0x0000_0000	18.5.3.2.10/18-47
0x288–0x2BC	Reserved	—	—	—
0x2C0	TMR_TXTS1_H*—Tx time stamp high (set 1)	R/W	0x0000_0000	18.5.3.2.11/18-48
0x2C4	TMR_TXTS1_L*—Tx time stamp high (set 1)	R/W	0x0000_0000	18.5.3.2.11/18-48
0x2C8	TMR_TXTS2_H*—Tx time stamp high (set 2)	R/W	0x0000_0000	18.5.3.2.11/18-48
0x2CC	TMR_TXTS2_L*—Tx time stamp high (set 2)	R/W	0x0000_0000	18.5.3.2.11/18-48
0x2D0–0x2FC	Reserved	—	—	—
eTSEC Receive Control and Status Registers				
0x300	RCTRL—Receive control register	R/W	0x0000_0000	18.5.3.3.1/18-48

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x304	RSTAT—Receive status register	w1c	0x0000_0000	18.5.3.3.2/18-50
0x308–0x30C	Reserved	—	—	—
0x310	RXIC—Receive interrupt coalescing register	R/W	0x0000_0000	18.5.3.3.3/18-52
0x314	RQUEUE*—Receive queue control register.	R/W	0x0080_0080	18.5.3.3.4/18-53
0x318–0x32C	Reserved	—	—	—
0x330	RBIFX*—Receive bit field extract control register	R/W	0x0000_0000	18.5.3.3.5/18-54
0x334	RQFAR*—Receive queue filing table address register	R/W	0x0000_0000	18.5.3.3.6/18-56
0x338	RQFCR*—Receive queue filing table control register	R/W	0x*nnnn_nnnn*	18.5.3.3.7/18-56
0x33C	RQFPR*—Receive queue filing table property register	R/W	0x*nnnn_nnnn*	18.5.3.3.8/18-57
0x340	MRBLR—Maximum receive buffer length register	R/W	0x0000_0000	18.5.3.3.9/18-61
0x344–0x380	Reserved	—	—	—
0x384	RBPTR0—RxBD pointer for ring 0	R/W	0x0000_0000	18.5.3.3.10/18-62
0x388	Reserved	—	—	—
0x38C	RBPTR1*—RxBD pointer for ring 1	R/W	0x0000_0000	18.5.3.3.10/18-62
0x390	Reserved	—	—	—
0x394	RBPTR2*—RxBD pointer for ring 2	R/W	0x0000_0000	18.5.3.3.10/18-62
0x398	Reserved	—	—	—
0x39C	RBPTR3*—RxBD pointer for ring 3	R/W	0x0000_0000	18.5.3.3.10/18-62
0x3A0	Reserved	—	—	—
0x3A4	RBPTR4*—RxBD pointer for ring 4	R/W	0x0000_0000	18.5.3.3.10/18-62
0x3A8	Reserved	—	—	—
0x3AC	RBPTR5*—RxBD pointer for ring 5	R/W	0x0000_0000	18.5.3.3.10/18-62
0x3B0	Reserved	—	—	—
0x3B4	RBPTR6*—RxBD pointer for ring 6	R/W	0x0000_0000	18.5.3.3.10/18-62
0x3B8	Reserved	—	—	—
0x3BC	RBPTR7*—RxBD pointer for ring 7	R/W	0x0000_0000	18.5.3.3.10/18-62
0x3C0–0x400	Reserved	—	—	—
0x404	RBASE0—RxBD base address of ring 0	R/W	0x0000_0000	18.5.3.3.11/18-62

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x408	Reserved	—	—	—
0x40C	RBASE1*—RxBD base address of ring 1	R/W	0x0000_0000	18.5.3.3.11/18-62
0x410	Reserved	—	—	—
0x414	RBASE2*—RxBD base address of ring 2	R/W	0x0000_0000	18.5.3.3.11/18-62
0x418	Reserved	—	—	—
0x41C	RBASE3*—RxBD base address of ring 3	R/W	0x0000_0000	18.5.3.3.11/18-62
0x420	Reserved	—	—	—
0x424	RBASE4*—RxBD base address of ring 4	R/W	0x0000_0000	18.5.3.3.11/18-62
0x428	Reserved	—	—	—
0x42C	RBASE5*—RxBD base address of ring 5	R/W	0x0000_0000	18.5.3.3.11/18-62
0x430	Reserved	—	—	—
0x434	RBASE6*—RxBD base address of ring 6	R/W	0x0000_0000	18.5.3.3.11/18-62
0x438	Reserved	—	—	—
0x43C	RBASE7*—RxBD base address of ring 7	R/W	0x0000_0000	18.5.3.3.11/18-62
0x440– 0x4BC	Reserved	—	—	—
0x4C0	TMR_RXTS_H*—Rx timer time stamp register high	R/W	0x0000_0000	18.5.3.3.12/18-63
0x4C4	TMR_RXTS_L*—Rx timer time stamp register low	R/W	0x0000_0000	18.5.3.3.12/18-63
0x4C8– 0x4FC	Reserved	—	—	—
eTSEC MAC Registers				
0x500	MACCFG1—MAC configuration register 1	R/W	0x0000_0000	18.5.3.5.1/18-66
0x504	MACCFG2—MAC configuration register 2	R/W	0x0000_7000	18.5.3.5.2/18-68
0x508	IPGIFG—Inter-packet/inter-frame gap register	R/W	0x4060_5060	18.5.3.5.3/18-70
0x50C	HAFDUP—Half-duplex control	R/W	0x00A1_F037	18.5.3.5.4/18-71
0x510	MAXFRM—Maximum frame length	R/W	0x0000_0600	18.5.3.5.5/18-72
0x514– 0x51C	Reserved	—	—	—
0x520	MIIMCFG—MII management configuration	R/W	0x0000_0007	18.5.3.5.6/18-72
0x524	MIIMCOM—MII management command	R/W	0x0000_0000	18.5.3.5.7/18-73
0x528	MIIMADD—MII management address	R/W	0x0000_0000	18.5.3.5.8/18-74
0x52C	MIIMCON—MII management control	WO	0x0000_0000	18.5.3.5.9/18-74

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x530	MIIMSTAT—MII management status	R	0x0000_0000	18.5.3.5.10/18-75
0x534	MIIMIND—MII management indicator	R	0x0000_0000	18.5.3.5.11/18-75
0x538	Reserved	—	—	—
0x53C	IFSTAT—Interface status	R	0x0000_0000	18.5.3.5.12/18-76
0x540	MACSTNADDR1—MAC station address register 1	R/W	0x0000_0000	18.5.3.5.13/18-76
0x544	MACSTNADDR2—MAC station address register 2	R/W	0x0000_0000	18.5.3.5.14/18-77
0x548	MAC01ADDR1*—MAC exact match address 1, part 1	R/W	0x0000_0000	18.5.3.5.15/18-78 18.5.3.5.16/18-78
0x54C	MAC01ADDR2*—MAC exact match address 1, part 2	R/W	0x0000_0000	
0x550	MAC02ADDR1*—MAC exact match address 2, part 1	R/W	0x0000_0000	
0x554	MAC02ADDR2*—MAC exact match address 2, part 2	R/W	0x0000_0000	
0x558	MAC03ADDR1*—MAC exact match address 3, part 1	R/W	0x0000_0000	
0x55C	MAC03ADDR2*—MAC exact match address 3, part 2	R/W	0x0000_0000	
0x560	MAC04ADDR1*—MAC exact match address 4, part 1	R/W	0x0000_0000	
0x564	MAC04ADDR2*—MAC exact match address 4, part 2	R/W	0x0000_0000	
0x568	MAC05ADDR1*—MAC exact match address 5, part 1	R/W	0x0000_0000	
0x56C	MAC05ADDR2*—MAC exact match address 5, part 2	R/W	0x0000_0000	

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x570	MAC06ADDR1*—MAC exact match address 6, part 1	R/W	0x0000_0000	18.5.3.5.15/18-78 18.5.3.5.16/18-78
0x574	MAC06ADDR2*—MAC exact match address 6, part 2	R/W	0x0000_0000	
0x578	MAC07ADDR1*—MAC exact match address 7, part 1	R/W	0x0000_0000	
0x57C	MAC07ADDR2*—MAC exact match address 7, part 2	R/W	0x0000_0000	
0x580	MAC08ADDR1*—MAC exact match address 8, part 1	R/W	0x0000_0000	
0x584	MAC08ADDR2*—MAC exact match address 8, part 2	R/W	0x0000_0000	
0x588	MAC09ADDR1*—MAC exact match address 9, part 1	R/W	0x0000_0000	
0x58C	MAC09ADDR2*—MAC exact match address 9, part 2	R/W	0x0000_0000	
0x590	MAC10ADDR1*—MAC exact match address 10, part 1	R/W	0x0000_0000	
0x594	MAC10ADDR2*—MAC exact match address 10, part 2	R/W	0x0000_0000	
0x598	MAC11ADDR1*—MAC exact match address 11, part 1	R/W	0x0000_0000	
0x59C	MAC11ADDR2*—MAC exact match address 11, part 2	R/W	0x0000_0000	
0x5A0	MAC12ADDR1*—MAC exact match address 12, part 1	R/W	0x0000_0000	
0x5A4	MAC12ADDR2*—MAC exact match address 12, part 2	R/W	0x0000_0000	
0x5A8	MAC13ADDR1*—MAC exact match address 13, part 1	R/W	0x0000_0000	
0x5AC	MAC13ADDR2*—MAC exact match address 13, part 2	R/W	0x0000_0000	
0x5B0	MAC14ADDR1*—MAC exact match address 14, part 1	R/W	0x0000_0000	
0x5B4	MAC14ADDR2*—MAC exact match address 14, part 2	R/W	0x0000_0000	
0x5B8	MAC15ADDR1*—MAC exact match address 15, part 1	R/W	0x0000_0000	
0x5BC	MAC15ADDR2*—MAC exact match address 15, part 2	R/W	0x0000_0000	
0x5C0– 0x67C	Reserved	—	—	—
eTSEC Transmit and Receive Counters				
0x680	TR64—Transmit and receive 64-byte frame counter	R/W	0x0000_0000	18.5.3.6.1/18-79
0x684	TR127—Transmit and receive 65- to 127-byte frame counter	R/W	0x0000_0000	18.5.3.6.2/18-80
0x688	TR255—Transmit and receive 128- to 255-byte frame counter	R/W	0x0000_0000	18.5.3.6.3/18-80
0x68C	TR511—Transmit and receive 256- to 511-byte frame counter	R/W	0x0000_0000	18.5.3.6.4/18-81
0x690	TR1K—Transmit and receive 512- to 1023-byte frame counter	R/W	0x0000_0000	18.5.3.6.5/18-81
0x694	TRMAX—Transmit and receive 1024- to 1518-byte frame counter	R/W	0x0000_0000	18.5.3.6.6/18-82

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x698	TRMGV—Transmit and receive 1519- to 1522-byte good VLAN frame count	R/W	0x0000_0000	18.5.3.6.7/18-82
eTSEC Receive Counters				
0x69C	RBYT—Receive byte counter	R/W	0x0000_0000	18.5.3.6.8/18-83
0x6A0	RPKT—Receive packet counter	R/W	0x0000_0000	18.5.3.6.9/18-83
0x6A4	RFCS—Receive FCS error counter	R/W	0x0000_0000	18.5.3.6.10/18-83
0x6A8	RMCA—Receive multicast packet counter	R/W	0x0000_0000	18.5.3.6.11/18-84
0x6AC	RBCA—Receive broadcast packet counter	R/W	0x0000_0000	18.5.3.6.12/18-84
0x6B0	RXCF—Receive control frame packet counter	R/W	0x0000_0000	18.5.3.6.13/18-85
0x6B4	RXPF—Receive PAUSE frame packet counter	R/W	0x0000_0000	18.5.3.6.14/18-85
0x6B8	RXUO—Receive unknown OP code counter	R/W	0x0000_0000	18.5.3.6.15/18-86
0x6BC	RALN—Receive alignment error counter	R/W	0x0000_0000	18.5.3.6.16/18-86
0x6C0	RFLR—Receive frame length error counter	R/W	0x0000_0000	18.5.3.6.17/18-87
0x6C4	RCDE—Receive code error counter	R/W	0x0000_0000	18.5.3.6.18/18-87
0x6C8	RCSE—Receive carrier sense error counter	R/W	0x0000_0000	18.5.3.6.19/18-88
0x6CC	RUND—Receive undersize packet counter	R/W	0x0000_0000	18.5.3.6.20/18-88
0x6D0	ROVR—Receive oversize packet counter	R/W	0x0000_0000	18.5.3.6.21/18-89
0x6D4	RFRG—Receive fragments counter	R/W	0x0000_0000	18.5.3.6.22/18-89
0x6D8	RJBR—Receive jabber counter	R/W	0x0000_0000	18.5.3.6.23/18-90
0x6DC	RDRP—Receive drop counter	R/W	0x0000_0000	18.5.3.6.24/18-90
eTSEC Transmit Counters				
0x6E0	TBYT—Transmit byte counter	R/W	0x0000_0000	18.5.3.6.25/18-91
0x6E4	TPKT—Transmit packet counter	R/W	0x0000_0000	18.5.3.6.26/18-91
0x6E8	TMCA—Transmit multicast packet counter	R/W	0x0000_0000	18.5.3.6.27/18-92
0x6EC	TBCA—Transmit broadcast packet counter	R/W	0x0000_0000	18.5.3.6.28/18-92
0x6F0	TXPF—Transmit PAUSE control frame counter	R/W	0x0000_0000	18.5.3.6.29/18-93
0x6F4	TDFR—Transmit deferral packet counter	R/W	0x0000_0000	18.5.3.6.30/18-93
0x6F8	TEDF—Transmit excessive deferral packet counter	R/W	0x0000_0000	18.5.3.6.31/18-94
0x6FC	TSCL—Transmit single collision packet counter	R/W	0x0000_0000	18.5.3.6.32/18-94
0x700	TMCL—Transmit multiple collision packet counter	R/W	0x0000_0000	18.5.3.6.33/18-95
0x704	TLCL—Transmit late collision packet counter	R/W	0x0000_0000	18.5.3.6.34/18-95

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x708	TXCL—Transmit excessive collision packet counter	R/W	0x0000_0000	18.5.3.6.35/18-96
0x70C	TNCL—Transmit total collision counter	R/W	0x0000_0000	18.5.3.6.36/18-96
0x710	Reserved	—	—	—
0x714	TDRP—Transmit drop frame counter	R/W	0x0000_0000	18.5.3.6.37/18-97
0x718	TJBR—Transmit jabber frame counter	R/W	0x0000_0000	18.5.3.6.38/18-97
0x71C	TFCS—Transmit FCS error counter	R/W	0x0000_0000	18.5.3.6.39/18-98
0x720	TXCF—Transmit control frame counter	R/W	0x0000_0000	18.5.3.6.40/18-98
0x724	TOVR—Transmit oversize frame counter	R/W	0x0000_0000	18.5.3.6.41/18-99
0x728	TUND—Transmit undersize frame counter	R/W	0x0000_0000	18.5.3.6.42/18-99
0x72C	TFRG—Transmit fragments frame counter	R/W	0x0000_0000	18.5.3.6.43/18-100
eTSEC Counter Control and TOE Statistics Registers				
0x730	CAR1—Carry register one register[2]	R	0x0000_0000	18.5.3.6.44/18-100
0x734	CAR2—Carry register two register [2]	R	0x0000_0000	18.5.3.6.45/18-101
0x738	CAM1—Carry register one mask register	R/W	0xFE03_FFFF	18.5.3.6.46/18-103
0x73C	CAM2—Carry register two mask register	R/W	0x000F_FFFD	18.5.3.6.47/18-104
0x740	RREJ*—Receive filer rejected packet counter	R/W	0x0000_0000	18.5.3.6.48/18-105
0x744–0x7FC	Reserved	—	—	—
Hash Function Registers				
0x800	IGADDR0—Individual/group address register 0	R/W	0x0000_0000	18.5.3.7.1/18-106
0x804	IGADDR1—Individual/group address register 1	R/W	0x0000_0000	
0x808	IGADDR2—Individual/group address register 2	R/W	0x0000_0000	
0x80C	IGADDR3—Individual/group address register 3	R/W	0x0000_0000	
0x810	IGADDR4—Individual/group address register 4	R/W	0x0000_0000	
0x814	IGADDR5—Individual/group address register 5	R/W	0x0000_0000	
0x818	IGADDR6—Individual/group address register 6	R/W	0x0000_0000	
0x81C	IGADDR7—Individual/group address register 7	R/W	0x0000_0000	
0x820–0x87C	Reserved	—	—	—

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x880	GADDR0—Group address register 0	R/W	0x0000_0000	18.5.3.7.2/18-106
0x884	GADDR1—Group address register 1	R/W	0x0000_0000	
0x888	GADDR2—Group address register 2	R/W	0x0000_0000	
0x88C	GADDR3—Group address register 3	R/W	0x0000_0000	
0x890	GADDR4—Group address register 4	R/W	0x0000_0000	
0x894	GADDR5—Group address register 5	R/W	0x0000_0000	
0x898	GADDR6—Group address register 6	R/W	0x0000_0000	
0x89C	GADDR7—Group address register 7	R/W	0x0000_0000	
0x8A0–0xAFC	Reserved	—	—	—
eTSEC DMA Attribute Registers				
0xB00–0xBF4	Reserved	—	—	—
0xBF8	ATTR—Attribute register	R/W	0x0000_0000	18.5.3.8.1/18-107
eTSEC Future Expansion Space				
–0xD94	Reserved	—	—	—
eTSEC IEEE 1588 Registers				
0xE00	TMR_CTRL*—Timer control register	R/W	0x0001_0000	18.5.3.9.1/18-108
0xE04	TMR_TEVENT*—time stamp event register	W1C	0x0000_0000	18.5.3.9.2/18-110
0xE08	TMR_TEMASK*—Timer event mask register	R/W	0x0000_0000	18.5.3.9.3/18-112
0xE0C	TMR_PEVENT*—time stamp event register	R/W	0x0000_0000	18.5.3.9.4/18-112
0xE10	TMR_PEMASK*—Timer event mask register	R/W	0x0000_0000	18.5.3.9.5/18-113
0xE14	TMR_STAT*—time stamp status register	R/W	0x0000_0000	18.5.3.9.6/18-114
0xE18	TMR_CNT_H*—timer counter high register	R/W	0x0000_0000	18.5.3.9.7/18-114
0xE1C	TMR_CNT_L*—timer counter low register	R/W	0x0000_0000	18.5.3.9.7/18-114
0xE20	TMR_ADD*—Timer drift compensation addend register	R/W	0x0000_0000	18.5.3.9.8/18-115
0xE24	TMR_ACC*—Timer accumulator register	R/W	0x0000_0000	18.5.3.9.9/18-116
0xE28	TMR_PRSC* -Timer prescale	R/W	0x0000_0002	18.5.3.9.10/18-116
0xE2C	Reserved	—	—	—
0xE30	TMROFF_H*—Timer offset high	R/W	0x0000_0000	18.5.3.9.11/18-117
0xE34	TMROFF_L*—Timer offset low	R/W	0x0000_0000	18.5.3.9.11/18-117

Table A-25. Enhanced Three-Speed Ethernet Controllers (eTSEC) Registers (continued)

eTSEC 1—Block Base Address 0x2_4000 eTSEC 2—Block Base Address 0x2_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xE40	TMR_ALARM1_H*—Timer alarm 1 high register	R/W	0xFFFF_FFFF	18.5.3.9.12/18-117
0xE44	TMR_ALARM1_L*—Timer alarm 1 high register	R/W	0xFFFF_FFFF	
0xE48	TMR_ALARM2_H*—Timer alarm 2 high register	R/W	0xFFFF_FFFF	
0xE4C	TMR_ALARM2_L*—Timer alarm 2 high register	R/W	0xFFFF_FFFF	
0xE50– 0xE7C	Reserved	—	—	—
0xE80	TMR_FIPER1*—Timer fixed period interval	R/W	0xFFFF_FFFF	18.5.3.9.13/18-118
0xE84	TMR_FIPER2*—Timer fixed period interval	R/W	0xFFFF_FFFF	
0xE88	TMR_FIPER*3—Timer fixed period interval	R/W	0xFFFF_FFFF	
0xEA0	TMR_ETTS1_H*—Time stamp of general purpose external trigger	R/W	0x0000_0000	18.5.3.9.14/18-119
0xEA4	TMR_ETTS1_L*—Time stamp of general purpose external trigger	R/W	0x0000_0000	
0xEA8	TMR_ETTS2_H*—Time stamp of general purpose external trigger	R/W	0x0000_0000	
0xEAC	TMR_ETTS2_L*—Time stamp of general purpose external trigger	R/W	0x0000_0000	
0xEB0– 0xFFF	Reserved	—	—	

[1] Reset value of ENCTRL is configured from the value of RCWH[TSEC*n*M] which is loaded during reset (See Section 4.3.2.2, "Reset Configuration Word High Register (RCWHR)")

[2] Cleared on read.

A.26 Enhanced Secure Digital Host Controller (eSDHC)

Table A-26. Enhanced Secure Digital Host Controller (eSDHC) Registers

eSDHC Registers—Block Base Address: 0x2_E000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	DMA system address (DSADDR)	R/W	0x0000_0008	11.4.1/11-5
0x004	Block attributes (BLKATTR)	R/W	0x0000_0008	11.4.2/11-6
0x008	Command argument (CMDARG)	R/W	0x0000_0000	11.4.3/11-7
0x00C	Command transfer type (XFERTYP)	R/W	0x0000_0000	11.4.4/11-7
0x010	Command response0 (CMDRSP0)	R	0x0000_0000	11.4.5/11-10
0x014	Command response1 (CMDRSP1)	R	0x0000_0000	11.4.5/11-10

Table A-26. Enhanced Secure Digital Host Controller (eSDHC) Registers (continued)

eSDHC Registers—Block Base Address: 0x2_E000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x018	Command response2 (CMDRSP2)	R	0x0000_0000	11.4.5/11-10
0x01C	Command response3 (CMDRSP3)	R	0x0000_0000	11.4.5/11-10
0x020	Data buffer access port (DATPORT)	R/W	0x0000_0000	11.4.6/11-11
0x024	Present state (PRSSTAT)	R	0xFF80_0000	11.4.7/11-12
0x028	Protocol control (PROCTL)	R/W	0x0000_0000	11.4.8/11-16
0x02C	System control (SYSCTL)	Mixed	0x0000_8000	11.4.9/11-18
0x030	Interrupt status (IRQSTAT)	w1c	0x0000_0000	11.4.10/11-21
0x034	Interrupt status enable (IRQSTATEN)	R/W	0x117F_013F	11.4.11/11-25
0x038	Interrupt signal enable (IRQSIGEN)	R/W	0x0000_0000	11.4.12/11-27
0x03C	Auto CMD12 status (AUTOC12ERR)	R	0x0000_0000	11.4.13/11-29
0x040	Host controller capabilities (HOSTCAPBLT)	R	0x01E3_0000	11.4.14/11-31
0x044[1]	Watermark level (WML)	R/W	0x0010_0010	11.4.15/11-32
0x050	Force event (FEVT)	W	0x0000_0000	11.4.16/11-32
0x0FC	Host controller version (HOSTVER)	R	0x0000_0001	11.4.17/11-34
0x40C	DMA control register (DCR)	R/W	0x0000_0000	11.4.18/11-35

[1] The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

A.27 Security Engine Controller (SEC)

Table A-27. Security Engine Controller (SEC) Registers

Security Engine Controller (SEC)—Block Base Address 0x3_0000					
Byte Offset (AD 17–0)	**SEC Module**	**Register**	**Access**	**Write By**	**Section/Page**
0x3_0100–0x3_01FF	Channel_1	Alternate location for Channel 1	See the Channel regions starting at address 0x3_1108 and the RCA bits in Table 17-21		
0x3_0200–0x3_02FF	Channel_2	Alternate location for Channel 2			
0x3_0300–0x3_03FF	Channel_3	Alternate location for Channel 3			
0x3_0400–0x3_04FF	Channel_4	Alternate location for Channel 4			

Table A-27. Security Engine Controller (SEC) Registers (continued)

Security Engine Controller (SEC)—Block Base Address 0x3_0000					
Byte Offset (AD 17–0)	**SEC Module**	**Register**	**Access**	**Write By**	**Section/Page**
0x3_1008	Controller	Interrupt enable	R/W	byte[1]	17.5.4.2/1717-50
0x3_1010		Interrupt status	R	—	17.5.4.2/1717-53
0x3_1018		Interrupt clear	R/W	byte	17.5.4.3/1717-54
0x3_1020		Identification	R	—	17.5.4.4/1717-54
0x3_1028		EU assignment status	R	—	17.5.4.1/1717-50
0x3_1030		Master control	R/W	byte	17.5.4.6/1717-55
0x3_1108	Channel_1	Configuration register	R/W	word	17.4.4.1/1717-37
0x3_1110		Pointer status	R/W	word	17.4.4.2/1717-41
0x3_1140		Current descriptor pointer	R	—	17.4.4.3/1717-43
0x3_1148		Fetch FIFO	W	word	17.4.4.4/1717-44
0x3_1180–0x3_11BF		Descriptor buffer	R	—	17.4.5.1/1717-45
0x3_11C0–0x3_11DF		Gather Link Table	R	—	17.4.5.2/1717-45
0x3_11E0–0x3_11FF		Scatter Link Table	R	—	17.4.5.2/1717-45
0x3_1208	Channel_2	Configuration register	R/W	word	17.4.4.1/1717-37
0x3_1210		Pointer status	R/W	word	17.4.4.2/1717-41
0x3_1240		Current descriptor pointer	R	—	17.4.4.3/1717-43
0x3_1248		Fetch FIFO	W	word	17.4.4.4/1717-44
0x3_1280–0x3_12BF		Descriptor buffer	R	—	17.4.5.1/1717-45
0x3_12C0–0x3_12DF		Gather Link Table	R	—	17.4.5.2/1717-45
0x3_12E0–0x3_12FF		Scatter Link Table	R	—	17.4.5.2/1717-45
0x3_1308	Channel_3	Configuration register	R/W	word	17.4.4.1/1717-37
0x3_1310		Pointer status	R/W	word	17.4.4.2/1717-41
0x3_1340		Current descriptor pointer	R	—	17.4.4.3/1717-43
0x3_1348		Fetch FIFO	W	word	17.4.4.4/1717-44
0x3_1380–0x3_13BF		Descriptor buffer	R	—	17.4.5.1/1717-45
0x3_13C0–0x3_13DF		Gather Link Table	R	—	17.4.5.2/1717-45
0x3_13E0–0x3_13FF		Scatter Link Table	R	—	17.4.5.2/1717-45

Table A-27. Security Engine Controller (SEC) Registers (continued)

Security Engine Controller (SEC)—Block Base Address 0x3_0000					
Byte Offset (AD 17–0)	**SEC Module**	**Register**	**Access**	**Write By**	**Section/Page**
0x3_1408	Channel_4	Configuration register	R/W	word	17.4.4.1/1717-37
0x3_1410		Pointer status	R/W	word	17.4.4.2/1717-41
0x3_1440		Current descriptor pointer	R	—	17.4.4.3/1717-43
0x3_1448		Fetch FIFO	W	word	17.4.4.4/1717-44
0x3_1480–0x3_14BF		Descriptor buffer	R	—	17.4.5.1/1717-45
0x3_14C0–0x3_14DF		Gather Link Table	R	—	17.4.5.2/1717-45
0x3_14E0–0x3_14FF		Scatter Link Table	R	—	17.4.5.2/1717-45
0x3_1500	Poly-Channel	Fetch FIFO Enqueue Count	R/W	word	17.4.3.1/1717-35
0x3_1508		Descriptor Finished Count	R/W	word	17.4.3.1/1717-36
0x3_1510		Data Bytes In Count	R/W	word	17.4.3.1/1717-36
0x3_1518		Data Bytes Out Count	R/W	word	17.4.3.1/1717-37
0x3_1BF8	Controller	IP block revision	R	—	17.5.4.5/1717-54
0x3_2000	DEU	Mode register	R/W	word	17.7.4.1/1717-108
0x3_2008		Key size register	R/W	word	17.7.4.2/1717-109
0x3_2010		Data size register	R/W	word	17.7.4.3/1717-110
0x3_2018		Reset control register	R/W	word	17.7.4.4/1717-110
0x3_2028		Status register	R	—	17.7.4.5/1717-111
0x3_2030		Interrupt status register	R/W	word	17.7.4.6/1717-112
0x3_2038		Interrupt mask register	R/W	word	17.7.4.7/1717-114
0x3_2050		EU-Go	W	word	17.7.4.8/1717-116
0x3_2100		IV register	R/W	word	17.7.4.9/1717-116
0x3_2400		Key 1 register	W	byte	17.7.4.10/1717-116
0x3_2408		Key 2 register	W	byte	17.7.4.10/1717-116
0x3_2410		Key 3 register	W	byte	17.7.4.10/1717-116
0x3_2800–0x3_2FFF		Input FIFO / Output FIFO	R/W[2]	byte	17.7.4.11/1717-116

Table A-27. Security Engine Controller (SEC) Registers (continued)

Security Engine Controller (SEC)—Block Base Address 0x3_0000					
Byte Offset (AD 17–0)	**SEC Module**	**Register**	**Access**	**Write By**	**Section/Page**
0x3_4000	AESU	Mode register	R/W	word	17.7.1.2/1717-58
0x3_4008		Key size register	R/W	word	17.7.1.3/1717-62
0x3_4010		Data size register	R/W	word	17.7.1.4/1717-62
0x3_4018		Reset control register	R/W	word	17.7.1.5/1717-63
0x3_4028		Status register	R	—	17.7.1.6/1717-63
0x3_4030		Interrupt status register	R/W	word	17.7.1.7/1717-64
0x3_4038		Interrupt mask register	R/W	word	17.7.1.8/1717-66
0x3_4040		ICV size register	R/W	word	17.7.1.9/1717-68
0x3_4050		End of message register	W	word	17.7.1.10/1717-68
0x3_4100–0x3_415F		Context	R/W	byte	17.7.1.11/1717-69
0x3_4400–0x3_441F		Key registers	R/W	byte	17.7.1.12/1717-87
0x3_4800–0x3_4FFF		Input FIFO / Output FIFO	R/W [1]	byte	17.7.1.12/1717-88
0x3_6000	MDEU	Mode register	R/W	word	17.7.6.2/1717-132
0x3_6008		Key size register	R/W	word	17.7.6.4/1717-136
0x3_6010		Data size register	R/W	word	17.7.6.5/1717-136
0x3_6018		Reset control register	R/W	word	17.7.6.6/1717-137
0x3_6028		Status register	R	—	17.7.6.7/1717-137
0x3_6030		Interrupt status register	R/W	word	17.7.6.8/1717-139
0x3_6038		Interrupt mask register	R/W	word	17.7.6.9/1717-140
0x3_6040		ICV size register	W	word	17.7.6.10/1717-141
0x3_6050		End of message reigster	W	word	17.7.6.11/1717-142
0x3_6100–0x3_6147		Context registers	R/W	byte	17.7.6.12/1717-142
0x3_6400–0x3_647F		Key registers	W	byte	17.7.6.13/1717-145
0x3_6800–0x3_6FFF		Input FIFO	W [1]	byte	17.7.6.14/1717-145

Table A-27. Security Engine Controller (SEC) Registers (continued)

Security Engine Controller (SEC)—Block Base Address 0x3_0000					
Byte Offset (AD 17–0)	**SEC Module**	**Register**	**Access**	**Write By**	**Section/Page**
0x3_8000	AFEU	Mode register	R/W	word	17.7.2.1/1717-89
0x3_8008		Key size register	R/W	word	17.7.2.2/1717-89
0x3_8010		Data size register	R/W	word	17.7.2.3/1717-90
0x3_8018		Reset control register	R/W	word	17.7.2.4/1717-91
0x3_8028		Status register	R	—	17.7.2.5/1717-91
0x3_8030		Interrupt status register	R/W	word	17.7.2.6/1717-92
0x3_8038		Interrupt mask register	R/W	word	17.7.2.7/1717-94
0x3_8050		End of message register	W	word	17.7.2.8/1717-96
0x3_8100–0x3_81FF		Context memory	R/W	byte	17.7.2.9/1717-96
0x3_8200		Context memory pointers	R/W	byte	17.7.2.9/1717-96
0x3_8400–0x3_840F		Key registers	W	byte	17.7.2.10/1717-97
0x3_8800–0x3_8FFF (3_8E00)		Input FIFO / Output FIFO (special context address)	R/W [1]	byte	17.7.2.10/1717-97
0x3_A000	RNGU	Mode register	R/W	word	17.7.8.1/1717-155
0x3_A010		Data size register	R/W	word	17.7.8.2/1717-155
0x3_A018		Reset control register	R/W	word	17.7.8.3/1717-155
0x3_A028		Status register	R	—	17.7.8.4/1717-156
0x3_A030		Interrupt status register	R/W	word	17.7.8.5/1717-157
0x3_A038		Interrupt mask register	R/W	word	17.7.8.6/1717-158
0x3_A050		End of message register	W	word	17.7.8.7/1717-159
0x3_A400–0x3_A43F		Entropy registers	W	word	17.7.8.8/1717-160
0x3_A800–0x3_AFFF		Output FIFO	R [1]	—	17.7.8.8/1717-160

Table A-27. Security Engine Controller (SEC) Registers (continued)

Security Engine Controller (SEC)—Block Base Address 0x3_0000					
Byte Offset (AD 17–0)	**SEC Module**	**Register**	**Access**	**Write By**	**Section/Page**
0x3_C000	PKEU	Mode register	R/W	word	17.7.7.1/1717-146
0x3_C008		Key size register	R/W	word	17.7.7.2/1717-146
0x3_C010		Data size register	R/W	word	17.7.7.4/1717-148
0x3_C018		Reset control register	R/W	word	17.7.7.5/1717-148
0x3_C028		Status register	R	—	17.7.7.6/1717-149
0x3_C030		Interrupt status register	R/W	word	17.7.7.7/1717-150
0x3_C038		Interrupt mask register	R/W	word	17.7.7.8/1717-152
0x3_C040		ABSize	R/W	word	17.7.7.3/1717-147
0x3_C050		End of message register	W	word	17.7.7.9/1717-153
0x3_C200–0x3_C27F		Parameter memory A0	R/W	byte	17.7.7.10/1717-153
0x3_C280–0x3_C2FF		Parameter memory A1	R/W	byte	
0x3_C300–0x3_C37F		Parameter memory A2	R/W	byte	
0x3_C380–0x3_C3FF		Parameter memory A3	R/W	byte	
0x3_C400–0x3_C47F		Parameter memory B0	R/W	byte	
0x3_C480–0x3_C4FF		Parameter memory B1	R/W	byte	
0x3_C500–0x3_C57F		Parameter memory B2	R/W	byte	
0x3_C580–0x3_C5FF		Parameter memory B3	R/W	byte	
0x3_C800–0x3_C9FF		Parameter memory N	R/W	byte	
0x3_CA00–0x3_CBFF		Parameter memory E	W	byte	

Table A-27. Security Engine Controller (SEC) Registers (continued)

Security Engine Controller (SEC)—Block Base Address 0x3_0000					
Byte Offset (AD 17–0)	**SEC Module**	**Register**	**Access**	**Write By**	**Section/Page**
0x3_E000	KEU	Mode register	R/W	word	17.7.5.1/1717-118
0x3_E008		Key size register	R/W	word	17.7.5.2/1717-119
0x3_E010		Data size register	R/W	word	17.7.5.3/1717-119
0x3_E018		Reset control register	R/W	word	17.7.5.4/1717-121
0x3_E028		Status register	R	—	17.7.5.5/1717-122
0x3_E030		Interrupt Status register	R/W	word	17.7.5.6/1717-123
0x3_E038		Interrupt Mask register	R/W	word	17.7.5.7/1717-125
0x3_E048		Data out register (f9 MAC)	R	—	17.7.5.8/1717-126
0x3_E050		End of message register	W	word	17.7.5.9/1717-127
0x3_E100		IV_1 register	R/W	byte	17.7.5.10/1717-127
0x3_E108		ICV_In register	R/W	byte	17.7.5.11/1717-128
0x3_E110		IV_2 register (FRESH)	R/W	byte	17.7.5.12/1717-128
0x3_E118		Context_1 register	R/W	byte	17.7.5.13/1717-129
0x3_E120		Context_2 register	R/W	byte	17.7.5.13/1717-129
0x3_E128		Context_3 register	R/W	byte	17.7.5.13/1717-129
0x3_E130		Context_4 register	R/W	byte	17.7.5.13/1717-129
0x3_E138		Context_5 register	R/W	byte	17.7.5.13/1717-129
0x3_E140		Context_6 register	R/W	byte	17.7.5.13/1717-129
0x3_E400		Key data register_1 (CK-high)	R/W	byte	17.7.5.14/1717-129
0x3_E408		Key data register_2 (CK-low)	R/W	byte	17.7.5.14/1717-129
0x3_E410		Key dataregister_3 (IK-high)	R/W	byte	17.7.5.15/1717-130
0x3_E418		Key data register_4 (IK-low)	R/W	byte	17.7.5.15/1717-130
0x3_E800–0x3_EFFF		Input FIFO / Output FIFO	R/W [1]	byte	17.7.5.16/1717-130

Table A-27. Security Engine Controller (SEC) Registers (continued)

Security Engine Controller (SEC)—Block Base Address 0x3_0000					
Byte Offset (AD 17–0)	**SEC Module**	**Register**	**Access**	**Write By**	**Section/Page**
0x3_F000	CRCU	Mode register	R/W	word	17.7.3.2/1717-98
0x3_F008		Key size register	R/W	word	17.7.3.3/1717-99
0x3_F010		Data size register	R/W	word	17.7.3.4/1717-100
0x3_F018		Reset control register	R/W	word	17.7.3.5/1717-100
0x3_F020		Control	R/W	word	17.7.3.6/1717-101
0x3_F028		Status register	R	—	17.7.3.7/1717-101
0x3_F030		Interrupt status register	R/W	word	17.7.3.8/1717-102
0x3_F038		Interrupt mask register	R/W	word	17.7.3.9/1717-104
0x3_F040		ICV size register	R/W	word	17.7.1.9/1717-68
0x3_F050		End of message register	W	word	17.7.3.11/1717-106
0x3_F108		Context register	R/W	byte	17.7.3.12/1717-106
0x3_F400		Key register	R/W	byte	17.7.3.13/1717-107
0x3_F800–0x3_FFFF		Input FIFO	W [1]	byte	17.7.3.14/1717-108

1 Byte accessibility is controlled by internal logic, particularly at FIFOs, to prevent unintended overwrites of partial words during writes, and to prevent unintended duplicate reads of partial data during reads. In addition, these bytes must be presented on the correct byte lanes for the intended destination.

2 For the EU FIFOs, write operations anywhere in the address range enqueue to the input FIFO, and read operations anywhere in the address range dequeue from the output FIFO. See the referenced section for more detailed information.

A.28 SerDes PHY

Table A-28. SerDes PHY Registers

SerDes PHY—Block Base Address 0xE_3000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	SRDS1CR0—SerDes1 Control Register 0	R/W	0x1100_CC30	19.3.1/19-5
0x004	SRDS1CR1—SerDes1 Control Register 1	R/W	0x0000_0040	19.3.2/19-9
0x008	SRDS1CR2—SerDes1 Control Register 2	R/W	0x0080_1C1C	19.3.3/19-11
0x00C	SRDS1CR3—SerDes1 Control Register 3	R/W	0x0101_0000	19.3.4/19-13
0x010	SRDS1CR4—SerDes1 Control Register 4	R/W	0x*nn*00_0*n*0*n*	19.3.5/19-16
0x014–0x01C	Reserved	—	—	—
0x020	SRDS1RSTCTL—SerDes1 Reset Control Register	R/W	0x0044_4500	19.3.6/19-17
0x024–0x0FC	Reserved	—	—	—

Table A-28. SerDes PHY Registers (continued)

SerDes PHY—Block Base Address 0xE_3000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x100	SRDS2CR0—SerDes2 Control Register 0	R/W	0x1100_CC30	19.3.1/19-5
0x104	SRDS2CR1—SerDes2 Control Register 1	R/W	0x0000_0040	19.3.2/19-9
0x108	SRDS2CR2—SerDes2 Control Register 2	R/W	0x0080_1C1C	19.3.3/19-11
0x10C	SRDS2CR3—SerDes2 Control Register 3	R/W	0x0101_0000	19.3.4/19-13
0x110	SRDS2CR4—SerDes2 Control Register 4	R/W	0x*nn*00_0*n*0*n*	19.3.5/19-16
0x114–0x11C	Reserved	—	—	—
0x120	SRDS2RSTCTL—SerDes2 Reset Control Register	R/W	0x0044_4500	19.3.6/19-17
0x124–0x1FC	Reserved	—	—	—

Appendix B
Revision History

This appendix provides a list of the major differences between the ***MPC8379E Integrated Host Processor Reference Manual***, revision 0 through revision 1.

B.1 Changes From Revision 0 to Revision 1

Major changes to the ***MPC8379E Integrated Host Processor Reference Manual***, from revision 0 to revision 1, are as follows:

Section, Page	Changes
Chapter 1/1-1	Remove references to CE-ATA features throughout chapter.
1.1/1-1	In Table 1-1, “Functionality of the MPC8379E, MPC8378E, and MPC8377E,” change MPC8379E SVR value from 80C0_0010 to 80C2_0010.
3.1/3-1	In Figure 3-2, “MPC8379E Signal Groupings (2 of 2),” and in Table 3-1, “MPC8379E Signal Reference by Functional Block,” update L1_SD_TX[0:1], L2_SD_TX[0:1], L1_SD_RX[0:1], L2_SD_RX[0:1] (and their complements) to: L1_SD_TXA/E, L2_SD_TXA/E, L1_SD_RXA/E, L2_SD_RXA/E (and their complements). In addition, remove signal SATA_CLK_IN.
4.3.3.1/4-23	Add the following text to the end of the first paragraph: “+ LCS0 is the default for GPCM, so GPCM controlled is used to read the reset configuration word from EEPROM. /LGTA should be high to avoid unintended early termination of the read cycle.”
4.3.3.1.1/4-24	Replace Figure 4-5, “Loading Reset Configuration Words from Local Bus,” and Figure 4-6, “Loading Reset Configuration Words from Local Bus (Continued)” with Figure 4-5.
5.3.2.5/5-24	In Table 5-32, “SICRL Bit Settings,” change DR_RX_ERROR_PWRFAULT to DR_PWRFAULT, change DR_TX_VALID_PCTL0 to DR_PCTL0, and change DR_TX_VALIDH_PCTL1 to DR_PCTL1.
5.3.2.7.1/5-30	Change “either” to say “enter,” as follows: “The DDR debug configuration enables a DDR memory controller to enter debug mode in which the DDR SDRAM source ID field and data valid strobe are driven onto one of two optional sets of pins.”
5.3.2.8/5-31	In first two paragraphs, update 18 Ω to 18.2 Ω.
5.3.2.8/5-31	In Table 5-34, “DDRCDR Field Descriptions,” update DDR_TYPE bit field description.

5.7.3/5-58	Add bullet list.
8.4.2/8-5	In Table 8-2, "IPIC External Signals—Detailed Signal Descriptions," modify the IRQ[7:0] Asserted State Meaning description as follows: "Asserted—When an external interrupt request signal is asserted, the priority is checked by the IPIC unit, and the interrupt is conditionally passed to the processor."
9.3.2.2/9-8	In Table 9-4, "Clock Signals—Detailed Signal Descriptions," update MCKE description to add the following: "The MCKE signals should be connected to the same rank of memory as the corresponding MCS and MODT signals. For example, MCKE[0] should be connected to the same rank of memory as MCS[0] and MODT[0]."
9.4.1.7/9-19	In Table 9-12, "DDR_SDRAM_CFG Field Descriptions," add new programming requirement for DDR_SDRAM_CFG[HSE] such that this bit should not be set if using automatic calibration. Also, modify note to say the following: "DDR1 (SDRAM_TYPE = 010) must use 8-beat bursts when using 32-bit bus mode (32_BE = 1) and 4-beat bursts when using 64-bit bus mode; DDR2 (SDRAM_TYPE = 011) must use 4-beat bursts, even when using 32-bit bus mode."
9.4.1.23/9-33	In Table 9-29, "CAPTURE_ECC Field Descriptions," extend bit field for ECE from 24:31 to 16:31 and modify the description, as follows: "Error capture ECC. Captures the ECC bits on the data path whenever errors are detected. 16:23—8-bit ECC code for 1st 32 bits 24:31—8-bit ECC code for 2nd 32 bits Note: In 64-bit mode, only 24:31 should be used, although 16:23 shows the 8-bit ECC code replicated."
9.4.1.27/9-36	In Table 9-33, "CAPTURE_ATTRIBUTES Field Descriptions," add the following bit field description to TSIZ: "000 4 double words 001 1 double word 010 2 double words 011 3 double words Others Reserved"
9.5.6/9-56	Add the note at end of section.
9.5.11/9-63	Add the following text to the end of the section: "In 32-bit mode, Table 9-48 is split into 2 halves. The first half, consisting of rows 0–31, is used to calculate the ECC bits for the first 32 data bits of any 64-bit granule of data. This always applies to the odd data beats on the DDR data bus. The second half of the table, consisting

	of rows 32–63, is used to calculate the ECC bits for the second 32 bits of any 64-bit granule of data. This always applies to the even data beats on the DDR data bus.”
9.6.1/9-67	In Table 9-52, “Programming Differences Between Memory Types,” for ODT_PD_EXIT, change it to be set to 0001 for DDR1; for FOUR_ACT, change it to be set for 00001 for DDR1.
Chapter 10/10-1	Remove references to atomic operations throughout chapter.
10.1.3/10-3	Add the sentence “The internal transaction address is limited to 32 bits, so all chip selects must fall within the 4-Gbyte window addressed by the internal transaction address.” Also, clarify the last sentence by changing it to the following: “Thus, with the eLBC in GPCM or FCM, or UPM mode, only one of the eight chip selects is active at any time for the duration of the transaction except in the case of UPM refresh where all UPM machines that are enabled for refresh have concurrent chip select assertion.”
10.2/10-4	In Table 10-1, “Signal Properties—Summary,” and Table 10-2, “Enhanced Local Bus Controller Detailed Signal Descriptions,” change “LA[6:31]” to “LA[7:31].” Also in Table 10-2, modify wording of LGPL*n* timing description to say its value is “driven”, rather than “drives”.
10.3.1.1/10-11	In Table 10-4, “BRn Field Descriptions,” clarify BR*n*[PS] field description with respect to FCM capabilities (only 8-bit data widths are supported with FCM).
10.3.1.2/10-12	Move Table 10-5, “Reset value of OR0 Register,” to Section 10.3.1.2.2, “Option Registers (ORn)—GPCM Mode.”
10.3.1.7/10-24	Update clarification regarding back-to-back special operations, to read as follows: “To avoid race conditions between software and a busy eLBC, registers that affect currently running special operation and LSOR must not be re-written before a pending special operation has been completed. The UPM and FCM have different indications of when such special operations are completed. The behavior of eLBC is unpredictable if special operation modes are altered between LSOR being written and the relevant memory controller completing that access.”
10.3.1.9/10-26	Change “error” to “error/events” in description of LTE registers.
10.3.1.14/10-31	In Table 10-21, “LBCR Field Descriptions,” change the LBCR[AHD] field state description as follows: “0 During address phases on the local bus, the LALE signal negates one platform clock period prior to the address being invalidated. For instance, at 33.3 MHz, this provides 3 ns of additional address hold time at the external address latch. 1 During address phases on the local bus, the LALE signal negates 0.5 platform clock period prior to the address being invalidated. This halves the address hold time, but extends the latch enable duration. This may be necessary for very high frequency designs.”

10.3.1.15/10-33	In Table 10-22, "LCRR Field Descriptions," modify PBYP field description, as follows: "PLL bypass. This bit should be set when using low bus clock frequencies (See device hardware specifications for applicable frequencies.) When in PLL bypass mode, incoming data is captured in the middle of the bus clock cycle. 0 The PLL is enabled. 1 The PLL is bypassed."
10.3.1.15/10-33	In Table 10-23, "FMR Field Descriptions," in CWTO field, change "(CW0-CW3)" to "(CW0, CW1, RBW and RSW)."
10.3.1.15/10-33	Update reset value to "*n*,*n*,0,0" for bits 27–31 in Figure 10-19, "Clock Ratio Register (LCRR)."
10.4.1.7/10-45	Update Figure 10-31 to deassert LCSn half a LCLK half cycle later.
10.4.2/10-46	In Figure 10-32, "Enhanced Local Bus to GPCM Device Interface," replace with a non-latching figure.
10.4.2/10-46	Update Figure 10-33, "GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0, CLKDIV = 4,8)."
10.4.2.1/10-47	In Table 10-30, "GPCM Read Control Signal Timing," update tARCS and tCSRP columns.
10.4.2.2/10-49	In Table 10-31, "GPCM Write Control Signal Timing," update t_{AWCS}, $t_{CSWP,}$ and t_{WC} columns.
10.4.2.3.2/10-51	Clarify dependency of Chip-Select and Write Enable Negation Timing on TRLX and CSNT by changing: "ORn[CSNT] controls the timing..." to "ORn[CSNT], along with ORn[TRLX], control the timing..."
10.4.3.4/10-72	Modify steps three and four.
10.4.3.4.1/10-72	In Table 10-36, "Boot Bank Field Values after Reset for FCM as Boot Controller," changed the setting for field SCY to read: From por_cfg_scy[1:3].
10.4.4.1.2/10-77	Clarify simultaneous chip selects during UPM refresh as follows: "Any banks assigned to a UPM are provided with the common UPMA refresh pattern if the RFEN bit of the corresponding UPM is set, concurrently. UPMA assigned banks, therefore, always receive refresh services when MAMR[RFEN] is set, while UPMB and UPMC assigned banks also receive (the same) refresh services if the corresponding MxMR[RFEN] bits are set. In this scenario, more than one chip select may assert at the same time, as refresh pattern runs for all banks assigned to UPM with RFEN bit set."
10.4.4.4.1/10-81	In the LAST and UTA field descriptions in Table 10-38, "RAM Word Field Descriptions" add the following note: "In case of UPM writes, program UTA and LAST in same RAM word.

	In case of UPM reads, program UTA and LAST in consecutive or same RAM words.”
10.4.4.4.1/10-81	In Table 10-38, “RAM Word Field Descriptions,” add the following note to fields LOOP and AMX: “AMX must not change the values in any RAM word which begins a loop.”
10.4.4.4.5/10-86	Specify a constraint that “Loop start word should not have an AMX change with regard to the previous word.”
10.4.4.4.7/10-87	Modify the last two sentences of the first paragraph, as follows: “The next address (NA) bit of the RAM word does not affect LA signals, unless AMX = 00 and chooses the column address for NA = 1.”
10.4.4.4.7/10-87	Add the following note: “AMX must not change values in any RAM word which begins a loop.”
10.4.4.4.7/10-87	Modify Table 10-40, “UPM Address Multiplexing.”
10.4.4.4.9/10-89	Add the following paragraph: “In case of UPM writes, program UTA and LAST in same RAM word. In case of UPM reads, program UTA and LAST in consecutive or same RAM words.”
10.4.4.5/10-90	Add the following text to the second paragraph: “The conditions are as follows: • The PLL must be enabled, that is, LCRR[PBYP] = 0. • DLT3 bit must be cleared in the same RAM word to avoid mid-sampling of read data. • LBCR[LPBSE] = 0 and MXMR[GPL4] = 1 The combination WAEN = 1 and UTA = 1 should be in the RAM word next to the word which gets frozen by LUPWAIT assertion. This condition limits the use of this mode to cases where the exact cycle of LUPWAIT assertion is predictable.”
10.5.1/10-91	Modify section to contain only the following sentence: “This section provides guidelines for interfacing peripherals in the various modes.”
10.5.1.1/10-91	Change sentence from: “For 8-bit devices, LA[30:31] should be used and for 16-bit devices, LA[31] should not be used. 32-bit devices use neither of these signals.” to: “For 8-bit devices, LA[30:31] should be used; for 16-bit devices, LA[30] should be used and LA[31] is unused; for 32-bit devices, LA[30:31] are not used.”
10.5.1.1/10-91	Replace section 10.5.1.2 “Non-Multiplexed Address and Data Buses for 25-Bit Addressing” with Section 10.5.1.1, “Multiplexed Address/Data Bus for 32-Bit Addressing.”
10.5.1.2/10-92	In addition, rename Figure 10-71 to “Non-Multiplexed Address and Data Buses,” and update LA[6:31] to LA[11:31].

10.5.1.3/10-92	Add section.
10.5.1.5/10-93	Change example frequency to 133 MHz.
10.5.1.5/10-93	Modify last sentence of second paragraph, as follows: "Typical values for the two propagation delays are in the order of 3–6 ns."
10.5.4.4/10-99	Change FMR[OP] = 01 to FMR[OP] = 1.
10.5.4.5/10-99	Change FMR[OP] = 01 to FMR[OP] = 1.
10.5.4.6/10-100	Change FMR[OP] = 01 to FMR[OP] = 1.
10.5.4.6/10-100	Update the first sentence in second paragraph to "Note that operations specified by OP5 and OP6 (status read) should never be skipped..."
10.5.7/10-103	Remove section "Interfacing to DSP Host Ports."
11.3/11-3	In Table 11-1, "Signal Properties," change SD_CD signal description to: "By default CDP = 0, and so the SDHC_CD signal should be tied high (meaning card present). If CDP is programmed to 1, the SDHC_CD signal should be tied low (card present, according to the standard). If not implemented, tie to a card present value according to the polarity set by the SCR[CDP]."
11.4/11-4	In Table 11-2, "eSDHC Memory Map," modify name of section and register from "System Control Register (SCR)" to "DMA Control Register (DCR)."
11.4.6/11-11	Add note saying, "When the internal DMA is not enabled and a write transaction is in operation, DATPORT must not be read."
11.4.17/11-34	In Table 11-25, "HOSTVER Field Descriptions," add "0x01 Freescale eSDHC version 2.0" to description for HOSTVER[VVN].
11.4.18/11-35	Update name of section and register from "System Control Register (SCR)" to "DMA Control Register (DCR)." Also, update values in RD_PF_SIZE field description in Table 11-26, "DCR Field Descriptions."
11.5.5/11-41	Add text saying, "Refer to SYSCTL[SDCLKFS] and SYSCTL[DVS] (see Section 21.4.14, "System Control Register (SYSCTL)") to select the divisor values."
11.5.5/11-41	In Figure 11-24, "Two Stages of Clock Divider," remove "1" from list in the first box (1st Divisor).
11.6.5/11-53	In Table 11-27, "Commands for MMC/SD," add argument field for ACMD23: "[31:23] stuff bits [22:0] number of blocks"
11.6.6/11-58	Add note saying, "When the internal DMA is not enabled and a write transaction is in operation, DATPORT must not be read."
12.4.1/12-3	In Table 12-2, "POTAR*n* Field Descriptions," revise TA field description to state that the windows must be aligned to the window size.

12.4.2/12-3	In Table 12-3, “POBAR*n* Field Descriptions,” revise BA field description to state that the windows must be aligned to the window size.
13.4.1/13-4	In Figure 13-2, “Outbound Message Interrupt Status Register (OMISR),” change bits 0 and 1 to w1c. In addition, change user access to “Mixed”as well as in memory map table. In addition, change register description sentence, as follows: “OMIMR can be read from the CSB or the PCI bus, but it can be cleared only from the PCI bus.” to “OMIMR can be read from the CSB or the PCI bus, but it can be written only from the PCI bus.”
13.4.8.1/13-10	In Table 13-11, “DMAMR*n* Field Descriptions,” modify entire PRC bit field description (bits 11–10), as follows: “00 1 PCI read 01 2 PCI read line 10 4 PCI read multiple 11 8 Reserved” Also, change TEM bit description to the following: “0 The DMA halts when a transfer error occurs. 1 The DMA completes the transfer regardless of whether a transfer error occurs. Regardless of the setting of TEM, if an error condition was detected during the DMA transfer, it will cause DMASRn[TE] to be set.”
13.4.8.2/13-12	In Table 13-12, “DMASRn Field Descriptions,” change CB bit description to the following: “It is cleared as a result of any of the following conditions: an error or completion of the DMA transfer.” In addition, change TE bit description to the following: “Set when there is an error condition during the DMA transfer.”
14.3.2.11/14-23	Add the following text: “Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.”
Chapter 15/15-1	Add note to introductory section saying, “The PCI Express engine does not support misaligned byte transfers. It must be DWORD aligned to the CSB bus.”
15.1/15-1	Update final paragraph to show that inbound I/O transactions are not supported.
15.3.1/15-5	Add block base address for PCI Express 1 and PCI Express 2 to header of Table 15-3, “PCI Express Memory Map.” Update access and reset value for PCI Express BIST register, PCI Express subsystem vendor ID register, and PCI Express subsystem ID register. Remove PCI Express subsystem vendor ID update register. Add “EP mode only” to base address registers 0 and 1.

15.4.1/15-15	Add note to end of section stating that the byte ordering for CSB bridge registers is little-endian.
15.4.1.1/15-15	In Table 15-5, "PCI Express Device ID Register Field Description," remove MPC8379 and MPC8379E from device ID list.
15.4.1.3/15-16	In Table 15-6, "PCI Express Command Register Fields Description," update bit 1 (MS) and bit 2 (BM) field descriptions to show bits work similarly in both EP/RC modes.
15.4.2.2/15-24	Update the access, reset value, and description of the PCI Express subsystem vendor ID register (0x2C). Add cross-reference to the PCI Express subsystem vendor ID Update register in Table 15-16, "PCI Express Subsystem Vendor ID Register Fields Description."
15.4.2.3/15-25	Update the access, reset value, and description for Figure 15-17, "PCI Express Subsystem ID Register."
15.4.2.6/15-26	In Table 15-20, "PCI Express Interrupt Pin Register Fields Description," update values in field description to show value for INTA only.
15.4.3.18/15-36	In Table 15-39, "PCI Express Interrupt Pin Register Fields Description," update values in field description to show value for INTA only.
15.4.4.3/15-39	In Table 15-43, "PCI Express Power Management Capabilities Register Fields Description," add descriptions for DSI and Version fields.
15.4.4.7/15-41	Add note to Figure 15-49, "PCI Express Next Capabilities Pointer." Also, modify figure in regard to RC mode.
15.4.4.7/15-41	In Table 15-47, "PCI Express Next Capabilities Pointer Field Descriptions," add the following sentence to bits 0–7: "The reset value is 0x70 in EP mode and 0x00 in RC mode."
15.4.4.8/15-42	In Table 15-48, "PCI Express Capabilities Register Fields Description," add description for ther version and interrupt number fields.
15.4.4.9/15-42	Update maximum payload size to 128 bytes.
15.4.4.12/15-45	Update Figure 15-54, "PCI Express Link Capabilities Register," and add reference to PCI Express Link Capabilities Update Register (PEX_LINKCAP_UPDATE).
15.4.4.15/15-46	Add explanation for how an End Point applications can configure this register with the PEX_SLCAP_UPDATE (gpex_slot_capability_register), offset 0x490. Update the reset value.
15.4.5/15-52	Add note to Figure 15-67, "PCI Express Extended Configuration Space."
15.4.5.1/15-53	Modify Figure 15-68, "PCI Express Advanced Error Reporting Capability ID Register."
15.4.5.11/15-61	Replace "PCI Express Correctable Error Source ID Register" section and "PCI Express Fatal/Non-Fatal Error Source Register" section with the "PCI Express Error Source Identification Register."

15.4.5.4/15-55	Modify Table 15-68, "PCI Express Uncorrectable Error Status Register Field Description."
15.4.5.5/15-56	In Figure 15-72," PCI Express Correctable Error Status Register," change access from "Read/Write" to "w1c."
15.4.6.2/15-63	Add this section.
15.4.6.3/15-64	Add this section.
15.4.6.4/15-65	Add this section.
15.4.6.7/15-68	Add this section.
15.4.6.8/15-68	Add this section
15.4.6.10/15-70	Add this section.
15.4.6.11/15-71	Add this section.
15.4.6.9/15-69	Add this section.
15.4.8.2/15-76	Add this section.
15.5.1/15-77	Add note to end of section stating that the byte ordering for CSB bridge registers is little-endian.
15.5.2.2/15-79	Add "The timer should be programmed to allow sufficient number fo clocks before the DMA tries to fetch the descriptors again" to register description:
15.5.3.1/15-81	Add TC bit to Figure 15-100, "PCI Express Outbound PIO Control Register (PEX_CSB_OBCTRL)," and Table 15-98, "PEX_CSB_OBCTRL Register Fields Description."
15.5.5.1/15-84	Add TC bit to Figure 15-104, "PCI Express Write DMA Control Register (PEX_WDMA_CTRL)," and Table 15-102, "PEX_WDMA_CTRL Register Fields Description."
15.5.5.3/15-85	Add sentence "For additional information see the PEX2 erratum in the errata document of the device" to the note in bit 0 field description in Table 15-104, "PEX_WDMA_STAT Register Fields Description."
15.5.5.4/15-86	Update title for Figure 15-107, "PCI Express Read DMA Control Register (PEX_RDMA_CTRL).
15.5.8.8/15-101	Add bits 16, 18, and 19 to Figure 15-128, "CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR)," and Table 15-126, "PEX_CSMISR Register Fields Description."
15.5.9.1/15-103	Add this section.
15.5.10.1/15-104	In Table 15-128, "PEX_OWAR0–PEX_OWAR3 Register Fields Description," replace last sentence with "This attribute is not applicable and must be cleared for configuration requests, I/O requests, and memory requests that are Message Signaled Interrupts."
15.5.10.2/15-105	Add the following to register description: "Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound

	window, or where an outbound translation window points back into an inbound window, are not allowed.”
15.5.11.1/15-107	Add the following to register description: “Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.”
15.5.12.2/15-109	Add the following to register description: “Inbound and outbound windows for the same bus should not overlap. Therefore, situations where an inbound window translation points back into an outbound window, or where an outbound translation window points back into an inbound window, are not allowed.”
15.5.12.2/15-109	Update bit field range in Figure 15-136, “PCI Express RC Inbound Window Translation Address Register n (PEX_RCIWTAR0–PEX_RCIWTAR3).”
15.5.12.3/15-110	Update bit field range in Figure 15-137, “PCI Express RC Inbound Window Base Address Register Low n (PEX_RCIWBARL0–PEX_RCIWBARL3).”
15.6.1.9/15-115	Replace “Configuration Space Addressing” section with “Configuration Space Access” section and subsections.
15.6.1.9/15-115	Add sentence saying, “The PCI Express controller generates a type 0 configuration when the primary bus number in the host bridge configuration register is equal to or less than the value of the secondary bus number.
15.6.1.10/15-119	In Table 15-141, “PCI Express RC Inbound Message Handling,” update the text and add references to the CSB System Miscellaneous Interrupt Status Register (PEX_CSMISR), CSB System Miscellaneous Interrupt Enable Register (PEX_CSMIER) and the PCI Express Root Error Status Register.
15.6.2.1/15-122	Replace old “EP Interrupt Generation” section and subsections with updated section and subsections.
15.6.2.2.2/15-124	Change all instances of “MSIR” to “MSIIR.”
15.6.5/15-127	Replace old “Hot Reset” section with new, updated “Hot Reset” section.
15.8.1/15-128	In Table 15-145, “DMA Descriptor Bit Fields Description,” update references to “address mapping register” to “outbound window address translation” throughout the table.
15.8.2/15-130	Update “bits” to “bytes” in third paragraph.
15.8.3/15-131	Update the “bits” to “bytes” in fifth paragraph.
15.8.4/15-116	Remove section “Register-Based DMA.”
15.8.4.1/15-132	In the second paragraph, add the following: “Non-contiguous valid descriptors are not supported. If the valid bit of a decriptor in the chain is not set, all of the succeeding descriptors should also have the valid bit as zero.”
15.8.4.4/15-134	Update first bullet so that “the desc_rd_timer field in the DMA control register” now reads “PEX_DMA_DSTMR[DSRT] (see Section 15.5.2.2, “PCI Express DMA Descriptor Timer Register (PEX_DMA_DSTMR)”).”

15.8.4.4/15-134	Update first sentence of fourth paragraph to include the phrase "... it first executes any remaining prefetched valid descriptors..."
16.1/16-1	Add eSATA to features list.
16.3.3.2/16-16	Change bits 10 and 24 to Reserved.
16.3.4.8/16-26	Update Figure 16-24, "PHY Control Configuration Register1 (PhyCtrlCfg1)" and Table 16-24, "PhyCtrlCfg1 Field Descriptions," to show bit 12.
16.3.8/16-34	Change "Modifies the protocol converter control register to place the PHY into loopback mode," to "Modifies the SerDes control register 1 to place the PHY into analog loopback mode."
16.3.8/16-34	Update numbered list.
Chapter 17/17-1	**Significant restructuring and content updating throughout chapter.**
17.1/17-3	Update arrows in Figure 17-1, "SEC Functional Modules."
17.7.1.11/17-69	Update definition of xtime in "Context and Operation for CMAC (OMAC1) Cipher Mode" subsection.
17.3.5/17-30	Update KEU f9 descriptor type in Table 17-10, "Descriptor Format Summary."
17.4.4.2/17-41	Update Figure 17-12, "Channel Status Register (CSR)," to show read/write access.
17.5.4.3/17-54	Add paragraph to note to clarify the RNG Done bit.
17.7.1.11.3/17-74	Add note that AES-CCM does not support zero-length AAD and payload simultaneously.
17.7.3.6/17-101	Update bit numbering of Figure 17-44, "CRCU Control Register," so that the first row reads 0–31 instead of 32–63.
17.7.3.12/17-106	Update the access to "Write only" for Figure 17-50, "CRCU Context Register (Write)."
17.7.3.12/17-106	Update the access to "Read only" for Figure 17-51, "CRCU Context Register (Read-Default Mode)."
17.7.3.12/17-106	Update the access to "Read only" for Figure 17-52, "CRCU Context Register (Read-Raw Mode)."
17.7.8.9/17-160	Add the following note: "Host reads of the RNGB FIFO should be performed on an 8-byte basis, regardless of how many bits of random number is actually required. Partial host reads can leave the RNGB FIFO in a state that result in a channel error."
Chapter 18/18-1	Throughout the chapter, update all TSEC_1588_XX signal names to TSEC_TMR_XX.
Chapter 18/18-1	Also, update all SerDes 2 references to SerDes 1.
Chapter 18/18-1	Also, remove text and references describing extraction of data to allocate in the L2 cache: In Figure 18-25 and Table 18-30, remove EX0–EX7 fields.

Remove text in Section 18.5.3.8.1, “Attribute Register (ATTR).”

In Figure 18-101 and Table 18-105, remove ELCWT and BDLWT fields.

Remove Section 18.5.3.8.2, “Attribute Extract Length and Extract Index Register (ATTRELI) {dma_rx_alocache}.”

Remove text in Section 18.6.6, “Buffer Descriptors.”

18.2/18-2 — Add clarification to 1588 features bullet item as follows: “(1588 not supported in conjunction with SGMII 10/100).”

18.4/18-6 — In Table 18-1, “eTSECn External Signals Network Interface Signal Properties,” update RX_ER description of RGMII and RTBI protocols from “Unused, output driven low” to “Unused.”

Also add TSEC_TMR_PP3 signal row and update signal names.

Also, change L1_SD_TX[n-1] (and complement) to L1_SD_TXA/E, L1_SD_RX[n-1] (and complement) to L1_SD_RXA/E, and change L1_SD_REF_CLK (and complement) to L1_SD_REF_CLK.

18.4.1/18-8 — In Table 18-2, “eTSEC Signals—Detailed Signal Descriptions,” for GTX_CLK125, add SGMII to list of protocols for which GTX_CLK125 is not used.

Also, correct the State Meaning description for the TSEC*n*_CRS, add TSEC_TMR_PP3 signal row, and update signal names.

Also, change L1_SD_TX[*n-1*] (and complement) to L1_SD_TXA/E, L1_SD_RX[*n-1*] (and complement) to L1_SD_RXA/E, and L1_SD_REF_CLK (and complement) to L1_SD_REF_CLK.

In addition, modify the following signal descriptions:

- L1_SD_TX0 and $\overline{\text{L1\_SD\_TX0}}$ to L1_SD_TXA and $\overline{\text{L1\_SD\_TXA}}$
- L1_SD_TXE and $\overline{\text{L1\_SD\_TXE}}$ to L1_SD_TXE and $\overline{\text{L1\_SD\_TXE}}$
- L1_SD_RXA and $\overline{\text{L1\_SD\_RXA}}$ to L1_SD_RXA and $\overline{\text{L1\_SD\_RXA}}$
- L1_SD_RXE and $\overline{\text{L1\_SD\_RXE}}$ to L1_SD_RXE and $\overline{\text{L1\_SD\_RXE}}$

18.5.2/18-12 — In Table 18-4, “Module Memory Map,” update default value of RQFCR and RQFPR registers from “all zeros” to “undefined.”

etsec_bg_body.fm: • Section 18.5.3.1.9, "TBI Physical Address Register (TBIPA) • Section 18.5.3.1.6, "Ethernet Control Register (ECNTRL) • Section 18.5, "Memory Map/Register Definition; Section 18.5.4, "Ten-Bit Interface (TBI)	• added register TBIPA • added missing ECNTRL[TBIM] bit • added text to sections relating RTBI to TBI

18.5.3.1.3/18-23	In Figure 18-4, "IEVENT Register Definition," and Table 18-8, "IEVENT Field Descriptions," add field FGPI, bit 27, with w1c access.
18.5.3.1.3/18-23	Add sub-bullet to third primary bullet in IEVENT register description, defining special function interrupts as FGPI,MSRO, MMWR, and MMRD.
18.5.3.1.4/18-27	In Figure 18-5, "IMASK Register Definition," and Table 18-9, "IMASK Field Descriptions," add field FGPIEN, bit 27.
18.5.3.1.6/18-31	In Figure 18-7, "ECNTRL Register Definition," add bit 26, TBIM. In Table 18-11, "ECNTRL Field Descriptions," and Table 18-12, "eTSEC Interface Configurations," update RMM field description. Also, add note to ECNTRL[R100M] bit description saying, "This bit must be cleared for 1-Gbps SGMII operation." Also added bit TBIM row.
18.5.3.1.9/18-35	Add section.
18.5.3.2.1/18-36	In Table 18-16, "TCTRL Field Descriptions," modify TFC_PAUSE field description.
18.5.3.3.1/18-48	In Table 18-27, "RCTRL Field Descriptions," modify RSF field description.
18.5.3.3.5/18-54	In Table 18-31, "RBIFX Field Descriptions," modify BnCTL = 01 field descriptions to clarify that arbitrary extraction of preamble is not supported in FIFO modes.
18.5.3.3.7/18-56	In Figure 18-28, "Receive Queue Filer Table Control Register Definition," update default value of RQFCR from "all zeros" to "undefined." Also, add field GPI, bit 0.
18.5.3.3.8/18-57	In Table 18-34, "RQFPR Field Descriptions," replace fourth paragraph (beginning with "Packets with a value in the length/type field .... ") with the following:

	"A value in the length/type field greater than 1500 and less than 1536 is treated as a type encoding by the parser. Since no recognized types exist in this range, the controller will not parse beyond the length/type field of any such frame."
18.5.3.3.8/18-57	In Table 18-34, "RQFPR Field Descriptions," add note stating: "Users of the eTSEC parser/filer should be aware of a difference in behavior between rev 1 and rev 2 silicon in cases where the Ethernet type/length field contains a value between 1500 and 1536. In rev 2 silicon, values between 1500 and 1536 are interpreted as a type. Since there are currently no valid types in this range publicly defined by IANA, the controller will not parse beyond the length/type field of any such frame. If the same packet is encountered with rev 1 silicon, parser/filer behavior is different. With rev 1 silicon, such packets are treated as payload length. S/W must confirm the parser and filer results by checking the type/length field after the packet has been written to memory to see if it falls in this range."
18.5.3.3.8/18-57	In Figure 18-29, "Receive Queue Filer Table Property IDs 0, 2–15 Register Definition," update default value of RQFCR from "all zeros" to "undefined."
18.5.3.5.2/18-68	In Table 18-40, "MACCFG2 Field Descriptions," add note to PAD/CRC]field description to include, "This bit must be set when in half-duplex mode (MACCFG2[Full Duplex] is cleared)."
18.5.3.5.4/18-71	In Figure 18-38, "Half-Duplex Register Definition," and Table 18-42, "HAFDUP Field Descriptions," update HAFDUP[collision window] field size from 22–31 to 26–31.
18.5.3.5.4/18-71	In Figure 18-38, "Half-Duplex Register Definition," update MACCFG1[Tx Flow] and MACCFG1[Rx Flow] field descriptions to include "Must be 0 if MACCFG2[Full Duplex] = 0."
18.5.3.5.6/18-72	In Table 18-43, "MAXFRM Descriptions," modify Maximum Frame field description.
18.5.3.5.9/18-74	Update Figure 18-43, "MII Mgmt Control Register Definition," to "Write-only."
18.5.3.6.25/18-91	Update second sentence in Table 18-79, "TBYT Field Descriptions," to read: "This count does not include preamble/SFD or jam bytes, except for half-duplex flow control (back-pressure triggered by TCTRL[THDF]=1). For THDF, the sum total of 'phantom' preamble bytes transmitted for flow control purposes is included in the TBYT increment value of the next frame to be transmitted, up to 65,535 bytes of frame and phantom preamble."
18.5.3.6.26/18-91	Change Table 18-80, "TPKT Field Descriptions," to show field TPKT as 22 bits (bits 10–31).
18.5.3.9/18-108	Add the following note after the third paragraph of this section: "NOTE: IEEE 1588 timestamping is not supported in conjunction with the SGMII 10/100 interface mode."

18.5.3.9.1/18-108	In Table 18-106, “TMR_CTRL Register Field Descriptions,” add “For nanosecond granularity on 1588 timer counter rate, the TCLK_PERIOD should be calculated using the following equation: TCLK_PERIOD = 10^9/Nominal_Frequency.”
18.5.3.9.1/18-108	In Table 18-106, “TMR_CTRL Register Field Descriptions” clarify description of CIPH bit field, stating that selecting an inverted phase input clock is not valid in conjunction with IEEE Std. 1588 functionality. In addition, change description for ETEP2 to: “External trigger 2 edge polarity” ; change description for ETEP1 to: External trigger 1 edge polarity.”
18.5.3.9.9/18-116	In Figure 18-108, “TMR_ACC Register Definition,” update access from “Read only” to “Read/Write.
18.5.3.9.12/18-117	In Figure 18-111, “TMR_ALARM1-2_H/L Register Definition,” update access from “Mixed” to “Read/Write.”
18.5.3.9.13/18-118	In Figure 18-112, “TMR_FIPERn Register Definition,” update access from “Mixed” to “Read/Write.”
18.5.4/18-119	Replace the introductory paragraph with the following text: “This section describes the ten-bit interface (TBI), reduced ten-bit interface (RTBI), and the TBI/RTBI MII set of registers. TBI and RTBI operate in the same manner (the only difference is that RTBI has reduced I/O signalling).”
18.5.4.3.10/18-131	In Table 18-133, “TBICON Field Descriptions,” modify Clock Select field description.
18.6.1.6/18-138	Add this section.
18.6.2.8/18-149	Clarify last sentence of Magic Packet Mode description to include multicast packets.
18.6.2.10/18-150	Clarify three sub-bullets under first primary bullet, noting that anything other than RXB, RXF, TXB, or TXF are classified as “error, diagnostic, or special interrupts.”
18.6.2.11/18-153	Update section.
18.6.2.13/18-153	In Table 18-146, “Reception Errors,” remove the note from the reception errors table entry for parser error.
18.6.4.2.1/18-163	Add the following statement to bulleted list: “The GPI field offers the user the ability to interrupt the core upon matching a rule that causes a frame to be filed to memory. Once the last RxBD corresponding to that frame is written to memory, the IEVENT[FGPI] event is asserted. This bit is set regardless of any interrupt coalescing that may be set.”
18.6.4.2.4/18-165	Add the following two paragraphs to the end of the section: “A functional interrupt is provided via use of the general purpose interrupt (GPI) bit in the filer table. When a property matches the value in the RQPROP entry at this index, and REJ = 0 and AND = 0, the filer will set IEVENT[FGPI] when the corresponding receive frame is written to memory. This allows the user to set up

	a filer rule where the core will be interrupted upon the reception of 'special' frames. If the timer is enabled (TMR_CTRL[TE] = 1), then the interrupt dedicated for timer events (in addition to the usual recieve, transmit and error interrupts) will be asserted."
18.6.5.2/18-171	Remove Parser/Data Extraction Logic from Figure 18-136, "1588 Timer Design Partition."
18.6.5.5/18-173	Add this section and corresponding subsections.
18.6.5.6/18-175	Add this section and corresponding subsections.
18.6.5.4.1/18-173	Update section, as follows: "The eTSEC receive filer has been enhanced with the addition of a general-purpose event bit. This event bit can be used in conjunction with filing table rules to identify 1588 packets and indicate these packets by setting special timer status register bits (TMR_STAT). Additionally, 1588 packets can be easily identified by upper-layer software by using the filer to queue all PTP packets to one or more predefined virtual queues."See Section 18.6.4.1.1, "Filing Rules," for further information."
18.6.6.3/18-182	In Table 18-159, "Receive Buffer Descriptor Field Descriptions," add recommendation to use 64-byte aligned receive buffer pointer addresses to description of Rx Data Buffer Pointer (offset 4–7, bits 0–31).
18.7.1.5/18-199	Add the following note: "SGMII mode utilizes the internal TBI PHY. The internal TBI PHY only auto-negotiates at 1 Gbps. However, 10 Mbps and 100 Mbps speeds are supported in SGMII mode. It is recommended that the external PHY inform the MAC if the desired link speed is not 1 Gbps. Software can perform MII management cycles to determine the external PHY link speed and program ECNTRL and MACCFG2 accordingly."
18.7.1.5/18-199	Add Table 18-172, "SGMII Interface Signal Configuration (4-Wire)."
18.7.1.5/18-199	In Table 18-173, "SGMII Mode Register Initialization Steps," remove references to TBICON[Enable Wrap] and TBICON[Comma Detect].
18.7.1.5/18-199	In Table 18-173, "SGMII Mode Register Initialization Steps," update the value for MIIMCON to "0000_0000_0000_0000_0001_0011_0100_0000" in the "Perform an MII Mgmt write cycle to TBI" row.
19.1.2/19-1	Add "Gen1i, Gen1m, Gen2i, and Gen2m electrical specifications are supported in SATA mode, compliant to *Serial ATA 2.5 Specification*" to list of features.
19.1.3/19-2	Update the following text: with corresponding device names: "The SerDes PHY block supports the following modes of operation: • SerDes1 — Two lanes running x1 SGMII at 1.25 Gbps (MPC8378E) — Two lanes running x1 SATA at 1.5 or 3.0 Gbps (MPC8377E, MPC8379E)

	• SerDes2 — Two lanes running x1 PCI Express at 2.5 Gbps (MPC8377E, MPC8378E) — One lane running x2 PCI Express at 2.5 Gbps (MPC8377E, MPC8378E) — Two lanes running x1 SATA at 1.5 or 3.0 Gbps (MPC8379E)”
19.2/19-3	In Table 19-1, “SerDes External Signals—Detailed Signal Descriptions,” in description for L1_SD_IMP_CAL_RX/L2_SD_IMP_CAL_RX and L1_SD_IMP_CAL_TX/L2_SD_IMP_CAL_TX, correct 200 W and 100 W to 200 Ω and 100 Ω.
19.3.6/19-17	Update bits 4–23 to show as Reserved in Figure 19-8, “SerDesn Reset Control Register (SRDSnRSTCTL),” and Table 19-8, “SRDSnRSTCTL Field Descriptions.”
20.3/20-8	In Table 20-4, “USB Interface Memory Map,” replace the following USBDR register reset values (in memory map and respective register figures): Address offset=0x23104: Value=0x0001_0011 Address offset=0x23124: Value=0x0000_0183 Address offset=0x23140: Value=0x0008_0000 Address offset=0x23164: Value=0
20.3.2.17/20-37–	
20.3.2.23/20-41	Change all ENDPT* register fields from 3-bit to 6-bit.
20.3.2.25/20-43	Update “If AGE_CNT_THRESH is equal to zero, priority state zero is always chosen.” to “ If AGE_CNT_THRESH is equal to zero, priority state one is always chosen.”
20.3.2.27/20-45	In Table 20-37, “SI_CTRL Register Field Descriptions,” update bit 31 (rd_prefetch_val) field description.
20.5.6/20-64	In Figure 20-41, “Queue Head Layout,” change RL field to bits 31–28.
20.6.1/20-70	Remove Table 20-65, “Default Values of Operational Register Space.” In addition, change first paragraph to the following: “After initial power-on or host controller reset (hardware or through USBCMD[RST]), all of the operational registers will be at their default values. After a hardware reset, only the operational registers will be at their default values.”
20.6.1/20-70	Remove the phrase “in the auxiliary power well” in the second sentence of the initial paragraph. Also remove the reset value table.
20.7.1/20-125	In Figure 20-61, “Endpoint Queue Head Layout,” update “Total Bytes” field to 15-bits wide instead of 14-bits wide.
20.7.2/20-128	In Figure 20-62, “Endpoint Transfer Descriptor (dTD),” update “Total Bytes” field to 15-bits wide instead of 14-bits wide.

21.3.1.5/21-9	In Table 21-8, “I2CnDR Field Description,” in DATA field description, modify last sentence to say, “Note that in both master receive and slave receive modes, the very first read is always a dummy read.”
21.5.5/21-23	Remove sentence, “For 1-byte transfers, a dummy read should be performed by the interrupt service routine (see Figure 20-11).”
22.2.2/22-3	In Table 22-2, “DUART Signals—Detailed Signal Descriptions,” for UART_RTS[1:2], update sentence in description from “Can be programmed to be automatically negated and asserted by either the receiver or transmitter” to “Can be programmed to be negated and asserted by either the receiver or transmitter.”
22.3.1.3/22-6	Update calculating percent error value step 1 calculation from “AFI = baud rate × 16” to “AFI = baud rate × 16 × divisor.”
23.2.3.3/23-5	Change beginning of the paragraph below Figure 23-3, “Multiple-Master Configuration,” to read, “The maximum sustained data rate that the SPI supports is input clock/50. However, the SPI can transfer a single character at much higher rates—input clock/4 in master mode and input clock/2 in slave mode, and subjected to the timing parameters of the interconnected devices, and board trace delays.”

Glossary

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this reference manual.

A

Architecture. A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible *implementations*.

Atomic access. A bus access that attempts to be part of a read-write operation to the same address uninterrupted by any other access to that address (the term refers to the fact that the transactions are indivisible). The Power Architecture technology implements atomic accesses through the **lwarx**/**stwcx.** instruction pair.

Autobaud. The process of determining a serial data rate by timing the width of a single bit.

B

Beat. A single state on the bus interface that may extend across multiple bus cycles. A transaction can be composed of multiple address or data *beats*.

Big-endian. A byte-ordering method in memory where the address *n* of a word corresponds to the *most-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 0, 1, 2, 3, with 0 being the *most-significant byte*. See *Little-endian*.

Boundedly undefined. A characteristic of certain operation results that are not rigidly prescribed by the Power Architecture technology. Boundedly-undefined results for a given operation may vary among implementations and between execution attempts in the same implementation.

Although the architecture does not prescribe the exact behavior for when results are allowed to be boundedly undefined, the results of executing instructions in contexts where results are allowed to be boundedly undefined are constrained to ones that could have been achieved by executing an arbitrary sequence of defined instructions, in valid form, starting in the state the machine was in before attempting to execute the given instruction.

Breakpoint. A programmable event that forces the core to take a breakpoint exception.

Burst. A multiple-beat data transfer whose total size is typically equal to a cache block.

Bus clock. Clock that causes the bus state transitions.

Bus master. The owner of the address or data bus; the device that initiates or requests the transaction.

C

Cache. High-speed memory containing recently accessed data or instructions (subset of main memory).

Cache block. A small region of contiguous memory that is copied from memory into a *cache*. The size of a cache block may vary among processors; the maximum block size is one *page*. In Power Architecture processors, *cache coherency* is maintained on a cache-block basis. Note that the term 'cache block' is often used interchangeably with 'cache line.'

Cache coherency. An attribute wherein an accurate and common view of memory is provided to all devices that share the same memory system. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor's cache.

Cache flush. An operation that removes from a cache any data from a specified address range. This operation ensures that any modified data within the specified address range is written back to main memory. This operation is generated typically by a Data Cache Block Flush (**dcbf**) instruction.

Caching-inhibited. A memory update policy in which the *cache* is bypassed and the load or store is performed to or from main memory.

Cast out. A *cache block* that must be written to memory when a cache miss causes a cache block to be replaced.

Changed bit. One of two *page history bits* found in each *page table entry* (PTE). The processor sets the changed bit if any store is performed into the *page*. See also *Page access history bits* and *Referenced bit*.

Clean. An operation that causes a cache block to be written to memory, if modified, and then left in a valid, unmodified state in the cache.

Clear. To cause a bit or bit field to register a value of zero. See also *Set*.

Context synchronization. An operation that ensures that all instructions in execution complete past the point where they can produce an *exception*, that all instructions in execution complete in the context in which they began execution, and that all subsequent instructions are *fetched* and executed in the new context. Context synchronization may result from executing specific instructions (such as **isync** or **rfi**) or when certain events occur (such as an exception).

Copy-back operation. A cache operation in which a cache line is copied back to memory to enforce cache coherency. Copy-back operations consist of snoop push-out operations and cache cast-out operations.

D

Direct-mapped cache. A cache in which each main memory address can appear in only one location within the cache; operates more quickly when the memory request is a cache hit.

Double data rate. Memory that allows data transfers at the start and end of a clock cycle. thereby doubling the data rate.

E

Effective address (EA). The 32-bit address specified for a load, store, or an instruction fetch. This address is then submitted to the MMU for translation to either a *physical memory* address or an I/O address.

Exclusive state. MEI state (E) in which only one caching device contains data that is also in system memory.

F

Fetch. Retrieving instructions from either the cache or main memory and placing them into the instruction queue.

Flush. An operation that causes a cache block to be invalidated and the data, if modified, to be written to memory.

Frame-check sequence (FCS). Specifies the standard 32-bit cyclic redundancy check (CRC) obtained using the standard CCITT-CRC polynomial on all fields except the preamble, SFD, and CRC.

G

General-purpose register (GPR). Any of the 32 registers in the general-purpose register file. These registers provide the source operands and destination results for all integer data manipulation instructions. Integer load instructions move data from memory to GPRs and store instructions move data from GPRs to memory.

Guarded. The guarded attribute pertains to out-of-order execution. When a page is designated as guarded, instructions and data cannot be accessed out-of-order.

H

Harvard architecture. An architectural model featuring separate caches and other memory management resources for instructions and data.

I

Illegal instructions. A class of instructions that are not implemented for a particular processor. These include instructions not defined by the architecture. In addition, for 32-bit implementations, instructions that are defined only for 64-bit implementations are considered to be illegal instructions. For 64-bit implementations instructions that are defined only for 32-bit implementations are considered to be illegal instructions.

Implementation. A particular processor that conforms to the architecture, but may differ from other architecture-compliant implementations for example in design, feature set, and implementation of *optional* features.

Inbound ATMU windows. Mappings that perform address translation from the external address space to the local address space, attach attributes and transaction types to the transaction, and map the transaction to its target interface.

In-order. An aspect of an operation that adheres to a sequential model. An operation is said to be performed in-order if, at the time that it is performed, it is known to be required by the sequential execution model.

Integer unit. An execution unit in the core responsible for executing integer instructions.

Inter-packet gap. The gap between the end of one Ethernet packet and the beginning of the next transmitted packet.

Instruction latency. The total number of clock cycles necessary to execute an instruction and make ready the results of that instruction.

K

Kill. An operation that causes a *cache block* to be invalidated without writing any modified data to memory.

L

L2 cache. Level-2 cache. See *Secondary cache*.

Latency. The number of clock cycles necessary to execute an instruction and make ready the results of that execution for a subsequent instruction.

Least-significant bit (lsb). The bit of least value in an address, register, field, data element, or instruction encoding.

Least-significant byte (LSB). The byte of least value in an address, register, data element, or instruction encoding.

Little-endian. A byte-ordering method in memory where the address *n* of a word corresponds to the *least-significant byte*. In an addressed memory word, the bytes are ordered (left to right) 3, 2, 1, 0, with 3 being the *most-significant byte*. See *Big-endian*.

Local access window. Mapping used to translate a region of memory to a particular target interface, such as the DDR SDRAM controller or the PCI controller. The local memory map is defined by a set of eight local access windows. The size of each window can be configured from 4 Kbytes to 2 Gbytes.

M

Media access control (MAC) sublayer. Sublayer that provides a logical connection between the MAC and its peer station. Its primary responsibility is to initialize, control, and manage the connection with the peer station.

Media-independent interface (MII) sublayer. Sublayer that provides a standard interface between the MAC layer and the physical layer for 10/100-Mbps operations. It isolates the MAC layer and the physical layer, enabling the MAC layer to be used with various implementations of the physical layer.

Medium-dependent interface (MDI) sublayer. Sublayer that defines different connector types for different physical media and PMD devices.

Memory access ordering. The specific order in which the processor performs load and store memory accesses and the order in which those accesses complete.

Memory-mapped accesses. Accesses whose addresses use the page or block address translation mechanisms provided by the MMU and that occur externally with the bus protocol defined for memory.

Memory coherency. An aspect of caching in which it is ensured that an accurate view of memory is provided to all devices that share system memory.

Memory consistency. Refers to agreement of levels of memory with respect to a single processor and system memory (for example, on-chip cache, secondary cache, and system memory).

Memory management unit (MMU). The functional unit that is capable of translating an effective (logical) *address* to a physical address, providing protection mechanisms, and defining caching methods.

Modified/exclusive/invalid (MEI). *Cache coherency* protocol used to manage caches on different devices that share a memory system. Note that neither the PowerPC ISA nor the Power ISA definitions specifies the implementation of an MEI protocol to ensure cache coherency.

Modified state. MEI state (M) in which one, and only one, caching device has the valid data for that address. The data at this address in external memory is not valid.

Most-significant bit (msb). The highest-order bit in an address, registers, data element, or instruction encoding.

Most-significant byte (MSB). The highest-order byte in an address, registers, data element, or instruction encoding.

N

NaN. An abbreviation for not a number; a symbolic entity encoded in floating-point format. There are two types of NaNs—signaling NaNs and quiet NaNs.

No-op. No-operation. A single-cycle operation that does not affect registers or generate bus activity.

O

OCeaN. (On-chip network) Non-blocking crossbar switch fabric. Enables full duplex port connections at 128Gb/s concurrent throughput and independent per port transaction queuing and flow control. Permits high bandwidth, high performance, as well as the execution of multiple data transactions.

Outbound ATMU windows. Mappings that perform address translations from local 32-bit address space to the address spaces of, which may be much larger than the local space. Outbound ATMU windows also map attributes such as transaction type or priority level.

P

Packet. A unit of binary data that can be routed through a network. Sometimes packet is used to refer to the frame plus the preamble and start frame delimiter (SFD).

Page. A region in memory. The OEA defines a page as a 4-Kbyte area of memory aligned on a 4-Kbyte boundary.

Page access history bits. The *changed* and *referenced* bits in the PTE keep track of the access history within the page. The referenced bit is set by the MMU whenever the page is accessed for a read or write operation. The changed bit is set when the page is stored into. See *Changed bit* and *Referenced bit*.

Page fault. A page fault is a condition that occurs when the processor attempts to access a memory location that does not reside within a *page* not currently resident in *physical memory*. A page fault exception condition occurs when a matching, valid *page table entry* (PTE[V] = 1) cannot be located.

Page table. A table in memory is comprised of *page table entries*, or PTEs. It is further organized into eight PTEs per PTEG (page table entry group). The number of PTEGs in the page table depends on the size of the page table (as specified in the SDR1 register).

Page table entry (PTE). Data structures containing information used to translate *effective address* to physical address on a 4-Kbyte page basis. A PTE consists of 8 bytes of information in a 32-bit processor and 16 bytes of information in a 64-bit processor.

Physical coding sublayer (PCS). Sublayer responsible for encoding and decoding data stream to and from the MAC sublayer.

Physical medium attachment (PMA) sublayer. Sublayer responsible for serializing code groups into a bit stream suitable for serial bit-oriented physical devices (SERDES) and vice versa. Synchronization is also performed for proper data decoding in this sublayer. The PMA sits between the PCS and the PMD sublayers.

Physical medium dependent (PMD) sublayer. Sublayer responsible for signal transmission. The typical PMD functionality includes amplifier, modulation, and wave shaping. Different PMD devices may support different media.

Physical memory. The actual memory that can be accessed through the system's memory bus.

Pipelining. A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one has completed.

Primary opcode. The most-significant 6 bits (bits 0–5) of the instruction encoding that identifies the type of instruction.

Program order. The order of instructions in an executing program. More specifically, this term is used to refer to the original order in which program instructions are fetched into the instruction queue from the cache.

Protection boundary. A boundary between *protection domains*.

Protection domain. A protection domain is a segment, a virtual page, a BAT area, or a range of unmapped effective addresses. It is defined only when the appropriate relocate bit in the MSR (IR or DR) is 1.

Q

Quad word. A group of 16 contiguous locations starting at an address divisible by 16.

Quiesce. To come to rest. The processor is said to quiesce when an exception is taken or a **sync** instruction is executed. The instruction stream is stopped at the decode stage and executing instructions are allowed to complete to create a controlled context for instructions that may be affected by out-of-order, parallel execution. See *Context synchronization*.

R

rA. The **r**A instruction field is used to specify a GPR to be used as a source or destination.

rB. The **r**B instruction field is used to specify a GPR to be used as a source.

rD. The **r**D instruction field is used to specify a GPR to be used as a destination.

rS. The **r**S instruction field is used to specify a GPR to be used as a source.

Record bit. Bit 31 (or the Rc bit) in the instruction encoding. When it is set, updates the condition register (CR) to reflect the result of the operation.

Reconciliation sublayer. Sublayer that maps the terminology and commands used in the MAC layer into electrical formats appropriate for the physical layer entities.

Reduced instruction set computing (RISC). An *architecture* characterized by fixed-length instructions with nonoverlapping functionality and by a separate set of load and store instructions that perform memory accesses.

Referenced bit. One of two *page history bits* found in each *page table entry*. The processor sets the *referenced bit* whenever the page is accessed for a read or write. See also *Page access history bits*.

Reservation. The processor establishes a reservation on a *cache block* of memory space when it executes an **lwarx** instruction to read a memory semaphore into a GPR.

Reservation station. A buffer between the dispatch and execute stages that allows instructions to be dispatched even though the results of instructions on which the dispatched instruction may depend are not available.

S

Secondary cache. A cache memory that is typically larger and has a longer access time than the primary cache. A secondary cache may be shared by multiple devices. Also referred to as L2, or level-2, cache.

Set (*v*). To write a nonzero value to a bit or bit field; the opposite of *clear*. The term ‘set’ may also be used to generally describe the updating of a bit or bit field.

Set (*n*). A subdivision of a *cache*. Cacheable data can be stored in a given location in one of the sets, typically corresponding to its lower-order address bits. Because several memory locations can map to the same location, cached data is typically placed in the set whose *cache block* corresponding to that address was used least recently. See *Set-associative*.

Set-associative. Aspect of cache organization in which the cache space is divided into sections, called *sets*. The cache controller associates a particular main memory address with the contents of a particular set, or region, within the cache.

Slave. The device addressed by a master device. The slave is identified in the address tenure and is responsible for supplying or latching the requested data for the master during the data tenure.

Snooping. Monitoring addresses driven by a bus master to detect the need for coherency actions.

Snoop push. Response to a snooped transaction that hits a modified cache block. The cache block is written to memory and made available to the snooping device.

Stall. An occurrence when an instruction cannot proceed to the next stage.

Sticky bit. A bit that when *set* must be cleared explicitly.

Superscalar machine. A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

Supervisor mode. The privileged operation state of a processor. In supervisor mode, software, typically the operating system, can access all control registers and can access the supervisor memory space, among other privileged operations.

Synchronization. A process to ensure that operations occur strictly *in order*. See *Context synchronization*.

System memory. The physical memory available to a processor.

T

Tenure. The period of bus mastership. There can be separate address bus tenures and data bus tenures.

Throughput. The measure of the number of instructions that are processed per clock cycle.

Time-division multiplex (TDM). A single serial channel used by several channels taking turns.

Transaction. A complete exchange between two bus devices. A transaction is typically comprised of an address tenure and one or more data tenures, which may overlap or occur separately from the address tenure. A transaction may be minimally comprised of an address tenure only.

Transfer termination. Signal that refers to both signals that acknowledge the transfer of individual beats (of both single-beat transfer and individual beats of a burst transfer) and to signals that mark the end of the tenure.

Translation lookaside buffer (TLB). A cache that holds recently-used *page table entries*.

U

User mode. The operating state of a processor used typically by application software. In user mode, software can access only certain control registers and can access only user memory space. No privileged operations can be performed. Also referred to as problem state.

V

Virtual address. An intermediate address used in the translation of an *effective address* to a physical address.

Virtual memory. The address space created using the memory management facilities of the processor. Program access to *virtual memory* is possible only when it coincides with *physical memory*.

W

Way. A location in the cache that holds a cache block, its tags, and status bits.

Word. A 32-bit data element.

Write-back. A cache memory update policy in which processor write cycles are directly written only to the cache. External memory is updated only indirectly, for example, when a modified cache block is *cast out* to make room for newer data.

Write-through. A cache memory update policy in which all processor write cycles are written to both the cache and memory.